Enhancing the Automatic Generation of Hints with Expert Seeding

John Stamper[1], Tiffany Barnes[2], Marvin Croy[3]

Carnegie Mellon University, Human-Computer Interaction Institute, Pittsburgh, PA[1]
University of North Carolina at Charlotte, Department of Computer Science, Charlotte, NC[2]
University of North Carolina at Charlotte, Department of Philosophy, Charlotte, NC[3]
john@stamper.org[1], tiffany.barnes@gmail.com[2], mjcroy@uncc.edu[3]

Abstract. The Hint Factory is an implementation of our novel method to automatically generate hints using past student data for a logic tutor. One disadvantage of the Hint Factory is the time needed to gather enough data on new problems in order to provide hints. In this paper we describe the use of expert sample solutions to "seed" the hint generation process. We show that just a few expert solutions give significant coverage (over 50%) for hints. This seeding method greatly speeds up the time needed to reliably generate hints. We discuss how this feature can be integrated into the Hint Factory and some potential pedagogical issues that the expert solutions introduce.

Key Words: Educational data mining, Markov decision process

1 Introduction

The goal of the Hint Factory is 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 computer aided instruction (CAI) without significantly rewriting the existing software. The Hint Factory is a novel technique that uses a Markov decision process (MDP), created from past student data, to generate specific contextualized hints for students using CAI.

We seek to make our data-driven methods effective quickly. One criticism of data-driven techniques is the amount of time it takes to achieve results for a new problem with no data. Although we have previously addressed this issue with a cold start analysis [1], this research provides an additional method to speed up hint giving capabilities. For this method, an expert (or experts) "seeds" the matrix by completing examples to new problems and using these examples to create the initial MDPs. The experiments presented here focus on how well hints can be provided from an initial expert seeding of the MDP. We hypothesized that hints derived from expert solutions could be used to provide hints in 50% of historical student solution steps. The expert time needed is quite low to achieve this level of hint coverage.

Our primary research implementation of the Hint Factory has been in a tutor to teach deductive logic in Philosophy and discrete mathematics at the college level [5].

In addition to the seeding experiment, this analysis examines additional problems in the logic domain order to further validate our previous work [1]. The analysis of the additional logic problems resulted in similar hint coverage to the problem (NCSU Proof 1) previously studied. This confirms our belief that the method is robust and effective in the logic domain.

2 Background and Related work

Historically, the research and development of intelligent tutors have relied on subject area experts to provide the background knowledge to give hints and feedback. Both cognitive tutors and constraint based tutors rely on "rules" that experts create [9]. This is a time consuming process, and requires the experts to not only understand the subject material, but also to understand the underlying processes used to give help and feedback. We believe that the development of intelligent tutors can be enhanced by using data collected from students solving problems. The amount of data being collected from CAI continues to grow at an exponential rate. Large data repositories like the PSLC DataShop have been created to store and analyze this data [7]. Data-driven methods applied to such large data repositories can enable the rapid creation of new intelligent tutoring systems, making them accessible for many more students.

Others have used collected student data with machine learning 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 [4]. Student data has been used to build initial models for an ITS, in an approach called Bootstrapping Novice Data (BND) [10]. Although the BND approach saves time in entering example problems, it still requires expert instructors and programmers to create a tutor interface and annotate the extracted production rules with appropriate hints. Similar to the goal of BND, we seek to use student data to directly create student models for an ITS. However, instead of using student behavior data to build a production rule system, our method generates MDPs that represent all student approaches to a particular problem, and use these MDPs directly to generate hints. RomanTutor is a ITS developed to teach astronauts to operate a robot arm on the International Space Station [11]. This tutor uses sequential pattern mining (SPM) over collected data to find the best sequence of steps at any given point. In this ill-defined domain, data mining has proved to be an effective way to provide feedback where the number of possible combinations would be too immense for experts to cover. SimStudent is an agent based tool for building student knowledge models by example [12]. SimStudent has been used with student log data to build a model that predicts student knowledge.

Our research using visualization tools to explore generated hints based on MDPs extracted from student data verified that the rules extracted by the MDP conformed to expert-derived rules and generated buggy rules that surprised experts [3]. Croy, Barnes, and Stamper applied the technique to visualize student proof approaches to allow teachers to identify problem areas for students. Barnes and Stamper 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 [1]. Our results indicated that extracted MDPs and our proposed hint-generating functions were able to provide hints over 80% of the time. The results also indicated that we can provide valuable tradeoffs between hint specificity and the amount of data used to create an MDP. The MDP method was successfully implemented into the Deep Thought logic tutor as part of the Hint Factory in a live classroom setting [2]. Fossati and colleagues have used the MDP method in the iList tutor used to teach linked lists and deliver "proactive feedback" based on previous student attempts [6].

One ITS authoring tool, CTAT, was given a feature to use demonstrated examples to learn ITS production rules [8]. 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. In many ways this is similar to the seeding approach presented here, but in our approach the expert need not supply the hints. Additionally, the example tracing tutors will only have the knowledge that the expert has added, while our methods would allow the tutor to continue to improve as additional expert or student problem attempts are added to them model. Finally, our method computes a value for problem states automatically, and uses this value to make decisions on which path to suggest even when multiple choices are reasonable. This ability to differentiate between several good solutions based on the specific context of student's current state remains the strong point of the Hint Factory.

3 Markov decision processes to create student models

The Hint Factory consists of the MDP generator and the hint provider. The MDP generator is an offline process that assigns values to the states that have occurred in student problem attempts. These values are then used by the hint provider to select the next "best" state at any point in the problem space.

A Markov decision process (MDP) is defined by its state set S, action set A, transition probabilities T: $S \times A \times S \rightarrow [0,1]$, and a reward function R: $S \times A \times S \rightarrow \Re$ [13]. The goal of using an MDP is to determine the best policy, or set of actions students have taken at each state *s* that maximize its expected cumulative utility (V-value) which corresponds to solving the given problem. The expected cumulative value function can be calculated recursively using equation (1). For a particular point in a student's logic proof, a state consists of the list of statements generated so far, and actions are the rules used at each step. Actions are directed arcs that connect consecutive states. Therefore, each proof attempt can be seen as a graph with a sequence of states connected by actions.

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. Next, value iteration 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), resulting in a bias toward short, correct solutions such as those an expert might derive. We apply value iteration

using a Bellman backup to iteratively assign values *V(s)* to all states in the MDP until the values on the left and right sides of equation (1) converge [13]. The equation for calculating the expected reward values *V(s)* for following an optimal policy from state *s* is given in equation (1), where *R(s,a)* is the reward for taking action *a* from state *s*, and $P_a(s, s')$ is the probability that action *a* will take state *s* to state *s´*. $P_a(s, s')$ is calculated by dividing the number of times action *a* is taken from state *s* to *s´* by the total number of actions leaving state *s*.

$$V(s) := \max_a \left(R(s,a) + \sum_{s'} P_a(s,s') \; V(s') \right) \quad (1)$$

Once value iteration is complete, the optimal solution in the MDP corresponds to taking an expert-like approach to solving the given problem, where from each state the best action to take is the one that leads to the next state with the highest expected reward value [2].

4 Method

We use historical data to estimate the availability of hints using the MDP and seeding approaches. We performed this experiment using student attempts at the Proof Tutorial problems 1-4, as given in Table 1, in the NC State University discrete math course from fall semesters 2003-2006. The givens are the premises that students will use to prove the conclusion in the tutorial. Before using the Proofs Tutorial as homework, students attend several lectures on propositional logic and complete fill-in-the-blank proofs.

Table 1. Description of Proofs Tutorial problems 1 through 4

Problem	Givens	Conclusion
1	If A then B, If C then D, not(If A then D)	B and not C
2	If A then B, If not C then D, not B or not D	If A then C
3	If (B or A) then C	If A then (If B then C)
4	A or (If B then C), B or C, If C then A	A

We generated an MDP for each semester of data separately, and Table 2 shows the number of states generated and number of total moves generated from each problem during each of the four semesters. The number of states represents the number of unique steps that were seen over all problem attempts, while the number of moves represents all student steps or state-action pairs. The number of moves gives a more accurate reflection of class behavior, and comparing states to moves gives a notion of how much repetition occurs in the dataset. Note that problem 1 was used in the original validation experiments [1]. From the table, several clear trends can be seen. First, the total number of attempts, states, and moves are lower in the Fall 2005 and significantly lower in Fall 2006 semesters. Second, problem 4 has significantly fewer

attempts in every semester when compared to the others. According to the course instructor, problem 4 is the hardest problem. The seeding data was provided by two subject area experts, who worked each problem several times, but for less than one hour per problem. An overview of their problem attempts is given in Table 3.

Table 2. Semester data, including attempts, moves, and states in the MDP for each semester

Problem	Semester	# Attempts	MDP states	# Moves
1	f3	172	206	711
1	f4	154	210	622
1	f5	123	94	500
1	f6	74	133	304
2	f3	138	162	628
2	f4	142	237	752
2	f5	105	122	503
2	f6	63	103	279
3	f3	139	145	648
3	f4	145	184	679
3	f5	113	103	577
3	f6	71	94	372
4	f3	103	46	166
4	f4	59	63	103
4	f5	34	30	48
4	f6	33	20	41

Table 3. Expert example seeding attempts, moves, and states for each problem

Problem	# Attempts	MDP states	# Moves
1	3	10	19
2	4	12	27
3	2	15	21
4	3	8	20

To verify our previous results for testing hint availability [1], we performed a cross-validation study, with each semester used as a test set while the remaining semesters are used in training sets for MDPs. Hints are available for a particular state in the test set if the MDP contains that state with a path to the goal state. We count these matching states for each move as "move matches." Table 4 shows the average percent move matches between each semester and the remaining combinations of training sets using one, two, and three semesters of data to construct MDPs. On average, one-semester source MDPs match 71.46% of the valid moves in a new semester of data. With two semesters of data the average move coverage reaches 77.32% for a 5.86% marginal increase. Adding a third semester of data results in an average coverage of 79.57%, a 2.25% marginal increase over two semesters. All the individual problems show a similar curve where the marginal return decreases after

each subsequent semester. For Problem 4 the total percentage of move matches is approximately 15-20% lower than the other problems, and this occurs since there are significantly fewer attempst on this more difficult problem.

Table 4. Average % move matches across problems comparing test sets and MDPs

Problem	1-sem. MDPs	2-sem. MDPs	3-sem. MDPs
1	72.79%	79.57%	82.32%
2	75.08%	80.58%	82.96%
3	79.01%	83.35%	84.89%
4	58.94%	65.77%	68.09%
Average	**71.46%**	**77.32%**	**79.57%**

Table 3 shows characteristics of the expert examples used for seeding the problem MDPs. Between two and four attempts were available for each of the problems and these attempts generated between 8 and 15 total states. Table 5 shows the results of comparing each semester of test data with the seeded MDPs and one-semester MDPs. Especially in problems 1 and 3, when compared to the semester data, the seeded MDP states were "high impact" states, which included the most used paths to solve the problems by the students. Comparing the unique state matches to move matches shows that although the seeded MDPs match only a small percentage of unique student problem states, they match a lot of the moves taken by students. It is interesting to note the move matches vary much wider than a semester of MDP data. This should be expected considering the small number of seeding attempts used. In fact, with further analysis of the individual problems we see for problems 1 and 3 there are two common solutions, that correspond to the expert seeds, resulting in high move coverage percent rates (62.08% and 53.33% respectively), while problems 2 and 4 have more than two common solutions, which results in much lower, but still promising move coverage considering the small number of expert seed attempts.

Table 5. Average % unique state and move matches for seeded and 1-semester MDPs

	Unique state matches		Move matches	
Problem	**Seeded MDP**	**1-sem. MDPs**	**Seeded MDP**	**1-sem MDPs**
1	6.22%	34.55%	62.08%	72.79%
2	11.40%	34.60%	29.82%	75.08%
3	7.69%	33.36%	53.33%	79.01%
4	12.46%	23.45%	26.57%	58.94%
Average	**9.44%**	**31.49%**	**42.95%**	**71.46%**

5 Revisiting the "cold start" problem

We previously explored how quickly an MDP can be used to provide hints to new students, or in other words, how long it takes to solve the cold start problem, for problem 1 [1]. In this his experiment we compare hint availability for incrementally constructed MDPs starting with no data to those starting with seed data. In both cases, hint availability is calculated for the current student attempt, and their states are then added to the MDP. For one trial, the method is given in Table 6. In this experiment, we calculate the hint availability (move matches) for each consecutive student with seeded and non-seeded MDPs. We repeat this process for 100,000 trials and plot the resulting hint availability curves for problem 3 in Figure 1. The curves for problems 1-4 were all similar, so we present only problem 3 here. Figure 1 shows that seeding shifts the initial starting point of hint availability from 0 to over 50%, giving a boost at the start. By 50 attempts the seeded set is just a few attempts ahead and by 100 attempts the 2 graphs are the same. This shows that seeding helps initial hint coverage, avoiding the steep wait for significant hint coverage when using incrementally constructed MDPs for hints. We note that in the initial boost, the seeded problems are covering "high impact" states, or those that are very frequent in the student data sets.

Table 6. 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.

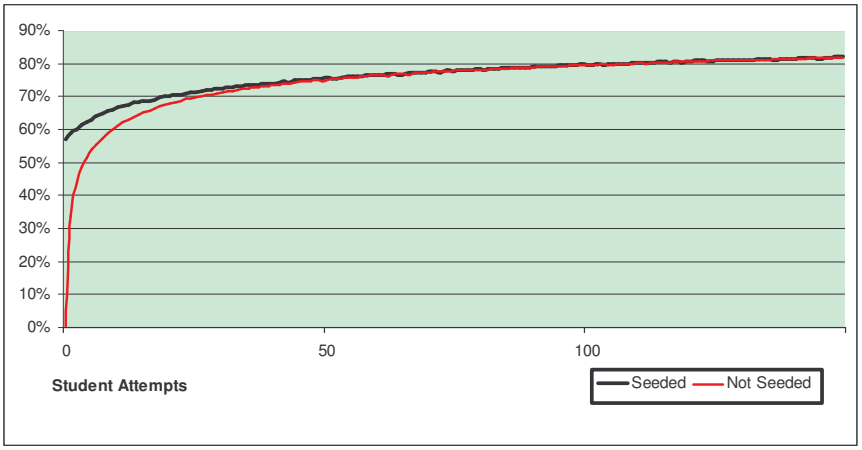

Figure 1. Percent hints available as attempts are added to the MDP, over 100,000 trials for Problem 3

Table 7 shows the number of attempts needed to achieve hint percentage thresholds with and without seeding for each of problems 1-4. Again we see that the seeding of each problem gives an initial boost that fades over time as more student attempts are added, which confirms our hypothesis.

Table 7. Number of attempts needed to achieve threshold % hints levels for seeded and unseeded MDPs constructed incrementally. Note that for problems 1 and 3, 50% hint coverage was achieved with seeds alone.

Problem		50%	55%	60%	65%	70%	75%	80%	85%	90%
1	Not seeded	8	11	14	20	30	46	80	154	360
1	Seeded	seeds	seeds	4	8	21	46	80	155	360
2	Not seeded	9	15	24	36	59	88	149	286	*
2	Seeded	2	3	16	22	46	80	146	286	*
3	Not seeded	5	7	10	16	27	50	110	266	*
3	Seeded	seeds	1	3	8	20	48	110	266	*
4	Not seeded	25	31	54	82	*	*	*	*	*
4	Seeded	12	22	53	80	*	*	*	*	*

(* means the method did not reach this percentage)

6 Conclusions and future work

The main contribution of this paper is to show how expert seeding can enhance the automatic generation of hints by reducing the amount of student data initially needed to effectively deliver hints. Although we believe that our data-driven method already ramps up quickly [1], seeding using expert examples enhances our ability to give hints. For the seeding problems 100% of all the seeded states appeared in each semester of data. Obviously, the educators know the most common solutions to the problems and by seeding the MDPs they can quickly get this data included to help jump-start the hint giving process. Additionally, the experiments presented here replicate and further validate our earlier work in solving the cold start problem.

We do see a few issues with seeding. One issue with using experts to seed the MDP for hint generation is that experts may unintentionally miss a very important solution to the problem. Students who try to solve the problem in this way would not receive hints and therefore may believe that they are doing something wrong. This problem, however, exists in traditional intelligent tutors as well. Further, the solutions that the expert provides will likely become more popular, since students receiving hints will likely use similar approaches to the experts. This is not necessarily a bad thing, but it could limit the ability of MDPs to provide broad coverage of the student solution space – since student solutions might be more limited if they make heavy use of seeded hints. Seeding would likely reinforce the expert solution for a long time to come even as additional data is acquired. To alleviate this problem the instructors can vary which problems receive hints so that clean data with no hints can be collected on every problem at some point. Alternatively, as enough student attempts are added to

the MDPs to generate hints, expert solutions could be removed from the data set to promote diversity in student answers. The seeding approach is very similar to the Bootstrapping Novice Data discussed in the related work section, and we believe it can be useful in many data-driven methods for generating intelligent tutoring capabilities.

In our current and future work, we are using machine learning methods to analyze our MDPs for problem structure and for generating new problems of similar difficulty. We are also building tools for teachers and researchers to visualize MDPs and allow teachers to write their own hints, and modify how the MDPs are used in generating hints.

References

1. Barnes, T., Stamper, J.: Toward Automatic Hint Generation for Logic Proof Tutoring Using Historical Student Data. In E. Aimeur, & B. Woolf (Eds.) Proceedings of the 9th International Conference on Intelligent Tutoring Systems (ITS 2008), pp. 373—382. Berlin, Germany: Springer Verlag (2008)
2. Barnes, T., Stamper, J., Lehmann, L., Croy, M.: A Pilot Study on Logic Proof Tutoring Using Hints Generated from Historical Student Data. In R. Baker, T. Barnes, J. Beck (Eds.) Proceedings of the 1st International Conference on Educational Data Mining (EDM 2008), pp. 197—201. Montreal, Canada. (2008)
3. 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)
4. 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)
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, pp. 145—155, Amsterdam, Netherlands: IOS Publishers (2007)
6. Fossati, D., Di Eugenio, B., Ohlsson, S., Brown, c., Chen, L., Cosejo, D. I learn from you, you learn from me: How to make iList learn from students. In V. Dimitrova, R. Mizoguchi, B. Du Boulay and A. Graesser (Eds.), Proc. 14th Intl. Conf. on Artificial Intelligence in Education, AIED 2009, pp. 186—195., Brighton, UK. IOS Press (2009)
7. Koedinger, K.R., Baker, R.S.J.d., Cunningham, K., Skogsholm, A., Leber, B., Stamper, J.: A Data Repository for the EDM commuity: The PSLC DataShop. To appear in Romero, C., Ventura, S., Pechenizkiy, M., Baker, R.S.J.d. (Eds.) Handbook of Educational Data Mining. Boca Raton, FL: CRC Press. (in press)
8. 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)
9. Mitrovic, A., Koedinger, K. & Martin, B.: A comparative analysis of cognitive tutoring and constraint-based modeling. User Modeling: 313—322. (2003)
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 (2004)

11. Nkambou, R., Mephu Nguifo, E., Fournier-Viger, P.: Using Knowledge Discovery Techniques to Support Tutoring in an Ill-Defined Domain. In E. Aimeur, & B. Woolf (Eds.) Intelligent Tutoring Systems (ITS 2008), pp. 395—405. Berlin: Springer Verlag. (2008)
12. Matsuda, N., Cohen, W. W., Sewall, J., Lacerda, G., & Koedinger, K. R.: Predicting students performance with SimStudent that learns cognitive skills from observation. In R. Luckin, K. R. Koedinger & J. Greer (Eds.), Proceedings of the international conference on Artificial Intelligence in Education, pp. 467—476. Amsterdam, Netherlands: IOS Press. (2007)
13. Sutton, S. & Barto, A.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)