Authoring Tutors with SimStudent: An Evaluation of Efficiency and Model Quality

Christopher J. MacLellan, Kenneth R. Koedinger, Noboru Matsuda

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

Abstract. Authoring Intelligent Tutoring Systems is expensive and time consuming. To reduce costs, the Cognitive Tutor Authoring Tools and the Example-Tracing Tutor paradigm were developed to make the tutor authoring process more efficient. Under this paradigm, tutors are constructed by demonstrating behavior directly in a tutor interface, reducing the need for programming expertise. This paper evaluates the efficiency of authoring a tutor with SimStudent, an extension to the Example-Tracing paradigm that is designed to produce greater generality in less time by induction from past demonstrations and feedback. We found that authoring an algebra tutor in SimStudent is faster than Example-Tracing while maintaining equivalent final model quality. Furthermore, we found that the SimStudent model generalizes beyond the problems that were used to author it.

1 Introduction

Intelligent Tutoring Systems (ITSs) are a widely used educational technology [1] that has been shown to improve learning over many traditional forms of instruction [2–7]. One challenge associated with ITSs is that they are difficult to build and require developers to make decisions about trade offs between power, usability, fidelity, and cost [8]. To overcome the challenge of authoring high-quality tutors, many authoring tools have been developed [8]. We focus on the Cognitive Tutor Authoring Tools (CTAT), which has been shown to decreases the time required to build a tutor by as much as 50% [9]. CTAT achieves these gains by providing a drag-and-drop interface builder and by providing support for authoring two types of tutors: Cognitive Tutors and Example-Tracing Tutors.

Cognitive Tutors provide step-by-step feedback to students while they solve problems by comparing their actions to a model of expert behavior for the given domain. This model uses production rules, if-then rules that map each state in a tutoring interface to a legal action that might be taken on that state [10]. These production rules are quite general, in that a single rule might apply to many states throughout problem solving. However, in general these models are costly to produce. It can take 200-300 hours of development to produce a Cognitive Tutor for one hour of instruction and tutor development usually requires multiple kinds of expertise (i.e., domain expertise, Cognitive Psychology, and Computer Science) [11, 8].

Example-Tracing Tutors were developed to reduce the costs of producing a Cognitive Tutor [9, 11]. These tutors reduce the technical costs of tutor development by allowing domain experts and Cognitive Psychologists to build a cognitive model by demonstration rather than by programming a production rule model. To build an expert model in this paradigm, the tutor author demonstrates every legal action at every step for every problem. The resulting cognitive model, called a behavior graph, is a simplified production rule model, where each production rule maps a single state to a single action. While some methods for generalization do exist, these models are still much less general than Cognitive Tutors. However, in practice this limitation is balanced out by the ease of authoring– in many cases individuals can learn to author tutors in one afternoon [9].

While CTAT drastically reduces the cost of authoring ITSs, the tutors that it can produce are at two ends of an authoring spectrum: Cognitive Tutors are difficult to produce, but are maximally general, while Example-Tracing Tutors are easy to produce, but are maximally specific. Recent extensions to Example-Tracing Tutors have addressed how to make Example-Tracing Tutors more general. Existing techniques include specifying sequences of actions that might be executed in any order, employing regular expressions or formulas for matching demonstrations, and duplicating behavior graph structures for many problems of similar type, an approach called mass-production [11]. While these techniques have improved Example-Tracing Tutor generality, more research into how general expert models might be produced without technical expertise is still an active area of research.

One promising development is the SimStudent architecture, a CTAT module that tries to bridge the gap between Example-Tracing Tutors and Cognitive Tutors by learning production rule models from demonstrations and problem-solving feedback [12]. Previous work has shown that authoring a model by tutoring (both demonstrations and feedback) is more efficient than demonstrations alone. However, the SimStudent approach to authoring has never been compared to the more widely used Example-Tracing approach.

We compare authoring time for a tutor built with SimStudent and Example-Tracing by using a Keystroke-Level Model (KLM) [13], a simple human information processing model that estimates how many seconds it would take a trained user to perform authoring actions. This analysis shows that SimStudent can reduce authoring time by as much as 50%, for domains that SimStudent has adequate background knowledge. Additionally, we evaluate the quality of the model produced by each approach and show that while both approaches produce models with equivalent quality by the end of authoring, SimStudent shows the ability to generalize from authored to unauthored problems along the way. Before showing these results, we review CTAT and how it can be used to author an Example-Tracing Tutor and then show how the SimStudent architecture can be used to author a tutor through CTAT.

2 Authoring an Example-Tracing Tutor in CTAT

In this section, we show an example of how CTAT can be used to author an Example-Tracing tutor for one- and two-step Algebra equation solving for a given tutor interface. For more details see [9].

To construct an Example-Tracing tutor, one demonstrates behavior directly in the tutoring interface. Traces of these actions are recorded in a behavior graph. A simple Algebra tutor interface and its associated behavior graph are shown in Figure 1.

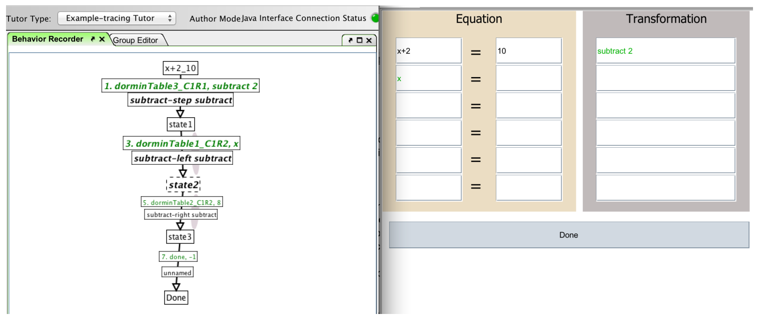

Tutor Type: Example-tracing Tutor

Fig. 1. The Algebra interface and the Behavior Graph produced from demonstrating behavior directly in the interface. The green text specifies the actions taken in the interface and the black text just below shows the author produced skill labels. The ellipsoids between the second and third state (partly occluded by the labels) signify that the actions can be executed in any order.

In this figure we see an interface for tutoring multi-step algebra equation solving (right) and a behavior graph (left). Each node represents a state of the tutoring interface, where the initial state represents the problem to solve. Each link coming out of a node represents an action that might be performed in the state the node represents. In Example Tracing each link is produced as a result of a single action demonstrated directly in the tutor interface, where many legal actions might be demonstrated for each state.

As an example of authoring, consider a tutor for solving the equation $x+2 = 10$ (using the interface shown in Figure 1). To construct this tutor the author would:

1. Create an empty behavior graph.
2. Input the equation into the interface.
3. Create the initial node of the behavior graph to represent this start state.
4. Demonstrate the first action, subtract 2 from both sides. This demonstration produces a new link in the behavior graph, which the author will label with to the knowledge necessary to perform that action (this label is useful for monitoring learning).

5. Demonstrate the second action, entering x as the new left side of the equation. A second link is produced and labeled in the behavior graph.
6. Next, the third action is demonstrated, entering 8 as the new right side of the equation. A third link is produced and labeled in the behavior graph.
7. The author performs the final action, clicking the done button. This adds a final link to the behavior graph, which the author labels as requiring the done skill.
8. Because the order of the second and third actions doesn't matter, the author either selects both links and marks them as being unordered or returns to the previous state by clicking on the node in the behavior graph and demonstrating the actions in reverse order.

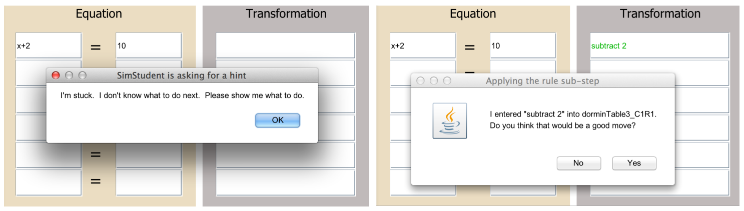
Figure 1 shows the resulting behavior graph (with the second and third links marked as unordered– denoted by the ellipsoids behind the skill names). For a given tutor interface, an author may produce many behavior graphs, each representing a different problem that might be solved in that interface. Other CTAT tools deploy the interface and associated behavior graphs as an ITS, a matter not discussed here.

3 Authoring using SimStudent

While the Example-Tracing approach has proven effective for authoring, the generality of the model is quite limited. To overcome this limitation the SimStudent architecture was created. This system extends Example-Tracing by inducing more general production rule models from demonstrations and tutoring feedback (for details on this rule induction see [12]). To summarize, SimStudent learns production rules from the demonstrations and refines the conditions on these production rules based on the author's feedback.

The process of authoring a tutor with SimStudent is similar to Example Tracing, in that the SimStudent asks for demonstrations when it does not know how to proceed. However, when SimStudent already has an applicable production rule, it fires the rule and shows the resulting action in the tutor interface. It then asks the tutor author for yes/no feedback on whether this action is correct. Based on the author's feedback, SimStudent refines the conditions of its production rules and proceeds to continue trying to solve the problem. If the author's feedback is negative, SimStudent may exhaust all of its applicable production rules. In these cases, SimStudent asks the user for a demonstration of the correct action. Figure 2 shows how SimStudent communicates with the tutor author to receive a demonstration or feedback.

When authoring models in SimStudent, the author does not have to specify that interface actions are unordered, as one would need to do in Example Tracing, because the production rules learned by SimStudent are applicable in any order, as long as their conditions are satisfied. It is worth noting that the process for authoring a tutor using SimStudent produces both behavior graphs, which might subsequently be used for Example Tracing, and a more general production rule model, which might be used in a full-fledged Cognitive Tutor.



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

I entered "subtract 2" into dorminTable3_C1R1.
Do you think that would be a good move?

Fig. 2. The image on the left shows SimStudent asking for a demonstration when it does not know how to proceed. The image on the right shows SimStudent asking for feedback on the action it took when it does know how to proceed.

4 Method

An Algebra tutor was authored using both the Example-Tracing and SimStudent approaches. In both cases, the tutor was authored to provide step-by-step feedback on the 20 algebra equations shown in Table 1, where they are organized by the skills necessary to solve them.

We estimated the average authoring time for each approach using the KLM technique, which involved breaking down each authoring action into its primitive steps (many mental pauses, point-and-click actions, and key presses) and then using timing data for how long the average user needs to complete these primitive steps. The KLM provides an accurate prediction of error-free task execution time for an expert user [13]. Both tutors were authored using CTAT and the same Algebra tutor interface, shown in Figures 1 and 2. As shown above, the authoring actions (e.g., providing demonstrations) differ only slightly between approaches; however, the frequency of these actions differs more substantially. In particular, many demonstrations are replaced with feedback when authoring in SimStudent. To compare timings between the two approaches we kept count of the number of authoring actions needed to author each problem, ignoring those actions that were identical between approaches (e.g., creating new behavior graph or start state).

Finally, after each problem demonstration, we evaluated the model quality in terms of the 20 problems that the finished tutor should be able to teach. To

Table 1. A tutor was developed to teach these 20 problems using the Example-Tracing and SimStudent approaches. The problem numberings denote the order in which problems were authored, so all problems of the same type were authored together.

Subtract	Add	Divide	Sub + Divide	Add + Divide
1. x+1=10	5. x-5=10	9. 3x=12	13. 5x+2=12	17. 2x-1=1
2. x+2=12	6. x-6=20	10. 4x=8	14. 7x+1=15	18. 3x-3=3
3. x+3=20	7. x-7=14	11. 2x=10	15. 2x+4=8	19. 5x-2=8
4. x+4=4	8. x-2=9	12. 7x=14	16. 3x+6=9	20. 7x-4=10

evaluate each model we computed a step and recall score, similar to previous studies [12]. The step score equals the number of correct actions suggested by the model divided by the total number of actions (both correct and incorrect) suggested by the model at each step. When the model suggests no actions, the step score is 0. The step score is averaged across all steps to get an overall step score that represents the quality of the model. The recall score is equal to 1 if the model suggests a correct action on a given step and 0 otherwise. The recall score is averaged across all steps to get an overall recall score. Recall assesses how complete a model is, in terms of the percentage of steps that can be tutored.

5 Results

5.1 Authoring Time

Each approach had two authoring actions. Authoring in Example Tracing consisted of demonstrating actions and specifying actions as unordered; whereas, authoring in SimStudent consisted of a slightly longer demonstration and required the author to give feedback on SimStudent's actions. Table 2 shows the number of seconds estimated for each of these actions using the KLM. These estimates were produced by breaking each action down in terms of their primitive steps (mental pauses, pointing and clicking, and keypresses) and summing the time it would take the average user to perform these steps, using previously computed estimates [13].

Figure 3 shows the cummulative time required to author 20 problems using each approach; these estimates were computed by counting the number of tutoring actions needed to author each problem and multiplying these counts by the time estimates shown in Table 2.

5.2 Model Quality

To evaluate the quality of each model we computed the step and recall scores on all 20 problems in the training set after each problem had been authored. This is meant to assess the quality in terms of the 20 problems each model is being built to teach. Figure 4 shows the step and recall scores of each approach after each problem had been authored.

Table 2. The KLM estimates of how long it would take an author to perform each authoring action.

Action	**Time (sec)**
Example-Tracing Demonstration	8.8
Example-Tracing Specify Unordered Actions	5.8
SimStudent Demonstration	10.4
SimStudent Feedback	2.4

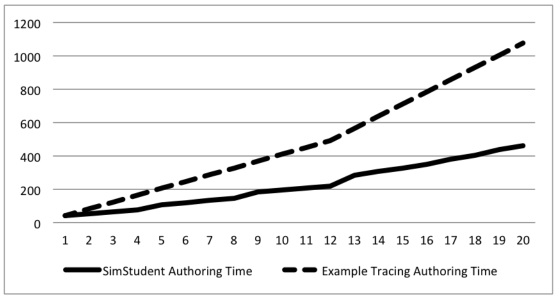


Fig. 3. Cummulative authoring time (in seconds) for each approach, as estimated by the KLM. This model only computes the time needed to perform actions that differ between approaches (demonstrations, specifying ordering, and feedback), so these estimates are slightly less than actual authoring time.

6 Discussion

The KLM analysis of the two approaches shows evidence that authoring using the SimStudent approach may yield improved authoring efficiency over the standard Example-Tracing approach. This efficiency gain was because SimStudent only required feedback, instead of demonstrations, when it had applicable production rules. Providing feedback (2.4 sec) takes much less time than performing a demonstration (8.8 sec for Example Tracing and 10.4 sec for SimStudent), so this results in a substantial decrease in authoring time. If SimStudent was used solely as a way to improve the efficiency of producing behavior graphs for an Example-Tracing tutor (and not as a way to author more general productions), then it appears authoring efficiency would improve.

When analyzing the model quality of the two approaches, it is important to note that by the end of the authoring process both tutors achieve 100% step and recall scores. However, the process each approach takes to get to 100% is quite different. Figure 4 shows that the Example-Tracing tutor linearly progresses towards perfect scores. Such linear progress is to be expected because it achieves perfect step and recall on all problems that have been authored and 0 step and recall on all problem that have not been authored.

For the SimStudent approach the progression is much different. After the first problem has been authored SimStudent has approximately 40% step and recall scores on the entire set of 20 problems (much larger than the 5% scores for Example-Tracing). This increase is due to the fact that SimStudent is (attempting to) learn general rules from the first problem and some of those rules transfer to the steps in other problems that have similar demands (e.g., knowing that the problem is done when you have x equals some number). After the first problem, SimStudent's step and recall scores jump every time it sees a new problem type because SimStudent learns new production rules to solve these new types

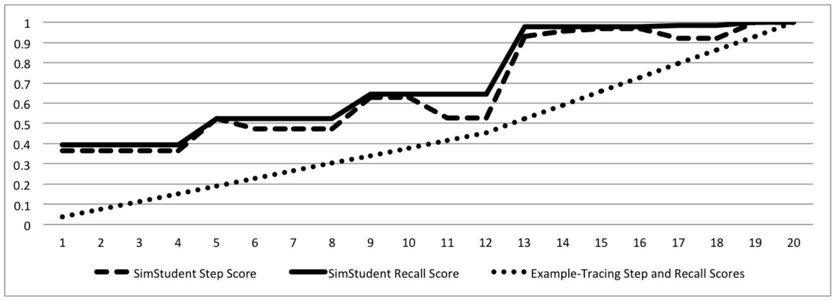


Fig. 4. The step and recall scores computed over all 20 problems after each problem has been authored (the x axis is # of problems authored so far). The Example-Tracing step and recall scores are identical at all points and are just shown with a single line. There is a slight increase in the slope of the Example-Tracing line at problem 12 because the problems transition from one to two step equations.

of problems (such as adding or dividing) that are useful in solving subsequent problems.

The greater generalization that SimStudent demonstrates within the 20-problem it gets trained on also applies beyond those 20 problems. That is, whereas the Example-Tracing model can only tutor on these 20 problems, the SimStudent model will work on a wider set of problems. For example, the SimStudent model can tutor problems of the same type that have different numbers and minor variations of these problems, such as "(x+2)=9" or "2+x=9." This is why we see plateaus in Figure 4 where SimStudent has already learned how to solve novel problems of the same type. Additionally, SimStudent can tutor some of the steps of more complex problems (e.g., finishing "5x + 10 = 7x" after 7x has been been subtracted from both sides) thus saving time in authoring those more complex problems.

Interestingly, we also observed that as SimStudent gets tutored on new problems its Step score sometimes decreases for previously tutored problems (though never enough to regress below the progress of Example-Tracing). This regression occurs because SimStudent is biased to learn the most general production rule conditions from the examples it sees and thus often overgeneralizes in its early rule acquisition. For example, when generalizing the conditions on the divide rule after getting positive feedback (e.g., when entering divide 2 for 2x=10 – problem #11), SimStudent may learn a rule without a pre-condition specifying a need for a coefficient and thus apply too broadly (e.g., divide 2 for x-2=9). In general, this results in behavior where SimStudent tries to apply productions where they are not applicable, such as trying to add on previous subtraction problems. This overgeneralization might be desirable when trying to model student errors (an application for which SimStudent has been used in the past), but when authoring an expert model of a tutor a decrease in step score on previously authored problems is not desirable. One way to minimize this effect would be to tutor problems in an interleaved vs. blocked fashion, as suggested by previous work [14]. By regularly returning to older problem types, SimStudent can receive negative feedback in the cases where it has overgeneralized. Alternatively, other approaches could be used to limit SimStudent's overgeneralization, such a using prior knowledge [15] to constrain the generalization.

One limitation of this analysis is that it does not account for the time it takes to develop domain predicates and primitive function operators for the SimStudent system, which are used for production rule learning. These are short pieces of code (roughly similar to writing functions in an Excel spreadsheet), but they do add development time that is not needed in the standard Example-Tracing approach. Despite this additional start-up cost, given the slopes of the lines in Figure 3 the SimStudent approach should eventually result in time savings as more problems are authored. The 16 predicates and 28 function operators used in this study [12] were developed for the algebra domain, but some may be applicable in other domains. Nevertheless, many domains will require new predicates and functions to be hand authored by someone with technical expertise, and this knowledge would need to be tested to ensure that it provides adaquate coverage of the given domain. Li and colleagues [16] have demonstrated how domain specific predicates and functions can be automatically acquired, eliminating or reducing this start-up knowledge engineering, but more work is still needed to demonstrate broader generality of this approach.

To summarize, we found that SimStudent decreases the amount of time needed to author a tutor over the standard Example-Tracing approach. This result is mainly due to the fact that less demonstrations are required with the SimStudent architecture. We also found that by the end of tutor authoring both approaches had equivalent model quality. Furthermore, we showed evidence that SimStudent produces a model that is more general than the specific demonstrations it sees, bridging the gap between an Example-Tracing Tutor and a full-fledged Cognitive Tutor. In some cases SimStudent overgeneralized, and we suggest ways that these overgeneralizations might be reduced. In conclusion, SimStudent appears to be a promising approach for reducing authoring time and producing more general models than standard Example Tracing.

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., Corbett, A.T.: Cognitive tutors: Technology bringing learning science to the classroom. The Cambridge Handbook of the Learning Sciences (2006) 61–78
2. Beal, C.R., Walles, R., Arroyo, I., Woolf, B.P.: On-line Tutoring for Math Achievement Testing: A Controlled Evaluation. Journal of Interactive Online Learning **6** (2007) 1–13
3. Graesser, A.C., Vanlehn, K., Rose, C., Jordan, P.W., Harter, D.: Intelligent Tutoring Systems with Conversational Dialogue. AI Magazine **22**(4) (December 2001) 39
4. 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
5. Mitrovic, A., Martin, B., Mayo, M.: Using evaluation to shape ITS design: Results and experiences with SQL-Tutor. User Modeling and User-Adapted Interaction **12**(2-3) (2002) 243–279
6. 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
7. Vanlehn, K.: The Relative Effectiveness of Human Tutoring, Intelligent Tutoring Systems, and Other Tutoring Systems. Educational Psychologist **46**(4) (October 2011) 197–221
8. 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
9. 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
10. Anderson, J.R., Corbett, A.T., Koedinger, K.R., Pelletier, R.: Cognitive Tutors: Lessons Learned. The Journal of Learning Sciences **4**(2) (July 1995) 167–207
11. 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**(2) (2009) 105–154
12. Matsuda, N., Cohen, W.W., Koedinger, K.R.: Creating Cognitive Tutors by Tutoring. International Journal of Artificial Intelligence in Education
13. Card, S.K., Moran, T.P., Newell, A.: The keystroke-level model for user performance time with interactive systems. Communications of the ACM **23**(7) (1980) 396–410
14. Li, N., Cohen, W.W., Koedinger, K.R.: Problem Order Implications for Learning Transfer. In Cerri, S., Clancey, W., eds.: International Conference on Intelligent Tutoring Systems. (March 2012) 1–10
15. MacLellan, C.J., Matsuda, N., Koedinger, K.R.: Toward a reflective SimStudent: Using experience to avoid generalization errors. In: AIED Workshop on Simulated Learners. (July 2013)
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