AgentX: Using Reinforcement Learning to Improve the Effectiveness of Intelligent Tutoring Systems

Kimberly N. Martin and Ivon Arroyo
Department of Computer Science
University of Massachusetts
140 Governors Drive
Amherst, MA 01003
{kmartin,ivon}@cs.umass.edu

Abstract

Reinforcement Learning (RL) can be used to train an agent to comply with the needs of a student using an intelligent tutoring system. In this paper, we introduce a method of increasing efficiency by way of customization of the hints provided by a tutoring system, by applying techniques from RL to gain knowledge about the usefulness of hints leading to the exclusion or introduction of other helpful hints.

Students are clustered into learning levels and can influence the agents method of selecting actions in each state in their cluster of affect. In addition, students can change learning levels based on their performance within the tutoring system and continue to affect the entire student population. The RL agent, AgentX, then uses the cluster information to create one optimal policy for all students in the cluster and begin to customize the help given to the cluster based on that optimal policy.

1 Introduction

The cost of tracing knowledge when students ask for help is high, as students need to be monitored after each step of the solution. The ITS requires a special interface to have the student interact with the system at each step, or explain to the tutoring system what steps have been done. Such is the case of ANDES [6] or the CMU Algebra tutor [8]. While trying to reduce the cost of Intelligent Tutoring Systems, one possibility is to try to infer students flaws based on the answers they enter or the hints they ask for. However, if students steps in a solution are not traced by asking the student after each step of the solution, and the student asks for help, how do we determine what hints to provide? One possibility is to show hints for the first step, and then for the second step if the student keeps asking for help and so on. However, the assumption cannot be made that the students seeking utility from the ITS are all at the same level when in fact

even within a single classroom, students will show a range of strengths and weaknesses. Some students may need help with the first step, others may be fine with a summary of the first step and need help on the second one. Efficiency could be improved by skipping hints that aid on skills that the student already knows. In an ITS that gives hints to a student in order to assist the student in reaching a correct solution, the hints are ordered by the ITS developer and may not reflect the true nature of the help needed by the student. Though feedback may be gathered through formative evaluations after the student has used the system for future enhancements, traditional tutoring systems get no feedback on the usefulness of the hints while the student is using the system.

Reinforcement Learning (RL) is a technique for learning actions in stochastic environments. While ITSs are becoming more adaptive, much of the customization is done based on student models that are built based on prior knowledge about what implies mastery. An improvement can be found in imploring RL, as optimal actions can be learned for each student, producing student and pedagogical models that self-modify themselves while learning how to teach. By combining techniques from RL with information from testing loaded a priori, the individual student adaptation is dynamic and the need for a pre-customized system is reduced.

In this paper, we will introduce a method of increasing the efficiency of hint sequencing and student performance by adding a RL agent to an ITS. In the agent, a policy is updated through policy iteration with each problem completed. The reward is calculated at the end of the problem and propagates back to all of the skills used in the problem updating the overall usefulness of hints of this skill type to the student currently using the system. With a state-value being associated with each possible trajectory of hint types, useful sequences begin to emerge and policy iteration produces an updated, more suitable policy.

2 Related Work

There exist intelligent tutoring systems that have implored techniques from Machine Learning (ML) in order to reduce the amount of knowledge engineering done at development [1, 3, 4, 5, 7]. These systems are modified so that the configurability of the system is done on the fly, making the system more adaptive to the student and simplifying the need for rigid constructs at development. ADVISOR [3] is an ML agent developed to simplify the structure of an ITS. ADVISOR parameterizes the teaching goals of the system so that they rely less on expert knowledge a priori and can be adjusted as needed. CLARISSE [1] is an ITS that uses Machine Learning to initialize the student model by way of classifying the student into learning groups. ANDES [7] is a Newtonian physics tutor that uses a Bayes Nets approach to create a student model to decide what type of help to make available to the student by keeping track of the students progress within a specific physics problem, their overall knowledge of physics and their abstract goals for solving the problem.

Our goal is to combine methods of clustering students and predicting the type and amount of help that is more useful to the student to boost overall efficiency of the ITS.

3 Reinforcement Learning Techniques

Reinforcement Learning [10] is used for learning how to act in a stochastic environment by interaction with the environment. When a student interacts with an ITS, there is no completely accurate method for predicting the students actions (answering) at each time step, so designing an agent that will learn the strengths and weaknesses of the student as they forge through each problem will assist in exposing helpful elements of the system that can then be exploited in order to make the students use of the ITS more productive. A policy, Π, defines the behavior of the agent at a given time and is a mapping from perceived states of the environment to actions to be taken when in those states. The state space is then made up of all possible states that the agent can perceive and the set of actions being all actions available to the agent from a perceived state, thus a reward function maps perceived states of the environment to a single number, a reward, indicating the intrinsic desirability of the state. The value of a state, $V(s)$, is the total amount of reward the agent can expect to accumulate over the future starting from that state. So, a policy is said to be optimal if the value for all states in the state space are optimal and the policy leads an agent from its current state through states that will lead it to the state with the highest expected return, R.

3.1 Calculating State Values

We use the Bellman equation (Equation 1) to assign the expected return from the best action from a state based on the current (optimal) policy to that state. It is written as

$$\sum_{s'} P_{ss'}^{\Pi(s)}[R_{ss'}^{\Pi(s)} + \gamma V(s')] \tag{1}$$

where $P_{ss'}^{\Pi(s)}$ is the transition probability, $R_{ss'}^{\Pi(s)}$ is the reward and γ is a discount rate.

3.2 Policy Iteration

Since AgentX interacts with the environment, it sees rewards often making it impractical to compute an optimal policy only once. Instead, we use a policy iteration technique that improves the policy after each time step. Policy iteration is the process combining policy evaluation, updating the value of the policy, and policy improvement, obtaining the best policy available. Policy iteration behaves like an anytime algorithm since it allows us to have some policy for each problem at all times and continues to check for a better policy. Figure 1 shows the policy iteration algorithm.

4 Experimental Design

The experiments and setup referred to in this paper are based on the Wayang Outpost [2] web-based intelligent tutoring system. A simplified overview of the architecture of the Wayang system is as follows: A problem is associated with a set of skills related to the problem. A problem has hints (that aid on a skill associated with the problem) for which order is significant. For the purpose of AgentX, the skills have been mapped to distinct

1. Initialization
   $V(s) \in R$ and $\Pi(s) \in A(s)$ arbitrarily for all $s \in S$
2. Policy Evaluation
   Repeat
   $\Delta \leftarrow 0$
   For each $s \in S$:
      $v \leftarrow V(s)$
      $V(s) \leftarrow \sum_{s'} P_{ss'}^{\Pi(s)}[R_{ss'}^{\Pi(s)} + \gamma V(s')]$
      $\Delta \leftarrow max(\Delta, |vV(s)|)$
   until $\Delta < \theta$ (a small positive number)
3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in S$:
      $b \leftarrow \Pi(s)$
      $\Pi(s) \leftarrow argmax_a \sum_s P_{ss'}^{\Pi(s)}[R_{ss'}^{\Pi(s)} + \gamma V(s')]$
   If $b \neq \Pi(s)$ then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop; else go to 2

Where S is the complete state space, s is the current state, $s\prime$ is the next state and $A(s)$ is the set of all actions available from state s.

Figure 1: Policy Iteration Algorithm.

letters A, B,..., P and the hints are then A_1, A_2,..., A_n, B_1, B_2,..., B_n,..., P_1, P_2,..., P_n where the order of the hints are preserved by their appearance in any problem (i.e., A_4 can never follow A_6).

4.1 State Space Reduction

Initially, the state space was comprised of every distinct path from the beginning of a problem to the end of the problem for all problems. Where a path is a sequence of hints that are seen and the end of a problem is the point at which a correct solution is provided or all hints for the problem have been shown. In order to reduce the complexity of the state space, we consider a distinct path to be a sequence of skills. This reduction speeds up the learning rate because it reduces the number of distinct states needed to be seen in optimizing the policy since the set of skills is small.

If in solving a problem, the student could see hints that aid on the following sequence of skills $A \rightarrow A \rightarrow K \rightarrow D \rightarrow F$ (as arriving at the solution to this problem involves steps that imply the use of these skills), or some subsequence of this sequence, then Figure 2 shows all of the states associated with this problem. Any subsequence of skills can be formed by moving up and to the right (zero or more spaces) in the tree.

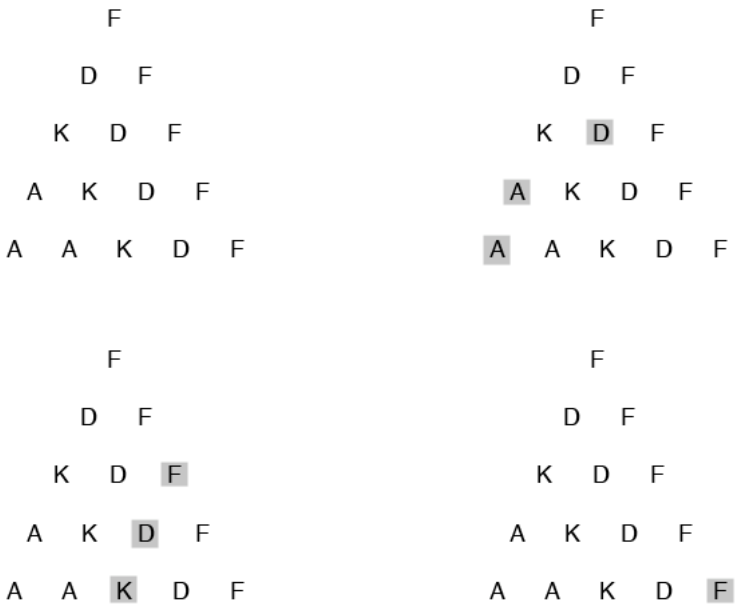


Figure 2: Possible skill trajectories from the state subspace for problem P.

4.2 Using Pretest Information

Wayang has the option that students take a computer-based pretest before using the ITS. The pretest uncovers the students strengths and weaknesses as they relate to problems in the system. With the computer-based pretest, information is easily accessible, and we can reduce the state space even further by excluding hints from skills where the student excels. In addition to the exclusion of the would be superfluous hints, we are able to use information about the weaknesses the student exhibits by initializing the value of the states that include skills of weakness with greater expected rewards, making the state more desirable to the agent instead of initializing each state with the same state-value.

An action in this system can be seen as moving from state to state where a state is a specific skill sequence containing skills that are related to the problem. Rewards occur only at the end of each problem, then propagate back to all states which are sub-states of the skill sequence.

4.3 Rewards

In acting in the system, the agent seeks states that will lead to greater rewards, then updates the value of each state effected by the action at the end of the problem. In order to guide the agent toward more desirable states, developing a reward structure that makes incorrect answers worse as the student receives more hints and correct answers better as students receive less hints allows us to shape the behavior of the action selec-

tion process at each state (Table 1). The reward for each problem is then the sum

Table 1: AgentX reward structure.

Action	Reward
correct answer	+10 hints seen
incorrect answer	- hints seen
no answer	+(1 / hints seen)

of the rewards given after each hint is seen. By influencing the agents with a reward[9] structure such as this, getting at correct answers sooner seems most desirable and speed up the process of reinforcement learning. The agent updates affected by the problem as it moves through the problem. An example shows that if the agent chooses state $A \rightarrow A \rightarrow K \rightarrow D \rightarrow F$ for the problem because its state value is the largest out of all eligible next states, then after the first hint from skill A is seen, state A is updated with the proper reward, after the second hint from skill A is seen, $A \rightarrow A$ is updated with the proper reward, etc.

4.4 Student Models

By sorting students into learning levels through clustering and re-clustering, student models can be used to speed up the policy iteration process for an individual student. Each problem completed by one student affects a cluster of students, diminishing the need for each problem to be seen by each student in order to judge the usefulness of help of a certain type to that student. Because the Wayang system is web-based and students use it all at the same time in classroom mode, a whole cluster of students that have a similar proficiency level or similar characteristics may be updating the shared value of a state. In the case where a student stays within their original cluster, all problems that this student completes will apply to the policy iteration process done on this specific process. In the case where a students learning level changes, they no longer have the ability to affect their former student cluster but are now re-classified into a new cluster, which is now their region of affect. They retrieve the state values and optimal policy of the new cluster and begin to have an effect on that cluster of students, thus making it possible to effect the entire population if they continue to change learning levels. Figure 3 shows the overall architecture of the system.

5 Experimental Setup

In creating the learning agent, we randomly generate student data for a student population of 1000. The random data is in the form of pre-test evaluation scores that allows the student to answer correctly, incorrectly, and not at all with different probabilities based on the data generated from the pre-test evaluation (Equation 2).

$$P(correct) + P(incorrect) + P(no\_answer) = 1 \quad (2)$$

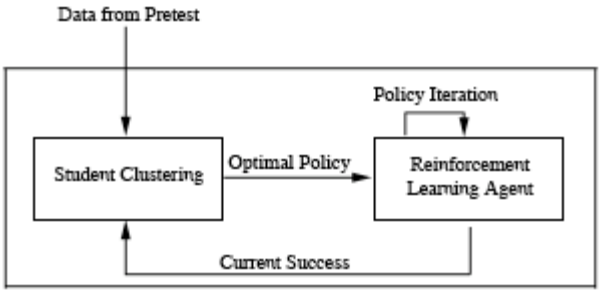


Figure 3: AgentX architecture.

As the student learns a skill, the probabilities are shifted away from answering incorrectly. Also, as the students actions are recorded the agent, the percentages for giving no answer and incorrect answers are able to effect the probability weightings.
The students are first sorted. The randomized pretest produces a numerical score for each of the skills utilized in the entire tutoring system, then we can use the harmonic mean of all scores to sort students into the multiple learning levels. Learning levels are created by the students expected success after having pretest results recorded and measured in percentiles. Table 2 shows the learning levels. Any student with no pretest data

Table 2: Learning levels for clustering students and their success measures.

Level	Expected	Current	Level	Expected	Current
L_1	90th percentile	90 - 100%	L_4	45th percentile	40 - 59%
L_2	80th percentile	80 - 89%	L_5	25th percentile	20 - 39%
L_3	65th percentile	60 - 79%	L_6	0th percentile	0 - 19%

available is automatically placed into learning level L4 since it contains the students who perform in the 50th percentile. Once the clusters are formed, after a short period of question answering (after x problems are attempted, where x is a small number such as 3 or 4), the students are able to change clusters based on their success within the tutor. The current success is measured by actions in percentages as seen in Equation 3.

$$current\ success = \#correct/(\#total\_hints\_seen - \#no\_answer) \quad (3)$$

So, answering correctly after each hint seen is 100% success and answering correctly after two hints are seen is 50% if the student answers incorrectly after the first hint and 100% if the student does not answer after the first hint.

While the learning levels are meant to achieve a certain amount of generalization over students, it is true that students who are in L_1 will perform better over all skills than that of any other grouping of students, which is why it is sufficient to use these learning levels even though the students may have different strengths. By the time students

attain L_1, they will be good at most skills and need fewer hints. While students in the middle levels will show distinctive strengths and weaknesses, but at different degrees of success, allowing the learning levels to properly sort them based on success. This clarifies a goal of the system to be able to cluster all students into L_1. Figure 4 shows the initial population of students within learning levels.

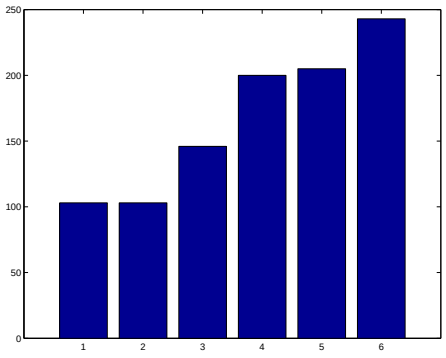


Figure 4: Student population in learning levels before any problems are attempted.

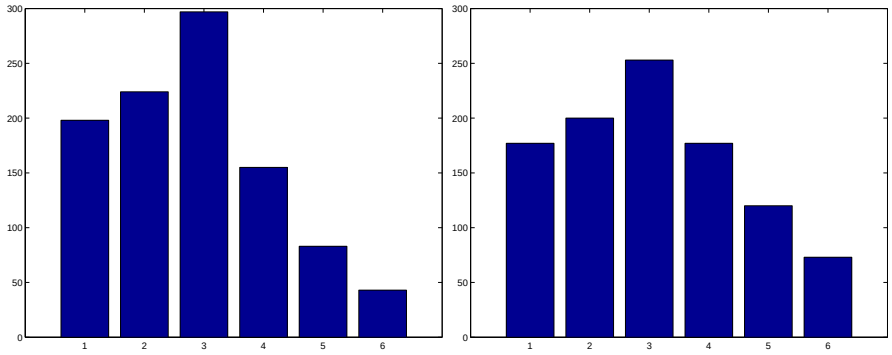
6 Results

In experimenting with different RL agents, we used an ϵ-greedy agent with different ϵ thresholds and a softmax agent. Figure 5a, 5b, and 5c shows the population of students in each learning level after 15 problems have been attempted by all students where the RL agent is ϵ-greedy with 10% exploration (ϵ = 0.1), softmax, and no RL agent respectively. Using the RL agent shifts the majority of the students towards the learning levels with success greater than 50% while the system without an RL agent maintains a slightly more normal (Gaussian) distribution of students about the learning level that includes 50% success. The average number of hints being shown in L_1 after 15 problems is reduced to 3.6 (three or four hints) as opposed to showing all hints (five).

7 Conclusions

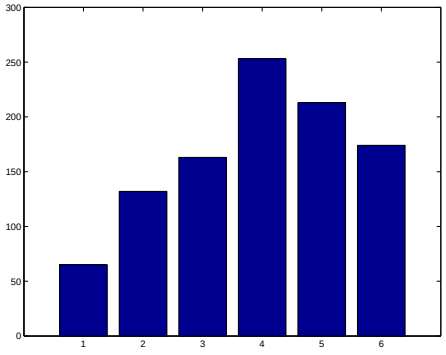
Using Reinforcement Learning agents can help to dynamically customize ITSs. In this paper we have shown that it is possible to boost student performance and a method for increasing the efficiency of an ITS after a small number of problems have been seen by incorporating a student model that allows the system to cluster students into learning levels and by choosing subsequences of all possible hints for a problem instead of simply showing all possible hints available to that problem.

Defining a reward structure based on a students progress within a problem and allowing their response to affect a region of other, similar students reduces the need to



(a) Using an ϵ-greedy RL agent

(b) Using a softmax RL agent



(c) Using no RL agent

Figure 5: Student population in learning levels after each student has attempted 15 problems.

see more distinct problems in creating a policy for how to act when faced with new skill sets and the need to solicit student feedback after each hint. With the goal to increase membership in learning level L_1 (90 - 100% success), which directly relates to the notion of increasing the efficiency of the system, we have shown that using an RL agent within an ITS can accomplish this.

References

[1] Esma Aimeur, Gilles Brassard, Hugo Dufort, and Sebastien Gambs. CLARISSE: A Machine Learning Tool to Initialize Student Models. In the *Proceedings of the 6th International Conference on Intelligent Tutoring Systems*. 2002.

[2] Carole R. Beal, Ivon Arroyo, James M. Royer, and Beverly P. Woolf. Wayang Outpost: A web-based multimedia intelligent tutoring system for high stakes math achievement tests. *Submitted to AERA 2003*.

[3] Joseph E. Beck, Beverly P. Woolf, and Carole R. Beal. ADVISOR: A machine learning architecture for intelligent tutor construction. In the *Proceedings of the 17th National Conference On Artificial Intelligence*. 2000.

[4] Joseph E. Beck and Beverly P. Woolf. High-level Student Modeling with Machine Learning. In the *Proceedings of the 5th International Conference on Intelligent Tutoring Systems*. 2000.

[5] Joseph E. Beck and Beverly P. Woolf. Using a Learning Agent with a Student Model. In the *Proceedings of the 4th International Conference on Intelligent Tutoring Systems*. pp. 6-15. 1998.

[6] Gertner, A. and VanLehn, K. Andes: A Coached Problem Solving Environment for Physics . In the *Proceedings of the 5th International Conference, ITS 2000* , Montreal Canada, June 2000.

[7] Gertner, A, Conati, C, and VanLehn, K.. Procedural help in Andes: Generating hints using a Bayesian network student model. In the *Proceedings of the 15th National Conference on Artificial Intelligence*. Madison, Wisconsin. 1998.

[8] Koedinger, K. R., Anderson, J.R., Hadley, W.H., and Mark, M . A. Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education, 8, 30-43*. 1997.

[9] Adam Laud and Gerald DeJong. The Influence of Reward on the Speed of Reinforcement Learning. In the *Proceedings of the 20th International Conference on Machine Learning (ICML-2003)*. 2003.

[10] R.S. Sutton and A.G. Barto. **Reinforcement Learning: An Introduction**. MIT Press, Cambridge, MA. 1998.