Matsuda, N., Cohen, W. W., & Koedinger, K. R. (2007, to appear). Predicting Students performance with SimStudent that learns cognitive skills from observation. In Proceedings of the international conference on Artificial Intelligence in Education. Draft – Do not quote

Predicting Students Performance with SimStudent that Learns Cognitive Skills from Observation

Noboru Matsuda, William W. Cohen, Gustavo Lacerda, Kenneth R. Koedinger

School of Computer Science

Carnegie Mellon University

Abstract: SimStudent is a machine-learning agent that learns cognitive skills by demonstration. SimStudent was originally built as a building block for Cognitive Tutor Authoring Tools to help an author build a cognitive model without heavy programming. In this paper, we evaluate a second use of SimStudent for student modeling for Intelligent Tutoring Systems. The basic idea is to have SimStudent observe human students solving problems with an intelligent tutoring system. It then induces a cognitive model that replicates their performance on these problems. If the model is accurate, it predicts the human students' performance on novel problems. An evaluation study with log data from genuine student-tutor interactions showed that when trained on 15 problems, SimStudent accurately predicted the human students' correct behavior more than 80% of the time (i.e., it can tell when the human students make correct steps). However, the current implementation of SimStudent does not accurately predict when the human students make errors. This paper gives a detailed description of the evaluation study and discusses the lessons learned.

1 Introduction

SimStudent was originally built as an intelligent building block for the Cognitive Tutor Authoring Tools, or CTAT for short [1]. In this authoring environment, an author can build a cognitive model that represents domain principles (i.e., context-based procedures) for a target task without heavy programming. The author can simply *demonstrate* the task (both correctly and incorrectly) using a tutor interface built with CTAT. SimStudent then learns how to perform the target task (or how to make a mistake) by generalizing the author's demonstration. The fundamental technology used in SimStudent is so-called *programming by demonstration* [2].

Since SimStudent can generalize actions taken in the tutor interface, theoretically speaking, SimStudent should be able to model domain principles that a human student acquired while solving problems with a Cognitive Tutor. Thus, the ultimate goal of the current project is to evaluate whether SimStudent can be used for student modeling. A preliminary study showed that SimStudent can model human students' *correct* behaviors, whereas modeling students' erroneous behaviors is quite challenging, partly because the human students make slips and random errors (i.e., non-systematic errors, which can not be observed consistently). Although building a descriptive model of student misconceptions (telling why a particular behavior was observed) might be challenging, building a predictive model of student performance (telling whether a student would perform a step correctly or not) might be tractable. Such a predictive model might give us further insight into designing a descriptive model of the student's knowledge, which represents the student's misconceptions at a fine level of granularity. Given all that is said, as an initial milestone towards our ultimate goal, the current paper addresses two research questions: (1) Can SimStudent learn domain principles by observing an individual student's performance? (2) Can SimStudent *predict* an individual student's behavior on novel problems given domain principles learned from the individual student?

2 Related Studies

There have been a wide variety of machine-learning techniques studied so far for student modeling, including synthetic, analytic, and stochastic methods. Ikeda et al [3] built an inductive logic learner

in conjunction with a truth maintenance system to inductively construct a hypothetical student model. Baffes et al [4] applied a machine learning technique to modify a given set of domain principles to be consistent with the student's incorrect behaviors. Similarly, Langley et al [5] applied a theory-refinement technique to model an individual student's behavior given a general model of the domain. MacLaren et al [6] applied a reinforcement learning method based on the ACT-R model [7] to model the process in which the students' knowledge activation patterns are strengthened. There are more studies on applications of machine-learning for student modeling.

The current study is about an application of a cutting-edge technology for student modeling. The proposed system is unique in two ways: (1) the fundamental framework used in SimStudent, inductive logic programming [8], is domain-general, and hence it is applicable to a wide range of domains, (2) SimStudent has a dual role in this context – to help the authoring of a cognitive model for the purposes of building a Cognitive Tutor, and to model a student's performance when he/she uses the Cognitive Tutor. The former cognitive model enables the Cognitive Tutor to perform model-tracing (see section 3.2), whereas the latter model allows the Cognitive Tutor, say, to proactively select a problem on which the student is likely to make a mistake.

3 Experimental Setting: Algebra I Cognitive Tutor

The Algebra I Tutor is a Cognitive Tutor developed by Carnegie Learning Inc. This tutor is used in real classroom situations for high school algebra in about 2000 schools nationwide in the United States [9].

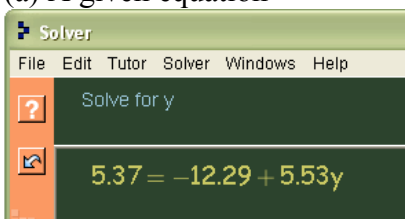
For the current study, we use tutor-interaction logs representing interactions between the Cognitive Tutor and individual human students. The current data were collected from a study conducted in a high school in an urban area of Pittsburgh. There were 81 students involved in the study. The students used the Cognitive Tutor individually to learn algebra equation solving. There were 15 sections taught by the tutor, which covered the most of the skills necessary to solve linear equations. In the current study, we focus on the log data for the first eight sections. The equations in those sections only contain a single unknown and have the form A+B=C+D where A, B, C, and D are either a constant or an unknown term in the form of N*x* or *x*/N, where N is a constant number.

3.1 Tutoring Interaction

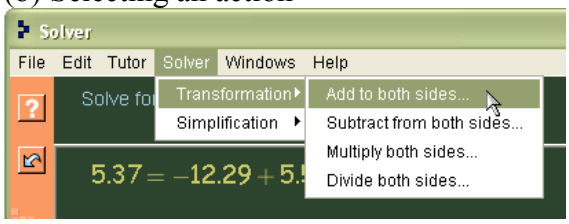
The unit of analysis in the current paper is a *problem-solving step* performed by the human students using the Cognitive Tutor interface. Here, a problem-solving step is slightly different from an *equation-transformation step*, which corresponds to a single line when solving an equation with a paper and a pencil.

There are two types of problem-solving steps: (1) an *action step* is to select an algebraic operation to transform an equation into another (e.g., "to declare to add 3*x* to the both sides of the equation"), and (2) a *type-in step* is to do a real arithmetic calculation (e.g., "to enter –4 as a result of adding 3*x* to –4–3*x*"). Performing these problem-solving steps, a given equation is transformed as follows: a student first selects an action and then applies it to both sides of the equation. For example, for the equation shown in Figure 1 (a), the student selected "Add to both sides" from the pull down menu (b), which in turn prompts the student to specify a value to add (c). This completes the first problem-solving step, which is an action step. The student then enters the left- and right-hand sides separately. Figure 1 (d) shows a moment where the student had just typed-in the left-hand side. Thus, entering a new equation is completed in two problem-solving steps, which are both type-in steps. In summary, three problem-solving steps correspond to a single equation-transformation step, which transforms an equation into another. Sometimes, however, the tutor carries out the type-in steps for the student, especially when new skills are introduced.

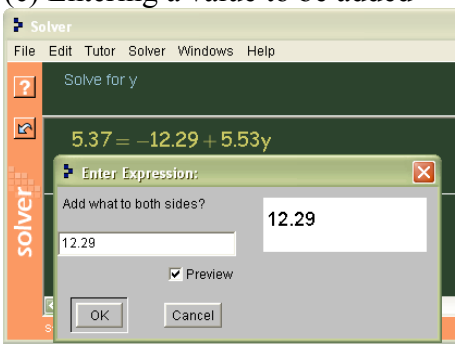
(a) A given equation



(b) Selecting an action



(c) Entering a value to be added



(d) Typing-in a left-hand side

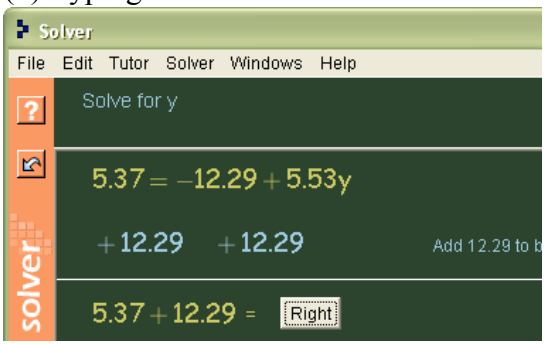


Figure 1: Screen shot for the Algebra I Tutor.

When a student makes an error, the tutor provides feedback. The student can also ask for a hint (by pressing the "[?]" button on the left side of the tutor window) when he/she gets stuck.
Every time a student performs a step, the tutor logs it. The log contains, among other things, (1) the equation in which the step was made, (2) the action taken (either the name of the algebraic operation selected from the menu for an action step, or the symbol "type-in" for a type-in step), (3) the value entered (e.g., the value specified to be added to both sides for the "add" action mentioned above, or the left- and right-hand side entered for the type-in steps), and (4) the "correctness" of the step, which is either "correct" (in case the student's steps is correct), "error" (the student's steps is incorrect), or "hint" (when the student asked for a hint).

3.2 Cognitive model and model tracing

As mentioned above, when a human student performs a problem-solving step, the tutor provides immediate feedback on the correctness of the step. This is possible because the tutor has a *cognitive model*, represented as a set of production rules that is sufficient to perform the target task. A cognitive model usually contains production rules not only for correct steps, but also for incorrect steps, so that the tutor can provide feedback on typical errors as well. When a step is performed, the tutor attempts to find a production rule that matches it. If the production rule is a correct one, then the tutor lets the student proceed to the next step. If the production rule is an incorrect one, then the tutor provides an error feedback message associated with the rule. This technique is called *model tracing*.

4 The Architecture of SimStudent

This section is a brief overview of SimStudent. We first explain how SimStudent learns cognitive skills from demonstrations. The double meaning of "demonstration" in the current context is then explained – a demonstration by an author who is building a Cognitive Tutor, and a "demonstration" in the tutor-interaction log representing real students' performance. Due to the space limitation, we only provide a brief overview. See Matsuda et al [10] for more details.

4.1 Modeling tutor-interaction for Algebra I Tutor

SimStudent learns a single production rule for each of the problem-solving steps. When demonstrating a problem-solving step, the demonstrator must specify two things; (1) a skill name, and (2) a focus of attention.

The skill name must be unique for unique steps and consistent throughout the demonstration. In the Algebra I Tutor, the skill name for an action step is an abbreviation of the action name shown in the tutor interface. For example, in the step shown in Figure 1 (a-c), the skill name to select "Add 12.29 to both sides" is called "add." The skill name for a type-in step consists of the skill name of the corresponding action step and "-typein." So, for example, the skill name for the step shown in Figure 1 (d) is called "add-typein."

The focus of attention is the set of elements on the student interface from which the value to be entered is determined. For example, the problem-solving step shown in Figure 1 ("add 12.29 to both sides") requires two elements, "5.37" and "-12.29+5.53y" as the focus of attention. When SimStudent is used for authoring, the author is asked to specify the focus of attention for each problem-solving step. There is, however, an issue getting the focus of attention when SimStudent is used to model real students' performance – the real students do not indicate their focus of attention. We have assumed that both the left-hand side and right-hand side are used as the focus of attention for the action steps. For the type-in steps, we presume that the skill name and the expression immediately above the expression to be typed-in are the focus of attention. So, for example, for the type-in step shown in Figure 1 (d), where "5.37+12.29" was typed-in, the skill name "Add 12.29 to both sides" and the expression "5.37" are used as the focus of attention.

4.2 Learning algorithm

Production rules are represented in the Jess production rule description language [11]. A production rule used in the Cognitive Tutors consists of three major parts: what, when, and how. The "what" part specifies which elements of the interface are involved in the production rule. The "when" part shows the conditions that should hold among the WME-paths for a production rule to be fired. The "how" part specifies what should be done with the interface elements ("what" part) to make a correct step to be performed perform.

SimStudent utilizes three different learning algorithms to learn these three parts separately. The "what" part is learned as a straightforward generalization of the focus of attention. The elements specified in the focus of attention are elements on the tutor interface. They can thus be uniquely identified in terms of their "location" in the interface. Constraints on the location are generalized from the most specific description using absolute positions (e.g., the 2nd and the 3rd cells) to the most general description saying "any element." A moderate generalization uses relative locations (e.g., two consecutive cells).

SimStudent use FOIL [12] to learn feature conditions. The target concept is an "applicability" of a particular skill given a focus of. When a step for a particular skill N is demonstrated, the step serves as a positive example for the skill N, and a negative example for all other skills. We provide FOIL with a set of feature predicates as the background knowledge to compose hypotheses for the target concept. Some examples of such feature predicates are isPolynomial(A), isNumeratorOf(A,B), isConstant(A). Once a hypothesis is found for the target concept, the body of the hypothesis becomes the feature condition on the left-hand side of the production rule. Suppose, for example, that FOIL found a hypothesis N(FN1, FN2) :- isPolynomial(FN1), isConstant(FN2). The left-hand feature condition for this production rule says that "the value of the first focus of attention must be polynomial and the second value must be a constant."

SimStudent uses the iterative-deepening depth-first search to learn an operator sequence for the right-hand side of the production rules. When a new instance of a particular skill is demonstrated, SimStudent searches for a shortest operator sequence that derives the demonstrated action from the

focus of attention for the all instances demonstrated thus far. Those operators are provides prior to learning as background knowledge.

5 Modeling Real Students

How well does SimStudent predict real students' incorrect steps? How often does SimStudent make incorrect prediction both in labeling a correct step as incorrect and incorrect step as correct? To answer these questions, an analysis of cost in prediction was conducted.

We are not only interested in observing SimStudent predicting real students correct steps, but predicting incorrect steps is equally (or even more) valuable for educational purposes.

5.1 Data: Tutor-interaction log

The students' learning log was converted into *problem files*. Each problem file contains the sequence of problem-solving steps made by a single real student for a single equation problem. There are three types of steps: *correct*, *buggy*, and *error*. The correct steps are those that were model-traced with a correct production rule by the Carnegie Learning Tutor. The buggy steps are those that were model-traced with a buggy production rule. The error steps were not model-traced either with a correct or a buggy production rule.

There were 1897 problems solved by 81 individual human students. There were total of 32709 problem-solving steps. These steps contain 21794 (66.7%) correct steps, 2336 (7.1%) buggy steps, 4567 (14.0%) error steps, and 4012 (12.3%) hint seeking steps[1]. Since the Algebra I Tutor created by Carnegie Learning has wide variety of buggy rules designed based on empirical studies, it is probably enough to model real students only based on correct and buggy steps. Also, we have found (by manually verifying data) that most of the error steps are likely to be due to slips and/or random errors in using interface, which makes the error steps most challenging to be modeled. Thus, we have only used correct and buggy steps for the current evaluation.

5.2 Method: Validation

We have applied a validation test to see how well SimStudent can predict if a real student's performance on a particular problem-solving step is correct at the first attempt.

For each of the real students, we have taken the first 15 problems for training and following five problems for testing. Using these problems, the validation was conducted for *each individual human student* separately as follows:

```
For each of the 15 training problems
    Train SimStudent on the correct and buggy steps performed by the real student
    For each of the five test problems
        Attempt to model-trace the correct steps performed by the student
```

Notice that only the steps correctly performed by a real student were model-traced for a test. This is because we are evaluating whether SimStudent could correctly *predict* if a real student succeed on a step or not. A student's performance on a step is coded as "success" if the *first attempt* at the step is correct, otherwise "error." The Cognitive Tutor forced real students to perform a step correctly to proceed to the next step; therefore, a chain of correct steps in a log represent an entire solution steps for the problem. Hence, for our purpose, it is sufficient to have SimStudent model-trace only those solution steps – if an attempt at model-tracing failed, it is a prediction that the real student would also fail to perform the step correctly at the first attempt.

[1] Asking a hint is logged as a problem-solving step. As mentioned in section 5.3.2, when a real student asked a hint on a step, the student's attempt was coded to be unsuccessful.

5.3 Result: Analysis of cost in prediction

62 students solved at least 20 problems. Three students were excluded due to apparent errors in the logging method. In average, there were 116.8 steps (7.8 steps per problem) for the 15 training problems, and 55.6 steps (11.1 steps per problem) for the five test problems.

There were 10 different skills in the log data. One of them had appeared in the training problems very rarely (27 times total across 59 students), and hence was excluded from the analysis. The remaining nine skills included five skills for the action steps (add, subtract, multiply, divide, and clt, which is to "combine like terms") and four skills for the type-in steps (add-typein, subtract-typein, multiply-typein, and divide-typein).

We first show an overall performance of model tracing and then discuss an analysis of cost in fitness of the model.

5.3.1 Learning curve – how well SimStudent learned cognitive skills?

Figure 2 shows the average ratio for the coverage of production-rule firing against the steps in the test problems aggregated across all skills and students. The x-axis shows a number of times that a particular skill is learned. In other words, the figure shows average ratio of the steps in the test problems that were model-traced correctly by a production rule learned after trained for a certain amount of times.

The average ratio of successful-model tracing increased as the opportunities to learn a particular skill increased. After trained for 16 times, more than 80% of the steps in test problems were model-traced with a production rule learned by SimStudent.

This result shows that SimStudent did actually learn cognitive skills from the tutor-interaction log performed by the real students. Below, we will explore two further issues: How many correct steps in the test problems were predicted as correct? Did SimStudent learn overly general rules hence they had a tendency to model-trace steps?

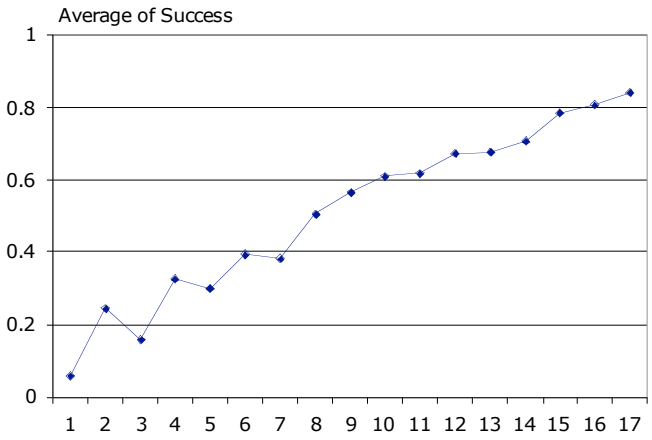


Figure 2: Learning curve showing average ratio of successful model-tracing; aggregated across all skills and all students. The x-axis shows a frequency of learning on a skill

5.3.2 Analysis of type of errors in predicting real students performance

The primary purpose of this study is to *predict* real students' performance hence SimStudent should only model-trace steps that the real students correctly performed and fail to model-trace steps that the real students failed. Thus, a better analysis is to count the number of matches between the real student's and SimStudent's performance on the training problems.

We define a result of a prediction as follows. Given a step performed by a real student and a result of model-tracing. A result of model-tracing is a *success* if there is a production rule fired reproduced the step; and an *error* otherwise. The real student's step is a *success* if he/she performed the step correctly at the first attempt, and an *error* otherwise[2]. Finally, we define a prediction to be (1) *true positive* (TP), if both the attempt of model-tracing and the student's performance were a success, (2) *false positive* (FP), if the attempt of model-tracing was a success while the student's

[2] This includes cases where the student requested a "hint" on the step.

performance was an error, (3) *false negative* (FN), if the attempt of model-tracing was an error, but the student's performance was a success, and (4) *true negative* (TN), if both the attempt of model tracing and the student's performance were an error.

We define following measures: (1) *Accuracy* = (TP+TN)/(TP+FN+FP+TN). (2) *Precision* = TP/(TP+FP). (3) *Recall* = TP/(TP+FN). (4) *E[rror]-Precision* = TN/(TN+FN). (5) *E[rror]-Recall* = TN/(TN/FP). In this study, we are particularly interested in E-Precision and E-Recall, because SimStudent ought to model not only the real student's successful performance, but also errors.

Figure 3 shows the result of the cost analysis in predicting real students' performance. The x-axis shows a frequency of learning on a skill. The y-axis shows an average for each measure aggregated across all students and skills.

As conjectured from the result of the previous section, the recall increased as SimStudent was trained on more steps – showing the ability for SimStudent to learn rules that model-trace steps that the real students succeeded.

That precision stayed high regardless of the number of training is rather surprising. It turned out, however, that the real students in the current particular log data made only a very few error steps – only 15% of the steps in the test problems were error steps and the ratio of error steps was stable across the different frequencies[3]. If SimStudent learned overly general rules, then the chance of successful model-tracing attempt increased with the relatively small number of false positives.

Finally, and most importantly, the E[rror]-precision increased slightly, but overall it stayed low. This means that SimStudent did not accurately predict real students' error steps. To our surprise, the E[rror]-recall decreased as learning progressed. This indicates that as learning progressed, SimStudent tended to learn more production rules that model-traced more steps performed incorrectly by human students. In other words, the current implementation of SimStudent is not correctly predicting human students' performance.

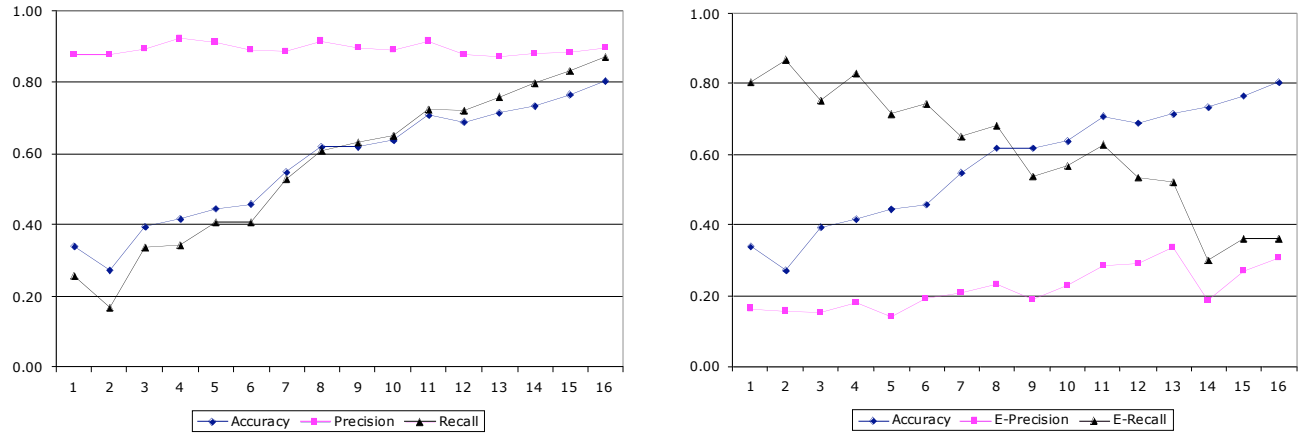


Figure 3: A cost analysis in prediction of the real student's performance. The X-axis shows a frequency of learning. The Y-axis shows an average of each measurement aggregated across all students and all skills.

5.3.3 Analysis of error of commission

On of the key concerns in the previous analysis is whether (or how often) SimStudent learned overly general rules. To address this issue, we need to "model-trace" SimStudent's rule firing. In other words, for each of the steps in the test problems, we ask SimStudent to show next steps that can be achieved, and assess a correctness for each of these steps assuming that we have an oracle for the judgment. Such evaluation is quite costly, and we have just not yet implemented the idea of using oracle for the analysis of error of commission. Instead, we have manually gauged the correctness of the rule firing by inspecting a conflict set of the production rules each time a step is to be model-traced. We have randomly selected three real students from the log data. There were

[3] This does not mean that the real students did not learn. Indeed, the *number* of error steps decreased, but the ratio of error steps to the correct steps stayed same – there was always about 15% of chance that the real students made an error step in this particular data set.

total of 74 steps in the test problems where 6 skills (add, divide, subtract, add-typein, divide-typein, and subtract-typein) were tested.

	Num of steps	Rule firing	
		True Firing	False Firing
add	11	90	136
add-typein	10	14	54
subtract	16	48	173
subtract-typein	8	12	21
divide	19	19	89
divide-typein	10	15	11
Total	74	198	484

Table 1: Total number of rules in the conflict set for each of the skills. The second column shows the frequency of the skill being model-traced in the test problems. The *False Firing* shows the number of incorrect rule application – i.e., if applied, the production rule generated an incorrect step.

Table 1 shows the total number of rules in the conflict set for each skill. The second column shows the frequency of the skill being tested. The *True Firing* shows the number of production rules that could be fired to generate a correct step. The *False firing* shows the number of production rules that, if applied, generated an incorrect step. There were surprisingly many false firing reported, meaning that SimStudent learned substantial number of overly general production rules. This observation agrees with the high precision mentioned above. It also explains why the E[rror]-recall rapidly decreased as the learning progressed – SimStudent quickly learned overly general rules, which resulted in covering more steps that the real students failed to perform correctly.

6 Conclusion

Using a genuine tutor-interaction log as "demonstrations" of real students performing their skills, SimStudent did learn to generate a student model that can predict more than 80% of the *correct* steps performed by real students.

However, it turned out that SimStudent does not accurately predict real students' *incorrect* steps. Explanations are produced by the student model that is overly general – SimStudent explains steps that were not performed correctly by real students (low E[rror]-precision). Also, SimStudent has rules in the conflict set that could explain behaviors different from the ones that were actually observed in real students.

Can we use SimStudent as a pedagogical component for an intelligent tutoring system? Currently, the answer leans toward the negative. We are still exploring the issues. One problem may be an incorrect model of students' prior skills. The solution may require different learning methods; the current SimStudent is designed for fast construction of cognitive models using programming by demonstration, not for student modeling.

7 References

1. Koedinger, K.R., V.A.W.M.M. Aleven, and N. Heffernan, *Toward a Rapid Development Environment for Cognitive Tutors*, in *Proceedings of the International Conference on Artificial Intelligence in Education*, U. Hoppe, F. Verdejo, and J. Kay, Editors. 2003, IOS Press: Amsterdam. p. 455-457.
2. Cypher, A., ed. *Watch what I do: Programming by Demonstration*. 1993, MIT Press: Cambridge, MA.
3. Ikeda, M. and R. Mizoguchi, *FITS: a framework for ITS - a computational model of tutoring.* International Journal of Artificial Intelligence in Education, 1994. **5**: p. 319-348.
4. Baffes, P. and R. Mooney, *Refinement-Based Student Modeling and Automated Bug Library Construction.* Journal of Artificial Intelligence in Education, 1996. **7**(1): p. 75-116.
5. Langley, P. and S. Ohlsson, *Automated cognitive modeling*, in *Proceedings of the Fourth National Conference on Artificial Intelligence*. 1984, AAAI: Melon Park, CA. p. 193-197.
6. MacLaren, B. and K.R. Koedinger, *When and why does mastery learning work: Instructional experiments with ACT-R "SimStudents".* in *Proceedings of the 6th International Conference on Intelligent Tutoring Systems*, S.A. Cerri, G. Gouarderes, and F. Paraguacu, Editors. 2002, Springer-Verlag: Berlin. p. 355-366.
7. Anderson, J.R., *Rules of the mind*. 1993, Hillsdale, NJ: Erlbaum.

8. Muggleton, S. and C. Feng, *Efficient Induction Of Logic Programs*, in *Inductive Logic Programming*, S. Muggleton, Editor. 1992, Academic Press: London, UK. p. 281-298.
9. Koedinger, K.R. and A. Corbett, *Cognitive Tutors: Technology Bringing Learning Sciences to the Classroom*, in *The Cambridge Handbook of the Learning Sciences*, R.K. Sawyer, Editor. 2006, Cambridge University Press: New York, NY. p. 61-78.
10. Matsuda, N., W.W. Cohen, and K.R. Koedinger, *Applying Programming by Demonstration in an Intelligent Authoring Tool for Cognitive Tutors*, in *AAAI Workshop on Human Comprehensible Machine Learning (Technical Report WS-05-04)*. 2005, AAAI association: Menlo Park, CA. p. 1-8.
11. Friedman-Hill, E., *Jess in Action: Java Rule-based Systems*. 2003, Greenwich, CT: Manning.
12. Quinlan, J.R., *Learning Logical Definitions from Relations.* Machine Learning, 1990. **5**(3): p. 239-266.