# A Machine Learning Approach for Automatic Student Model Discovery

Nan Li and Noboru Matsuda and William W. Cohen and Kenneth R. Koedinger, Carnegie Mellon University

Student modeling is one of the key factors that affects automated tutoring systems in making instructional decisions. A student model is a model to predict the probability of a student making errors on given problems. A good student model that matches with student behavior patterns often provides useful information on learning task difficulty and transfer of learning between related problems, and thus often yields better instruction. Manual construction of such models usually requires substantial human effort, and may still miss distinctions in content and learning that have important instructional implications. In this paper, we propose an approach that automatically discovers student models using a state-of-art machine learning agent, SimStudent. We show that the discovered model is of higher quality than human-generated models, and demonstrate how the discovered model can be used to improve a tutoring system's instruction strategy.

## 1. INTRODUCTION

A student model is a set of *knowledge components (KC)* encoded in intelligent tutors to model how students solve problems. The set of KCs includes the component skills, concepts, or percepts that a student must acquire to be successful on the target tasks. For example, a KC in algebra can be how students should proceed given problems of the form *Nv=N* (e.g. $3x = 6$). It provides important information to automated tutoring systems in making instructional decisions. Better student models match with real student behavior. They are capable of predicting task difficulty and transfer of learning between related problems, and often yield better instruction. Traditional ways to construct student models include structured interviews, think-aloud protocols, rational analysis, and so on. However, these methods are often time-consuming, and require expert input. More importantly, they are highly subjective. Previous studies [Koedinger and Nathan 2004; Koedinger and McLaughlin 2010] have shown that human engineering of these models often ignores distinctions in content and learning that have important instructional implications. Other methods such as Learning Factor Analysis (LFA) [Cen et al. 2006] apply an automated search technique to discover student models. It has been shown that these automated methods are able to find better student models than human-generated ones. Nevertheless, one key limitation of LFA is that it carries out the search process only within the space of human-provided factors. If a better model exists but requires unknown factors, LFA will not find it.

To address this issue, we propose a method that automatically discovers student models not depending on human-provided factors. The system uses a state-of-art machine learning agent, SimStudent [Matsuda et al. 2009], to acquire skill knowledge. Each skill corresponds to a KC that students need to learn. The model then labels each observation of a real student based on skill application. We evaluated the approach in algebra using real student data. Experiment results show that the discovered model fits with real student data better than human-generated models, and provides useful insights in finding better instructional methods.

In the following sections, we begin with a review of SimStudent. Next, we report experiment results that demonstrate the benefits of the SimStudent model over the human-generated model. After this, we discuss the possible improvements that can be made to a tutoring system suggested by the SimStudent model. In closing, we discuss related work as well as future directions for this work.

Author's address: Nan Li; email: nli1@cs.cmu.edu; Nobour Matsuda; email: Noboru.Matsuda@cs.cmu.edu; William W. Cohen; email: wcohen@cs.cmu.edu; Kenneth R. Koedinger; email: koedinger@cmu.edu; 5000 Forbes Ave, Pittsburgh, PA 15232

## 2. A REVIEW OF SIMSTUDENT

SimStudent is an intelligent agent that inductively learns skills to solve problems from demonstrated solutions and from problem solving experience. It is a realization of programming by demonstration [Lau and Weld 1998] and employs inductive logic programming [Muggleton and de Raedt 1994] as one of its learning mechanisms. For more details about SimStudent, please refer to Matsuda et al. [2009].

### 2.1 Input

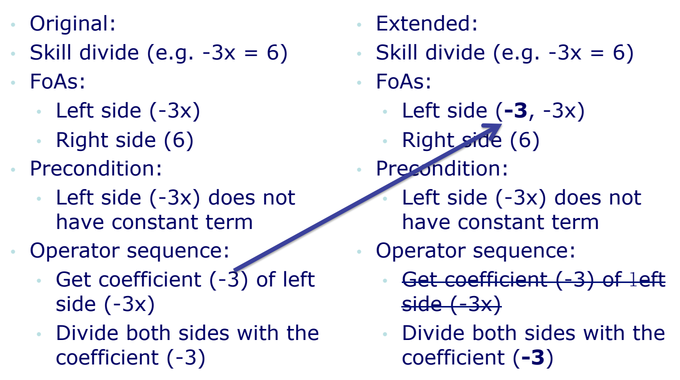
SimStudent is given a set of *feature predicate symbols* and a set of *operator symbols* as prior knowledge before learning. Each predicate is a boolean function that describes relations among objects in the domain (e.g. *(has-coefficient -3x)*). Operators specify basic manipulations (e.g. *(add 1 2)*, *(coefficient -3x)*) that SimStudent can apply to objects in the problem solving interface, like numbers or character strings. Operators are divided into two groups, domain-independent operators and domain-specific operators. Domain-independent operators (e.g. *(add 1 2)*) are basic manipulations that are applicable across multiple domains. Real students usually have knowledge of these simple skills prior to class. Domain-specific operators (e.g. *(add-term 5x-5 5)*, *(coefficient -3x)*), on the other hand, are more complicated manipulations that are associated with only one domain. From a learner modeling perspective, beginning students may not know domain-specific operators and thus providing such operators to SimStudent may produce learning behavior that is distinctly different from human students [Matsuda et al. 2009]. Operators in SimStudent (whether domain-independent or domain-specific) have no explicit encoding of preconditions and effects. This matches the intuition that human students often "know how" without "knowing when".

During the learning process, given the current state of the problem (e.g., *-3x = 6*), SimStudent first tries to find an appropriate *production rule* (skill knowledge acquired by SimStudent) that proposes a plan for the next step (e.g. *(coefficient -3x ?coef) (divide ?coef)*). If it finds one, it executes the plan, performs an action in the system interface, and waits for feedback from the human user/author/tutor. If the user provides positive feedback, SimStudent continues to the next step. If not, SimStudent records this negative feedback and may try again. If SimStudent does not find a production rule that generates a correct action, it requests a demonstration of the next step, which the user performs in the interface. SimStudent uses any negative feedback to modify existing productions. It uses the next-step demonstration, if provided, to learn a new production rule.

In the experiments we describe here, the user/author/tutor role is simulated by a hand-engineered algebra tutor [Koedinger et al. 1995], which provides SimStudent with feedback and next-step demonstrations as needed via an API. For each demonstrated step, the user/tutor specifies a tuple of ⟨*selection, action, input*⟩ (SAI tuple) for a skill. SimStudent is given *a skill label* (e.g. "divide") generated by the cognitive tutor, which corresponds to the type of skill applied. "Selection" in the SAI tuple (e.g. *-3x* and *6* for *-3x = 6*) is associated with elements in the graphical user interfaces (GUI). It shows where a "focus of attention" is —that is, where to look for relevant information. "Action" (e.g. entering some text) indicates what action to take with the "input" (e.g. *(divide -3)* for problem *-3x = 6*). In this example, the full plan might be to first retrieve coefficient and then to divide by it (e.g. *(coefficient -3x ?coef) (divide ?coef)*), but the tutor only demonstrates the final action (e.g., *(divide -3)*) to SimStudent. Taken together, the given information forms one record indexed by the skill label, *R*=⟨*label,* ⟨*selection, action, input*⟩⟩ (e.g. *R*=⟨*divide,* ⟨*(-3x, 6), input text, (divide -3)*⟩⟩*)*. In learning, SimStudent acquires one production rule for each skill label, based on the set of associated records gathered at that point.

### 2.2 Production Rules

The output of the learning agent is represented as production rules. Each production rule corresponds to one knowledge component. The left side of Figure 1 shows an example of a production rule learned by SimStudent. A production rule indicates "when" (precondition) to apply a rule to what information found "where" (focus of attention (FoA)) in the interface and "how" (operator sequence) the problem state should be changed. For example, the rule shown in the left side of Figure 1 would be read as "given a left-hand side



- Original:
- Skill divide (e.g. -3x = 6)
- FoAs:
  - Left side (-3x)
  - Right side (6)
- Precondition:
  - Left side (-3x) does not have constant term
- Operator sequence:
  - Get coefficient (-3) of left side (-3x)
  - Divide both sides with the coefficient (-3)

- Extended:
- Skill divide (e.g. -3x = 6)
- FoAs:
  - Left side (**-3**, -3x)
  - Right side (6)
- Precondition:
  - Left side (-3x) does not have constant term
- Operator sequence:
  - ~~Get coefficient (-3) of left side (-3x)~~
  - Divide both sides with the coefficient (**-3**)

Fig. 1. Original and extended production rules for divide in a readable format.

(i.e. *-3x*) and a right-hand side (i.e. *6*) of the equation, when the left-hand side does not have a constant term, then get the coefficient of the term on the left-hand side and divide both sides by the coefficient." The focus of attention of the production represents paths through the task-specific GUI interface that retrieve the items needed by the operator sequence. The precondition of a production rule includes a set of feature tests, representing preconditions for applying the rule. The operator sequence specifies a plan to execute.

## 2.3 Learning Mechanism

SimStudent uses three different learning components for the three parts of the production rules. The first component (the "where learner") learns how to focus attention on the relevant aspects of the interface by generalizing paths from the element for the interface as a whole to the specific elements of the interface that have the information needed to execute the operator sequence. The elements in the GUI are organized in a tree structure. In the algebra domain, the root node is a table node that links to columns, and each column has multiple cells as children. The "where learner's" task is to find the right paths in the tree to reach the nodes in the focus-of-attention (e.g. *Cell 11* and *Cell 21*). A FoA (e.g. *Cell 21*) can be reached either 1) by the path to its exact position (e.g. *Cell 21*) in the tree, 2) by a generalized path (e.g. *Cell 2?*, *Cell ??*) to its position. Therefore, given a set of FoAs from positive records, for each position, the "where learner" searches for one least general path that covers all of the FoAs at that position.

The second part of the learning mechanism is a precondition learner (the "when learner", which acquires the precondition of the production rule using the given feature predicates. The precondition learner utilizes FOIL [Quinlan 1990], an inductive logic programming system that learns Horn clauses from both positive and negative examples expressed as relations. For each rule, the precondition learner creates a new predicate that corresponds to the precondition of the rule, and sets it as the target relation for FOIL to learn. The arguments of the new predicate are associated with the FoAs. Each training record serves as either a positive or a negative example for FOIL. For example, *(precondition-divide -3x 6)* is a positive example for the new predicate *(precondition-divide ?$FoA_1$ ?$FoA_2$)*. The precondition learner also computes the truthfulness of all predicates bound with all possible permutations of FoA values, and sends it as input to FOIL. Given these inputs, FOIL will acquire a set of clauses formed by feature predicates describing the precondition predicate.

The last component is the operator sequence learner (the "how learner"). For each positive record, $R_i$, the learner takes the FoAs, $FoAs_i$, as the initial state, and sets the step, $step_i$, as the goal state. We say an operator sequence explains a FoAs-step pair, $\langle FoAs_i, step_i \rangle$, if the system takes $FoAs_i$ as an initial state and yields $step_i$ after applying the operators. For example, with the FoAs-step pair in the example, $\langle$*(-3x, 6), (divide -3)*$\rangle$, the operator sequence *(coefficient -3x ?coef) (divide ?coef)* is a possible explanation for this pair. The learner searches for the shortest operator sequence that explains all of the $\langle$*FoAs, step*$\rangle$ pairs using iterative-deepening depth-first search.

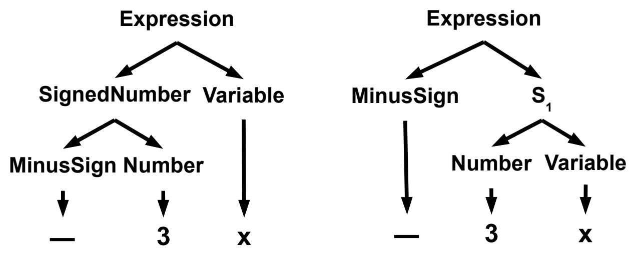


Fig. 2. Correct and incorrect parse trees for $-3x$.

Last, although SimStudent tries to learn one rule for each label, it might fail to do so (e.g., when no operator sequence can explain all records). In that case, SimStudent learns a disjunctive rule just for the last record. This effectively splits the records into two clusters. Later, for each new record, SimStudent tries to acquire a rule for each of the clusters with the new record, and stops whenever it successfully learns a rule with one of the clusters, or creates another new cluster.

## 2.4 Extending SimStudent to Learn Deep Features

Previous study [Chi et al. 1981] has shown that one of the key differences between experts and novices is that experts view the world in terms of deep features, whereas novices only see shallow features. Recently, we have extended SimStudent to support acquisition of deep features using Li et al. [2010]'s algorithm. They model deep feature learning as a grammar induction problem. In the algebra domain, expressions are modeled with a probabilistic context free grammar (PCFG), and the deep features (e.g., "coefficient") are intermediate symbols in the grammar rules. Moreover, Li et al. [2010] showed that student errors can be modeled as incorrect parsing, as shown at the right side of Figure 2. Li et al. [2010]'s deep feature learner extends an earlier PCFG learner [Li et al. 2009] to support feature learning and transfer learning.

The input of the system is a set of observation-feature pairs such as ⟨*-3x, -3*⟩. The output is a PCFG with a designated intermediate symbol in one of the rules set as the target feature. The learning process contains two steps. The system first acquires the grammar using Li et al. [2009]'s algorithm. After that, the feature learner tries to identify an intermediate symbol in one of the rules as the target feature. To do this, the system builds parse trees for all of the observation sequences, and picks the intermediate symbol that corresponds to the most training records as the deep feature. To model transfer learning, Li et al. [2010] further extend the feature learner to acquire PCFGs based on previously acquired knowledge. When the learner is given a new learning task, it first uses the known grammar to build parse trees for each new record in a bottom-up fashion, and stops when there is no rule that could further merge two parse trees into a single tree. The learner then switches to the original learner and acquires new grammar rules as needed. Having acquired the grammar for deep features, when a new problem is given to the system, the learner will extract the deep feature by first building the parse tree of the problem based on the acquired grammar, and then extracting the subsequence associated with the feature symbol from the parse tree as the target feature. However, this model is only capable of learning and extracting deep features without using them to solve problems.

As we have mentioned above, SimStudent is able to acquire production rules in solving complicated problems, but requires a set of operators given as prior knowledge. Some of the operators are domain-specific, and require expert knowledge to build them. On the other hand, the feature learner acquires the deep features that are essential for effective learning, but is limited to information extraction tasks. In order to both reduce the amount of prior knowledge engineering needed for SimStudent and to extend the deep feature learner's capability, we integrated the deep feature learner into SimStudent.

**Extending Perceptual Learning.** Previously, the FoAs encoded in production rules are always associated with paths to elements in the GUI (such as cells in the algebra example). Intuitively, the deep features discussed above represent perceptual information–however, it is *domain-specific, learned* perceptual information. To exploit this information, we extend the perceptual hierarchy for the GUI to further include the most probable parse trees from the learned PCFG in the contents of the leaf nodes. We implement this by appending the parse trees to their associated leaf nodes, marking the appended nodes as type "subcell". In the algebra example, this extension means that cells representing algebraic expressions (e.g., those corresponding to left-hand sides or right-hand sides of the equation) are linked to parse trees for these expressions. Using *-3x* as an example, the extended hierarchy includes the parse tree for *-3x* as shown on the left side of Figure 2 as a subtree connected to the cell node associated with *-3x*. With this extension, the coefficient (*-3*) of *-3x* is now explicitly represented in the percept hierarchy. Hence if the extended SimStudent includes this subcell as a FoA in production rules, as shown at the right side of Figure 1, the production rule would no longer need the domain-specific engineered operator "coefficient".

However, extending the percept hierarchy presents challenges to the original "where learner". First of all, since the extended subcells are not associated with GUI elements, we can no longer depend on the tutor to specify FoAs for SimStudent. Nor can we simply put all of the subcells in the parse trees as FoAs: if we did, the acquired production rules would contain redundant information that might hurt the generalization capability of the "where learner". For example, for problem *-3x=6*, among all inserted subcells, only *-3* is a relevant FoA in solving the problem. Second, the paths to the relevant FoAs are typically more diverse: for example, for problems *-3x=6* and *4x=8*, the original where learner would not be able to find one set of generalized paths that explain both training examples, since *-3x* has eight nodes in its parse tree, while *4x* has only five. To address these challenges, we extend the original "where learner" to support acquisition of FoAs with redundant and non-fixed length FoA lists.

To do this, SimStudent first includes all of the inserted subcells as candidate FoAs, and calls the operator sequence learner to find a plan that explains all of the training examples. The "where learner" then removes all of the subcells that are not used by the operator sequence from the candidate FoA list. Since all of the training records share the same operator sequence, the number of FoAs remained for each record should be the same. Next, the "where learner" arranges the remained subcell FoAs based on their orderings of being used by the operator sequences. After this process, the "where learner" now has a set of FoA lists that contains fixed number of FoAs ordered in the same fashion. We can then switch to the original "where learner" to find the least general paths for the updated FoA lists. In our example for skill "divide", as shown at the right side of Figure 1, the FoAs of the production rule would contain three elements, the left-hand side and right-hand side cells which are the same as the original rule, and a coefficient subcell which corresponds to the left child of the variable term. Note that since we removed the redundant subcells, the acquired production rule now works with both *-3x=6* and *4x=8*.

**Extending Precondition Acquisition.** In addition to extending the feature learner, we also extend the vocabulary of feature symbols provided to the precondition learner. As implied by its name, the deep feature learner acquires information that reveal essential features of the problem state. It is natural to think that these deep features could also be used in describing desired situations to fire a production rule. Therefore, we construct a set of *grammar features* that are associated with the acquired PCFG. The set of new predicates describe positions of a subcell in the parse tree. For example, we create a new predicate called "is-left-child-of", which should be true for *(is-left-child-of -3 -3x)* based on the parse tree shown in the left side of Figure 2. Importantly, these new predicates are not domain-specific (although they are specific to the PCFG-based approach to deep feature learning). All of the grammar feature predicates are then included in the set of existing feature predicates for the precondition learner to use later.

## 3. EXPERIMENT STUDY

### 3.1 Method

In order to evaluate the effectiveness of the proposed approach, we carried out a study using an algebra dataset. We compared the SimStudent model with a human-generated KC model by first coding the real student steps using the two models, and then testing how well the two model codings fit with real student data.

For the human-generated model, the real student steps were first coded using the "action" label associated with a correct step transaction, where an action corresponds to a mathematical operation(s) to transform an equation into another. As a result, there were nine KCs defined (called the Action KC model) – add, subtract, multiply, divide, distribute, clt (combine like terms), mt (simplify multiplication), and rf (reduce a fraction). Four KCs associated with the basic arithmetic operations (i.e., add, subtract, multiply, and divide) were then further split into two KCs for each, namely a skill to identify an appropriate basic operator and a skill to actually execute the basic operator. The former is called a transformation skill whereas the latter is a typein skill. As a consequence, there were 12 KCs defined (called the Action-Typein KC model). Not all steps in the algebra dataset can be coded with these KC models – some steps are about a transformation that we do not include in the Action KC model (e.g., simplify division). There were 9487 steps that can be coded by both KC models mentioned above. The "default" KC model, which were defined by the productions implemented for the cognitive tutor, has only 6809 steps that can be coded. To make a fair comparison between the "default" and "Action- Typein" KC models, we took the intersection of those 9487 and 6809 steps. As a result, there were 6507 steps that can be coded by both the default and the Action-Typein KC models. We then defined a new KC model, called the Balanced-Action-Typein KC model that has the same set of KCs as the Action-Typein model but is only associated with these 6507 steps, and used this KC model to compare with the SimStudent model.

To generate the SimStudent model, SimStudent was tutored on how to solve linear equations by interacting with a Carnegie Learning Algebra I Tutor like a human student. We selected 40 problems that were used to teach real students as the training set for SimStudent. Given all of the acquired production rules, for each step a real student performed, we assigned the applicable production rule as the KC associated with that step. In cases where there was no applicable production rule, we coded the step using the human-generated KC model (Balanced-Action-Typein). Each time a student encounters a step using some KC is considered as an "opportunity" for that student to show mastery of that KC.

Having finished coding real student steps with both models (the SimStudent model and the human-generated model), we used the Additive Factor Model (AFM) [Cen et al. 2006] to validate the coded steps. AFM is an instance of logistic regression that models student success using each student, each KC, and the KC by opportunity interaction as independent variables,

$$\ln \frac{p_{ij}}{1 - p_{ij}} = \theta_i + \sum_k \beta_k Q_{kj} + \sum_k \beta_k Q_{kj} (\gamma_k N_{ik}) \tag{1}$$

Where:

$i$. represents a student i.

$j$. represents a step j.

$k$. represents a skill or KC k.

$p_{ij}$. is the probability that student i would be correct on step j.

$\theta_i$. is the coefficient for proficiency of student i.

$\beta_k$. is coefficient for difficulty of the skill or KC k

$Q_{kj}$. is the Q-matrix cell for step j using skill k.

$\gamma_k$. is the coefficient for the learning rate of skill k;

$N_{ik}$. is the number of practice opportunities student i has had on the skill k;

We utilized DataShop [Koedinger et al. 2010], a large repository that contains datasets from various educational domains as well as a set of associated visualization and analysis tools, to facilitate the process of evaluation, which includes generating learning curve visualization, AFM parameter estimation, and evaluation statistics including AIC (Akaike Information Criterion) and cross validation.

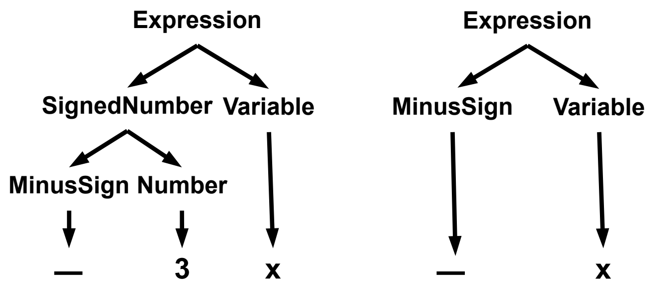


Fig. 3. Different parse trees for -3x and -x.

### 3.2 Dataset

We analyzed data from 71 students who used an Carnegie Learning Algebra I Tutor unit on equation solving. The students were typical students at a vocational-technical school in a rural/suburban area outside of Pittsburgh, PA. The problems varied in complexity, for example, from simpler problems like *3x=6* to harder problems like *x/-5+7=2*. A total of 19,683 transactions between the students and the Algebra Tutor were recorded, where each transaction represents an attempt or inquiry made by the student, and the feedback given by the tutor.

### 3.3 Measurements

To test whether the generated model fits with real student data, we used AIC and a 3-fold cross validation. AIC measures the fit to student data while penalizing over-fitting. We did not use BIC (Bayesian Information Criterion) as the fit metric, because based on past analysis across multiple DataShop datasets, it has been shown that AIC is a better predictor of cross validation than BIC is. The cross validation was performed over three folds with the constraint that each of the three training sets must have data points for each student and KC. We also report the root mean-squared error (RMSE) averaged over three test sets.

### 3.4 Experiment Result and Implications on Instructional Decision

The SimStudent model contains 21 KCs. Both the AIC (6448) and the cross validation RMSE (0.3997) are lower than the human-generated model (AIC 6529 and cross validation 0.4034). This indicates that the SimStudent model better fits with real student data without over-fitting.

In order to understand whether the differences are significant or not, we carried out two significance tests. The first significance test evaluates whether the SimStudent model is actually able to make better predictions than the human-generated model. During the cross validation process, each student step was used once as the test problem. We took the predicated error rates generated by the two KC models for each step during testing. Then, we compared the KC models' predictions with the real student error rate (0 if the student was correct at the first attempt, and 1 otherwise). After removing ties, among all 6494 student steps, the SimStudent model made a better prediction than the human-generated KC model in 4260 steps. A sign test on this shows that the SimStudent model is significantly ($p < 0.001$) better in predicting real student behavior than the human-generated model. In the second test, due to the random nature of the folding process in cross validation, we evaluated whether the lower RMSE achieved by the SimStudent model was consistent or by chance. To do this, we repeated the cross validation 20 times, and calculated the RMSE for both models. Across the 20 runs, the SimStudent model consistently outperformed the human-generated model. Thus, a paired t-test shows the SimStudent model is significantly ($p < 0.001$) better than the human-generated model. Also note that differences between competitors in the KDD Cup 2010 (https://pslcdatashop.web.cmu.edu/KDDCup/Leaderboard) have also been in this range of thousands

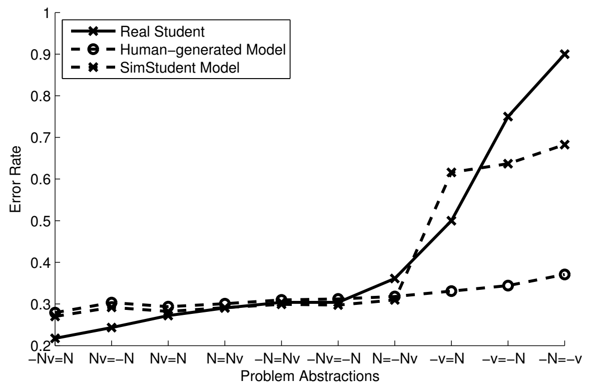


Fig. 4. Error rates for real students and predicted error rates from two student models.

in RMSE. Therefore, we conclude that the SimStudent model is a better student model than the human-generated KC model.

We can inspect the data more closely to get a better qualitative understanding of why the SimStudent model is better and what implications there might be for improved instruction. Among the 21 KCs learned by the SimStudent model, there were 17 transformation KCs and four typein KCs. It is hard to map the SimStudent KC model directly to the expert model. Approximately speaking, the distribute, clt, mt, rf KCs as well as the four typein KCs are similar to the KCs defined in the expert model. The transformation skills associated with the basic arithmetic operators (i.e. add, subtract, multiply and divide) are further split into finer grain sizes based on different problem forms.

One example of such split is that SimStudent created two KCs for division. The first KC (simSt-divide) corresponds to problems of the form $Ax{=}B$, where both $A$ and $B$ are signed numbers, whereas the second KC (simSt-divide-1) is specifically associated with problems of the form $-x{=}A$, where A is a signed number. This is caused by the different parse trees for $Ax$ vs $-x$ as shown in Figure 3. To solve $Ax{=}B$, SimStudent simply needs to divide both sides with the signed number $A$. On the other hand, since $-x$ does not have $-1$ represented explicitly in the parse tree, SimStudent needs to see $-x$ as $-1x$, and then to extract $-1$ as the coefficient. If SimStudent is a good model of human learning, we expect the same to be true for human students. That is, real students should have greater difficulty in making the correct move on steps like $-x = 6$ than on steps like $-3x = 6$ because of the need to convert (perhaps just mentally) $-x$ to $-1x$. To evaluate this hypothesis, we computed the average error rates for a relevant set of problem types – these are shown with the solid line in Figure 4 with the problem types defined in forms like $-Nv{=}N$, where the $N$s are any integrate number and the $v$ is a variable (e.g., $-3x{=}6$ is an instance of $-Nv{=}N$ and $-6{=}-x$ is an instance of $-N{=}-v$). We also calculated the mean of the predicted error rates for each problem type for both the human-generated model and the SimStudent model. Consistent with the hypothesis, as shown in Figure 4, we see that problems of the form $Ax{=}B$ (average error rate 0.283) are much simpler than problems of the form $-x{=}A$ (average error rate 0.719). The human-generated model predicts all problem types with similar error rates (average predicted error rate for $Ax{=}B$ 0.302, average predicted error rate for $-x{=}A$ 0.334), and thus fails to capture the difficulty difference between the two problem types ($Ax{=}B$ and $-x{=}A$). The SimStudent model, on the other hand, fits with the real student error rates much better. It predicts higher error rates (0.633 on average) for problems of the form $-x{=}A$ than problems of the form $Ax{=}B$ (0.291 on average).

SimStudent's split of the original division KC into two KCs, simSt-divide and simSt-divide-1, suggests that the tutor should teach real students to solve two types of division problems separately. In other words, when tutoring students with division problems, we should include two subsets of problems, one subset corresponding to simSt-divide problems ($Ax{=}B$), and one specifically for simSt-divide-1 problems ($-x{=}A$). We should perhaps also include explicit instruction that highlights for students that $-x$ is the same as $-1x$.

## 4. RELATED WORK

The objective of this paper is using a machine learning agent, SimStudent, to automatically construct student models. There has been considerable work on comparing the quality of alternative cognitive models. LFA automatically discovers student models, but is limited to the space of the human-provided factors. Other works such as [Pavlik et al. 2009; Villano 1992] are less dependent on human labeling, but may suffer from challenges in interpreting the results. In contrast, the SimStudent approach has the benefit that the acquired production rules have a precise and usually straightforward interpretation. Baffes and Mooney [1996] applies theory refinement to the problem of modeling incorrect student behavior. Other systems [Tatsuoka 1983; Barnes 2005] use Q-matrix to find knowledge structure from student response data. None of the above approaches use simulated students to construct cognitive models.

Other research on creating simulated students [Vanlehn et al. 1994; Chan and Chou 1997; Pentti Hietala 1998] also share some resemblance to our work. VanLehn [1990] created a learning system and evaluated whether it was able to learn procedural "bugs" like real students. Biswas et al. [2005]'s system learns causal relations from a conceptual map created by students. None of the above approaches compared the system with learning curve data. To the best of our knowledge, our work is the first combination of the two whereby we use cognitive model evaluation techniques to assess the quality of a simulated learner.

## 5. CONCLUSION AND FUTURE WORK

In this paper, we introduced an innovative application of a machine-learning agent, SimStudent, for an automatic discovery of student models. An empirical study showed that a SimStudent generated student model was a better predictor of real students learning performance than a human-coded student model. The basic idea is to have SimStudent learn to solve the same problems that human students did and use the productions that SimStudent generated as knowledge components to codify problem-solving steps. We then used these KC coded steps to validate the models prediction. Unlike the human-engineered student model, the SimStudent generated student model has a clear connection between the features of the domain contents and knowledge components. An advantage of the SimStudent approach of student modeling over previous techniques like LFA is that it does not depend heavily on the human-engineered features. SimStudent can automatically discover a need to split a purported KC or skill into more than one skill. During SimStudents learning, a failure of generalization for a particular KC results in learning disjunctive rules. Discovering such disjuncts is equivalent to splitting a KC in LFA, however, whereas human needs to provide potential factors to LFA as the basis for a possible split, SimStudent can learn such factors. The use of the perceptual learning component, implemented using a probabilistic context-free grammar learner, is a key feature of SimStudent for these purposes as we hypothesized that a major part of human expertise, even in academic domains like algebra, is such perceptual learning.

Our evaluation demonstrated that representing the rules SimStudent learns in the student model improves the accuracy of model prediction, and showed how the SimStudent model could provide important instructional implications. Much of human expertise is only tacitly known. For instance, we know the grammar of our first language but do not know what we know. Similarly, most algebra experts have no explicit awareness of subtle transformations they have acquired like the one above (seeing *-x* as *-1x*). Even though such instructional designers may be experts in a domain they have thus have some blind spots regarding subtle perceptual differences like this one, which may make a real difference for novice learners. A machine learning agent, like SimStudent, can help get past such blind spots by revealing challenges in the learning process that experts may not be aware of.

The current study used a single dataset in a single domain. The generality and validity of the proposed student-modeling technique could be extended by training SimStudent with one dataset and applying a discovered KC model to another dataset. For instance, the experiment dataset was from one high school. An interesting future study would be to examine data from other schools or grade levels, and evaluate the generality of the proposal technique. We should also apply this approach in other domains such as stoichiometry, fraction addition and so on. The Pittsburgh of Science of Learning Centers DataShop contains over 200 datasets in algebra and other domains that could be used for such cross-dataset or cross-domain validation.

## 6. ACKNOWLEDGEMENTS

The research reported here was supported by National Science Foundation Award No. DRL-0910176 and the Institute of Education Sciences, U.S. Department of Education, through Grant R305A090519 to Carnegie Mellon University. The opinions expressed are those of the authors and do not represent views of the Institute or the U.S. Department of Education. This work is also supported in part by the Pittsburgh Science of Learning Center, which is funded by the National Science Foundation Award No. SBE-0836012.

## REFERENCES

Baffes, P. T. and Mooney, R. J. 1996. A novel application of theory refinement to student modeling. In *Proceedings of the thirteenth national conference on Artificial intelligence*. AAAI Press, 403–408.

Barnes, T. 2005. The Q-matrix method: Mining student response data for knowledge. In *Proceedings AAAI Workshop Educational Data Mining*. Pittsburgh, PA, 1–8.

Biswas, G., Schwartz, D., Leelawong, K., and Vye, N. 2005. Learning by teaching: A new agent paradigm for educational software. *Applied Artificial Intelligence 19*, 363–392.

Cen, H., Koedinger, K., and Junker, B. 2006. Learning factors analysis - a general method for cognitive model evaluation and improvement. In *Proceedings of the 8th International Conference on Intelligent Tutoring Systems*. 164–175.

Chan, T.-W. and Chou, C.-Y. 1997. Exploring the design of computer supports for reciprocal tutoring. *International Journal of Artificial Intelligence in Education 8*, 1–29.

Chi, M. T. H., Feltovich, P. J., and Glaser, R. 1981. Categorization and representation of physics problems by experts and novices. *Cognitive Science 5,* 2, 121–152.

Koedinger, K. R., Anderson, J. R., Hadley, W. H., and Mark, M. A. 1995. Intelligent Tutoring Goes to School in the Big City. In *Proceedings of the 7th International Conference on Artificial Intelligence in education.*

Koedinger, K. R., Baker, R. S., Cunningham, K., Skogsholm, A., Leber, B., and Stamper, J. 2010. A data repository for the EDM community: The PSLC DataShop.

Koedinger, K. R. and McLaughlin, E. A. 2010. Seeing language learning inside the math: Cognitive analysis yields transfer. In *Proceedings of the 32nd Annual Conference of the Cognitive Science Society*. Austin, TX, 471–476.

Koedinger, K. R. and Nathan, M. J. 2004. The real story behind story problems: Effects of representations on quantitative reasoning. *The Journal of Learning Sciences 13,* 2, 129–164.

Lau, T. and Weld, D. S. 1998. Programming by demonstration: An inductive learning formulation. In *Proceedings of the 1999 International Conference on Intelligence User Interfaces*. 145–152.

Li, N., Cohen, W. W., and Koedinger, K. R. 2010. A computational model of accelerated future learning through feature recognition. In *Proceedings of 10th International Conference on Intelligent Tutoring Systems*. 368–370.

Li, N., Kambhampati, S., and Yoon, S. 2009. Learning probabilistic hierarchical task networks to capture user preferences. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*. Pasadena, CA.

Matsuda, N., Lee, A., Cohen, W. W., and Koedinger, K. R. 2009. A computational model of how learner errors arise from weak prior knowledge. In *Proceedings of Conference of the Cognitive Science Society.*

Muggleton, S. and de Raedt, L. 1994. Inductive logic programming: Theory and methods. *Journal of Logic Programming 19*, 629–679.

Pavlik, P. I., Cen, H., and Koedinger, K. R. 2009. Learning Factors Transfer Analysis: Using Learning Curve Analysis to Automatically Generate Domain Models. In *Proceedings of 2nd International Conference on Educational Data Mining.* 121–130.

Pentti Hietala, T. N. 1998. The competence of learning companion agents. *International Journal of Artificial Intelligence in Education 9*, 178–192.

Quinlan, J. R. 1990. Learning logical definitions from relations. *Machine Learning 5,* 3, 239–266.

Tatsuoka, K. K. 1983. Rule space: An approach for dealing with misconceptions based on item response theory. *Journal of Educational Measurement*, 345–354.

VanLehn, K. 1990. *Mind Bugs: The Origins of Procedural Misconceptions.* MIT Press, Cambridge, MA, USA.

Vanlehn, K., Ohlsson, S., and Nason, R. 1994. Applications of simulated students: an exploration. *Journal of Artificial Intelligence in Education 5*, 135–175.

Villano, M. 1992. Probabilistic student models: Bayesian belief networks and knowledge space theory. In *Proceedings of the 2nd International Conference on Intelligent Tutoring Systems*. Heidelberg, 491–498.