SimStudent: Improving Tutor Quality and Reducing Authoring Costs

Christopher J. MacLellan, Eliane Stampfer Wiese,
Noboru Matsuda, Kenneth R. Koedinger

Human-Computer Interaction Institute
Carnegie Mellon University, Pittsburgh, PA 15213, USA
cmaclell@cs.cmu.edu, stampfer@cs.cmu.edu,
noboru.matsuda@cs.cmu.edu, kodedinger@cmu.edu

Abstract. Intelligent Tutoring Systems are effective at improving learning, but are costly to produce. In this paper, we review SimStudent, a system designed to aid authors in improving the quality of their tutors while simultaneously reducing authoring costs. This system works by inducing general production rule models from user demonstrations and feedback. We discuss results demonstrating that SimStudent learned models better fit student data than expert authored models and that it is efficient to use. In addition, we provide an example of how a simple algebra tutor would be constructed using this paradigm. Lastly, we review future directions for this work.

1 Introduction

Intelligent tutoring systems are effective at improving learning [1–4], but despite widespread massive use of math Cognitive Tutors (more than 500k students per year complete a Carnegie Learning tutor course), they have not been widely adopted more generally (e.g., in online education platforms such as Khan Academy, Coursera, etc.). Perhaps intelligent tutors have not been adopted because their learning benefits are not thought to outweigh the costs of their development. This problem is particularly pronounced for massive online education platforms, which have a large quantities of content that vary widely across domains (Khan academy has about 500 hours of videos spread over 40 units in Math, Science, Economics, and Humanities). Our work aims to increase the value of intelligent tutoring systems by improving both sides of the cost-benefit equation - building higher quality tutors that lead to more robust learning while also decreasing authoring time.

Previous estimates [5, 6] have suggested that it takes 200-300 hours to author an intelligent tutor for one hour of instruction. A further difficulty is that tutor authoring also requires multiple kinds of expertise: experts in psychology, computer science, and the content domains are usually needed when building tutors. In an effort to lower the amount of time and expertise needed to produce tutoring systems, many tutor authoring tools have been developed [5].

One such authoring tool is Cognitive Tutor Authoring Tools (CTAT) [7, 6]. This system provides tools for constructing drag-and-drop tutor interfaces, authoring Cognitive Tutors, and authoring Example-Tracing Tutors. The Cognitive Tutor is more general, but more costly to produce. Authoring in this paradigm consists of manually constructing production rules that define which actions are appropriate given the current problem-solving state; e.g., if there is a constant on both sides of the equation, then subtract one of those constants from both sides. These production rules can generalize to a wide range of problems, as long as the 'if' part of the production rule is met. Authoring an Example-Tracing Tutor consists of demonstrating every possible action for every state directly in the tutor interface (e.g., for the equation 4 + x = 2x - 5, the author would demonstrate subtracting 4 from both sides). These demonstrations comprise a behavior graph, which specifies which actions are legal in each state. While authoring these tutors is generally much easier (students can learn to build example-tracing tutors in an afternoon), they are much more specific than Cognitive Tutors. Demonstrations with Example-Tracing tutors can be generalized to new problems that share the same underlying structure (e.g., demonstrating 4 + x = 2x-5 could be generalized to 10 + x = 3x - 6 but not to 4 + x = 5) using a technique called mass production, but problems with different structure require additional demonstrations. These tutors are at two extremes: Cognitive Tutors are difficult to produce, but they are quite general; whereas, Example-tracing tutors are easy to produce, but are quite specific. Our goal is to combine the best of both worlds in an authoring tool that makes general tutors easy to build.

SimStudent, our authoring system, uses machine learning techniques to try and bridge the gap between Cognitive Tutors and Example-Tracing Tutors. It does this by learning general production rule models from demonstrations and feedback. In this paper, we summarize how this system works, give a step-by-step example of how a tutor might be authored with SimStudent, and discuss the different lines of research we are currently pursuing with SimStudent.

2 The SimStudent Architecture

SimStudent[1] was created for three purposes: 1) to advance theories of human learning; 2) to explore the learning-by-teaching phenomenon; and 3) to improve the authoring of intelligent tutors. We briefly review the SimStudent architecture, discuss prior findings, and then describe how SimStudent can be used to author tutors.

SimStudent learns from four sources of knowledge (see Figure 1). The first two sources are built before SimStudent starts learning, and the other two sources come from SimStudent's learning environment. First, SimStudent needs to recognize relevant features of the tutor interface (e.g., numbers, operators, the equals sign). These 'feature predicates' are constructed by writing small Java functions, the equivalent of writing regular expressions, to identify key features in the interface. Second, SimStudent starts with a certain level of prior knowledge (e.g.,

[1] For more details on SimStudent see http://www.simstudent.org/

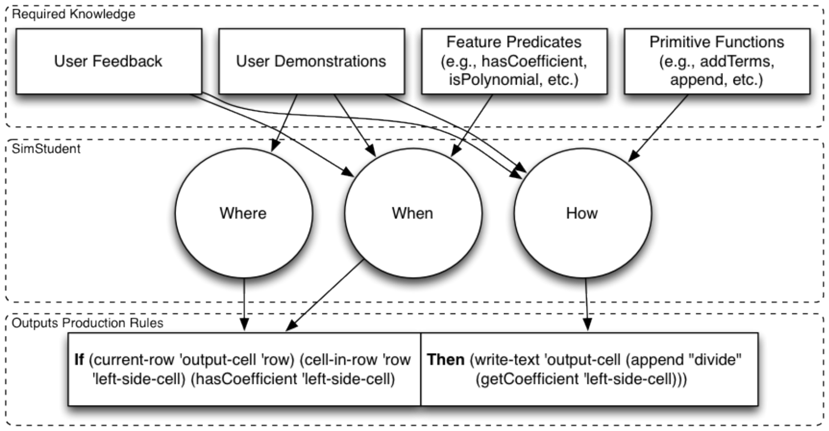

Required Knowledge
User Feedback
User Demonstrations
Feature Predicates (e.g., hasCoefficient, isPolynomial, etc.)
Primitive Functions (e.g., addTerms, append, etc.)
SimStudent
Where
When
How
Outputs Production Rules
If (current-row 'output-cell 'row) (cell-in-row 'row 'left-side-cell) (hasCoefficient 'left-side-cell)
Then (write-text 'output-cell (append "divide" (getCoefficient 'left-side-cell)))

Fig. 1. The knowledge (squares) and learning processes (circles) utilized by the SimStudent system.

SimStudent for algebra can add two numbers at the beginning); these 'primitive functions' are also small Java functions, similar to basic Excel formulas, for performing mental and interface actions. Third, within the learning environment, SimStudent is provided with author demonstrations. These consist of the author solving sample problem steps. Fourth, SimStudent learns from yes/no correctness feedback when it attempts steps in a problem. After tutor problems have been demonstrated, SimStudent will learn new rules, attempt to apply them to new problems, and will ask the author for verification that the rules were applied correctly.

Given these four sources of knowledge, SimStudent employs three learning mechanisms to produce general production rules. These three types of learning are called 'how' learning, 'where' learning, and 'when' learning. How learning identifies sequences of primitive function operators that would have plausibly produced the user demonstrations (e.g., going from 4+4x = 5 to 4x = 1 could be caused by subtracting the constant '4' from both sides or by subtracting the coefficient of 'x' from both sides). How learning generates the 'then' part of the production rules. Where learning identifies which interface elements are relevant to each demonstration, (e.g., learning that all the interface elements in the last used row are relevant). Lastly, When learning identifies the conditions under which a given sequence of operators is applicable. The Where and When learning jointly produce the 'if' part of a production rule. As the author demonstrates problem steps the three mechanisms learn new production rules. Once production rules are learned, SimStudent attempts to use those rules to solve practice problems. The rules are refined when the author provides correctness feedback on each step of the problem.

SimStudent enables us to test if the How, Where, and When mechanisms are reasonable approximations of how human students learn from demonstration and feedback. Indeed, empirical work indicates that models generated by SimStudent better fit student tutor data than models hand authored by domain experts [8]. These results were replicated across three different domains (algebra, stoichiometry, and fraction addition). SimStudent may produce better results because it is less susceptible to "expert blind spots" [9] than domain experts. These blind spots refer to knowledge that an expert doesn't realize they know. For example, a domain expert might view $-x = 4$ and $-1x = 4$ as equivalent, but the SimStudent model recognizes that additional knowledge is needed in the first case since the -1 coefficient is implicit. Improved student models are likely to result in better student learning [10] because they guide interface design, problem selection, and assessment of student knowledge. Continuing the example above, the original model for students' extraction of a negative coefficient lumped $-x$ together with $-3x$, $-5x$, etc. That model assumes that practice on any of those examples would lead to improved performance on other examples within the group. In contrast, the SimStudent model would provide additional practice for $-x$ and would not assume automatic transfer from $-3x$ to $-x$. These findings, that SimStudent can create better models and that better models result in better student learning, show promise for leveraging SimStudent to create more effective tutors.

In addition to theory building, SimStudent has been used as a teachable agent. Instead of asking students to learn directly from the tutor, students are tasked with teaching SimStudent so that it can pass a quiz on the domain content. The learning-by-teaching paradigms aim to take advantage of the "protoge effect," so called because students have been found to be more motivated to learn on behalf of a teachable agent than to learn for themselves [11]. Results [12] suggest that learning-by-teaching is as effective as a Cognitive Tutor for students who have reached a basic level of competency. This work seems to imply that we don't need an expert model to teach students since they can learn simply by teaching the SimStudent agent. However, the students are still receiving feedback on how the SimStudent agent does on each quiz, and grading the quizzes is done using an expert model. Therefore, it is still necessary to author good expert models, even in a learning-by-teaching paradigm.

A third line of SimStudent research investigates the authoring of expert models for use in both tutoring systems and teachable agents. One study found that higher quality models are produced by providing SimStudent with both demonstrations and feedback, compared with only giving it demonstrations [13]. A follow up study showed that authoring an Algebra tutor with SimStudent is more efficient than authoring an equivalent tutor using Example Tracing and that the model learned by SimStudent is more general [14], when the background knowledge had already been authored. Overall, research on the SimStudent system suggests that it might be a viable tool for efficiently authoring tutoring content that is general and of high quality.

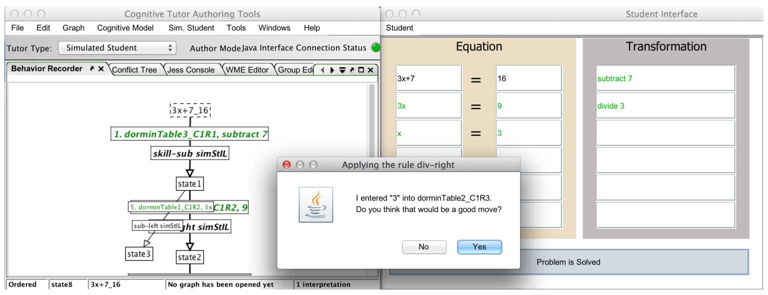
3 An Example of Authoring with SimStudent

Authoring with SimStudent is similar to authoring with CTAT. Details of authoring a tutor with CTAT are written up elsewhere (http://ctat.pact.cs.cmu.edu/), so we focus on the aspects of authoring that are unique to SimStudent: authoring background knowledge and tutoring the SimStudent system interactively. This example shows how to construct a simple algebra tutor using SimStudent.

The first step in authoring a SimStudent tutor is to construct the feature predicates that SimStudent will use to interpret the world. These feature predicates are small Java functions that returns True if a feature is present in an interface element and False otherwise. One example might be the "HasCoefficient" feature, which would be True for $3x$ but False for $x + 1$. SimStudent uses feature predicates to recognize important features in the tutor interface. For the algebra domain we have authored 16 feature predicates. These predicates tend to be relatively general, so they can be reused from one tutor to the next.

The next step in authoring is to construct primitive function operators. These operators are similar to the feature predicates, in that they are small java functions, but they take two inputs (taken either from interface elements or from the outputs of other primitive function operators) and return a single value. One example of a primitive function operator is "AddTerm": when given two numbers it returns their sum. These operators enable SimStudent to explain demonstrations and to take actions in the tutor interface. For the algebra domain we have authored 28 primitive function operators. Similar to feature predicates, these functions tend to be reusable across tutors.

After constructing background knowledge, authoring is done in the tutor interface using CTAT and running SimStudent's interactive learning module. SimStudent tries to solve the problem loaded into the interface by firing an applicable production rule and taking the step determined by the rule. After each step it asks for feedback on the correctness of that action (see Figure 2). If the author provides positive feedback to SimStudent, then it will continue solving the problem. If the feedback is negative, SimStudent will try other applicable production rules. When it runs out of production rules that apply to the current



Tutor Type: Simulated Student

Author Mode Java Interface Connection Status

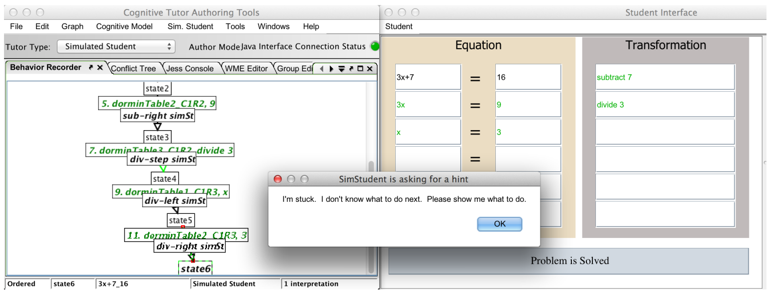
Equation			Transformation
3x+7	=	16	subtract 7
3x	=	9	divide 3
x	=	3	

I entered "3" into dorminTable2_C1R3.
Do you think that would be a good move?

Problem is Solved

Ordered | state8 | 3x+7_16 | No graph has been opened yet | 1 interpretation

Fig. 2. SimStudent asking for correctness feedback.



SimStudent is asking for a hint

I'm stuck. I don't know what to do next. Please show me what to do.

Equation			Transformation
3x+7	=	16	subtract 7
3x	=	9	divide 3
x	=	3	

Fig. 3. SimStudent asking for a demonstration.

step, it will ask the user to do that step and then use its learning mechanisms to learn a new production rule from that demonstration (see Figure 3).

After tutoring, SimStudent produces a behavior graph (shown on the left side in Figures 2 and 3) and a production rule file. The behavior graph can power an Example-Tracing tutor and the production rule file can run a Cognitive Tutor.

4 Future Work

The SimStudent architecture shows promise as a tool for simultaneously increasing authoring efficiency and model quality [14, 13], but more research still needs to be done. In terms of efficiency, few studies have directly compared the efficiency of different authoring approaches. We are exploring different usability and interaction models as a method for evaluating different approaches (e.g., the goals, operators, methods, and selections rules models). In terms of model quality, we are working to identify key performance metrics for general models. For example, in addition to evaluating accuracy and recall of a model for correct behavior, we are also looking at accuracy and recall of incorrect behavior. SimStudent can learn incorrect productions from correct instruction by making incorrect induction due to suboptimal background knowledge [15]. These plausible, but incorrect, inductions can be used to identify bug rules that might be missed by experts. If seems likely that the SimStudent models will have higher quality when evaluated in terms of both correct and incorrect behavior then when evaluated in terms of correct behavior alone (based on previous findings related to the expert blind spot).

In addition to evaluating efficiency and quality, we are also interested in exploring how to increase SimStudent's generality. To accomplish this, we have been exploring approaches for automatically learning the background feature predicates from tutoring [16]. By reducing or eliminating the need to author this predicate knowledge, we will make it easier to apply SimStudent to new domains. Additionally, we have been exploring how this feature predicate learning can be used to apply SimStudent to learning models for open-ended tasks, such as educational games [17].

Utilizing these new improvements, we are exploring the effectiveness of the SimStudent architecture for authoring content for a MOOC platform, such as Khan Academy. We are planning to recreate some of the MOOC instruction using SimStudent and to produce evidence that the cost-benefit of creating intelligent tutors for these platforms is worth it. There is a great potential for intelligent tutors to have a broader impact (through MOOCs and other avenues), if we can demonstrate that authoring tools can lower the cost to tutor authoring while jointly improving tutor quality and student learning. It is our hope that SimStudent, and other general tutor authoring platforms, can help achieve this goal.

Acknowledgments

This work was supported in part by a Graduate Training Grant awarded to Carnegie Mellon University by the Department of Education (#R305B090023) and by the Pittsburgh Science of Learning Center, which is funded by the NSF (#SBE-0836012). This work was also supported in part by National Science Foundation Awards (#DRL-0910176 and #DRL-1252440) and the Institute of Education Sciences, U.S. Department of Education (#R305A090519). All opinions expressed in this article are those of the authors and do not necessarily reflect the position of the sponsoring agency.

References

1. Koedinger, K.R., Anderson, J.R.: Intelligent Tutoring Goes To School in the Big City. International Journal of Artificial Intelligence in Education **8** (1997) 1–14
2. Pane, J.F., Griffin, B.A., McCaffrey, D.F., Karam, R.: Effectiveness of Cognitive Tutor Algebra I at Scale. Technical report, RAND Corporation, Santa Monica, CA (March 2013)
3. Ritter, S., Anderson, J.R., Koedinger, K.R., Corbett, A.T.: Cognitive Tutor: Applied research in mathematics education. Psychonomic Bulletin & Review **14**(2) (2007) 249–255
4. Vanlehn, K., Lynch, C., Schulze, K., Shapiro, J.A., Shelby, R., Taylor, L., Treacy, D., Weinstein, A., Windersgill, M.: The Andes Physics Tutoring System: Five Years of Evaluations. In McCalla, G., Looi, C.K., Bredeweg, B., Breuker, J., eds.: Artificial Intelligence in Engineering. (2005) 678–685
5. Murray, T.: An Overview of Intelligent Tutoring System Authoring Tools: Updated analysis of the state of the art. In Murray, Ainsworth, Blessing, eds.: Authoring tools for advanced technology learning environments. Kluwer Academic Publishers, Netherlands (2003) 493–546
6. Aleven, V., McLaren, B.M., Sewall, J., Koedinger, K.R.: A New Paradigm for Intelligent Tutoring Systems: Example-Tracing Tutors. International Journal of Artificial Intelligence in Education **19** (2009) 105–154
7. Aleven, V., McLaren, B.M., Sewall, J., Koedinger, K.R.: The cognitive tutor authoring tools (CTAT): Preliminary evaluation of efficiency gains. In Ikeda, M., Ashley, K.D., Tak-Wai, C., eds.: International Conference on Intelligent Tutoring Systems, Springer (2006) 61–70

8. Li, N., Stampfer, E., Cohen, W.W., Koedinger, K.R.: General and Efficient Cognitive Model Discovery Using a Simulated Student. In Knauff, M., Paulen, M., Sebanz, N., Wachsmuth, I., eds.: Proceedings of the 35th Annual Meeting of the Cognitive Science Society, Austin: TX (2013)
9. Nathan, M., Koedinger, K.R., Alibali, M.: Expert Blind Spot: When Content Knowledge Eclipses Pedagogical Content Knowledge. In: Third International Conference on Cognitive Science. (2001) 644–648
10. Koedinger, K.R., Stamper, J., McLaughlin, E., Nixon, T.: Using Data-Driven Discovery of Better Student Models to Improve Student Learning. In Lane, H.C., Yacef, K., Mostow, J., Pavlik, P., eds.: Artificial Intelligence in Engineering, Memphis, TN (July 2013) 421–430
11. Chase, C., Chin, D.B., Oppezzo, M., Schwartz, D.L.: Teachable Agents and the Protégé Effect. Journal of Science Education and Technology **18** (2009) 334–352
12. Matsuda, N., Keiser, V., Raizada, R., Tu, A., Stylianides, G., Cohen, W.W., Koedinger, K.R.: Learning by Teaching SimStudent: Technical Accomplishments and an Initial Use with Students. In Aleven, V., Kay, J., Mostow, J., eds.: International Conference on Intelligent Tutoring Systems. (2010) 317–326
13. Matsuda, N., Cohen, W.W., Koedinger, K.R.: Teaching the Teacher: Tutoring SimStudent Leads to More Effective Cognitive Tutor Authoring. International Journal of Artificial Intelligence in Education
14. MacLellan, C.J., Koedinger, K.R., Matsuda, N.: Authoring Tutors with SimStudent: An Evaluation of Efficiency and Model Quality. In Trausen-Matu, S., Boyer, K., eds.: International Conference on Intelligent Tutoring Systems. (June 2014)
15. Matsuda, N., Lee, A., Cohen, W.W., Koedinger, K.R.: A Computational Model of How Learner Errors Arise from Weak Prior Knowledge. In Taatgen, N., van Rijn, H., eds.: Annual Conference of the Cognitive Science Society, Austin, TX (2009) 1288–1293
16. Li, N., Schreiber, A.J., Cohen, W.W., Koedinger, K.R.: Efficient Complex Skill Acquisition Through Representation Learning. Advances in Cognitive Systems **2** (2012) 149–166
17. Harpstead, E., MacLellan, C., Koedinger, K.R., Aleven, V., Dow, S.P., Myers, B.: Investigating the Solution Space of an Open-Ended Educational Game Using Conceptual Feature Extraction. In: International Conference on Educational Data Mining. (2013)