Human Computation and Serious Games: Papers from the 2012 AIIDE Joint Workshop
AAAI Technical Report WS-12-17

A Review of Student Modeling Techniques in Intelligent Tutoring Systems

Brent Harrison and **David L. Roberts**
North Carolina State University
Raleigh, North Carolina 27606

Abstract

In this paper, we survey techniques used in intelligent tutoring systems (ITSs) to model student knowledge. The three techniques that we review in detail are knowledge tracing, performance factor analysis, and matrix factorization. We also briefly cover other techniques that have been used. This review is meant to be a repository of knowledge for those who want to integrate these techniques into serious games. It is also meant to increase awareness and interest as to the techniques available that can be integrated into serious games.

Introduction

The ability to model players can be a great aid to a serious game designer. Being able to determine what the player knows or how they are feeling can be used to greatly improve the learning experience of that player during gameplay. To do this, however, researchers must be able to create models of player behavior, or *player models*. In this paper, we are defining *player model* to mean a descriptive computational model that captures some aspect of player knowledge or behavior.

Similar issues arise when looking at intelligent tutoring systems (ITSs). An ITS must maintain a model of current student knowledge that is updated whenever the student answers a question or completes a task. As a result, there is a wealth of information on student modeling and knowledge modeling techniques with respect to ITSs. The goal of this paper is to encourage the exploration of student modeling techniques present in ITSs for use in serious games. To encourage this exploration, this paper presents a brief survey of student modeling techniques used in ITSs as well as examples of these systems in practice. The techniques that we will explore in detail are knowledge tracing, performance factor analysis, and matrix factorization.

Knowledge Tracing

The most commonly used technique for student modeling is knowledge tracing. Knowledge tracing involves monitoring a student's peformance on various questions and then uses their performance on these questions to update a model that describes their mastery of the skills, or knowledge concepts (KCs), required to answer the questions.

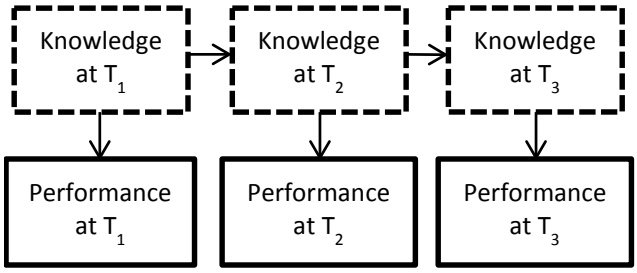

Figure 1: A DBN representing student knowledge over time. Boxes with dashed outlines indicate hidden nodes in the graph while boxes with solid outlines represent observable nodes in the graph.

Formulation

The original knowledge tracing technique (Corbett and Anderson 1994) used a simple dynamic Bayesian network (DBN). An example of a DBN can be seen in Figure 1. A DBN is a graphical model used to represent sequential data. There are two types of nodes in a DBN: hidden nodes and observable nodes. As the name implies, hidden nodes represent some information that is unknown whereas observable nodes represent information that can be seen. In knowledge tracing, the student's knowledge state is represented as a hidden node with two possible values, *learned* or *unlearned*. The observable nodes of this model are the student's performance on a given item. At every time step, which in this case would be every time the student answers a question, the student's knowledge state is either learned or unlearned with a certain probability. These probabilities are calculated using the following formula:

$$p(L_n) = p(L_{n-1}|e) + (1 - p(L_{n-1}|e)) * p(T) \quad (1)$$

In the above equation, $p(L_n)$ is the probability that the student is in the learned state at time n, e is the current set of evidence (whether or not the question was correctly answered), and $p(T)$ is the probability of transitioning from the unlearned state to the learned state. As you can see in the

equation, the probability that a student is in a learned state at a given time is the sum of two separate probabilities. The first of these is the probability that the student was in the learned state in the previous time step given that student's evidence. The second probability used is the probability that the student transitioned into the learned state if he was not in the learned state during the last time step. It is important to note that this model implies that a student can not *forget* once they have entered the learned state. In other words, it is not possible to transition from the learned state back to the unlearned state at any time.

Using equation 1, it is possible to make predictions about a student's performance on future questions. At each step, the probability that a student will answer a question correctly can be calculated using the following equation:

$$p(C_{i,s}) = p(L_{r,s}) * (1 - p(S_r)) + (1 - p(L_{r,s})) * p(G_r) \quad (2)$$

Here, $p(C_{i,s})$ is the probability that a student, s, answers question i correctly. In the second half of the equation, $p(L_{r,s})$ is the probability that the student, s, is in the learned state for the rule r that is needed to solve question i, $p(S_r)$ is the probability that a student will *slip* (answer incorrectly on an already known topic), and $p(G_r)$ is the probability that a student will *guess* correctly on a topic that is unknown. So, the probability that a student will answer correctly is also the sum of two probabilities. The first of these is the probability that the student has learned the topic times the probability that they do not *slip*. The second of these is the probability that the student has not learned the topic times the probability that the student has correctly guessed the answer. In this model, there are four parameters which must be fit: the guess parameter $p(G_r)$, the slip parameter $p(S_r)$, the initial probability of knowing a skill $p(L_0)$, and the transition probability $p(T)$.

Extensions and Examples

One problem that was soon discovered with the knowledge tracing model was that it consistently overestimated student performance. Corbett and Bhatnagar (1997) show that the standard knowledge tracing model will, on average, overestimate student performance by 8%. They claim that students are learning suboptimal rules that do not transfer from the tutoring system to the test. Corbett and Bhatnagar account for this behavior by adding a third state to this model. The new states of student knowledge are: unlearned, learned incorrectly, and learned correctly. To actually make predictions, Corbett and Bhatnagar included an additive term that is proportional to how well the student is able to acquire ideal rules. They showed that this extension to the framework essentially eliminated the overestimation that they had been seeing.

Another issue with the simple model presented above is that it requires that there only be one KC associated with each task. Therefore, observing student performance on a given task only updates the probability of a single KC being in the learned state. Conati *et al.* (2002) relax this restriction with the student model in the *Andes* system (Vanlehn et al. 2005). In other words, one task can have more than one parent node in the Bayesian network. In this system, credit assignment is done to determine which KC is most likely to have contributed to the performance of the student. This is achieved by examining rule prior probabilities. If the prior probability of one concept is much higher than the other, then it will receive most of the credit for the solution through Bayesian update rules.

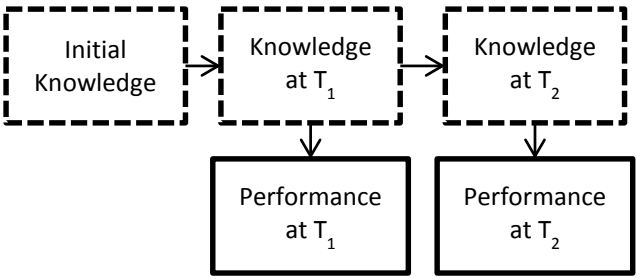

Figure 2: A DBN showing the initial student knowledge node added in. In the model put forth by Pardos and Heffernan, there would be an initial knowledge node for each student that could be modeled.

This model also suffers from the *identifiability problem* (Beck 2007). This occurs when the same training data can be fit equally well by different parameter values. This can have very noticeable effects on the generalizability of the model. Beck (2007) offers a solution using Dirichlet priors to guide parameter values towards their means in order to raise the probability that these parameters all converge to the same maximum.

Baker *et al.* (2008) claim that this introduces a new problem: *model degeneracy*. Model degeneracy occurs when parameter values cause the model to violate its conceptual foundations (such as students being more likely to answer correctly if they do not know a skill). Baker *et al.* address these issues by adding contextualized estimations of the guess and slip parameters. Recall that in the original model these parameters remained constant across all observations. In this model, they examine logs of student performances and determine whether each response was a slip or a guess by examining that student's future performance. They then construct models through machine learning that will extract rules about when a guess or a slip is likely to occur. They show that this method produces models that are less vulnerable to model degeneracy while being comparable to previous approaches in addressing the identifiability problem.

Pardos and Heffernan (2010) introduce a prior probability over student knowledge in order to add individualization to the model. This way, the model will be able to take into account individual differences between students. This was done by adding in student prior knowledge as the first node in the network. An example of this network can be seen in Figure 2. They showed that the best way to initialize this parameter was to use the student's average performance on all observed questions except for the one being predicted and then let this value be updated during parameter fitting.

Recently, the knowledge tracing model was applied to the

interactive narrative-based learning environment *Crystal Island*. To accomplish this, Rowe and Lester (2010) had to identify which in-game behaviors were most indicative of student knowledge and construct the DBN using this knowledge. They showed that using this technique to model student knowledge in a serious game had promise as it performed better than a random baseline classifier for several knowledge thresholds.

Possible Applications to Serious Games

For use in serious games, knowledge tracing would need to be applied to a game in which the tasks that the player were asked to complete had one knowledge concept that determined whether the player could complete it or not. Consider a puzzle that can only be solved by applying a certain KC. In this example, knowledge tracing could be used to predict that player's performance on this task, which can then be used to scale the difficulty so that it is appropriate for the current player.

Performance Factor Analysis

Originally introduced as an alternative to knowledge tracing, performance factor analysis (PFA) addresses some of the problems that are present in the knowledge tracing model. The main problem that PFA addresses is that of a task requiring multiple KCs to complete. Since PFA involves a sum over all contributing KCs, it is able to handle multiple KCs contributing to student performance.

Formulation

PFA (Pavlik, Cen, and Koedinger 2009) is based on a technique for educational data mining known as learning factors analysis (LFA) (Cen, Koedinger, and Junker 2006). In order to better understand PFA, we will first provide an overview of LFA.

The LFA model decomposes learning into three factors that describe different aspects of learning and KCs: the student's ability, the KC's easiness, and the KC's learning rate. Formally, the LFA model is given by the equation below

$$m(s, j \in KCs, n) = \alpha_s + \sum_{j \in KCs} (\beta_j + \gamma_j n_{s,j}) \qquad (3)$$

The formula *m* represents the accumulated learning for a student s using a KC j. In the above equation, α is used to encode student ability, β is used to represent the *easiness* of a KC, and $n_{s,j}$ is used to determine the weight of prior observations since γ is added for each observation. In order to convert these m values into predictions of probability, one must use the following equation:

$$p(m) = \frac{1}{1 + e^{-m}} \qquad (4)$$

For each task that will be presented to the student, the KCs associated with that question are stored in a Q-matrix. This is used to determine the KCs that best describe each task based on observed data.

The main issue with LFA is that it does not take into account student performance beyond the frequency of prior observations. In order to account for this, Pavlik *et al.* (2009) extended the LFA model to be appropriate for predicting student performance. They did this by, first, eliminating the α parameter since student ability is typically not known beforehand in an ITS. They also replace the n variable with two parameters, c and f, which track the student's correct responses and incorrect responses respectively. These two variables are scaled by the variables γ and ρ. The resulting equation is given by

$$m(s, j \in KCs, c, f) = \sum_{j \in KCs} (\beta_j + \gamma_j c_{s,j} + \rho_j f_{s,j}) \qquad (5)$$

Equation 4 is still used to convert these m values into probability predictions. In this model, there are parameters that are fit using logistic regression to create a model that best explains the training data. These parameters are β, γ, α, and/or ρ.

Extensions and Examples

One of the main issues with the PFA model is that, while it does take past successes and failures into account, it does not take into account the order that these successes and failures occurred in. Gong *et al.* (2011) extended this framework to include a decay factor which would decrease the effect that older questions had on current student knowledge. They showed that this version of PFA had a higher prediction accuracy than the original framework by a significant margin.

Another issue with the framework as it was originally formulated is that it assumes that the context in which a KC is learned has no effect on whether or not the concept is learned. In other words, mastery of two different tasks that consist of the same two KCs will result in the same amount of perceived learning for these two KCs. Pavlik *et al.* (2011) propose a new way of constructing the Q-matrix to help determine which KCs are most likely to be learned at a given time. In the original formulation, KCs control an entire column of the Q-matrix and so when they are fit, they are fit for all tasks. In the proposed method, each *cell* of the Q-matrix is fit. As a result of this, KC associations are found for specific tasks rather than finding KC associations across all tasks. Pavlik *et al.* tested this method on a least common multiples (LCM) skills dataset and compared the Q-matrix produced by their method against the Q-matrix produced by PFA. Pavlik *et al.* found that PFA did not associate any KCs to tasks in the dataset whereas their method found several associations.

Chi *et al.* (2011) explored adding in support for other types of instructional intervention into the PFA model. Their model, which they have named the instructional factors analysis model (IFM), also receives input about what they call *tell actions*. In the ITS that they study, a *tell action* is an instructional intervention where the system tells the student what steps to take next. Previous iterations of PFA do not take these actions into account since they do not have an immediately observable effect on the student. To account for

this, Chi *et al.* add in a variable to account for the number of *tell actions* that the student has previously received on a given KC to equation 5. They showed that IFM outperformed the standard implementation of PFA on data collected by a natural language physics tutor named Cordillera.

Possible Applications to Serious Games

PFA can be used in serious games in which many concepts may be necessary in order to complete a task. Using some of the methods mentioned above, a player's mastery of these individual concepts can be gleaned and then additional instruction/guidance can be done. Also, these concepts can be used to determine which concepts need to be improved upon so that the player can overcome certain challenges.

Matrix Factorization

Recently, matrix factorization has been used to predict student performance in tutoring systems. Matrix factorization is a technique that was originally used in recommender systems. The goal of a recommender system is to either recommend new items to a user based on that user's history, or to predict how a user will rate an item based on how they rated items in the past. Using this, Thai-Nghe *et al.* (2011a) claimed that the problem of predicting student performance is similar, computationally speaking, to that of predicting user ratings. With this in mind, they converted the matrix factorization algorithm into a form that makes it fit for predicting student performance.

Formulation

Matrix factorization is the task of approximating a matrix X as the product of two smaller matrices W and H. Here, $W \in \mathbb{R}^{U \times K}$ is a matrix containing the latent factors that describe each student and $H \in \mathbb{R}^{I \times K}$ is a matrix containing the latent factors that describe each task. In this framework, the performance of student u one task i is predicted using the following equation:

$$\widehat{p}_{ui} = \sum_{k=1}^{K} w_{uk} h_{ik} = (WH^T) \tag{6}$$

To use this equation, however, we must find optimal values for the elements of the matrices W and H. These values are typically found using gradient descent in order to minimize some error function. An example of this technique will be done with the aid of Figure 3. Let's assume that the matrices W and H seen in Figure 3 were obtained after training for $K = 2$ latent factors. If we wanted to predict the performance of Student 2 on Task 2 (which are shown in bold and italics in the figure), we would simply do the following:

$\widehat{p}_{ui} = (WH^T) = 0.11 * 0.94 + 0.32 * 0.63 = 0.305$ In this example, the student's predicted performance is 0.305. Since performance is typically reported a 0 if the student incorrectly answers a question and a 1 if the student correctly answers a question, we can conclude that it is unlikely that the student will correctly answer the question.

As it is, this model does not take into account *student bias*, the ability of the student, or the *task bias*, the difficulty of the task. To incorporate these into this model, three new variables are introduced: the global average performance μ, the student bias b_u, and the task bias b_i. The new equation for prediction becomes

$$\widehat{p}_{ui} = \mu + b_u + b_i + \sum_{k=1}^{K} w_{uk} h_{ik} \tag{7}$$

In the above equation, b_u is the average amount that the performance of student u deviates from μ and b_i is the average amount that the performance of task i deviates from μ. When training this model, these values are updated using the predictions made from the current values of W and H.

Now the only thing the model lacks is a way to describe time. To do this, Thai-Nghe *et al.* (2011) used a technique called *tensor factorization*. Tensor factorization is a generalized form of matrix factorization. Specifically, a new matrix Q is introduced which describes the context of a task (the time). After adding this extra component to the model, the equation for prediction becomes

$$\widehat{p}_{ui} = \mu + b_u + b_i + (\sum_{k=1}^{K} w_k h_k \phi_k) \tag{8}$$

$$\phi_k = \frac{\sum_{t=(T-T_{max}+1)}^{T} q_k(t)}{T_{max}} \tag{9}$$

In the above equations, q is a vector containing the K latent factors that affect time, and ϕ is the average value of q_k looking T_{max} steps into the past. In other words, this model will use the average performance over the last T_{max} tasks to predict the student's performance on the current task. It is interesting to note that this model does not have explicit parameters that account for *guess* and *slip* actions. This is because they are assumed to be accounted for as latent factors that describe the student and the tasks they perform.

Extensions and Examples

An issue with matrix factorization that was pointed out by Thai-Nghe *et al.* (2011b) is that it only considers the relationship between the student and tasks/skills. They point out that there are several other relationships that could be considered in an ITS, such as the relationship between tasks and the skills they require. To address this, they extend the matrix factorization method by using multi-relational matrix factorization (MRMF) (Lippert et al. 2008). MRMF is a generalization of matrix factorization in which multiple relationships and entities (such as *students* or *tasks*) can be used. Thai-Nghe *et al.* evaluated this method on the *KDD cup 2010* educational data mining dataset for *algebra* and *bridge to algebra*[1]. They showed that using MRMF results in an overall lower root mean squared error than both normal matrix factorization as well as knowledge tracing.

Recently, Zook *et al.* (2012) used tensor factorization to create models of player expertise in a skill-based mission game. In this work, they described missions in terms of what

[1] http://www.sigkdd.org/kddcup/index.php?section=2010&method=data

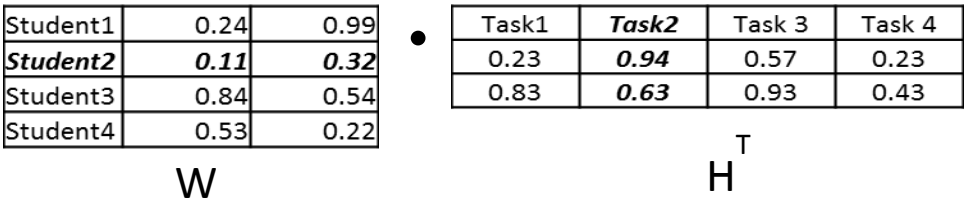

Figure 3: An example of using matrix factorization for student performance prediction.

skills are necessary to complete them. Using tensor factorization, Zook *et al.* were able to predict how player performance would change over time. They hypothesized that they can use this predicted player performance to customize the missions that a player would experience. What is interesting about this is that they could customize missions in order to produce a desired performance curve. This is also an example of a student model from an ITS being used outside of the ITS field. In this case, it has been applied to a game meant solely for entertainment.

Possible Applications to Serious Games

The main benefit of matrix factorization is that the knowledge concepts that are related to a task do not need to be explicitly known. This, however, is also its greatest weakness. If it is used to predict performance in a serious game, it will be difficult to determine why a player performed poorly or why they performed well. One possible application of matrix factorization to serious games is to use it to sequence tasks such that the player gets a mixture of tasks that they are skilled at and tasks that they are not skilled at.

Choosing Between Methods

As we have shown, each of these three methods have distinctive strengths and weaknesses; however, it may not be clear when one technique should be used over another. The easiest way to determine this is to examine how KCs relate to ITS questions (or tasks). If each question/task has only one required KC, then knowledge tracing is the strongest technique to use. If multiple KCs are required to perform tasks, then PFA is the most suitable technique. While techniques exist that allow knowledge tracing to handle tasks that rely on multiple KCs (such as the techniques used in the Andes system), it has been shown (Gong, Beck, and Heffernan 2010) that PFA will still outperform knowledge tracing on these tasks.

While knowledge tracing and PFA are ideal when the exact KCs are known beforehand, it is not always guaranteed that this knowledge will be available. When this is the case, matrix factorization is ideal since one does not need to know the exact KCs that influence student performance. Matrix factorization only requires that the number of latent factors be provided.

Other Methods

Due to length constraints, it is not possible to go into great detail about all of the significant student modeling techniques that exist in ITSs. In this section, we will provide a brief overview of other techniques and example systems that do not fit into the categories that we have outlined in the above sections.

In 1999, Zhou and Evens proposed a simple student model in which student answers are classified based on their correctness. The system then uses a planner along with the answer classification to determine the type of instructional intervention to execute next. Examples of interventions include giving the student a hint about the correct answer or giving the student related questions.

Stathacopoulou *et al.* (1999) proposed a *neuro-fuzzy* model which operates in four stages. The first stage involves fuzzifying numerical data that contributes to the evaluation of certain characteristics into one of three categories: small medium or large. The second stage involves converting these measurements into student characteristics such as learning speed. The third stage involves using a neural network to calculate the final fuzzy sets of student characteristics. The last stage involves using a backpropagation network to create nonfuzzy assessment of student characteristics.

There has been an increasing amount of research done in applying ensemble methods to student modeling. In machine learning, an ensemble method is a way to combine the output of many different models in order to improve prediction accuracy on a problem. Baker *et al.* (2011) used an ensemble of several popular student modeling techniques (such as knowledge tracing and PFA) created using linear regression to predict student performance on a genetics dataset. They found that ensembling these techniques did not provide any significant gains over using the base classifiers by themselves. It is important to note that they hypothesize that the reason that ensembling did not perform well is because the base models were, overall, too similar to each other.

Pardos *et al.* (2012) hypothesize that the reason for the poor performance, however, was dataset size. In their evaluation, they used a dataset from the ASSISTments platform (Razzaq et al. 2007) which contains 15 times the responses of the genetics dataset used by Baker *et al*. They tested several different ensembling techniques with this dataset and found that the best performing ensemble performed 10% better than the best performing individual student model.

Conclusion

In this paper, we have reviewed several techniques used in ITSs to model student knowledge with the hope that this will create an interest in transferring these models to serious games. We have also given examples of when these models have been used outside of ITSs in an effort to show that such a transfer is possible. This paper is meant to create discussion about modeling techniques that exist in other fields (ITSs in particular) and how they might be applied to the problems that researchers in serious games face today.

References

Baker, R.; Pardos, Z.; Gowda, S.; Nooraei, B.; and Heffernan, N. 2011. Ensembling predictions of student knowledge within intelligent tutoring systems. *User Modeling, Adaption and Personalization* 13–24.

Baker, R.; Corbett, A.; and Aleven, V. 2008. More accurate student modeling through contextual estimation of slip and guess probabilities in bayesian knowledge tracing. In *Intelligent Tutoring Systems*, 406–415. Springer.

Beck, J. 2007. Difficulties in inferring student knowledge from observations (and why you should care). In *Educational Data Mining: Supplementary Proceedings of the 13 th International Conference of Artificial Intelligence in Education*, 21–30.

Cen, H.; Koedinger, K.; and Junker, B. 2006. Learning factors analysis–a general method for cognitive model evaluation and improvement. In *Intelligent Tutoring Systems*, 164–175. Springer.

Chi, M.; Koedinger, K.; Gordon, G.; and Jordan, P. 2011. Instructional factors analysis: A cognitive model for multiple instructional interventions. *EDM*.

Conati, C.; Gertner, A.; and Vanlehn, K. 2002. Using bayesian networks to manage uncertainty in student modeling. *User Modeling and User-Adapted Interaction* 12(4):371–417.

Corbett, A., and Anderson, J. 1994. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User modeling and user-adapted interaction* 4(4):253–278.

Corbett, A., and Bhatnagar, A. 1997. Student modeling in the act programming tutor: Adjusting a procedural learning model with declarative knowledge. *COURSES AND LECTURES-INTERNATIONAL CENTRE FOR MECHANICAL SCIENCES* 243–254.

Gong, Y.; Beck, J.; and Heffernan, N. 2010. Comparing knowledge tracing and performance factor analysis by using multiple model fitting procedures. In *Intelligent Tutoring Systems*, 35–44. Springer.

Gong, Y.; Beck, J.; and Heffernan, N. 2011. How to construct more accurate student models: Comparing and optimizing knowledge tracing and performance factor analysis. *International Journal of Artificial Intelligence in Education* 21(1):27–46.

Lippert, C.; Weber, S.; Huang, Y.; Tresp, V.; Schubert, M.; and Kriegel, H. 2008. Relation prediction in multi-relational domains using matrix factorization. In *Proceedings of the NIPS 2008 Workshop: Structured Input-Structured Output, Vancouver, Canada*.

Pardos, Z., and Heffernan, N. 2010. Modeling individualization in a bayesian networks implementation of knowledge tracing. *User Modeling, Adaptation, and Personalization* 255–266.

Pardos, Z.; Gowda, S.; Baker, R.; and Heffernan, N. 2012. The sum is greater than the parts: ensembling models of student knowledge in educational software. *ACM SIGKDD Explorations Newsletter* 13(2):37–44.

Pavlik, P.; Cen, H.; and Koedinger, K. 2009. Performance factors analysis–a new alternative to knowledge tracing. In *Proceeding of the 2009 conference on Artificial Intelligence in Education: Building Learning Systems that Care: From Knowledge Representation to Affective Modelling*, 531–538. IOS Press.

Pavlik, P.; Yudelson, M.; and Koedinger, K. 2011. Using contextual factors analysis to explain transfer of least common multiple skills. In *Artificial Intelligence in Education*, 256–263. Springer.

Razzaq, L.; Heffernan, N.; Feng, M.; and Pardos, Z. 2007. Developing fine-grained transfer models in the assistment system. *Journal of Technology, Instruction, Cognition, and Learning* 5(3):289–304.

Rowe, J., and Lester, J. 2010. Modeling user knowledge with dynamic bayesian networks in interactive narrative environments. In *Proceedings of the Sixth International Conference on Artificial Intelligence in Interactive Digital Entertainment*, 57–62.

Stathacopoulou, R.; Magoulas, G.; and Grigoriadou, M. 1999. Neural network-based fuzzy modeling of the student in intelligent tutoring systems. In *Neural Networks, 1999. IJCNN'99. International Joint Conference on*, volume 5, 3517–3521. IEEE.

Thai-Nghe, N.; Drumond, L.; Horváth, T.; Krohn-Grimberghe, A.; Nanopoulos, A.; and Schmidt-Thieme, L. 2011a. Factorization techniques for predicting student performance. *Educational Recommender Systems and Technologies: Practices and Challenges (In press). IGI Global*.

Thai-Nghe, N.; Drumond, L.; Horváth, T.; Schmidt-Thieme, L.; et al. 2011b. Multi-relational factorization models for predicting student performance. *Proceedings of the KDD Workshop on Knowledge Discovery in Educational Data*.

Thai-Nghe, N.; Horvath, T.; and Schmidt-Thieme, L. 2011. Context-aware factorization for personalized student's task recommendation. In *Proceedings of the International Workshop on Personalization Approaches in Learning Environments*, volume 732, 13–18.

Vanlehn, K.; Lynch, C.; Schulze, K.; Shapiro, J.; Shelby, R.; Taylor, L.; Treacy, D.; Weinstein, A.; and Wintersgill, M. 2005. The andes physics tutoring system: Lessons learned. *International Journal of Artificial Intelligence in Education* 15(3):147–204.

Zook, A.; Lee-Urban, S.; Drinkwater, M.; and Riedl, M. 2012. Skill-based mission generation: A data-driven temporal player modeling approach.