Directing Development Effort with Simulated Students

Joseph E. Beck

Robotics Institute
Carnegie Mellon University
Pittsburgh, PA. U.S.A.
joseph.beck@cmu.edu

Abstract. Our goal is to find a methodology for directing development effort in an intelligent tutoring system (ITS). Given that ITS have several AI reasoning components, as well as content to present, evaluating them is a challenging task. Due to these difficulties, few evaluation studies to measure the impact of individual components have been performed. Our architecture evaluates the efficacy of each component of an ITS and considers the impact of a particular teaching goal when determining whether a particular component needs improving. For our AnimalWatch tutor, we found that for certain goals the tutor itself, rather than its reasoning components, needed improvement. We have found that it is necessary to know what the system's teaching goals are before deciding which component is the limiting factor on performance.

1 Introduction and Motivation

An intelligent tutoring system (ITS) is a complex piece of software. In addition to using a student model (SM) to interpret the student's actions and a pedagogical module to selecting teaching decisions, there is also the tutor itself. ITS designers must be concerned with how hints actually appear on the screen, the pedagogical principles used, what types of problems and feedback are available, etc.

It is not surprising that little work has been done at assessing which components of the system work and which do not. The work by Shute [6] at testing the efficacy of her tutor's pedagogical decisions is one of the rare examples of this type of evaluation. Given that this research showed the existing pedagogical module might not have been helpful, there is considerable potential gain from using this type of evaluation.

Unfortunately, there are two factors working against this approach. First, human evaluation studies are expensive. Even if subjects aren't paid, getting an ITS ready for "prime time," installing it in the lab to be tested, and diagnosing why it doesn't work on their computers costs a large amount of time. Second, there is the problem of combinatorics: studying whether the pedagogical module is helpful or not gives 2 experimental conditions. If one is interested in the SM's effectiveness that gives 4 conditions. If there are 3 different types of hints, and you are curious which one works best... Finding sufficient subjects for all of these conditions can be difficult.

This difficulty is unfortunate, as examining each component's impact not only tells us how well our past efforts worked, but it also informs us where we should direct our future efforts. If an evaluation shows that improving the SM would greatly enhance performance, it might not be sensible to spend effort on improving the ITS's hints.

Therefore, our goal is to find a low-cost way to assess the performance of each component of an ITS. If we can determine which components are not performing as well as they should be, we will have a method for directing development efforts in an ITS. Our approach for this is to use simulations of actual students [7] We conduct this research in the context of the AnimalWatch [1] arithmetic tutor.

2 Prior use of Simulated Students

VanLehn et al. [7] performed some of the first work with simulated students, or *simulees*. These simulees were constructed via a detailed cognitive task analysis of how students solve problems. This work emphasized constructing a runnable model of student behavior and using it to test systems. The testing was primarily to make sure the tutor's behavior was not overtly incorrect. For example, the simulees were able to detect that some words were used and only defined in later lessons. This idea of using such "fake" data for evaluating a tutoring system is appealing.

Other work in using simulated students for ITS design was done by Mertz [4]. This work constructed a learning agent based on the Soar architecture [5]. The problems and instructions were recoded so as to be understandable by the Soar agent, and then the simulated learner would attempt to solve the task. The Soar agent screened an intelligent tutor for weaknesses in its design. Since Soar learns from trial to trial, sub procedures that are useful will be reused in later exercises. This learning allows the tutor designer to consider ordering of exercises, or even how to best place the components for solving a particular exercise.

However, this process is time intensive: every piece of textual instruction and every problem situation within the tutor must be converted into a representation understandable by Soar. Furthermore, to determine if the ITS's lessons are well constructed, it is necessary to examine the production rules learned by the Soar-agent.

This technique uncovered several potential flaws in the tutor's design: missequenced problems, needless divergence from standards used in other problems, and inefficient problem solving procedures. This list of detected problems is interesting and impressive. Although expensive to construct, it is not clear that an automatic detection scheme that did not involve fine-grained cognitive modeling would be capable of doing this.

A drawback of such schemes is that they can be difficult to construct and require expert knowledge. Furthermore, if the same cognitive modeling approach is used to design and to test the system, there is a problem with circular reasoning.

3 Our Architecture

Although this paper explores using ADVISOR as an evaluation tool, ADVISOR's goal is to automate decision-making in an ITS. ADVISOR learns how students behave by observing a population of users and, with these data, generates a set of teaching strategies that optimize instruction. The teaching strategies adjust the tutor's behavior to try to meet a configurable teaching goal. Currently, teaching goals are specified on a per problem basis. So goals such as "learn the curriculum quickly" are not permitted. However, goals such as "Present problems so that the student makes

one mistake per problem, limit the amount of help the tutor gives, and try not to have it take too much or too little time," *are* permitted. Of course, it is necessary to quantify exactly how important each restriction is, how much time is "too little," etc.

To build ADVISOR, we used a layered ML architecture. This design allowed each component to be considered and optimized independently of other aspects of the project. One layer of learning is concerned with describing student behavior in different contexts. The other layer determines how to map this description of the student's behavior onto a correct teaching action, see Fig. 1.

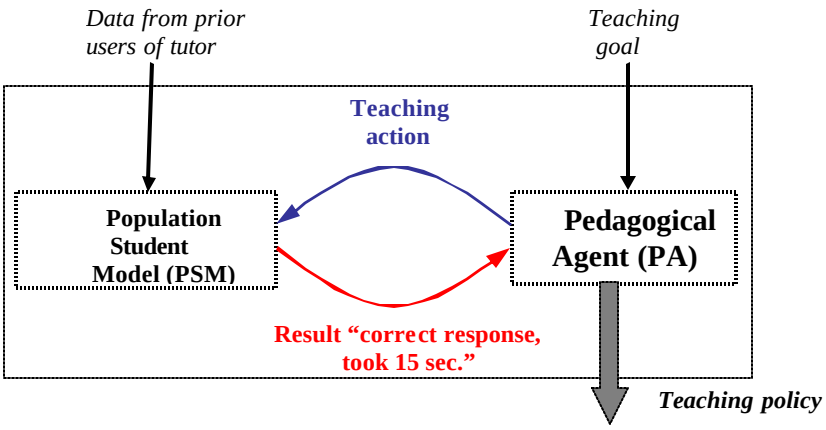

Fig. 1. ADVISOR architecture

3.1 The Population Student Model

The population student model (PSM) is responsible for predicting how the student will react in given situation. The PSM is constructed by data mining logs of previous users of the tutor. The current PSM used 10,000 interactions with students. Note that an "interaction" refers to the student submitting a response to the tutor, not to the number of actual users. In this case, 100 students used the tutor for 3 to 5 hours.

When we analyzed the interactions between students and the AnimalWatch tutor, we had 48 features to describe the current situation [2], and each interaction was cast in terms of those features. Given a feature vector to describe the current situation, and labeled training instances, it is straightforward to use a supervised machine learning technique to construct a model that maps the set of descriptive features to how the student will behave. Specifically, the PSM takes the set of features and predicts:

1. How long the student will take to respond
2. Whether the student's response will be correct

This PSM was constructed using linear regression [2]. However, the specific technique used is not important for this discussion.

Two important points are that the PSM makes different predictions for different students, and it is capable of acting as a simulee. Since the PSM makes its predictions based on features that describe (among other things) the student using the tutor, it will give different predictions for different students. So, although it is derived from data

from a population, it is not a "stereotype" approach to student modeling. The PSM can take an arbitrary state and predict how the student will behave. For example, when queried about how the student will perform in state ***S***, the PSM would compute:

1. Probability of correct response, and the time the student required.
2. Probability of an incorrect response, and the time the student required.

The two probabilities will sum up to 1.0, but the predicted times differ.

3.2 The Pedagogical Agent

The Pedagogical Agent's (PA) job is to interact with the PSM, and to use it to compute how it should teach to meet the specified teaching goal. By experimenting with the PSM, the PA is able to determine a policy (i.e. strategy) to meet the goal. The PA makes the same teaching decisions that the AnimalWatch tutor does:

- The topic on which the student will work (e.g. subtract whole numbers)
- The problem on which the student will work (e.g. 6-4 Vs. 1003-847)
- What feedback to provide to the student in event he makes a mistake

Since the PA examines the effects of teaching actions on this simulation of a student, it is not necessary to experiment with actual students to determine how to teach them. This architecture was tested in a classroom and found to work [3].

Finally, we have a simulation of an ITS, specifically the AnimalWatch mathematics tutor (ADVISOR is not specific to a particular ITS). This simulation determines which teaching actions are available to the PA at each point.

Although several simulations are required for ADVISOR, building each of them is not difficult. The hardest is a simulation of the student. We have chosen to build a simulation of the student (the PSM) that works at a coarse grain size by only predicting time and correctness of the student response. This coarseness limits the scope of the current ADVISOR architecture, but, by avoiding fine-grained details, model construction cost is lower and the simulation is more computationally tractable.

The original purpose of ADVISOR was to learn an optimal teaching policy. However, its component architecture and its ability to reason without using additional human subjects, make it ideal for examining the performance of each part of an ITS.

4 Goals

The goal of this research is to use the ADVISOR architecture to determine how to direct engineering effort on an ITS. This work is being done in the context of the AnimalWatch tutoring system, which is designed to teach arithmetic.

To evaluate ADVISOR, we examine how well it can perform in what we call the *target time* task. For this task, ADVISOR must get students through a problem in a specific amount of time. If ADVISOR takes the specified amount of time it receives a reward of 1.0, but is penalized linearly for taking a longer or shorter amount of time. For example, for a penalty of 0.025 per second and a goal of 45 seconds, if a student finished a problem in 38 seconds, ADVISOR would receive a reward of 1.0 - (45-38) * 0.025 = 0.825. ADVISOR's task is to get as large a reward as possible.

An important point to note is that the objective of a high target time is not to make the student's life difficult or to mislead him. Rather, it is a method for controlling

problem difficulty. A teacher interested in basic skills practice (e.g. math fact retrieval) could set a goal of 5 seconds as a target time, a teacher interested in making students think about each problem could assign a goal of 45 seconds.

For a goal of 45 seconds, ADVISOR cannot give a very simple problem to the student since he would solve it very quickly (e.g. repeatedly presenting 1+1 would not achieve a high reward). With respect to feedback, if a student provides an incorrect response after 5 seconds, ADVISOR can provide non-specific help such as "try again." If a student enters an incorrect response after 40 seconds, the tutor will have to provide effective help in order to avoid being penalized. So ADVISOR's strategy will have to be dynamic based on how the student is progressing through the problem.

It is possible to argue that mistakes rather than time are a better measure of student performance. There is merit to this argument, but neither metric does a good job at telling the entire story. Observing sixth-graders using the AnimalWatch tutor shows several cases of a student getting a problem such as "6+7" and answering it immediately. Other students stop, put up six fingers, and then count with their fingers to reach the final total. Stating that both of these students made no mistakes glosses over tremendous differences in how challenged they were by this problem.

There is nothing specific about the ADVISOR architecture requiring it to optimize time. We have run experiments where ADVSOR had to minimize the amount of help students received, the number of mistakes they made, while also being heavily penalized for providing problems the students can solve without error. Of course, this goal is somewhat contradictory, and all facets cannot be simultaneously optimized, but ADVISOR did mange to find a solution that was somewhat surprising.

We tested ADVISOR with 120 different target times (from 1 second up to 120 seconds). For these tests we used a variety of simulated students. At the start of each simulation run a student was generated randomly according to the distribution of characteristics in the population we used to train the PSM (e.g. proficiency at certain topics, score on Piaget test of development). Fig. 2 shows performance on this task. For example, when given a target time goal of 36 seconds, ADVISOR received an average reward of 0.6 (averaged across 500 runs of the simulation). This set of simulation runs was done with the standard PSM, but a fairly weak PA. Performance starts out fairly low (0.45), for time goals of 25 seconds it reaches its maximum (0.80), and performance smoothly falls off from there.

One question is why doesn't ADVISOR achieve a performance of 1.0 for all time goals? If it has a simulation of the student to work with, a simulation of the tutor, and can look ahead to see longer term effects, what is holding it back? These restrictions are precisely the set of issues that are (or should be) considered when evaluating an ITS: the accuracy of the SM, the quality of the pedagogical model's decisions, and the quality of the actual tutorial interventions. Although hard to evaluate in the general case, considering the target time task and seeing performance graphically makes this clearer. What are the limiting factors of ADVISOR's performance?

1. *Weakness in the Pedagogical Agent.* The PA is not capable of reasoning perfectly with the information provided by the PSM. To search far enough ahead to consider all of the possibilities is not possible.
2. *Inaccuracy of the Population Student Model.* The PSM is not a perfect model of students. When the PA queries it to determine the effect of a teaching action, the PSM returns two possibilities, each with an associated

probability. The first is the state resulting from an incorrect response; the second is the possibility the student will be correct. Since the PA is uncertain about the effects of its actions, this limits possible performance.

3. *Lack of flexibility of AnimalWatch tutor*. AnimalWatch has a broad, but limited, selection of hints and problem types. Perhaps in some situations, a certain kind of hint would be ideal but does not exist?

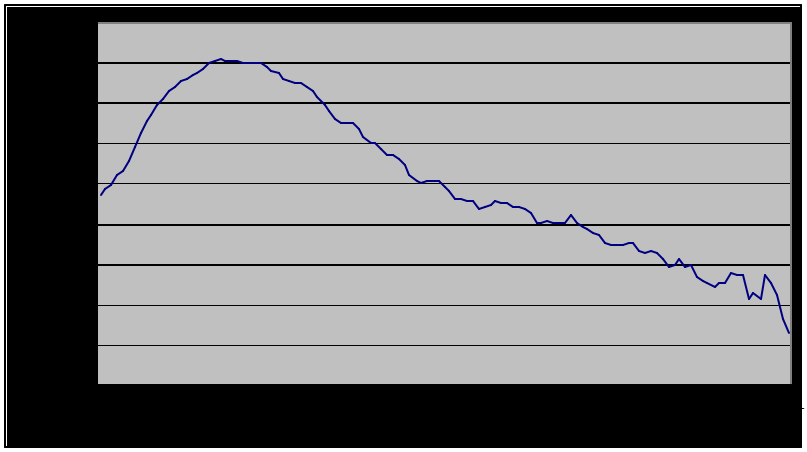

Fig. 2. ADVISOR's performance

All of these act to limit ADVISOR's performance. The questions we must ask is *when* are each of these the limiting factor, and *how much* can be gained by improving each component. We now construct a set of experiments to determine this.

5 Methodology and Experiments

To determine what constrains ADVISOR's performance, we vary the performance of the PA and PSM. Specifically, we attempt to construct an "optimal" version of each.

5.1 Optimizing the PA

A PA in an actual ITS must reason quickly. We set an upper limit of 0.25 seconds per decision as acceptable performance, since students are often impatient. With this limit, the PA cannot search as deeply to estimate the future repercussions of the PA's decision. We call a PA that can reason in less than 0.25 seconds per decision an "efficient" PA. Fig. 2 is from a PA of this type.

How well could a PA perform that had unlimited time to reason? Obviously, no ITS will ever have that much time, but removing the time constraint lets us explore the *maximum* impact of making the tutor's reasoning more efficient. We simulate this by allowing the PA to search many moves ahead and use either a heuristic function or rollouts to evaluate the leaf nodes. For use in an actual tutor, this does not give acceptable response times. Even in simulation runs, this is not practical.

Fig. 3 shows this performance. Efforts at improving the PA can move the curve representing efficient performance closer optimal performance. Possible ways to do this include making the search more efficient by using pruning techniques, or using machine learning to select actions. *However this is done, the maximum payoff for the design and engineering effort is the curve marked "Perfect PA."*

Note that for most time range, performance is still not close to 1.0. This indicates that there are other weaknesses in ADVISOR's reasoning.

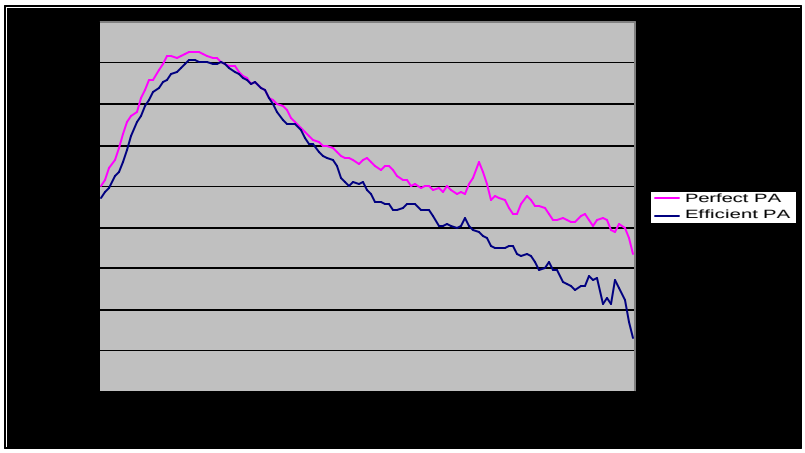

Fig. 3. ADVISOR with perfect pedagogical reasoning

5.2 Optimizing the PSM

How can we improve the performance of the optimal PA that is shown in Fig. 3? One method is to simulate how ADVISOR would perform given a perfect model of the student. Obviously, we don't have such a model. However, we can pretend we do by making the PSM's predictions deterministic. When the PA considers an action, it queries the PSM and receives two possible outcomes for whether the student will give the correct response or not. When the PA is done considering possible actions, and performs one of them, it informs the PSM of this fact. At this point, the PSM tells the PA which of the two outcomes actually occurs. Thus, the PA can plan ahead with some uncertainty, but is forced to accept an unlucky outcome.

We can simulate a "perfect" PSM by determining ahead of time which of the two events will occur. When the PA queries the PSM for possible outcomes, the PSM immediately decides which of the two will occur. It then sets the probability of that outcome to 1.0 and the probability of the other outcome to 0. This knowledge of how the student will behave allows the PA to plan perfectly.

Obviously this model is only perfect in the context of the simulation, so isn't applicable for a deployed tutor. However, it lets us examine the effect of having a perfect SM. Analyzing performance of a perfect SM gives us insight into the maximum gains we can get from improving student modeling through techniques such as eye tracking, finding a better set of features to provide to the PSM, etc.

Fig. 4 shows ADVISOR's performance with a perfect PSM. The top curve refers to a version of ADVISOR with an optimal PA and a perfect PSM. The bottom curve is the performance with an optimal PA with the standard PSM (the same as the top curve from Fig. 3). For time goals of less than 15 seconds there is little gain from improving the PSM. However, for time goals greater than 20 seconds there is a large difference in performance, so spending effort on the PSM makes sense *in this context.*

5.3 Impact of AnimalWatch

We have examined how the weaknesses in the PSM and PA limit performance. In the top curve of Fig. 4, ADVISOR has a perfect model of the student, and can search enough interactions ahead to achieve near-optimal performance with this model. Since this removes two of the three listed constraints on ADVISOR's performance, we must consider the fact that the AnimalWatch tutor itself is a constraint. Any remaining deficiencies in performance (i.e. anything below 1.0) are the fault of the tutor itself. For low target times, the tutor is the major constraint on performance. For high target times the tutor is not the main limiting factor.

6 Results

We have examined how weaknesses in the PSM (i.e. the student model in a typical ITS), the PA (i.e. the pedagogical module), and AnimalWatch (a proxy for a generic ITS) limit performance. What can we do with this knowledge?

Fig. 5 shows graphically the potential gains from improving each component of the system. The area between the bottom two curves represents the possible gains from improving the pedagogical agent. The area between the middle and top curves represents possible gain from improving the SM. The area above the top curve is how much can be gained by making the AnimalWatch tutor more flexible.

The first item to notice is that work at improving the system *must be in the context of a particular goal.* If the goal is to build a system where students practice basic math fact retrieval (e.g. a target time of 5 seconds per problem), spending effort at improving the system's ability to reason about the student and predict how he will act (i.e. the PSM or student model) is a waste of time. Similarly, improving the tutor's ability to select teaching actions (i.e. the PA or pedagogical module) is probably not productive. The best way to achieve gains is to broaden the scope of the AnimalWatch tutor; the lack of performance is not the fault of the AI. This fits with our observations of the tutor. AnimalWatch was designed to present students with somewhat lengthy word problems. For target times of 5 seconds, questions such as "What is 5-3?" would be more appropriate, and did not exist in our system.

For target times of 15 seconds or greater, there is a large gain for improving performance of the PSM. Thus, if researchers (or the teachers) are interested in having students solve problems of this type, continuing research on student modeling for this task is warranted. Target times must be around 60 seconds before finding methods to improve the tutor's pedagogical reasoning become profitable

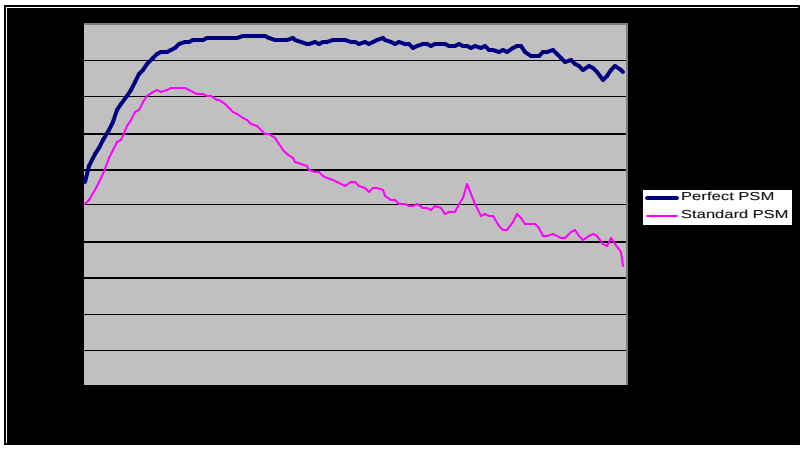

Fig. 4. ADVISOR with a perfect student model

Thus, before tinkering with a system or performing research to improve it, it is necessary to determine the goals for the system. If the system will be presenting problems that require students to spend more than 60 seconds on the problem, there is not much point is finding ways to improve the PA. Although we have found the PSM might be hard to improve, there is a large potential gain for most target times.

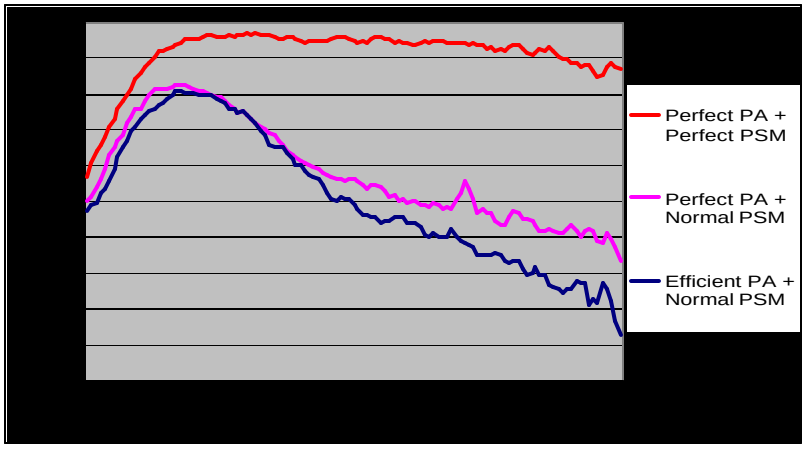

Fig 5. Directing development effort

7 Conclusions and Future Work

Although ADVISOR was designed to learn how to teach students, we have had success at applying it to answering questions about how to direct development efforts. By first enumerating constraints on performance, and then simulating their removal, it is possible to experimentally determine effects of improving various part of the tutor.

This technique permits designers to not conduct expensive ablation studies. Although such studies are known to be useful, the relative lack of them indicates a lack of eagerness to undertake an expensive (and possibly fruitless!) evaluation.

Therefore, by computationally modeling student performance and simulating a tutoring system, we can build systems whose teaching policies can be easily altered. This adaptability is a benefit to end users of the tutor. But we can also adjust the system's performance in the lab to direct future design work.

Two large outstanding issues are the grain-size of the PSM's reasoning and the PA's teaching goals. The PSM predicts student actions at a gross level, and does not attempt to reason at a cognitive level. This restricts the ADVISOR architecture to not being able to reason bout why a student might be confused.

Currently, teaching goals are on a per-problem basis. In principle, it is possible to construct a PA that reasons about the curriculum, but that would require many interactions with the PSM to predict that far into the future. This search is not computationally tractable or likely to be accurate. One possibility is to construct a higher level agent responsible for guiding the student through the curriculum, and have this agent interact with a different, more coarse-grained, PSM. Both of these are hard problems, but the ADVISOR architecture is not incapable of this scaling.

We have presented the results of using ADVISOR to perform an ablative evaluation of the AnimalWatch tutor. Prior research has shown the potential of simulating students. The results presented suggest that simulating other components of an ITS has considerable power for economically addressing a variety of issues.

Acknowledgements

This work has been funded from 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. Beverly Woolf, Carole Beal, Ivon Arroyo, Rachel Wing, and David Marshall were instrumental in creating and evaluating AnimalWatch.

References

1. Arroyo, I.. Learning Algebra with the Computer. In *Workshop on "Learning Algebra with the Computer" at the Fifth International Conference on Intelligent Tutoring Systems* 2000
2. Beck, J.E. and Woolf, B.P.. High-level Student Modeling with Machine Learning. In *Fifth International Conference on Intelligent Tutoring Systems* p. 584-593. 2000
3. Beck, J.E., Woolf, B.P. and Beal, C.R.. ADVISOR: A machine learning architecture for intelligent tutor construction. In *Seventeenth National Conference on Artificial Intelligence*. p. 552-557. 2000
4. Mertz, J.S., Using a Simulated Student for Instructional Design. *Artificial Intelligence in Education*, 1997. **8**: p. 116-141.
5. Newell, A., *Unified Theories of Cognition*. 1990, Cambridge, Massachusetts: Harvard Universtiy Press.
6. Shute, V.. SMART Evaluation: Cognitive Diagnosis, Mastery Learning and Remediation. In *Artificial Intelligence in Education*. p. 123-130. 1995
7. VanLehn, K., Ohlsson, S. and Nason, R., Applications of Simulated Students: An Exploration. *Journal of Artificial Intelligence in Education*, 1994. **5**(2).