Toward automatic hint generation for logic proof tutoring using historical student data

Tiffany Barnes, John Stamper

University of North Carolina at Charlotte, Computer Science Department,
9201 University City Blvd., Charlotte, NC 28223, USA
tbarnes2@uncc.edu, john@stamper.org

Abstract. We have proposed a novel application of Markov decision processes (MDPs), a reinforcement learning technique, to automatically generate hints for an intelligent tutor that learns. We demonstrate the feasibility of this approach by extracting MDPs from four semesters of student solutions in a logic proof tutor, and calculating the probability that we will be able to generate hints at any point in a given problem. Our results indicate that extracted MDPs and our proposed hint-generating functions will be able to provide hints over 80% of the time. Our results also indicate that we can provide valuable tradeoffs between hint specificity and the amount of data used to create an MDP.

1 Introduction

Logic proof construction is an important skill for both computer science and philosophy. However, in our experience, this topic is of particular difficulty for students, especially in determining a strategy to derive a conclusion from given premises. Once students apply rules that are easy for them, they often become stuck, performing unnecessary steps or giving up when the next step is unclear. In one-on-one tutoring sessions, however, suggesting one or more intermediate goal states helps many students achieve complete proofs.

Our long-term goal is to provide real-time, individualized hints to support on-going student proof construction efforts. In this paper, we present our system for automatically generating strategic hints using historical data and Markov Decision Processes, and the results of two experiments demonstrating the feasibility of automated hint generation. The first is something like a cross-validation study, comparing the hints we can generate using various semesters of data for MDP creation. The second is a simulation of creating MDPs incrementally as students work proofs, and calculating the probability of being able to generate hints as new attempts are added to the MDP.

The Proofs Tutorial is a computer-aided instructional (CAI) tool implemented on NovaNET (http://www.pearsondigital.com/novanet/). This program has been used for practice and feedback in writing proofs in our university-level discrete mathematics courses since 2002. In the tutorial, students type in consecutive lines of a proof, which consist of 4 parts: the premise, reference lines, the axiom used, and the substitutions

which allow the axiom to be applied. After the student enters these 4 parts to a line, the premise, reference lines, axiom, and substitutions are verified. If a mistake is made, a warning message is shown, and the line is deleted (but saved for later analysis). In this work, we examine student solutions to Proof 1, as in Table 1.

Table 1. Sample Proof 1 Solution

	Premise	Line	Reason
1.	a → b		Given
2.	c → d		Given
3.	¬ (a → d)		Given
	¬ a v d	**3**	**rule IM (error)**
4.	a ^ ¬ d	3	rule IM implication
5.	a	4	rule S simplification
	b	**4**	**rule MP (error)**
6.	b	1,5	rule MP modus ponens
7.	¬ d	4	rule S simplification
8.	¬c	2,7	rule MT modus tollens
9.	b ^ ¬c	6,8	rule CJ conjunction

Our goal in this work is to augment the Proofs Tutorial with goal feedback that helps focus student attention on an appropriate next sub-goal. This type of feedback has been shown to improve learning and skill transfer over minimal and condition violation feedback [1]. Since the Proofs Tutorial has been used for several years, we have a large corpus of data to use in building student models from historical data. We create a student model for each problem, and use it to generate intelligent hints. As a new student works a problem, we record his or her sequence of actions as a state. If the current state is matched in the model, and the matched state has a successor closer to the goal, we enable a Hint Button to give contextual help. From the successor state with the highest reward value, we derive a hint sequence: 1) indicate a goal expression to derive, 2) indicate the rule to apply next, 3) indicate the premises (lines) where the rule can be used, and 4) a bottom-out hint combining 1-3.

Table 2 shows an example hint sequence, generated using the solution in Table 1 for a student solving proof 1 and requesting a hint after line 3. If a student's state is not found in the model, the Hint Button will be disabled. Such a student will not get goal feedback. However, we can add the student's action and its correctness to our database, and periodically run reinforcement learning to update the reward function values. Before an update is applied, we could test the update by examining new MDP states to ensure that unusual solutions have not superseded existing good solutions.

Table 2: Example hint sequence derived from example student solution

Hint #	Hint Text
1	Try to derive: a ^ ¬ d
2	Use line 3, ¬ (a → d) to derive it
3	Use the rule: IM, implication
4	Enter a ^ ¬ d with ref. line 3 and rule IM implication

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 [2]. 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 [3]. However, constraint-based tutors can only provide condition violation feedback, not goal-oriented feedback, that has been shown to be more effective [1].

Example-based authoring tools such as CTAT use demonstrated examples to learn ITS production rules [4]. 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) [5]. 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 amount of time students took to solve arithmetic problems, and to adapt instruction to minimize this time while meeting teacher-set instructional goals [6]. 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 [7].

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 [8], 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 with expert-derived rules and generated buggy rules that surprised experts. In [9], we applied the technique to visualize student proof approaches to allow teachers to identify problem areas for students.

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 Markov decision processes to create student models

A Markov decision process (MDP) is defined by its state set S, action set A, transition probabilities P, and a reward function R [10]. 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 [10]. 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':

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

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 [8]. 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.

4 Method

We use historical data to estimate the availability of hints using different types of state-matching functions and differing datasets for training. We use data from the four fall semesters of 2003-2006 (denoted f3-f6), where an average of 220 students take the discrete math course each fall. Students in this course are typically engineering students in their 2nd or 3rd years, but most have not been exposed to a course in logic. Students attend several lectures on logic and then use the Proofs Tutorial to solve 10 proofs. Sixty percent of students used direct proof when solving proof 1. We extracted 537 of students' first attempts at direct solutions to proof 1.

The data were validated by hand, by extracting all premises generated by students, and removing those that 1) were false or unjustifiable, or 2) were of improper format. We also remove all student steps using axioms Conjunction, Double Negation, and Commutative, since students are allowed to skip these steps in the tutorial. After cleaning the data, there were 523 attempts at proof 1. Of these, 381 (73%) were complete and 142 (27%) were partial proofs, indicating that most students completed the proof. The average lengths, including errors, were 13 and 10 steps, respectively, for completed and partial proofs. When excluding errors and removed steps, the average number of lines in each student proof is 6.3 steps.

The validation process took about 2 hours for an experienced instructor, and could be automated using the existing truth and syntax-checking program in our tutorial. We realized that on rare occasions, errors are not properly detected in the tutorial (less than 10 premises were removed). We plan to correct this in future work.

We performed two experiments to explore the capability of our method to generate automated hints. In each experiment, we isolated the data into training and test sets, where the training set was used to generate the Markov Decision Process (MDP) as described above, and the test set was used to explore hint availability. The process for comparing the test set to the MDP consists of several steps. Because of the structure of the tutorial, we first removed all error states from the MDP and from student attempts before comparison, since the tutorial provides error messages and deletes the corresponding error from the student proof. Then, each attempt in the test set is mapped onto a sequence of states. For each test state, there are two requirements for a hint to be available: 1) there must be a "matching" state in the MDP, and 2) the "matching" state must have a successor state in the MDP (i.e. it cannot be a dead end). The closer the match between a test state and the corresponding MDP state, the more context-specific the hint based on that match will be.

4.2 Matching functions

To maximize the probability that our generated hints are in line with a student's current strategy, we seek to give hints based on states very similar to the current state. We considered four matching functions: 1) ordered (exact), 2) unordered, 3) ordered minus the latest premise, and 4) unordered minus the latest premise. An ordered, or exact, state match means that another student has taken the same sequence of steps in solving the proof. An unordered state match means that there is a state with exactly the same premises, but they were not necessarily reached in the same order. An "ordered-1" match looks for an exact match between the student's previous state and an MDP state. An "unordered-1" match looks for an unordered match between the student's previous state and an MDP state. Once a match is made, we generate a hint using the optimal successor state from the matching state. The more specific the match, the more contextualized the hint. Hints generated using unordered matches will reveal steps taken by other students in the same problem state, but who might be using a different approach to problem solving, so these hints may differ from hints based on ordered matches.

To determine hint availability, we calculated two numbers for each match type. The first is "***move matches***": the percentage of test states, or "moves", including duplicates, with matches in the MDP. The second is the "***unique matches***": the percentage of unique test states that have matches in the MDP. *Move matches* gives us a measure of the probability that a hint is available for each move. *Unique matches* reflects the percent overlap in test and training sets, and could indicate if one class is particularly different from the training set.

5 Experiment 1: Comparing classes

In this experiment, we explored the ability of our system to provide hints using one, two, three, or four semesters of data to build MDPs. Similar to a cross-validation study, each semester is used as a test set while all the remaining semesters are used as training sets for MDPs. This experiment provides us insight into the number of semesters of data we might need to provide hints a reasonable percentage of the time while students are solving proofs. Table 3 presents the data for each semester. Semester f5 was unusual: there were a small number of states, but a large number of moves, suggesting that students solved this proof in very similar ways.

Table 3. Semester data, including attempts, moves, and states in the MDP for each semester

Semester	# Attempts	MDP states	# Moves
f3	172	206	711
f4	154	210	622
f5	123	94	500
f6	74	133	304

We hypothesized that we could provide hints a majority of the time using just one semester as our MDP training data. Table 4 shows the percent ordered matches between each semester and the remaining combinations of training sets. We were very encouraged by these data, suggesting that our system would provide highly contextualized hints over sixty-six percent of the time, in the worst case, after just one semester of training. In all cases, adding more data increased the probability of providing hints, though we do see diminishing returns when comparing the marginal increase between 1-2 (6.8%) and 2-3 (2.8%) semesters of data.

Table 4. Average % move matches across semesters using the ordered test sets and MDPs

Test set	1-sem. MDPs	2-sem. MDPs	3-sem. MDPs
f3	68.73%	75.67%	78.62%
f4	69.77%	77.71%	81.03%
f5	86.33%	90.80%	92.00%
f6	66.34%	74.12%	77.63%
Average	**72.79%**	**79.57%**	**82.32%**

Our experiments using the remaining matching techniques showed consistent increases going from 1-semester MDPs up to 2-semester MDPs, as expected. However, the increases between 2- and 3-semester MDPs are decreasing, suggesting consistent diminishing returns for adding more data to the MDPs.

Table 5 lists the average percent matches for each of our experiments using the four match functions. This table gives an indication of the tradeoffs between using multiple semesters of data versus multiple techniques for matching. Here, we see that, on average, for 72% of moves, we can provide highly contextualized (ordered) hints using just one semester of data. With two semesters of data, we can provide

these hints almost 80% of the time, but this only increases to 82% for three semesters of data. If we wished to provide hints after collecting just one semester of data, we could provide less contextualized hints for those who don't have ordered matches in the MDP. There is a nearly identical increase in the match rate, to almost 80%, by providing hints using either unordered or ordered-1 searches. We can provide hints an additional five percent of the time if we add the unordered-1 match function.

When analyzing these data, we observed a skew in all statistics because of the unusual distribution of states and moves in f5. We therefore repeated all experiments excluding f5, and the results are given in Table 6. The differences caused by skew in f5 had a smaller effect moving from top left to bottom right, suggesting that more data or less sensitive matching can mitigate the effect of unusual training data.

Table 5. Comparison of % move matches across multiple semesters and matching techniques

Matching	1-sem. MDPs	2-sem. MDPs	3-sem. MDPs
Ordered	72.79%	79.57%	82.32%
Unordered	79.62%	85.22%	87.26%
Ordered-1	79.88%	87.84%	91.57%
Unordered-1	85.00%	91.50%	93.96%

Table 6. Comparison of % move matches, excluding f5 from all sets

Test set	1-sem. MDPs	2-sem. MDPs
Ordered	70.97%	78.05%
Unordered	78.69%	83.59%
Ord-1	79.02%	87.99%
Unord-1	85.77%	91.86%

Table 7 shows the marginal increase, with ordered as a baseline, of each matching technique for each MDP size, to illustrate the tradeoffs between additional data and matching technique. When considering matching functions, the easiest technical change is from ordered to ordered-1, where one premise is removed from the test state before comparison with the MDP states. In all cases, the probability of providing these hints is higher than that of providing hints based on unordered matches. This is probably because there is some inherent partial ordering in proofs, so only limited benefit is seen from reordering premises. When an ordered hint cannot be matched, it is perhaps more likely that the student has just performed a step that no one else has done before, rather than generating a new ordering of steps, so the benefit of ordered-1 can exceed that of unordered. Providing the unordered search requires us to maintain 2 separate MDPs to make the search more efficient, so there are both time and space tradeoffs to using unordered matching. However, adding unordered-1 after adding unordered provides a very large difference in our capability to provide hints, with little investment in time.

As part of this study we also compared the unique states across semesters, as shown in Table 8. This gives us a measure of the percent overlap between MDPs. Using 3 semesters of data with ordered matching, or using 1 semester of data with

unordered-1 matching, both give us over 50% matching of states across MDPs. When compared with the much higher move matches, this suggests that although a new semester may bring many more different solution steps, the ones actually used for complete solutions already exist and are those most often used by students.

Table 7. Marginal increases when comparing matching techniques to ordered

Technique	1-sem. ordered	2-sem. ordered	3-sem. ordered
Unordered	6.83%	5.65%	4.94%
Ordered-1	7.09%	8.27%	9.25%
Unordered-1	12.21%	11.93%	11.64%

Table 8. Unique state % matches across semesters and techniques

Test set	1-sem. MDPs	2-sem. MDPs	3-sem. MDPs
Ordered	34.55%	45.84%	51.93%
Unordered	43.62%	55.23%	59.90%
Ordered-1	48.25%	63.07%	71.39%
Unordered-1	57.28%	71.98%	77.87%

6 Experiment 2: Exploring the "cold start" problem

One critique of using data to generate hints has been the expected time needed for the method to be applied to a new problem, or in other words, the "cold start" issue. Our hypothesis was that a relatively low number of attempts would be needed to build an MDP that could provide hints to a majority of students. One method for building our hint MDP would be to incrementally add MDP states as students solve proofs. This experiment explores how quickly such an MDP is able to provide hints to new students, or in other words, how long it takes to solve the cold start problem. For one trial, the method is given in Table 9.

Table 9: Method for one trial of the cold-start simulation.

1. Let Test = {all 523 student attempts}
2. Randomly choose and remove the next attempt a from the Test set.
3. Add a's states and recalculate the MDP.
4. Randomly choose and remove the next attempt b from the Test set.
5. Compute the number of matches between b and MDP.
6. If Test is non-empty, then let a:=b and go to step 3. Otherwise, stop.

For this experiment, we used the ordered and unordered matching functions, and plotted the resulting average matches over 100,000 trials, as plotted in Figure 1. These graphs show a very quick rise in ability to provide hints to students, that can be fit using power functions, whether using ordered or unordered MDP states and matching.

Clearly, the availability to give hints ramps up very quickly. Table 10 lists the number of attempts needed in the MDP versus target hint percentages. For the unordered matching function, the 50% threshold is reached at just 8 student attempts and the 75% threshold at 49 attempts. For ordered matching, 50% occurs on attempt 11 and 75% on attempt 88. These data are encouraging, suggesting that instructors using our MDP hint generator could seed the data to jump-start new problems. By allowing the instructor to enter as few as 8 to 11 example solutions to a problem, the method might already be capable of automatically generating hints for 50% of student moves.

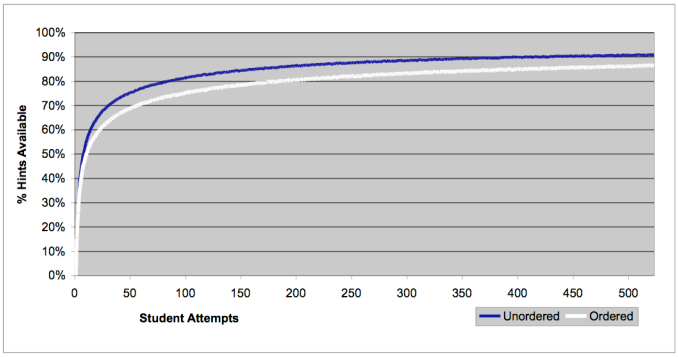

Figure 1. Percent hints available as attempts are added to the MDP, over 100,000 trials

Table 10. Number of attempts needed to achieve threshold % hints levels

	50%	55%	60%	65%	70%	75%	80%	85%	90%
Un-Ordered	8	11	14	20	30	46	80	154	360
Ordered	11	15	22	33	55	85	162	362	?

7 Conclusions and future work

We have proposed and explored the feasibility of an approach to mining Markov decision processes from student work to automatically generate hints. This approach 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. In addition, the generated hints are not created by hand as in example-based tutors, but created based on past student work. Our novel MDP-based approach enables us to automatically provide highly contextual hints, and also allow our knowledge model to 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 hints can be generated.

In our future work, we plan to empirically validate our findings with actual students, and to measure the impact of our generated hints on learning. We will

continue to explore ways to learn general rules to build intelligent hints, feedback and help with greater coverage and robustness. For instance, we plan to group students according to their class performance and behavior in solving proofs, and create tailored MDPs for each group of students. In [8], we proposed three modified reward functions that may benefit different types of students. The first, used in this experiment, is the "expert" function that rewards solutions with fewer steps and errors. This is most related to existing knowledge-based help systems that emphasize efficiency, and the steps in expert solutions are probably most understandable to high-performing students. Other students may better understand less efficient approaches that have been taken by many students, inspiring our second reward function that suggests steps that many students have taken in successful past solutions. On the other hand, at-risk or highly frustrated students may benefit from hints that help them avoid complex or error-prone solutions as in our least-error prone reward function.

We plan to explore machine learning techniques to generalize our models to other problems in the same domain. We also plan to apply our MDP approach to less procedural domains, where creating goal feedback may be difficult, but providing insight into prior student solutions may help current students.

References

1. McKendree, J.: Effective Feedback Content for Tutoring Complex Skills. *Human-Computer Interaction* 5(4), pp. 381—413. Lawrence Earlbaum (1990)
2. Murray, T.: Authoring intelligent tutoring systems: An analysis of the state of the art. *Intl. J. Artificial Intelligence in Education*, 10: 98-129. (1999)
3. Mitrovic, A., Koedinger, K. & Martin, B.: A comparative analysis of cognitive tutoring and constraint-based modeling. *User Modeling*: 313-322. (2003)
4. 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)
5. 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)
6. 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)
7. Merceron, A. & Yacef, K.: Educational Data Mining: a Case Study. In *12th Intl. Conf. on Artificial Intelligence in Education*, Amsterdam, The Netherlands,IOS Press (2005)
8. 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)
9. 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)
10. Sutton, S. &Barto, A.: *Reinforcement Learning: An Introduction*. MIT Press, Cambridge (1998)