ADVISOR: A Machine Learning Architecture for Intelligent Tutor Construction

Joseph E. Beck and **Beverly Park Woolf**
Computer Science Department
University of Massachusetts
Amherst, MA 01003
USA
{beck, bev}@cs.umass.edu

Carole R. Beal
Psychology Department
University of Massachusetts
Amherst, MA 01003
USA
cbeal@psych.umass.edu

Abstract

We have constructed ADVISOR, a two-agent machine learning architecture for intelligent tutoring systems (ITS). The purpose of this architecture is to centralize the reasoning of an ITS into a single component to allow customization of teaching goals and to simplify improving the ITS. The first agent is responsible for learning a model of how students perform using the tutor in a variety of contexts. The second agent is provided this model of student behavior and a goal specifying the desired educational objective. Reinforcement learning is used by this agent to derive a teaching policy that meets the specified educational goal. Component evaluation studies show each agent performs adequately in isolation. We have also conducted an evaluation with actual students of the complete architecture. Results show ADVISOR was successful in learning a teaching policy that met the educational objective provided. Although this set of machine learning agents has been integrated with a specific intelligent tutor, the general technique could be applied to a broad class of ITS.

Introduction

AnimalWatch is an intelligent tutor for teaching arithmetic to grade school students. The goal of the tutor is to improve girls' self-confidence in their ability to do math, with the long-term goal of increasing the number of women in mathematical and scientific occupations. AnimalWatch has been shown to increase the self-confidence of girls who use it. The tutor maintains a student model of how students perform for each topic in the domain, and uses this model to select a topic to present to the student, to construct a problem at the appropriate level of difficulty for the student, and to customize feedback. This adaptation is done using a set of teaching heuristics.

A major factor contributing to women's lower participation in science, engineering and mathematics careers is that, beginning in junior high school, many girls begin to doubt their ability to learn mathematics(Beller & Gafni 1996). As a result, girls are typically more likely than boys to progress no further in mathematics than eighth grade algebra, and are subsequently under-prepared for many science and math-intensive majors and programs at the university and graduate school levels. This project focuses on development of an intelligent tutor to provide effective, confidence-enhancing mathematics instruction for girls in elementary school.

We have constructed a learning agent that models student behavior at a coarse level of granularity for a mathematics tutor. Rather than focusing on whether the student knows a particular piece of knowledge, the learning agent determines how likely the student is to answer a problem correctly and how long he will take to generate this response. To construct this model, we used traces from previous users of the tutor to train the machine learning (ML) agent. This model is combined with a reinforcement learning agent to produce a configurable teaching policy.

Research goals and previous work

Intelligent tutoring systems (ITS) have not been integrated into schools and corporate training as rapidly as would be expected. One reason for this is the lack of configurability of such systems by organizations deploying ITS(Bloom 1996). Once constructed, an ITS's performance is fixed. Also, the cost to construct ITS is very high, both in terms of number of hours required, and the amount of expertise needed. Our high-level goal is to reduce the amount of knowledge engineering (and expert knowledge) needed to construct an ITS and to make the tutor's teaching goals parameterized so they can be altered as needed. Machine learning is a useful technique for automating knowledge engineering tasks and improving adaptivity.

A second goal is to make a tutor's reasoning defensible. Currently, most teaching decisions are made by cognitive and/or pedagogical theories that may or may not have been well-evaluated, and almost certainly have not been extensively studied in the context of an ITS. We aim to bypass this step, and instead direct decision making by relying on observational data of students using intelligent tutors. A potential advantage of this architecture is to centralize the tutor's reasoning in one component. Research that improves this single piece affects the entire ITS.

There has been some progress made at using ML to simplify ITS construction. The ASSERT system(Baffes & Mooney 1996) used theory refinement to automatically construct buggy rules, and could build teaching examples

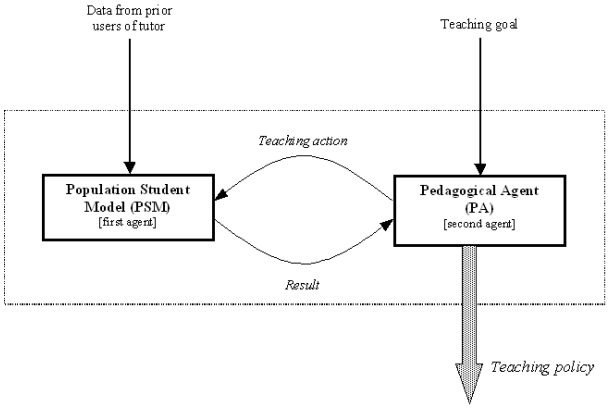

Figure 1: Overview of ADVISOR architecture.

to show students the fallacy behind their misconceptions. Given the typical difficulty in compiling bug lists(Burton 1982), this could be a substantial savings. Work on Input Output Agent modeling (IOAM)(Chiu & Webb 1998) has used ML to accurately predict mistakes students will make solving subtraction problems.

ANDES(Gertner, Conati, & VanLehn 1998) constructs a Bayesian network that represents a solution to the physics problem on which the student is working, and uses this network to recognize the student's plan(Conati *et al.* 1997) and determine on which skills he needs help(Gertner, Conati, & VanLehn 1998). This is a excellent example of using a single reasoning mechanism for many decisions within an ITS, and of principled reasoning techniques to direct teaching.

Architecture

To achieve our research goals, we constructed ADVISOR, which is a set of ML agents that learn how to teach students. Figure 1 provides an overview of this process. First, students using AnimalWatch are observed, and these data provided to a learning agent that models student behavior. This Population Student Model (PSM) is responsible for taking a context, and predicting how students will act in this situation.

The second component is the pedagogical agent (PA), that is given a high-level learning goal and the PSM. An example of such a high-level goal is "Students should make precisely 1 mistake per problem." Students making fewer than one mistake may not be sufficiently challenged, while those making too many mistakes may become frustrated. The PA's task is to experiment with the PSM to find a teaching policy that will meet the provided teaching goal.

The area contained in the dashed box represents a simulation that will occur several times during training before actual students use the tutor. The pedagogical agent considers a teaching action (selecting a particular topic, constructing a certain problem, or providing certain feedback) and observes how the simulation of the student reacts to this event. After learning which actions under which conditions meet the teaching goal and which do not, it outputs a teaching policy for the AnimalWatch tutor to use with real students. We define a teaching policy as a function that maps the current state of the tutor and student to a teaching action. Thus, from a given situation, the teaching policy can be used to direct all of the tutor's decision making. Currently, AnimalWatch's teaching policy is implemented with a set of heuristics. Our goal is to replace this with an ML-derived policy.

Population model

The purpose of the population model is to take as input a particular situation, and determine how the student would behave in that situation. Viewed this way, it is an executable student model. Combined with a tutoring system, this executable student model can act as a simulated student, or simulee(VanLehn, Ohlsson, & Nason 1994).

Constructing the PSM Building the PSM requires a database of prior users of the tutor. We have deployed AnimalWatch for two previous studies; one in a fourth-grade classroom (9 year olds), the other in a fifth-grade (10 year olds) classroom. When the student is presented with an opportunity to provide an answer, the system takes a "snap shot" of its current state. This picture consists of information from 4 main areas:

1. **Student:** The student's level of prior proficiency and level of cognitive development(Arroyo *et al.* 1999)
2. **Topic:** The difficulty of the current topic and the type of mathematics operand/operators.
3. **Problem:** The difficulty of the current problem.
4. **Context:** The student's current efforts at answering this question, and the hints he has seen.

After the student makes his next response, the system records the time he required to make this response and whether it was correct. These two pieces of information, time to respond and correctness of response, are what the learning agent will predict. We gathered 11,000 training instances from the previous deployments of AnimalWatch.

The above information specifies the inputs and outputs of the learning agent. For a function approximator, we used linear regression (after trying naive Bayesian classifiers and decision trees). Linear regression allows quick experimentation with models (unlike gradient descent techniques, which can take substantial time to learn), and works well with continuous variables. There were two regression models. Each took as input 48 features from the above 4 categories. The first model output the expected amount of time the student would require to generate his *next* response (in log-milliseconds). The second model output the probability the student's *next* response would be correct. Note, the model does not try to predict the student's longer term performance on the current problem, only his immediate action.

Validating the PSM To determine if the PSM is sufficiently accurate, we compare its predictions to how the students in the training dataset actually performed. Figure 2 shows its accuracy for predicting how long students required to generate a response, the PSM's predictions correlated at 0.629 ($R^2 \approx 0.4$) with actual performance. Training on half the dataset and testing on the other half dropped this correlation to 0.619, which is still very strong for predicting as noisy a variable as time.

The PSM is also responsible for predicting whether the student's response will be correct. Figure 3 shows the

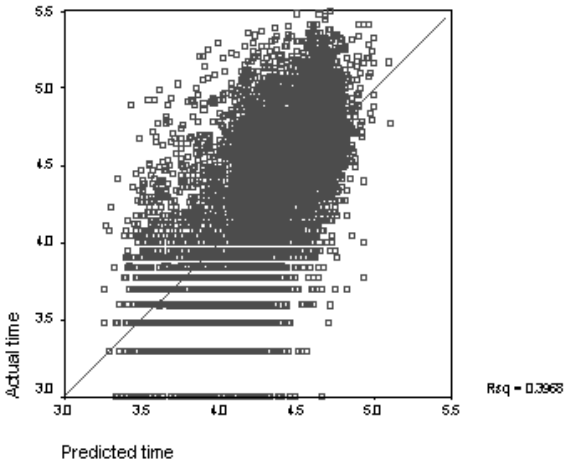

Figure 2: Accuracy for predicting response time

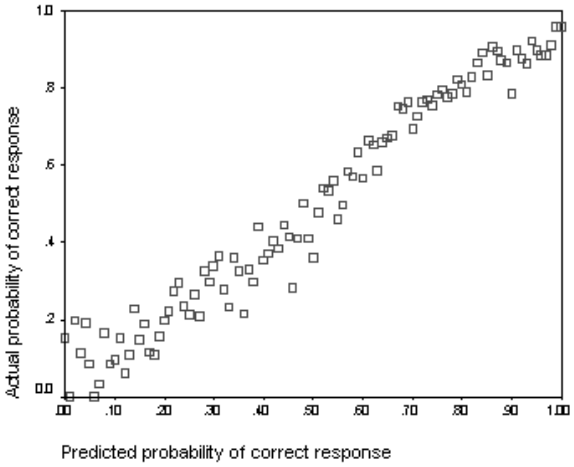

Figure 3: Accuracy for predicting response correctness.

model's performance at this task. The data were binned, so (for example) the right-most datapoint refers to cases where the model thought there was a probability of 1.0 the student would answer a question correctly. In these cases, there was roughly a 0.95 probability the student would generate a correct response. Interestingly, this model only correlates at 0.484 (0.481 with split-half validation) with the student's actual performance, for an $R^2 \approx 0.23$. Figure 3's correlation is much higher, but this is for averaged, not individual student performance.

In Figure 3, if the model were perfectly accurate, there should be no datapoints aside from those at 0.0 (leftmost edge of x-axis) and 1.0 (rightmost edge of x-axis). Students either generate a correct response or an incorrect response, there is no middle ground. So for individual cases, the PSM may be inaccurate. However, averaged across similar cases, the PSM's performance at predicting correctness is quite good. For making critical, stand alone predictions, this is not acceptable. As part of a simulation that uses millions of simulated trajectories, it seems to be adequate to have a probabilistic model such as this.

Pedagogical agent

The pedagogical agent (PA) uses the simulation of the student as described by the PSM, and experiments with different teaching policies to achieve an externally provided teaching goal.

An interesting question is why the pedagogical agent is needed. Why not use the PSM to directly learn/predict the relevant teaching goal in question? If the goal is for the student to make one mistake per problem, simply have the PSM predict the number of future mistakes. The difficulty with this is that the PSM's observations come under a particular teaching policy. If this teaching policy changes, the PSM's long-term predictions become worthless. Consider a PSM that is trained by observing a tutor that never provides any feedback. When asked to predict how many mistakes a student will make on a problem, its estimate will be very high, particularly if the teaching policy has changed so the tutor now provides feedback. Given that we are using training data from a tutor that probably has different teaching goals, and consequently a different teaching policy, this situation is not acceptable.

Thus, the PSM only makes short range predictions: "if the student has knowledge K, is working on problem P, has seen hints h1 and h2, and is now provided hint H $\rightarrow$ he will answer correctly with probability P after time T." A prediction of this form is *independent* of the teaching policy used by the tutor. The PSM is not interested why the student and tutor are in this situation, or indeed, what will happen afterwards. All the PSM is asked to predict is what the student will do immediately from this situation. Thus, the PSM is used to generate a model of state transitions that describe student behavior.

Constructing the pedagogical agent The pedagogical agent is a reinforcement learning (RL)(Sutton & Barto 1998) agent. RL can operate with a model of the environment (the Population Student Model), and a reward function (the pedagogical goal the agent is trying to achieve). This architecture of using a simulation is similar to that used in other complex RL tasks such as TD-gammon(Tesauro 1995) and elevator dispatching(Crites & Barto 1998).

For our RL mechanism, we decided to use temporal-difference (TD) learning(Sutton & Barto 1998). Specifically, TD(0) with state-value learning. The benefit of state-value learning is that there may be several actions that result in a similar state (e.g. the student gets the right answer in a small amount of time), and it is efficient to learn about these together. This is particularly relevant if the agent is trying to learn about the student in real-time.

We gave our first deployable PA the goal of minimizing the amount of time students require to solve a problem. This should result in the PA selecting easier topics, building easier problems, and not giving hints that will slow down the student's problem solving efforts (such as a long-winded interactive hint, when the student has made a minor mistake).

This goal (i.e. reward function for the RL agent) was chosen since it is relatively straightforward to optimize, and should not be a troublesome policy for actual students to use.

Specifically, when the (simulated) student solved a problem, the RL agent was given a reward inversely proportional to the amount of time the student required to solve the problem. Thus, this is a bounded horizon task, and discounting is not required. During training, the PA used ϵ-greedy exploration. This means that a certain percentage of the time, the PA selected a random action rather than the one it thought best. This helped prevent the PA from getting stuck in a locally good, but globally sub-optimal policy. Over time, ϵ was gradually reduced. Once the agent was deployed in the "real-world", however, ϵ was set to 0.

The PA used a linear function approximator to map the state to a future expected reward. This was used for all of the tutor's decision-making. E.g. to select a hint, the tutor would add the information about the topic, problem, and student's current efforts to the state. Combining the information about each possible hint with the current state, it would query the PSM to determine how a student would likely respond to the hint. The output of the PSM was given to the linear function approximator that predicted future expected reward, and the action corresponding to whichever "afterstate" had the highest value was selected.

Validating the pedagogical agent To determine if the pedagogical agent would interact correctly with the PSM, we constructed a simulation. This simulation contained all of the typical steps in a tutoring session. First the PA selected a topic and a problem. The PSM was then invoked to see if the student would make a mistake or answer correctly. In event of an incorrect answer, the PA was responsible for finding the hint that appeared best. From the PA's standpoint, this was identical to interacting with an actual student.

Figure 4 shows the improvement of the PA's performance at minimizing the amount of time students spent per problem. The x-axis is the number of trials (in thousands) the agent has spent learning. The y-axis is the exponential average of the amount of time the simulated student required to solve a problem. Performance started at around 40 seconds per problem, and eventually reduced to around 16 seconds per problem.

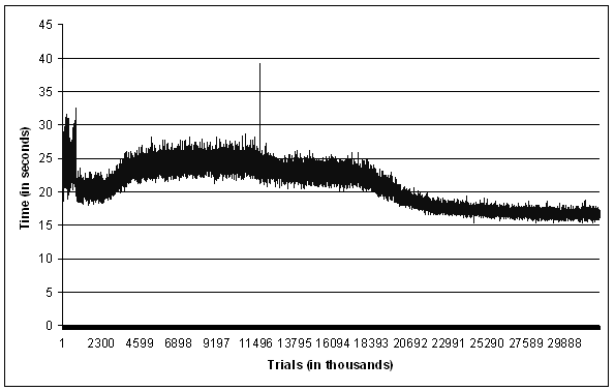

Figure 4: Improvement of pedagogical agent.

Compromises

Ideally, ADVISOR would have complete control over AnimalWatch's teaching actions. However, there were two difficulties with this. First, there was a limit on how frequently students could see a particular word problem template. Templates are skeletons for word problems, and problems are dynamically created with numbers in a range specified by the template. If all of the templates for a particular topic had been seen recently, the tutor was not permitted to select that topic. Furthermore, when ADVISOR's goal is to minimize the amount of time per problem, it has a tendency to select templates that involve small numbers. There is a limited supply of such templates. Thus, with with some probability (currently 0.1), the tutor uses a random level of difficulty to select a problem template. These issues are not the fault of the AI components, but rather result from our limited number of templates; as more are entered these restrictions can be removed.

Second, the classic AnimalWatch tutor only selects topics for which the student is "ready" (i.e. has not mastered, but has some understanding of the prerequisites). All of the training data provided to the PSM were biased in this manner. That is, there were no instances of students being shown a topic for which they did not understand the prerequisites. This makes it difficult for the PA to predict how students will react when presented with a problem for which they are not prepared. We had difficulty finding a graceful solution to this, and ultimately compromised. When using ADVISOR, the list of topics would first be screened so that only "ready" topics were considered. ADVISOR then used its standard decision-making strategies on this abridged list of topics. This difficulty is more fundamental to the AI architecture. By combining short- and long-range prediction agents, our architecture allows training data to be generalized to novel situations. However, it is unclear how to resolve such severe biases in the training data.

Validating ADVISOR

Given that both the PSM and PA have been validated, it is still necessary to the combined ADVISOR architecture. It is possible that the simulation of the PA and PSM working together is incorrect. Therefore, we performed an evaluation study comparing the performance of the classic AnimalWatch tutor with an ML version whose goal was to minimize the amount of time per problem. A possible objection is that time explicitly appears in the PSM, so is not an interesting choice for a reward. However, it is difficult to conceive of a reward that does not appear in the PSM. If the PA is being rewarded for something that cannot be directly observed, the question arises as to how the reward signal is generated. If the PA is rewarded for an observable event (such as time per problem), why not have the PSM predict (a version of) this to improve the model's accuracy?

This experiment was conducted with an urban, sixth grade class in a magnet school for students in the arts. This is

in contrast to our earlier studies, which were used to construct the PSM, which utilized rural/suburban fourth and fifth grade students. Students were randomly assigned to one of two groups. One condition used the classic AnimalWatch tutor and served as a control. The second, experimental, group used a tutor which reasoned using the ADVISOR architecture. The only difference between the two conditions was the AI method used for selecting topics, problems, and feedback. Students were assigned to an experimental condition based on where they sat on the first day. To avoid problems with similar students sitting near each other, physically adjacent computers were in opposite conditions (e.g. experimental, control, experimental, control, ...). Unfortunately, classes did not fill the entire lab, so some machines were unused. We wound up with a split of 60% of the students using the ML version (N=58), and 40% (N=39) using classic AnimalWatch.

Results

Students taught with ADVISOR averaged 27.7 seconds to solve a problem, while students using the classic version of AnimalWatch averaged 39.7 seconds. This difference was significant at P<0.001 (P$\approx 10^{-23}$ (2-tailed t-test)). Just as important, the difference is meaningful: reducing average times by 30% is a large reduction. Thus, the agent made noticeable progress in its goal of reducing the amount of time students spent per problem. We are not arguing this is necessarily the best pedagogical goal, just what we asked the agent to do.

To ensure that students in each condition were equally balanced with respect to math/computer ability, we examined their scores on a Piaget test of cognitive ability and their performance on the first problem seen using the tutor. The Piaget test has been found to correlate with both with speed and accuracy of problem solving on our tutor. Students in the control group had an average score of 6.7 (out of 10), while students in the experimental group had an average score of 6.6. This difference is insignificant (P=0.73, 2-tailed t-test).

Another metric for ensuring students were equally balanced is to examine first problem performance. The first problems students see (in both groups) is very easy, and almost always an addition of whole numbers problem. Thus, differences in time are largely a function of how experienced students are at learning a new interface as well as their math ability. Students in the control group required 59.8 seconds to solve the first problem, while students in the experimental group required 64.3 seconds. This difference is not significant (P$= 0.63$, 2-tailed t-test). Thus we feel safe in assuming the students in the two groups did not significantly differ in their incoming math abilities.

Further evidence of ADVISOR's ability to adapt instruction can be seen in Table 1. Students in the experimental group solved whole (P<0.001) and fraction problems (P=0.02) significantly faster than students in the control group. Prefraction problems are relatively homogeneous, so it is difficult to improve speed much on these. Students in the experimental group saw relatively fewer whole number, but more prefraction and fraction problems. Experimental group students saw more fractions problems, even though such problems take more time on average, because of the restriction AnimalWatch placed on topic selection. Experimental group students finished the whole number and prefraction topics relatively quickly, so were forced to work on fraction problems (as these were the only "ready" topics).

		Control	Experimental
Whole	Percentage	73.6%	60.0%
	Time	43.4 sec	**28.1 sec**
Prefract	Percentage	19.3%	27.3%
	Time	22.7 sec	21.7 sec
Fraction	Percentage	7.2%	12.7%
	Time	44.5 sec	**38.5 sec**

Table 1: Summary of performance by topic area.

Particularly impressive is that experimental group students were faster at solving fraction problems in spite of having a significantly lower Piaget score. Students in the control group *who got to fraction problems* averaged a Piaget score of 7.9, while students in the experimental group averaged 7.3; this difference is significant at P<0.001. Thus, in spite of being less restrictive about which students saw fraction (i.e. difficult) problems, the experimental group was still able to get through such problems more quickly.

Conclusions

We have constructed an intelligent tutor that makes its teaching decisions using a novel AI architecture. The ADVISOR architecture first learns a model of how the population of students performs using the tutor. Then this model is used to train an RL agent that is responsible for attaining a predefined teaching goal. Due to the state representation we have chosen, it is possible to train the RL agent with data obtained from a tutor that uses a different teaching strategy. It is also possible to generalize learned policies across different populations, as our agents were trained with rural fourth- and fifth-graders but were tested with urban sixth graders.

We have evaluated each component of our architecture in isolation and found performance to be acceptable. Our evaluation study of the entire architecture with students showed that it learned a teaching policy that resulted in students correctly answering questions in a reduced amount of time (compared to the traditional tutor). This provides evidence that it has learned to optimize its teaching to meet the specified goal. For our first experiments with ADVISOR we used a fairly straightforward goal. Allowing for more general teaching goals is part of our future work.

This combination of the PSM and PA is used by ADVISOR to make all of its teaching decisions. This is advantageous, since as we make progress at improving these agents, all of the tutor's teaching becomes more accurate and/or more adaptive. Although these agents have been constructed and evaluated within the context of AnimalWatch, the general architecture should apply to other intelligent tutors.

Data-driven techniques for tutor construction are becoming more feasible. Ten years ago, the idea of 11,000 train-

ing instances and evaluation studies with 100 students would have been unrealistic. As studies become larger, it becomes less necessary to rely on pedagogical theory to guide decision-making and more possible to directly construct models from statistical patterns in the data. Such models appear to generalize across populations, as the PSM and PA were constructed with a different population of students than those who used the final ML tutor.

Future work

Our future goals include improving the performance of the PSM and PA. The PSM is accurate at predicting time, and somewhat accurate at predicting the correctness of student responses. A more sophisticated function approximation technique, such as neural networks, could result in increased accuracy. This strategy also applies to the PA, which could benefit from a more general learning scheme.

We are also investigating improving performance via online, real-time learning. This would enhance the tutor's adaptivity. The tutor records how students perform when using the tutor–in fact it saves the same state information that was used to generate the PSM. However, the PSM is not being updated online. What is needed is some means of integrating the current student's actions with the PSM, while giving these actions higher weight (otherwise the thousands of datapoints from the population will swamp any information about the individual student).

Even if the PSM can be updated online, it is necessary to modify the pedagogical agent's teaching policy to reflect this. This is the technically more challenging task. Reinforcement learning takes many CPU cycles to converge. Given that we are running on fairly typical classroom computers, valid questions are how much learning can we hope to achieve in a short amount of time, and how can we ensure the online learning does not interfere with the response time of the tutor. Prior research has shown benefits to adding information from individual students to population models(Beck & Woolf 1998). However, we have not yet found methods for updating the teaching policy while the student is using the tutor.

ADVISOR adapted its teaching to fit a (relatively) straightforward goal: "minimize the amount of time students spend on a problem." More complicated goals would both be a tougher test of the architecture, but more pedagogically useful. For instance, a system that receives its highest reward for students taking 30 to 45 seconds per problem would probably do a better job at keeping the problems "hard enough" for students. However, such goals are more difficult to learn, but should not require major adjustment to our architecture. We hope to achieve this by scaling up the pedagogical agent with more complicated learning mechanisms for our PSM and PA.

A more ambitious plan is to allow larger-scale teaching goals to be provided to the PA. Currently, goals must be phrased on a per-problem level. Ideally, it would be possible to provide goals such as, "Have the students master prefractions material as quickly as possible." This is beyond the scope of our simulation used to train the PSM and PA. Scaling up this simulation to work across multiple problems, and allowing reward to be (very) delayed is a challenging task.

Acknowledgements

This work has been funded by the National Science Foundation program HRD 9714757. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. We acknowledge the contributions of David Marshall in the implementation of AnimalWatch, and Ivon Arroyo for her work with implementation and design of AnimalWatch's hinting.

References

Arroyo, I.; Beck, J. E.; Schultz, K.; and Woolf. 1999. Piagetian psychology in intelligent tutoring systems. In *Proceedings of the Ninth International Conference on Artificial Intelligence in Education*.

Baffes, P., and Mooney, R. 1996. A novel application of theory refinement to student modeling. In *Proceedings of American Association for Artificial Intelligence*, 403–408.

Beck, J. E., and Woolf, B. P. 1998. Using a learning agent with a student model. In *Proceedings Fourth International Conference on Intelligent Tutoring Systems*, 6–15.

Beller, M., and Gafni, N. 1996. The 1991 international assessment of educational progress in mathematics and sciences: The gender differences perspective. *Journal of Educational Psychology* 88:365–377.

Bloom, C. P. 1996. Promoting the transfer of adanced training technologies. In Frasson, C.; Gauthier, G.; and Lesgold, A., eds., *Proceedings of Third International Conference on Intelligent Tutoring Systems*, 1–10. Springer.

Burton, R. 1982. Diagnosing bugs in a simple procedural skill. In Sleeman, and Brown., eds., *Intelligent Tutoring Systems*, 157–182. Academic Press.

Chiu, B. C., and Webb, G. I. 1998. Using decision trees for agent modeling: Improving prediction performance. *User Modeling and User Adapted Interaction* 8(1-2):131–152.

Conati, C.; Gertner, A.; VanLehn, K.; and Druzdel, M. 1997. On-line student modeling for coached problem solving using bayesian networks. In *Proceedings of the Seventh International Conference on User Modeling*, 231–242.

Crites, R., and Barto, A. 1998. Elevator group control using multiple reinforcement learning agents. *Machine Learning* 33:235–262.

Gertner, A. S.; Conati, C.; and VanLehn, K. 1998. Procedural help in ANDES: Generating hints using a Bayesian network student model. In *Fifteenth National Conference on Artificial Intelligence*, 106–111.

Sutton, R., and Barto, A. 1998. *An Introduction to Reinforcement Learning*. MIT Press.

Tesauro, G. 1995. Temporal difference learning and TD-Gammon. *Communications of the ACM* 38(3).

VanLehn, K.; Ohlsson, S.; and Nason, R. 1994. Applications of simulated students: An exploration. *Journal of Artificial Intelligence in Education* 5(2):135–175.