SimStudent: Authoring Expert Models by Tutoring

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, koedinger@cmu.edu

Abstract. One aim of the Generalized Intelligent Framework for Tutoring (GIFT) is to reduce the time and cost of authoring Intelligent Tutoring Systems. Recent work with SimStudent offers a promising approach to the efficient authoring of expert models and misconception libraries. SimStudent works by inducing general production rule models from author demonstrations and feedback. Importantly, the demonstration and feedback takes place directly in the tutor interface and requires no programming. Empirical results have shown that models induced by SimStudent fit student data better than models hand-authored by domain experts. Additionally, an analysis with the Goals, Operators, Methods, and Selection rules (GOMS) model showed that authoring with SimStudent is more efficient than authoring with current approaches, namely Example-Tracing. This paper reviews those results and provides an example of constructing a simple algebra tutor with SimStudent. This work with SimStudent presents several concepts that may be useful in the design and development of GIFT: modularization to allow for tutor authoring by non-programmers, generation of likely student misconceptions as a byproduct of expert-model creation, and methods for comparing and evaluating authoring tools.

Keywords. Tutor Authoring Tools; Machine Learning; Expert Model; Production Rules; Learning by Tutoring; Learning by Demonstrating

1 Introduction

Intelligent tutoring systems are effective at improving learning [1-4], but development costs remain a formidable obstacle to their general adoption [5]. As an example, despite widespread use of math Cognitive Tutors (more than 500k students per year complete a Carnegie Learning tutor course), they have not been widely used more broadly (e.g., in online education platforms such as Khan Academy, Coursera, etc.), perhaps because their learning benefits are not thought to outweigh the costs of their development. Authoring costs are particularly pronounced for massive online education platforms, which have 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). Similar to the goals that motivate GIFT, 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.

SimStudent is an outgrowth of the Cognitive Tutor Authoring Tools (CTAT) [6, 7]. CTAT provides tools for constructing drag-and-drop tutor interfaces, authoring Example-Tracing Tutors, and creating an expert model for Model-Tracing Tutors. The Model-Tracing 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 applicable. 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 [7]), they are much more specific than Model-Tracing 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 structures require additional demonstrations. These types of tutors are two ends of a spectrum: Model-Tracing Tutors are difficult to produce, but they are quite general; 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 [8], uses machine-learning techniques to try and bridge the gap between Example-Tracing Tutors and Model-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.

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

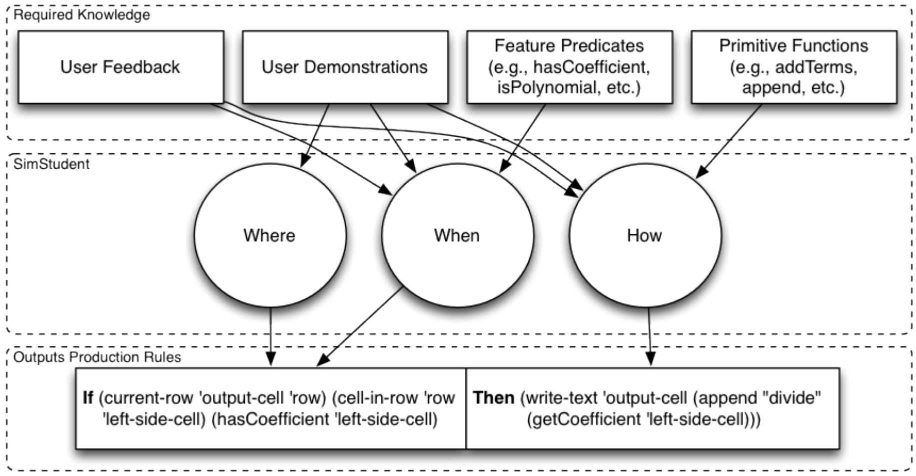

Fig. 1. The knowledge (squares) and learning processes (circles) utilized by the SimStudent system.

SimStudent learns from four sources of knowledge [8] (see Figure 1). Feature predicates and primitive functions are built before SimStudent starts learning, and User Feedback and User Demonstrations 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., 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 'user demonstrations.' These consist of the author solving sample problem steps. Fourth, SimStudent learns from 'user feedback,' which is 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. Based on this author feedback, the condition statements of these rules are refined.

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 [9]. 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" [10] 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 because the -1 coefficient is implicit. Improved student models are likely to result in better student learning [11] 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 "protégé 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 [12]. Results [13] 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.

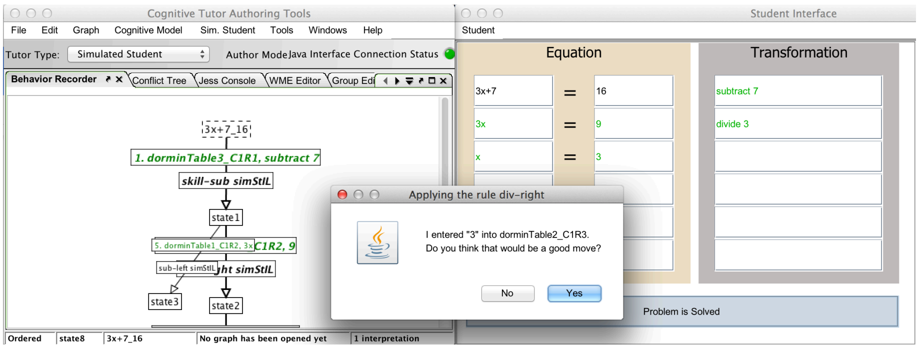

Fig. 2. SimStudent asking for correctness feedback.

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 [8]. 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.

3.1 Authoring Background Knowledge

The SimStudent system separates the authoring of background knowledge from the construction of the expert model. Constructing the background knowledge requires basic programming skills; since the expert model is created through interactive tutoring, it requires no programming at all. The first class of background knowledge, feature predicates, are small Java functions that return 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 second class of knowledge, primitive function 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, primitive functions tend to be reusable across tutors.

3.2 Tutoring SimStudent Interactively

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 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.

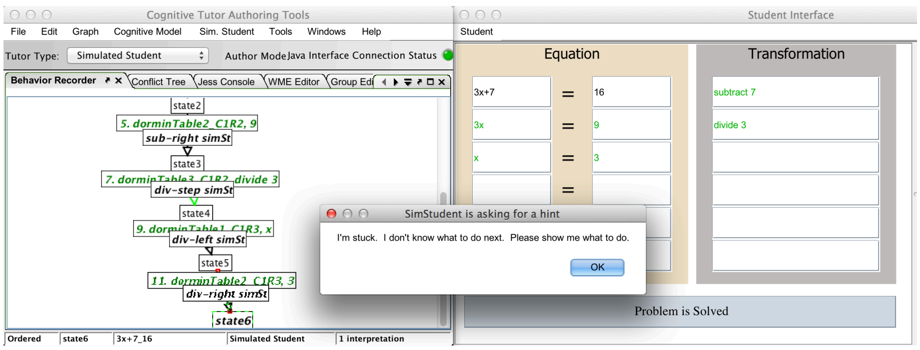

Fig. 3. SimStudent asking for a demonstration.

4 Future Work

The SimStudent architecture shows promise as a tool for simultaneously increasing authoring efficiency and model quality [8, 14], 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. As an example, when SimStudent is taught to divide both sides by 3 for the problem 3x = 6, it might incorrectly learn a rule for dividing both sides by any number on the left side of the equation (an error students commonly make). When SimStudent incorrectly solves a subsequent problem using this bug rule, it will receive negative correctness feedback and refine its rule to only apply to dividing by the coefficient. SimStudent could be modified to request a hint message for this misconception during authoring, such as text pointing out that the both sides of the equation are divided by the coefficient of x, not just any number. After this modification, it would be worth evaluating how authoring with SimStudent compares to Example-tracing or hand authoring in terms of number of bug rules identified.

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.

5 Recommendations for GIFT

Based on our research with the SimStudent system, we have three recommendations for the Generalized Intelligent Framework for Tutoring. First, as with many authoring frameworks, authoring expert models in GIFT is a challenging problem. As such, it may benefit from a tool like SimStudent to aid in this authoring process. SimStudent's automatically constructed expert models perform better than hand-authored models for multiple domains because they are not susceptible to expert blind spots. At the same time, in the process of generating these expert models, SimStudent makes errors that are often helpful in predicting human students' mistakes. These errors could form the basis of a misconceptions library, before any data is gathered from real students. Exploring how SimStudent's expert models and misconceptions could be utilized by GIFT may be a worthwhile direction for future work. This integration could take one of two forms: A SimStudent-like module could be constructed for GIFT that would allow authors to construct the domain knowledge by tutoring GIFT directly in the tutoring application or SimStudent could be configured to work with the tutoring application and then the production rule file generated by SimStudent could be converted into one of the domain knowledge formats acceptable to GIFT.

Second, we recommend that GIFT separate authoring of knowledge that is domain specific from the authoring of knowledge that is tutor specific. Domain general knowledge is already separated from domain specific knowledge in GIFT, but our research has found that domain specific knowledge is often reusable across tutoring applications. In the SimStudent system, we separated the construction of background domain knowledge (algebra features and operators), which tends to be reusable across tutors for the same domain (algebra), from the construction of an expert model for a specific tutor (how to solve particular algebra problems in the tutor interface). This was particularly useful because domain-specific background knowledge requires some Java programming abilities, whereas tutor-specific knowledge only requires the ability to demonstrate solutions in the tutor. This separation is useful because it allows domain experts, who may not know how to program, to construct the expert model for the tutor, if adequate domain knowledge already exists. Furthermore, our work with SimStudent has shown that domain-specific background knowledge tends to transfer across different tutors in the same domain and sometimes even across domains. For example, the feature predicates for extracting numbers and words from problem descriptions work in fraction addition tutors, algebra tutors, and chemistry tutors.

Finally, the modularity of GIFT makes it ideal for measuring the usability and efficiency of different combinations of authoring approaches and tools. We have used the GOMS model to evaluate the efficiency of different expert model authoring approaches (SimStudent and Example Tracing) in the context of CTAT. GIFT would benefit from similar analyses. Future GIFT research might explore how similar usability models can be employed for measuring the efficiency of different aspects of tutor authoring in a way that is comparable to other systems.

6 Acknowledgements

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, 1–14 (1997).
2. Pane, J.F., Griffin, B.A., McCaffrey, D.F., Karam, R.: Effectiveness of Cognitive Tutor Algebra I at Scale. RAND Corporation, Santa Monica, CA (2013).
3. Ritter, S., Anderson, J.R., Koedinger, K.R., Corbett, A.T.: Cognitive Tutor: Applied research in mathematics education. Psychonomic Bulletin & Review. 14, 249–255 (2007).
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. Presented at the Artificial Intelligence in Engineering (2005).
5. Sottilare, R.A., Holden, H.K.: Motivations for a Generalized Intelligent Framework for Tutoring (GIFT) for Authoring, Instruction, and Analysis. Presented at the AIED 2013 Workshop on Recommendations for Authoring, Instructional Strategies and Analysis for Intelligent Tutoring Systems (ITS): Towards the Development of a Generalized Intelligent Framework for Tutoring (GIFT) June 21 (2013).
6. Aleven, V., McLaren, B.M., Sewall, J., Koedinger, K.R.: The cognitive tutor authoring tools (CTAT): Preliminary evaluation of efficiency gains. Presented at the International Conference on Intelligent Tutoring Systems (2006).
7. Aleven, V., McLaren, B.M., Sewall, J., Koedinger, K.R.: Example-Tracing Tutors: A New Paradigm for Intelligent Tutoring Systems. International Journal of Artificial Intelligence in Education. 19, 105–154 (2009).
8. 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.
9. Li, N., Stampfer, E., Cohen, W.W., Koedinger, K.R.: General and Efficient Cognitive Model Discovery Using a Simulated Student. Presented at the Proceedings of the 35th Annual Meeting of the Cognitive Science Society, Austin: TX (2013).
10. Nathan, M., Koedinger, K.R., Alibali, M.: Expert Blind Spot: When Content Knowledge Eclipses Pedagogical Content Knowledge. Presented at the Third International Conference on Cognitive Science (2001).
11. Koedinger, K.R., Stamper, J., McLaughlin, E., Nixon, T.: Using Data-Driven Discovery of Better Student Models to Improve Student Learning. Presented at the Artificial Intelligence in Engineering, Memphis, TN July (2013).
12. Chase, C., Chin, D.B., Oppezzo, M., Schwartz, D.L.: Teachable Agents and the Protégé Effect. Journal of Science Education and Technology. 18, 334–352 (2009).
13. 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. Presented at the International Conference on Intelligent Tutoring Systems (2010).

14. MacLellan, C.J., Koedinger, K.R., Matsuda, N.: Authoring Tutors with SimStudent: An Evaluation of Efficiency and Model Quality. Presented at the 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. Presented at the Annual Conference of the Cognitive Science Society, Austin, TX (2009).
16. Li, N., Schreiber, A.J., Cohen, W.W., Koedinger, K.R.: Efficient Complex Skill Acquisition Through Representation Learning. Advances In Cognitive Systems. 2, 149–166 (2012).
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. Presented at the International Conference on Educational Data Mining (2013).