International Journal of Artificial Intelligence in Education 21 (2011) 153–167
DOI 10.3233/JAI-2011-021
IOS Press

Enhancing the Automatic Generation of Hints with Expert Seeding

John Stamper, *Human-Computer Interaction Institute, Carnegie Mellon University*
john@stamper.org
Tiffany Barnes, *Department of Computer Science, University of North Carolina at Charlotte*
tiffany.barnes@gmail.com
Marvin Croy, *Department of Philosophy, University of North Carolina at Charlotte*
mjcroy@uncc.edu

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.

Keywords. Educational data mining, markov decision process, hints

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 (Barnes & Stamper, 2008), this research provides an overview of extensions to our method to speed up hint giving capabilities. Specifically we focus on expert seeding where 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 (Croy, Barnes, & Stamper, 2007). In addition to the seeding experiment, this analysis examines additional problems in the logic domain order to further validate our previous work (Barnes & Stamper, 2008). The analysis of the additional logic problems resulted in similar hint coverage to the problem previously studied. This confirms our belief that the method is robust and effective in the logic domain.

BACKGROUND AND RELATED WORK

Marking student work as right or wrong is a simple form of feedback that can often be automated, but automatically generating effective formative feedback is a much more complex problem. Shute's review of the literature suggests that effective formative feedback be multidimensional and credible, specific but not evaluative, and infrequent but timely (Shute, 2008). Determining the timing and frequency of hints is a particular challenge, but studies suggest that offering hints on demand, instead of proactively, can have positive effects on learning (Razzaq & Heffernan, 2010). While some studies have suggested as much as 72% of help-seeking behaviors can be unproductive (Aleven et al., 2004), Shih's work suggests that some of these behaviors are in fact helpful (Shih, Koedinger, & Scheines, 2008). Shih argues that using help to achieve a bottom-out hint can be seen as looking for a worked example, an effective learning strategy.

Based on the hint and help literature, we devised a strategy of automatically generating hints to be as specific as possible, derived on-demand, and directed to the student's problem-solving goal, to provide the right type of help at the right time. Based on our experience in teaching logic for many years, we have observed that students often know how to execute the steps needed to solve logic proof problems but may have trouble choosing what to do next. These observations confirm that our on-demand, context-specific system could address the needs of students solving logic proof problems, but the research in our current study was needed to evaluate whether our implemented system achieved that goal.

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 (Mitrovic, Koedinger, & Martin, 2003). 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 (Koedinger et al., 2010). 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 (Beck, Woolf, & Beal, 2000). Student data has been used to build initial models for an ITS, in an approach called Bootstrapping Novice Data (BND) (McLaren, et al., 2004). 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 (Nkambou, Nguifo, & Fournier-Viger, 2008). 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 (Matsuda, et al., 2007). 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 (Barnes & Stamper, 2007). 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 (Barnes & Stamper, 2008). 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 (Barnes et al., 2008). 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 (Fossati et al., 2009).

One ITS authoring tool, CTAT, was given a feature to use demonstrated examples to learn ITS production rules (Koedinger et al., 2004). 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.

MARKOV DECISION PROCESSES TO CREATE STUDENT MODELS

The foundation of 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: $S \times A \times S \longrightarrow [0, 1]$, and a reward function : $S \times A \times S \longrightarrow \Re$ (Sutton & Barto, 1998). 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 (Sutton & Barto, 1998). 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 (Barnes et al., 2008).

THE HINT FACTORY FRAMEWORK

The Hint Factory framework consists of two main parts, the MDP generator and the hint provider. The MDP generator is an offline process that implements the MDP method, but the hint provider must be integrated with the CAI. The goal of the framework is to make this integration easy with minimal modifications. In our experiment, the modifications to the existing CAI included tracking the student actions, passing them to the hint provider, and adding a hint button to the interface. Some work must be done to word the hints, but this only needed to be done once.

Deep Thought

Deep Thought is a custom CAI tool that allows students to practice solving logic proofs (Croy, 1999) (Croy, 2000). As shown in Fig. 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 that was created from the MDP generator into memory.

The MDP generator applied to Deep Thought

Each action that a student makes while solving a proof in the Deep Thought tutor generates a transaction in a student log file. The MDP generator is used as an offline process that turns each student solution into a Markov chain consisting of states and actions. All Markov chains for a problem are then combined to create an MDP for that problem containing probabilities based on the frequency of each action in the student logs. Then value iteration is applied using parameters set by the educators creating the MDPs.

A subset of example data from a student log file showing a student solving a Deep Thought problem can be seen in Table 1. Each student step is given a step number. The original statements describe what is seen on the screen when the student starts the step. The conclusion is the statement students are trying to reach, and in the Deep Thought tutor students can work backwards from the conclusion. In Table 1, however, the student has solved the problem in the forward direction which can be seen since the conclusion data does not change. The action describes the rule the student has applied by pressing a button in Deep Thought. The "New" column shows any new statements that have been created. The correct column states whether the step was performed correctly. In Table 1, the student makes one error on step 3, which does not add any new statements. The Solved column is a Boolean field stating whether the student has completed the problem.

When the MDP generator is applied to the data in Table 1, a Markov chain consisting of six states is created. This Markov chain can be seen in Fig. 2. Each state is connected via the action that was used to reach that state. The state information can be seen listed on the right of Fig. 2 and represents the statements that can be seen in the interface when a student is in that state.

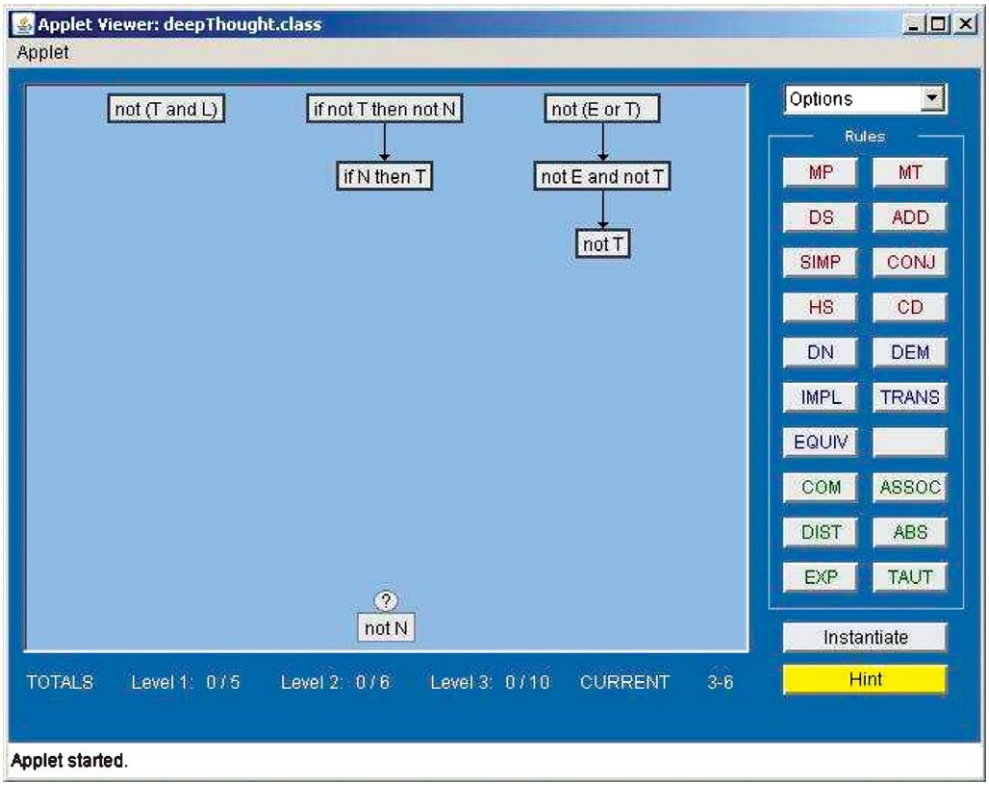

Fig. 1. An example of the Deep Thought Interface, with problem 3.6 partially completed. The statements at the top are "premises" which are given at the problem start, and the student is trying to prove the statement "not N", as denoted with a question mark.

Table 1
Example data showing a student solution to problem 1.2 in the Deep Thought tutor

Step	Original Statements	Conclusion	Action	New	Correct	Solved
1	(FvG)>H,IvF,~I&J	H	SIMP	~I	Y	0
2	(FvG)>H,IvF,~I&J, ~I	H	DS	F	Y	0
3	(FvG)>H,IvF,~I&J, ~I, F	H	HS		N	0
4	(FvG)>H,IvF,~I&J, ~I, F	H	ADD	FvG	Y	0
5	(FvG)>H,IvF,~I&J, ~I, F, FvG	H	MP		Y	1

The hint provider applied to Deep Thought

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.

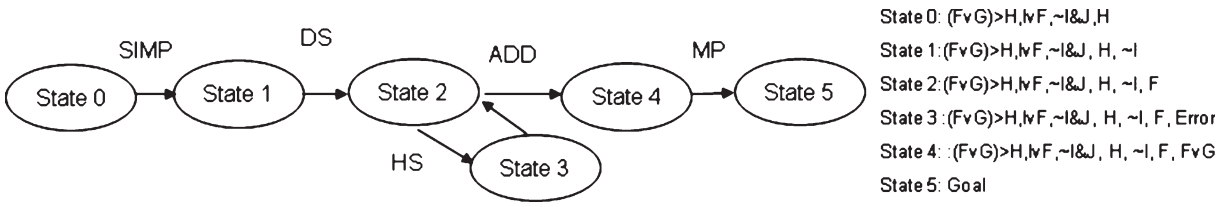

Fig. 2. Markov Chain showing the states and actions created from the example data shown in Table 1.

Table 2
Example hint sequence derived from example student state seen in Fig. 1

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

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 (McKendree, 1999). 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 (VanLehn, 2006).

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 Fig. 1, the hint sequence seen in Table 2 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.

Extending the Hint Factory framework

The previous implementation of the Hint Factory was based on the MDP method. While the method has been successfully used in tutors (Barnes, et al., 2008) (Fossati, et al., 2010), it is currently applied only to specific problems. We have identified several opportunities to improve the ability to give hints even on problems that have no previous data associated with them. We introduce this extended framework as the "Cascading Hint Factory" which can be seen in Fig. 3.

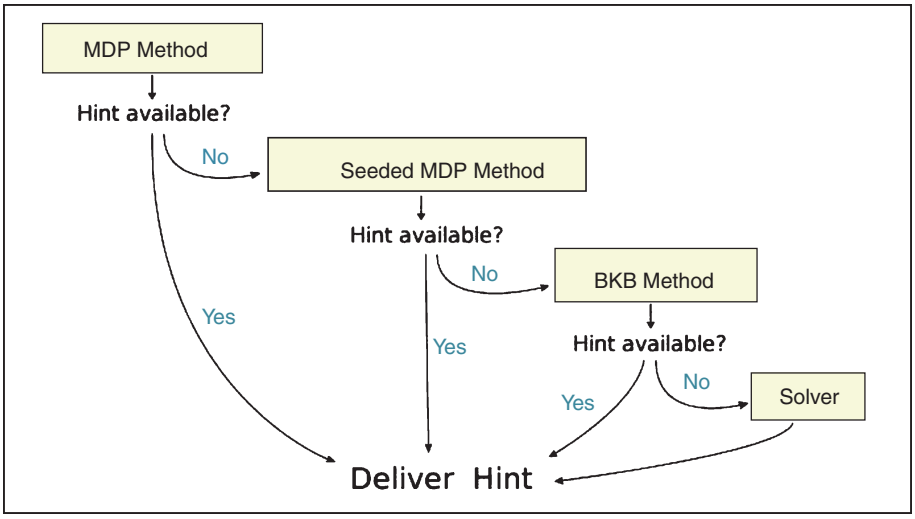

Fig. 3. Cascading Hint Factory framework for automatically generating hints.

The idea of the cascading Hint Factory is to provide a framework for the automatic generation of hints. Starting with our MDP method, which is the most context specific, the system can try and produce a hint from the student model if it exists. If the current student state does not exist in our model the MDP method is unable to give a hint, so our framework looks to the expert generated "seeded MDP" which is described in this paper. The seeded MDP is created using the MDP method, but the data to create the MDP is gathered from experts solving the problems. These expert generated MDPs contain many of the common solutions that experts predict will be seen, but may not support student thought processes as well as the MDP method. If a hint is still not available with the seeded MDP, the framework passes the state on to the Bayesian Knowledge Base (BKB) method (Santos & Santos, 1998). This method is used to expand our individual MDPs into a full corpus of domain knowledge. Bayesian networks (BN) have found use in many areas of research including AI in education. Their main use has been in knowledge tracing, where the probability of knowing a certain skill can be modeled with a Bayesian Network (BN) (Arroyo & Woolf, 2005) (Baker, Corbett, & Aleven, 2008). Bayesian networks have been used in model tracing in the Andes physic tutor, which predicts the strategy that a student is using (Conati et al., 1997). A BKB is a generalization of a BN where each node has independence from the others (Santos & Santos, 1998). BKBs have been used with success in areas where complete knowledge is not available and a full BN cannot be constructed (Haddawy, 1994). Figure 4 shows an example of a partial BN on the left and the resulting BKB records on the right. In our implementation each node in the BKB represents a group of individual model components where each component consists of a state, actions, and new states. These nodes closely resemble rules in rule based tutors.

In order to provide a hint using the BKB the features of the current state are matched against the records in a Bayesian knowledge base that was created from similar problems. If appropriate BKB records exist, they can be used to point to the next best action and a hint can be given. If, however, the BKB fails to match any records for a given state, the cascading hint factory can still give a hint using a low level solver. Solvers exist and can be used in a number of domains such as logic (McCune, & Shumsky, 2000) and Geometry (Bouma et al., 1995). Although the solver can provide a solution path that can be used to give a hints, these hints will be the least context specific and least likely to correspond to the way students learn and solve problems.

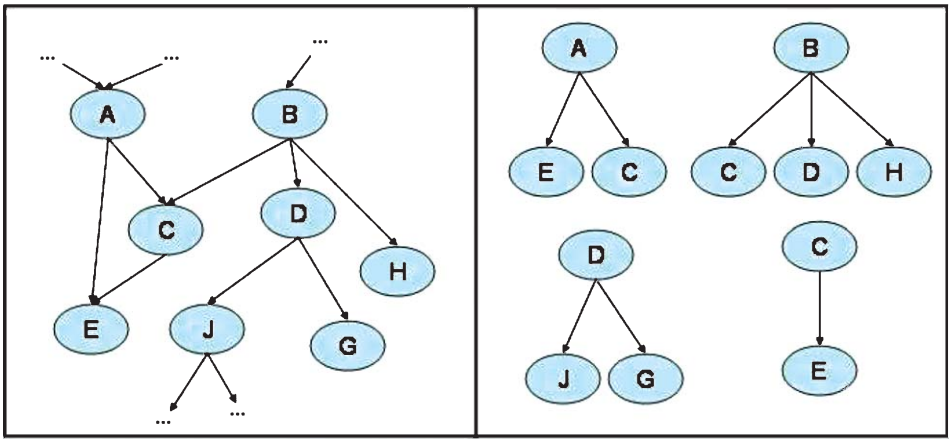

Fig. 4. Partial Bayesian Network (left) and extracted BKB Model Components (right) (state labels are the same for both).

Table 3
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

The Cascading Hint Factory provides extensions to our original Hint Factory to provide a wider range of hint coverage in an autonomous manner. In the remainder of this paper we examine the seeded MDP method which can be used to jump start hint giving capabilities when data to create a traditional MDP or BKB does not exist.

EXPERT SEEDING METHOD

Expert seeding applied to the Hint Factory uses expert solutions to initially generate the MDPs. To generate the solutions the expert only needs to go through the tutor and solve problems in the way they believe students will solve the problems. By using the logs created from the interaction of the expert with the tutor, "seeded" MDPs can be generated. This addition of a few expert solutions allows the system to ramp up much more quickly. We compare MDPs generated with and without these expert solutions added.

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 3, 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.

We generated an MDP for each semester of data separately, and Table 4 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 (Barnes & Stamper, 2008) while the others were newly created for this research. 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 and many students fail to attempt this 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 5.

Table 4
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 5
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 (Barnes & Stamper, 2008), 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 6 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 attempts on this more difficult problem. These results are consistent with our previous research and further show the method is useful in this domain.

Table 5 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 7 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

Table 6
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 7
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%

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.

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 (Barnes & Stamper, 2008). 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 8. 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 Fig. 5. The curves for problems 1-4 were all similar, so we present only problem 3 here. Figure 5 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 8
Method for one trial of the cold-start simulation

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

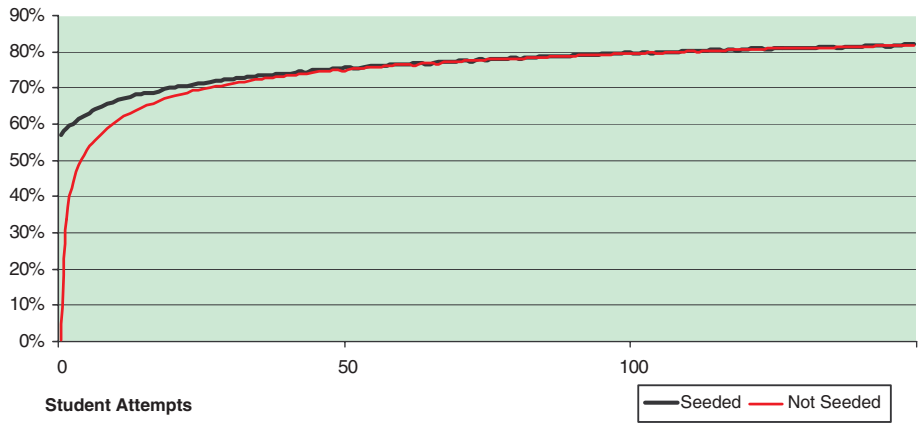

Fig. 5. Percent hints available as attempts are added to the MDP, over 100,000 trials for Problem 3.

Table 9
Number of attempts needed to achieve threshold % hints levels for seeded and un-seeded 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).

Table 9 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.

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 (Barnes & Stamper, 2008), 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 issue on additional problems.

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.

We also showed how expert seeding fits into our Cascading Hint Factory framework. In the cascading system, one or more of our hint generating methods could be employed to generate hints for any encountered

student states. Combined with the ability for teachers to annotate the MDP, this implementation would be more autonomous and provide intelligent hints with a much smaller time investment for development than the development time needed for more traditional intelligent tutors. We envision the cascading hint factory attaching to existing computer aided instruction, collecting student data, building student models, and delivering hints, with little expert human involvement. This framework will facilitate automatic hint generation allows for the quick addition to context-specific hints to existing CAI tools, and the cascading hint factory can also serve as a repository to collect empirical data about learning in the CAI's content domain. This can be used to better understand how students solve problems in the given domain through the implicit domain models being created through the MDP and BKBs.

In our current and future work, we are exploring more extensions to our hint generation techniques to provide knowledge assessment, varied hint types, and user adaptation. While we created only one MDP from a specific problem date set, it is possible to cluster different types of students and then create separate MDPs with different reward functions for each cluster. MDPs could be generated for 1) expert, 2) typical, and 3) least error-prone groups of students. The reward function that has been implemented reflects an expert reward function, where the value for a state reflects the shortest path to the goal state. Alternatively, when the Hint Button is pressed, we could select a personalized 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. If there is sufficient data, we can create separate MDPs for students in particular groups, such as high, low, and medium performers on a previous exercise, learning styles, GPA, or other factors, and use these to generate personalized hints. These hints would be contextualized both within the problem and by student characteristics.

In this research, we focused on the domain of deductive logic. Additional research is addressing how the Hint Factory methods can be applied in other domains. As stated previously, others have applied our work to a tutor for computer science (Fossati et al. 2009), and we are looking at other domains including mathematics and chemistry. This research will show how well the Hint Factory will generalize.

We are using machine learning methods to analyze our MDPs for problem structure and for generating new problems of similar difficulty. This process capitalizes on the feature extraction used in the BKB method. We are also building tools for teachers and researchers to visualize MDPs. These tools will allow teachers to identify how students are actually solving the problems and see specific points where students are having difficulty. The visualization tools will also enable teachers to write their own hints, and modify how the MDPs are used in generating hints.

REFERENCES

Aleven, V., McLaren, B., Roll, I., & Koedinger, K. (2004). Toward tutoring help seeking: Applying cognitive modeling to meta-cognitive skills. *Proceedings of the Seventh International Conference on Intelligent Tutoring Systems.*

Arroyo, I., Woolf, B. (2005). Inferring learning and attitudes from a Bayesian Network of log file data. In C.K. Looie, G. McCalla, B. Bredeweg, and J. Breuker (eds.) *Proceedings of the 12th International Conference on Artificial Intelligence in Education*. pp. 33-40. Amsterdam: ISO Press.

Baker, R., Corbett, A.T., Aleven, V. (2008). More Accurate Student Modeling through Contextual Estimation of Slip and Guess Probabilities in Bayesian Knowledge Tracing. *In E. Aimeur, & B. Woolf (Eds.) Proceedings of the 9th International Conference on Intelligent Tutoring Systems (ITS 2008)*, pp. 406-415. Berlin, Germany: Springer Verlag.

Barnes, T., Stamper, J. (2008). 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.

Barnes, T., Stamper, J., Lehmann, L., Croy, M. (2008). 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.

Barnes, T. & Stamper, J. (2007). 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.

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

Bouma, W., Fudos, I., Hoffmann, C. M., Cai, J., and Paige, R. (1995). A geometric constraint solver. *Computer Aided Design*. 27, pp. 487-501.

Conati, C., Gertner, A., VanLehn, K., and Druzdzel, M. (1997). Online student modeling for coached problem solving using Bayesian networks. *In Proceedings of the Sixth International Conference on User Modeling*, Sardinia, Italy, pages 231-242. User Modeling, Springer-Verlag.

Croy, M., Barnes, T. & Stamper, J. (2007). 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.

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

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

Fossati, D., Di Eugenio, B., Ohlsson, S., Brown, C., Chen, L., Cosejo, D. (2009). 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.

Haddawy, P. (1994). Generating Bayesian Networks from Probability Logic Knowledge Bases. *In Proceeding of the 10th Annual Conference on Uncertainty in Artificial Intelligence (UAI 1994)*. pp 262-269.

Koedinger, K.R., Baker, R.S.J.d., Cunningham, K., Skogsholm, A., Leber, B., Stamper, J. (2010). A Data Repository for the EDM community: The PSLC DataShop. *In Romero, C., Ventura, S., Pechenizkiy, M., Baker, R.S.J.d. (Eds.), Handbook of Educational Data Mining*. Boca Raton, FL: CRC Press.

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

McCune, W. and Shumsky, O. (2000). "Ivy: A Preprocessor and Proof Checker for First-order Logic". Chapter 16 in Computer-Aided Reasoning: ACL2 Case Studies (ed. M. Kaufmann, P. Manolios, and J Moore), Kluwer Academic Publishers.

Mitrovic, A., Koedinger, K. & Martin, B. (2003). A comparative analysis of cognitive tutoring and constraint-based modeling. *User Modeling*: 313-322.

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

McLaren, B., Koedinger, K., Schneider, M., Harrer, A., & Bollen, L. (2004). 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.

Nkambou, R., Mephu Nguifo, E., Fournier-Viger, P. (2008). 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.

Matsuda, N., Cohen, W. W., Sewall, J., Lacerda, G., & Koedinger, K. R. (2007). 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.

Razzaq, L. & Heffernan, N. (2010) Hints: Is It Better to Give or Wait to be Asked? In Aleven, V., Kay, J & Mostow, J. (Eds.), *Proc. 10th Intelligent Tutoring Systems (ITS2010)* Part 1. Springer. Pages 349-358.

Santos, E., Jr., & Santos, E. (1996). Bayesian Knowledge-bases. Technical report AFIT/EN/TR96-05, Department of Electrical and Computer Engineering, Air Force Institute of Technology.

Shih, B., Kenneth R. Koedinger, and Richard Scheines. (2008). A Response Time Model For Bottom-Out Hints as Worked Examples. In *Educational Data Mining*, pp. 117-126.

Shute, V.J. (2008). Focus on formative feedback. *Review of Educational Research*, 78(1): p. 153-189.

Stamper, J., Barnes, T., Croy, M. (2010). Using a Bayesian Knowledge Base for Hint Selection on Domain Specific Problems. *In Baker, R., Merceron, A., Pavlik, P. (Eds.), Proceedings of the 3rd International Conference on Educational Data Mining (EDM 2010)*, pp. 327-328. Pittsburgh, PA.

Sutton, S. & Barto, A. (1998). Reinforcement Learning: An Introduction. MIT Press, Cambridge.

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