Toward the extraction of production rules for solving logic proofs

Tiffany Barnes, John Stamper
Department of Computer Science, University of North Carolina at Charlotte
tbarnes2@uncc.edu, jcstampe@uncc.edu

Abstract: In building intelligent tutoring systems, it is critical to be able to understand and diagnose student responses in interactive problem solving. However, building this understanding into the tutor is a time-intensive process usually conducted by subject experts. Much of this time is spent in building production rules that model all the ways a student might solve a problem. We propose a novel application of Markov decision processes (MDPs), a reinforcement learning technique, to automatically extract production rules for an intelligent tutor that learns. We demonstrate the feasibility of this approach by extracting MDPs from student solutions in a logic proof tutor, and using these to analyze and visualize student work. Our results indicate that extracted MDPs contain many production rules generated by domain experts and reveal errors that experts do not always predict. These MDPs also help us identify areas for improvement in the tutor.

Keywords: educational data mining, Markov decision processes

1. Introduction

According to the ACM computing curriculum, discrete mathematics is a core course in computer science, and an important topic in this course is solving formal logic proofs. However, this topic is of particular difficulty for students, who are unfamiliar with logic rules and manipulating symbols. To allow students extra practice and help in writing logic proofs, we are building an intelligent tutoring system on top of our existing proof verifying program. Our experience in teaching discrete math, and in student surveys, indicate that students particularly need feedback when they get stuck.

The problem of offering individualized help and feedback is not unique to logic proofs. Through adaptation to individual learners, intelligent tutoring systems (ITS) can have significant effects on learning [1]. However, building one hour of adaptive instruction takes between 100-1000 hours of work of subject experts, instructional designers, and programmers [2], and a large part of this time is used in developing production rules that are used to model student behavior and progress. A variety of approaches have been used to reduce the development time for ITSs, including ITS authoring tools (such as ASSERT and CTAT), or building constraint-based student models instead of production rule systems. ASSERT is an ITS authoring system that uses theory refinement to learn student models from an existing knowledge base and student data [3]. Constraint-based tutors, which look for violations of problem constraints, require less time to construct and have been favorably compared to cognitive tutors, particularly for problems that may not be heavily procedural [4].

Some systems, including RIDES, DIAG, and CTAT use teacher-authored or demonstrated examples to develop ITS production rules. RIDES is a “Tutor in a Box” system used to build training systems for military equipment usage, while DIAG was built as an expert diagnostic system that generates context-specific feedback for students [2]. These systems cannot be easily generalized, however, to learn from student data. CTAT has been used to develop “pseudo-tutors” for subjects including genetics, Java, and truth tables [5]. This system has also been used with data to build initial models for an ITS, in an approach called Bootstrapping Novice Data (BND) [6].

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, we propose to generate Markov Decision Processes that represent all student approaches to a particular problem, and use these MDPs directly to generate feedback. We believe one of the most important contributions of this work is the ability to generate feedback based on frequent, low-error student solutions.

We propose a method of automatically generating production rules using previous student data to reduce the expert knowledge needed to generate intelligent, context-dependent feedback. The system we propose is capable of continued refinement as new data is provided. We illustrate our approach by applying MDPs to analyze student work in solving formal logic proofs. This example is meant to demonstrate the applicability of using MDPs to collect and model student behavior and generate a graph of student responses that can be used as the basis for a production rule system.

2. Background and Proofs Tutorial Context

Several computer-based teaching systems, including Deep Thought [7], CPT [8] and the Logic-ITA [9] have been built to support teaching and learning of logic proofs. Of these, the Logic-ITA is the most intelligent, verifying proof statements as a student enters them, and providing feedback after the proof is complete on student performance. Logic-ITA also has facilities for considerable logging and teacher feedback to support exploration of student performance [9], but does not offer students help in planning their work. In this research, we propose to augment our own existing Proofs Tutorial, with a cognitive architecture derived using educational data mining, that can provide students feedback to avoid error-prone solutions, find optimal solutions, and inform students of other student approaches.

In [10], the first author has applied educational data mining to analyze completed formal proof solutions for automatic feedback generation. However, this work did not take into account student errors, and could only provide general indications of student approaches, as opposed to feedback tailored to a student’s current progress. In this work, we explore all student attempts at proof solutions, including partial proofs and incorrect rule applications, and use visualization tools to learn how this work can be extended to automatically extract a production rule system to add to our logic proof tutorial. In [11], the second author performed a pilot study to extract Markov decision processes for a simple proof from three semesters of student data from Deep Thought, and verified that the rules extracted by the MDP conformed with expert-derived rules and generated buggy rules that surprised experts. In this work, we apply the technique and extend it with visualization tools to new data from the Proofs Tutorial.

The Proofs Tutorial is a computer-aided learning tool implemented on NovaNET (http://www.pearsondigital.com/novanet/). This program has been used for practice and feedback in writing proofs in university discrete mathematics courses taught by the first author and others at North Carolina State University since 2002. In the Proofs Tutorial, students are assigned a set of 10 problems that range from simpler logical equivalence applications to more complex inference proofs. (The tutorial can check arbitrary proofs, but we use it for a standard set of exercises). In the tutorial, students type in consecutive lines of a proof, which consist of 4 parts: the statement, 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 statement, reference lines, axiom, and substitutions are verified. If any of these conditions does not hold, a warning messageis shown, and the line is deleted (but saved for later analysis).

In this work, we examine student solutions to Proof 1. Table 1 lists an example student solution. Figure 2 is a graphical representation of this proof, with givens as white circles, errors as orange circles, and premises as rectangles.

Table 1: Sample Proof 1 Solution (red lines are errors)

Statement	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)**
b	**1**	**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

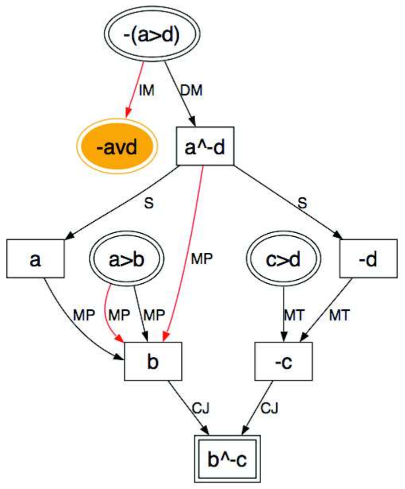


Figure 2: Graph of Proof 1 Solution
(Red lines are errors)

3. Markov Decision Processes and ACT-R

A Markov decision process (MDP) is defined by its state set S, action set A, transition probabilities P, and a reward function R [12]. 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. 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. We propose instead, to create multiple functions to deliver different types of feedback, such as functions that could:

1. Find expert-like paths. To derive this policy, we would assign a high reward to the goal state, negative rewards to error states and smaller negative rewards for taking an action. This function would return an optimal (short and correct) choice, and hence expert feedback, at every point in the MDP.
2. Find a typical path to the goal state. We could derive this policy by assigning high rewards to successful paths that many students have taken. Vygotsky's theory of the zone of proximal development [13] states that students are able to learn new things that are closest to what they already know. Presumably, frequent actions could be those that more students feel fluent using. Therefore, paths based on typical student behavior may be more helpful than optimal solutions, which may be above a student's current ability to understand.
3. Find paths with low probabilities of errors. It is possible that some approaches could be much less error-prone than others. We could this policy by assigning large penalties to error states. Students often need to learn a simple method that is easily understood, rather than an elegant but complex solution.

These MDPs are also intended to serve as the basis for learning a production rule system that will perform model tracing, as in a cognitive tutor, while a student is solving a problem. ACT-R Theory [1] is a cognitive architecture that has been successfully applied in the creation of cognitive tutors, and consists of declarative and procedural components. The procedural module contains a production rules system, and creating its production rules is the most time consuming part of developing an ITS based on ACT-R. A production rule system consists of the current problem state (called working memory in ACT-R), a set of production rules, and a rule interpreter. A production rule can be seen as a condition-action pair such as "if A then C" with associated probability x. An example production rule for solving proofs is "if the goal is to prove b^-c, then set as subgoal to prove b". The rule interpreter performs model tracing to find a sequence of productions that produce the actions a student has taken. This allows for individualized help, tracks student mastery (using the correctness of the productions being applied), and can provide feedback on recognized errors.

Production rules in ACT-R are of a general nature to allow them to apply to multiple problems. However, they could be written very specifically to apply to a particular problem. An MDP extracted for a particular problem could be seen as a set of production rules that describe the actions a student might take from each problem state. In this case, MDP state-action pairs are production rules, where the full current problem state is the condition and the action is the applying a particular axiom.

By building an MDP using data from several semesters, we generate a significant number of rules and record probabilities that reflect different student approaches. We can use this MDP as it is to perform model tracing as a student is working a proof. If a student is working the problem in a way that has already been encountered, we can use the MDP to provide feedback. If he or she asks for help, we can direct that student to optimal, frequent, or simply away from error-prone paths, based on that particular student and/or path. Similarly to constraint-based tutors [4], if a student is solving a proof in a way that is not already present in our MDP, we simply add their steps to our model. If such a student does commit an error, only default feedback will be given.

As a first step, MDPs created from student data can be used to add intelligent feedback to every problem. This would require storing the MDP, adding a process to the end of the statement checking to match the student's state to the MDP, and if it is found, adding a hint option that would reveal the optimal next choice in the MDP. If a student's state is not found in the MDP, we can add it (storing its correctness), and periodically run reinforcement learning to update the reward function values.

Our next step will be to apply machine learning to the MDP to learn more general rules for building a true production rule system. The list of axioms itself can be used as general production rules, and represent most valid student actions. For example, Modus Ponens can be expressed as the production rule: “If A and A$\rightarrow$B, then apply Modus Ponens to get B”. However, there are no priorities assigned to the axioms themselves. We can cycle through the states of the MDP, examining all cases where the premises hold. We can use the MDP reward functions or the frequency of student use, or a combination of these, to set priorities for each axiom. Then when more than one axiom applies, we know which one was most frequently used with success (and therefore which one should fire).

Alternatively, we could apply machine learning techniques to our MDPs to find common subgraphs in student approached, or build hierarchical MDPs to group substrategies in proof solutions. We believe this approach could be done in a general way so that it could be applied to other domains. Toward that end, we have generated visualizations of our MDPs to investigate the structure of student solutions, what errors they commit, and where they occur most frequently. This process can provide insight into where machine learning could provide the most benefit. In addition, this analysis can help us discover areas for improvement in the ITS.

4. Method

This experiment uses data from the four fall semesters of 2003-2006, where an average of 120 students take the discrete math course at NC State University each fall. Students in this course are typically engineering and computer science students in their second or third year of college, but most have not been exposed to a course in logic. Students attend several lectures on propositional logic and complete an online homework where students complete truth tables and fill in the blanks in partially-completed proofs. Students then use the Proofs Tutorial to solve 10 proofs as homework, directly or using proof by contradiction. The majority of students used direct proof to solve proof 1. We extracted 429 of students’ first attempts at direct solutions to proof 1 from the Proofs Tutorial. We then removed invalid data (such as proofs with only one step to reach the conclusion), resulting in 416 student proofs. Of these, 283 (70%) were complete and 133 (30%) were partial proofs. Due to storage limitations, a few (6) of these proofs may have been completed by students but not fully recorded. The average lengths were 13 and 10 lines, respectively, for completed and partial proofs. This indicates that students did attempt to complete the proof.

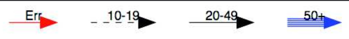
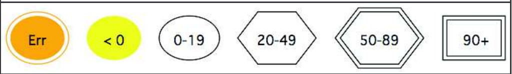
After cleaning the data, we load the proofs into a database and build an MDP for the data. We then set a large reward for the goal state (100) and penalties for incorrect states (10) and a cost for taking each action (1). Setting a non-zero cost on actions causes the MDP to penalize longer solutions (but we set this at 1/10 the cost of taking an incorrect step). These values may need to be adjusted for different sizes of MDPs. We apply the value iteration Bellman backup reinforcement learning technique to assign reward values to all states in the MDP. 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 value 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 [12]. The rewards 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. For the purposes of this paper, just one step of value iteration was performed to cascade goal values through the MDP.

We then ran the MDP for each semester of data, and for the aggregate. This resulted in a set of states, reward values, and actions for each MDP. On average, the four semesters yielded 158 states (ranging from 95-226 states in a semester). The most frequent approaches to problems were very similar across semesters, and the most frequent errors were also repeated. Therefore, in the remainder of this paper, we examine the aggregate MDP. Using Excel®, we assigned labels to each state in the MDP (just using the latest premise added), colors for errors, state values, and action frequencies, and prepared the data for display. We used GraphViz to display the output and convert into pictures. Table 2 shows the legend for nodes and edges. After graphing each MDP, we continually refined the data being displayed to explore questions about the student data. We present our findings in the following section.

Table 2: Legend for MDP edges and nodes

Edges (Values=Frequency)	Nodes (Values=Rewards)
	

5. Results

The aggregate MDP run on all four semesters of data has a total of 547 states (each semester alone generated between 95-226 unique states). Since it is hard to view statically, we omit it here. From interactive exploration, we found that 90% of all student errors related to explaining their actions, and a great majority of these were on the simplification rule. We plan to improve the interface for this explanation. We also found that students commit a great number of errors in the first step but less as they progress. To assist in visualizing student behavior, we plan to build a tool for teacher visualization that will allow for pruning nodes below a certain reward or frequency, and also for highlighting all the correct or incorrect applications of a particular action. We demonstrate some of these views in Figures 3-4.

Figure 3 shows the aggregate MDP restricted to only valid states and frequent state-action pairs. The graph shows only one frequent successful path – indicating an optimal solution that students can perform. This path also corresponds to an expert solution. On this path, it seems that errors are occurring in going from state (a^-d) to (a), demonstrating our observation that applying simplification is difficult for students. We note many errors at the start, indicating that students have trouble getting started. Following the shaded path, second from the lowest in Figure 4, we observe that several (20-49) students apply IM in various ways; this path is *very* error-prone and does not (frequently) lead to a solution. This indicates a need to help students plan proof strategies. We can detect this type of path in the MDP and offer hints to help avoid it.

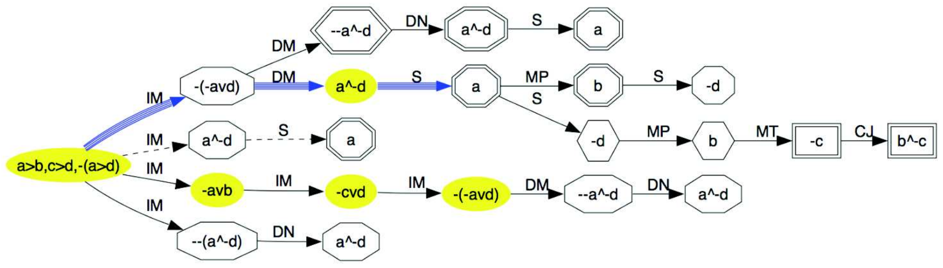


Figure 3: View of MDP restricted to valid states and frequent actions

To further examine student approaches, we expand this graph to include error states in Figure 4. In most errors, students find the correct premise (e.g. –d, which is correct) but have trouble explaining how the rule was applied to get the premise. For example, the rule for S (simplification) states (p^q)→p, so to obtain (a^-d)→-d the student must show that we substitute p=-d and q=a. We conclude that we need a better interface to help students show how a rule applies. For example, in Deep Thought, students are not required to perform substitutions if the program can detect them itself, and for simplification, the program asks: Right or Left? Anecdotally and intuitively, students seem to have less trouble with this approach. All but two of the remaining error states (indicated with darker shading and a double border) are due to substitution errors. (Start) → (–(a^-d)) is an incorrect application of IM (it's missing a not), while Contrapositive (CP) was frequently applied to obtain –c from c>d and –d (which needs Modus Tollens, MT). These findings indicate that more practice might be needed with these rules in the context of negated variables.

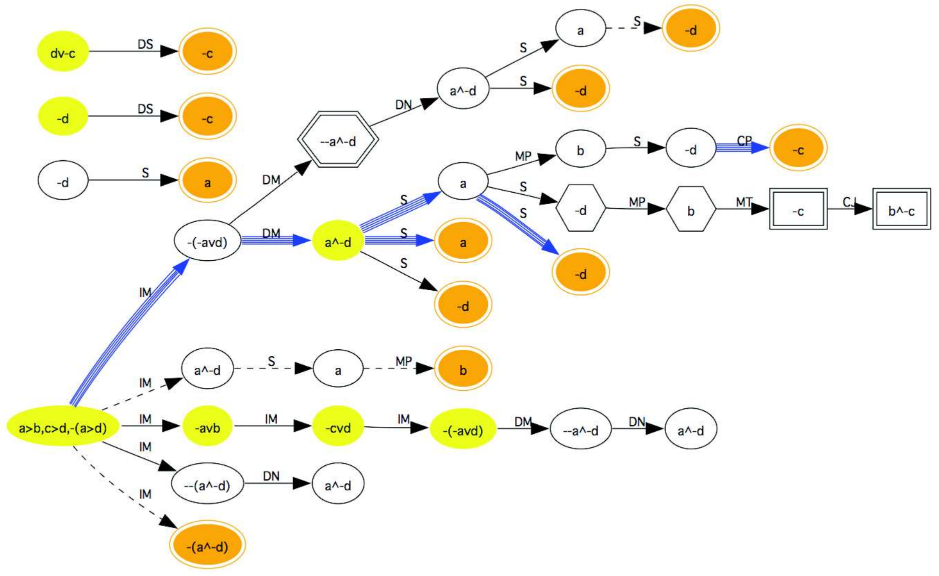


Figure 4: View of MDP restricted to frequent states and actions

These visualizations are useful in predicting what the MDP will find when generating feedback based on frequent responses. However, this does not yield insight into a more general application of MDPs to proofs and what types of production rules might be generated for general problems. To make more general production rules for proof problems, we will take MDPs from several problems and attempt to learn general rules. For instance, a production rule often applied by experts is, "if (p → q) and (p) are premises then apply MP to obtain the new premise (q)". To learn this, we need to break MDP states down into their constituent premises.

We created Figure 5, (which is a graph, not an MDP), by mapping all correct states with a common last premise into a single node, which corresponds to grouping unique action/premise pairs. We then eliminated all low-frequency actions and the simplification rule (since we plan to change the interface to improve usage of this rule). This new visualization of the paths of student solutions allows us to track frequent solution attempts regardless of order. In other words, our previous MDP views correspond to unique paths, while this graph shows us relationships between consecutive steps in the proof regardless of what was done before. In Figure 5, there are some dead-end paths, meaning that several students started proofs in these directions but few students were able to reach the solution this way. These dead-end paths can be used to derive feedback to students that their approach may not be productive. We can also use Figure 5 to derive most frequent orderings of student solutions. To do so, we start at the (top) start node and choose a frequent (wide) edge, and repeat without visiting nodes twice, until we reach the solution, as in Figure 6.

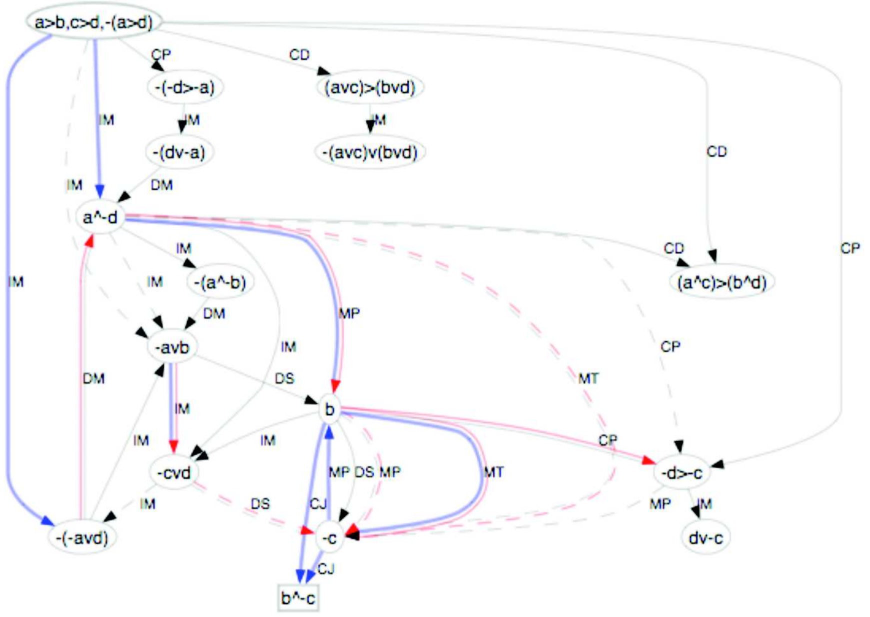


Figure 5: Graph showing inferences, unique premises and frequent (>9) actions

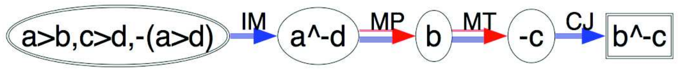


Figure 6: Most frequent proof solution sequences, derived from Figure 5

Some secondary approaches are shown in Figure 7, demonstrating how a teacher could use Figure 5 by following particular paths but not repeating any nodes visited, to find unique solutions to the proof. These approaches demonstrate students' frequent preference to use Disjunctive Syllogism (DS), even though these solutions are longer.

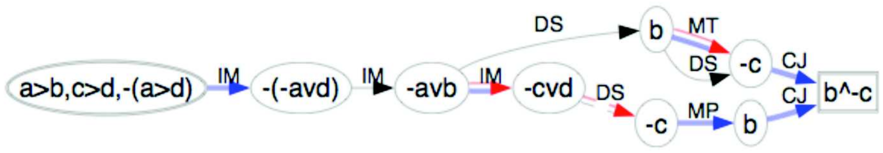


Figure 7: Secondary proof approaches derived from Figure 5

In both Figures 6 and 7, we see that there are errors (on the double edges) leading to and from states containing negated variables. This observation reflects our experience that students need more practice in using rules when variables are of negated, and in applying rules in sequence to achieve a goal.

6. Conclusions and Future Work

We have proposed a general approach to mining Markov decision processes from student work to automatically generate production rules and discussed how this approach can be applied to a particular CBT. 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. We have explored visualizations of the Markov decision processes extracted from solutions to a formal logic proof to determine how to improve the tutor and how we might proceed in building useful production rules and feedback. We believe that the process we have applied to creating problem visualizations can be useful in learning about other problem-solving processes from student data, instead of creating expert systems by hand. From these visualizations, we have learned:

1. Although we hypothesized that students need help in planning, this did not seem to be the case. Instead, students needed help on getting started.
2. As we expected, students need more practice with negation.
3. The overwhelming majority of student errors were in explaining rule applications. We plan to add a better interface for this step.

We have also concluded that the extracted MDPs will be useful in generating student feedback. The extracted MDP does contain a frequent expert-like path and contains a significant number of correct paths and student errors. Our tutor can already classify many errors students make. Adding the MDP to this tutor will enable it to model student mastery, provide hints, and feedback on errors. This MDP can constantly learn from new student data. We note that on cold start for a new problem that has no student data, the system will still act as a problem-solving environment, but after even one semester of data is collected, a limited amount of feedback can be generated. As more data are added, more automated feedback can be generated. In our future work we plan to implement this system in our tutor and in Deep Thought, a logic tutor created by Marvin Croy [7]. Once implemented, we will test the feedback generated based on the MDP, and we will continue to explore ways to learn general rules to build production rule systems with greater coverage and robustness.

A novel contribution of this work is the idea of providing feedback on the most frequent approach to a problem. It is often the case that students are not ready to apply optimal solutions to a problem. However, detecting this readiness is a challenging user modeling problem. It may be possible that student feedback based on frequency rather than optimality can provide most students with the help they need. This type of feedback may also encourage student reflection, when the tutor is no longer all-knowing, but has a memory that students tap into instead. In our future work we plan to investigate the impact of this type of feedback on student learning.

7. References

[1] Anderson, J., Gluck, K. 2001. What role do cognitive architectures play in intelligent tutoring systems? In D. Klahr & S. Carver (Eds.) *Cognition & Instruction: 25 years of progress*, 227-262. Erlbaum.

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

[3] Baffes, P. & Mooney, R.J. 1996. A novel application of theory refinement to student modeling. *Proc. AAAI-96*, pp. 403-408, Portland, OR, August 1996.

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

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

[6] 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, August 30, 2004.

[7] Croy, M. 2000. "Problem Solving, Working Backwards, and Graphic Proof Representation," *Teaching Philosophy*, 23(2), 169-187.

[8] Scheines, R., & Sieg, W. 1994. Computer environments for proof construction. *Interactive Learning Environments*, 4(2), 159-169.

[9] Merceron, A., & Yacef, K. 2005. Educational data mining: a case study. In: C.-K. Looi, G. McCalla, B. Bredeweg & J. Breuker (eds.), (*Proc. 12th Intl. Conf. Artificial Intelligence*), pp. 467–474. Amsterdam: IOS Press.

[10] Barnes, T. 2006. Evaluation of the q-matrix method in understanding student logic proofs. *Proc. 19th Intl. Florida AI Research Society Conference* (*FLAIRS 2006)*, Melbourne Beach, FL, May 11-13, 2006.

[11] Stamper. J. 2006. Automating the Generation of Production Rules for Intelligent Tutoring Systems. *Proc. Intl. Conf. Computer-Aided Learning (ICL2006)*. Austria, Sep. 27-29, 2006.

[12] Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA.

[13] Vygotsky, L. (1986). *Thought and language*. Cambridge, MA: MIT Press.