Authoring Tutors with Complex Solutions: A Comparative Analysis of Example Tracing and SimStudent

Christopher J. MacLellan[1], Erik Harpstead[1], Eliane Stampfer Wiese[1], Mengfan Zou[2], Noboru Matsuda[1], Vincent Aleven[1], and Kenneth R. Koedinger[1]

[1] Carnegie Mellon University, Pittsburgh PA, USA,
{cmaclell, eharpste, stampfer, noboru.matsuda, aleven, koedinger}@cs.cmu.edu,
[2] Tsinghua University, Beijing, China,
zmf11@mails.tsinghua.edu.cn

Abstract. Problems with many solutions and solution paths are on the frontier of what non-programmers can author with existing tutor authoring tools. Popular approaches such as Example Tracing, which allow authors to build tutors by demonstrating steps directly in the tutor interface. This approach encounters difficulties for problems with more complex solution spaces because the author needs to demonstrate a large number of actions. By using SimStudent, a simulated learner, it is possible to induce general rules from author demonstrations and feedback, enabling efficient support for complexity. In this paper, we present a framework for understanding solution space complexity and analyze the abilities of Example Tracing and SimStudent for authoring problems in an experimental design tutor. We found that both non-programming approaches support authoring of this complex problem. The SimStudent approach is 90% more efficient than Example Tracing, but requires special attention to ensure model completeness. Example Tracing, on the other hand, requires more demonstrations, but reliably arrives at a complete model. In general, Example Tracing's simplicity makes it good for a wide range problems, a reason for why it is currently the most widely used authoring approach. However, SimStudent's improved efficiency makes it a promising non-programmer approach, especially when solution spaces become more complex. Finally, this work demonstrates how simulated learners can be used to efficiently author models for tutoring systems.

Keywords: Tutor Authoring, Intelligent Tutoring Systems, Cognitive Modeling, Programming-by-Demonstration

1 Introduction

Intelligent Tutoring Systems (ITSs) are effective at improving student learning across many domains– from mathematics to experimental design [10, 13, 5]. ITSs

also employ a variety of pedagogical approaches for learning by doing, including intelligent novice [7], invention [12], and learning by teaching [9]. Many of these approaches require systems that can model complex solution spaces that accommodate multiple correct solutions to a problem and/or multiple possible paths to each solution. Further, modeling complex spaces can be desirable pedagogically: student errors during problem solving can provide valuable learning opportunities, and therefore may be desirable behaviors. Mathan and Koedingers spreadsheet tutor provides experimental support for this view– a tutor that allowed exploration of incorrect solutions led to better learning compared to one that enforced a narrower, more efficient solution path [7]. However, building tutoring systems for complex solution spaces has generally required programming. What options are available to the non-programmer? Authoring tools have radically reduced the difficulties and costs of tutor building [2, 6], and have allowed authoring without programming. Through the demonstration of examples directly in the tutor interface, an author can designate multiple correct solutions, and many correct paths to each solution. Yet, the capabilities of these tools for authoring problems with complex solution spaces has never been systematically analyzed.

In this paper, we define the concept of solution space complexity and, through a case study, explore how two authoring approaches deal with this complexity. Both approaches (Example Tracing and SimStudent) are part of the Cognitive Tutor Authoring Tools (CTAT) [1]. Our case study uses the domain of introductory experimental design, as problems in this area follow simple constraints (only vary one thing at a time), but solutions can be arbitrarily complex depending on how many variables are in the experiment and how many values each can take.

2 Solution Space Complexity

Solution spaces have varying degrees of complexity. Our framework for examining complexity considers both how many correct solutions satisfy a problem and how many paths lead to each solution. Within this formulation, we discuss how easily a non-programmer can author tutors that support many solutions and/or many paths to a solution.

How might this formulation of complexity apply to an experimental design tutor? Introductory problems in this domain teach the control of variables strategy (only manipulating a single variable between experimental conditions to allow for causal attribution) [3]. Due to the combinatorial nature of experiments (i.e., multiple conditions, variables, and variable values), the degree of complexity in a particular problem depends on how it is presented. To illustrate, imagine that students are asked to design an experiment to determine how increasing the heat of a burner affects the melting rate of ice in a pot (see Figure 1). The following tutor prompts (alternatives to the prompt in Figure 1) highlight how different problem framings will affect the solution complexity:

One solution with one path Design an experiment to determine how increasing the heat of a Bunsen burner affects the rate at which ice in a pot will

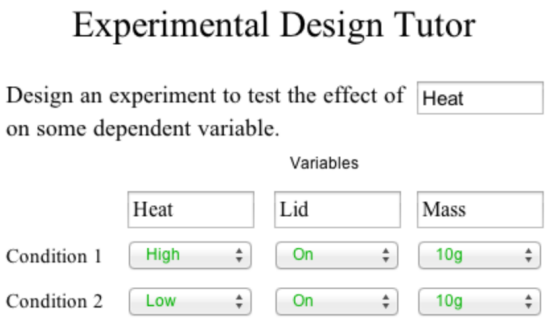

Fig. 1. Experimental design tutor interface

melt by assigning the first legal value to the variables in left to right, top down order as they appear in the table.

One solution and many paths Design an experiment to determine how increasing the heat of a Bunsen burner affects the rate at which ice in a pot will melt by assigning the first legal value to variables.

Many solutions each with one path Design an experiment to determine how increasing the heat of a Bunsen burner affects the rate at which ice in a pot will melt by assigning values to variables in left to right, top down order as they appear in the table.

Many solutions with many paths Design an experiment to determine how increasing the heat of a Bunsen burner affects the rate at which ice in a pot will melt.

While these examples show that solution space complexity can be qualitatively changed (i.e., one solution vs. many solutions) by reframing a problem, quantitative changes are also possible. For example, adding a fourth variable to the interface in Figure 1 would require two more steps per solution path (setting the variable for each condition), while adding another value to each variable increases the number of possible options at each step of the solution path. As this example illustrates, solution space complexity is not an inherent property of a domain, but rather arises from an authors design choices.

3 Tutor Authoring

Our analysis focuses on the Cognitive Tutor Authoring Tools (CTAT), as CTAT is the most widely used tutor authoring tool and the approaches it supports are representative of authoring tools in general [2]. CTAT supports non-programmers in building both tutor interfaces and cognitive models (for providing feedback). Cognitive models can be constructed with Example Tracing or SimStudent. In this section, we step through how Example-Tracing and SimStudent approaches would be applied by non-programmers to the experimental design task, using the

interface shown in Figure 1. Further, we discuss the features of each approach for handling solution space complexity in the context of this example.

3.1 Example Tracing

When building an Example-Tracing tutor in CTAT, the author demonstrates correct solutions directly in the tutoring interface. These demonstrated steps are recorded in a behavior graph. Each node in the behavior graph represents a state of the tutoring interface, and each link represents an action that moves the student from one node to another. In Example Tracing each link is produced as a result of a single action demonstrated directly in the tutor interface; many legal actions might be demonstrated for each state, creating branches in the behavior graph.

Figure 2 shows an example of our experimental design tutor interface and an associated behavior graph. The particular prompt chosen has 8 solutions and many paths to each solution. These alternative paths correspond to different orders in which the variables in the experimental design can be assigned. The Example-Tracing approach allows authors to specify that groups of actions can be executed in any order. In the context of our example, this functionality allows the author to demonstrate one path to each of the 8 unique solutions (these 8 paths are visible in Figure 2) and then specify that the actions along that path can be executed in any order. Unordered action groups are denoted in the behavior graph by colored ellipsoids.

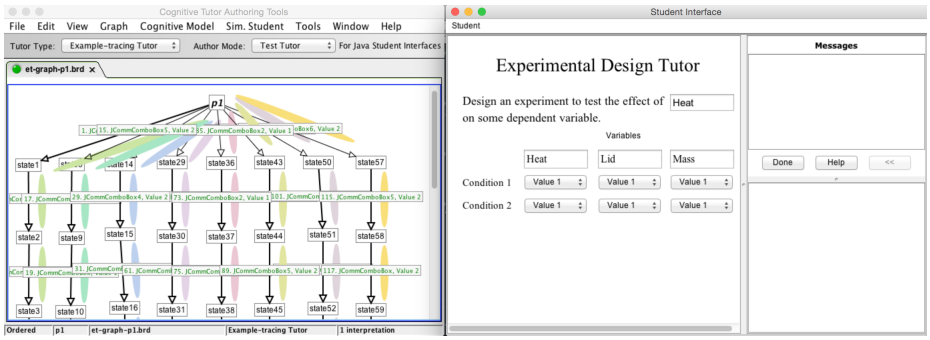

Fig. 2. An experimental design tutor (right) and its associated behavior graph (left). This tutor supports students in designing an experiment to test the effect of heat on a dependet variable. The correct answer is to pick two different values for the "Heat" variable and to hold the values constant for other variables.

Once a behavior graph has been constructed for a specific problem (e.g. determine the effect of heat on ice melting), that behavior graph can be generalized to other problems (e.g. determine the effect of sunlight on plant growth) using mass production. The mass production feature allows the author to replace specific values in the interface with variables and then to instantiate an arbitrary

number of behavior graphs with different values for the variables. This approach is powerful for supporting many different problems that have identical behavior graph structure, such as replacing all instances of "heat" with another variable, "sunlight". However, if a problem varies in the structure of its behavior graph, such as asking the student to manipulate a variable in the second column instead of the first (e.g., "lid" instead of "heat"), then a new behavior graph would need to be built to reflect the change in the column of interest.

How efficient is Example Tracing in building a complete cognitive model for the experimental design problem? The complete model consists of 3 behavior graphs (one for each of the three variable columns that could be manipulated). Each graph took 56 demonstrations and required 8 unordered action groups to be specified. Thus, the complete cognitive model required 168 demonstrations and 24 unordered group specifications. Using estimates from a previously developed Keystroke-Level Model [6], which approximates the time needed for an error-free expert to perform each interface action, we estimate that this model would take about 27 minutes to build using Example Tracing. Notably, the ability to specify unordered action groups offers substantial efficiency gains - without it, authoring would take almost 100 hours. Furthermore, with mass production, this model can generalize to any set of authored variables.

3.2 SimStudent

While the Example-Tracing behavior graph creates links from user demonstrations, the SimStudent system extends these capabilities by inducing production rule models from demonstrations and feedback (for details on this rule induction see [8]). In the experimental design tutor, SimStudent might learn a rule that sets one of the variables to an arbitrary value when no values for that variable have been assigned. Then, it might learn different rules for setting a variables second value based on whether or not it is being manipulated.

Authoring with SimStudent is similar to Example Tracing in that SimStudent asks for demonstrations when it does not know how to proceed. However, when SimStudent already has an applicable rule, it fires the rule and shows the resulting action in the tutor interface. It then asks the author for feedback on that action. If the feedback is positive, SimStudent may refine the conditions of its production rules before continuing to solve the problem. If the feedback is negative, SimStudent will try firing a different rule. When SimStudent exhausts all of its applicable rules, it asks the author to demonstrate a correct action. Figure 3 shows how SimStudent asks for demonstrations and feedback. When authoring with SimStudent, the author does not have to specify rule order - as long as a rule's conditions are satisfied, it is applicable. Authoring with SimStudent produces both a behavior graph (of the demonstrations and actions SimStudent took in the interface) and a production rule model.

To evaluate the efficiency of the SimStudent approach we constructed a complete model for the experimental design tutor. It can be difficult to determine when a SimStudent model is correct and complete from the authoring interactions alone. In most cases the SimStudent model is evaluated with set of held-out

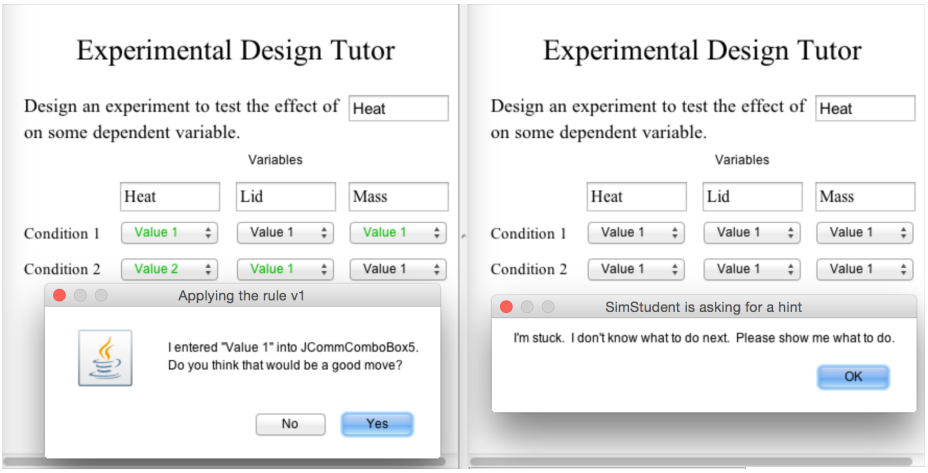

Fig. 3. SimStudent asking for feedback (left) and for a demonstration (right).

test problems (i.e., unit tests). However, in this case the learned rules were simple enough to evaluate by direct inspection. We noticed that SimStudent learned one correct strategy, but had not explored other solutions. This is typical of SimStudent - once it learns a particular strategy it applies it repeatedly. Therefore, authors must give it additional demonstrations of alternative paths. With the experimental design tutor, we noticed that SimStudent was always choosing the first value for non-manipulated variables, so we gave it additional demonstrations where non-manipulated variables took values besides those demonstrated on the initial run.

Ultimately, SimStudent acquired a complete model after 7 demonstrations and 23 feedback responses. Using the same Keystroke-Level Model from [6], we estimate that building a cognitive model using SimStudent would take an error-free expert about 2.12 minutes – much shorter than Example Tracing. Like Example Tracing, the model produced by SimStudent can work with arbitrary variables. Unlike Example Tracing, the learned model can work for unauthored variables; for example, students could define their own variables while using the tutor. This level of generality could be useful in inquiry-based learning environments [4]. Finally, if another variable column was added to the tutor, the SimStudent model would be able to function without modification. For Example Tracing, such a change would constitute a change to the behavior graph structure, so a completely new behavior graphs would need to be authored to support this addition.

4 Discussion

Both Example Tracing and SimStudent can create tutors for problems with complex solution spaces. However, our analysis shows that the two approaches

differ in terms of their efficiency and, as a result, how many solutions and paths they can handle in practice.

First, the Example-Tracing approach worked very well, even though the experimental design problems have a combinatorial structure. In particular, unordered action groups and mass production drastically reduced the number of demonstrations needed to cover the solution space, 168 vs. 40,362. The simplicity of Example Tracing combined with the power afforded by these features is likely why Example Tracing is the most widely used authoring approach today [2].

The SimStudent approach was more efficient than Example Tracing (approx. 2.12 vs. 27 minutes), but this comparison requires several caveats. The machine learning mechanisms of SimStudent generalize demonstrations and feedback into rules, which allows SimStudent to only model unique actions and the conditions under which they apply. However, this means SimStudent may not acquire a complete model. In the experimental design case study, SimStudent at first only learned that non-manipulated variables take their first value (rather than any value that is constant across conditions). In general, this problem arises when SimStudent acquires a model that can provide at least one correct solution for any problem. In these situations, it never prompts an author to provide alternative demonstrations; leading an unsuspecting author to create an incomplete model. A related complication is determining when the SimStudent model is complete. While determining the completeness of models in both Example Tracing and SimStudent can be difficult, authors must attempt to infer completeness from SimStudent's problem solving performance– a method that can be rather opaque at times. Thus, an open area for simulated learning systems is how best to evaluate the quality of learned models.

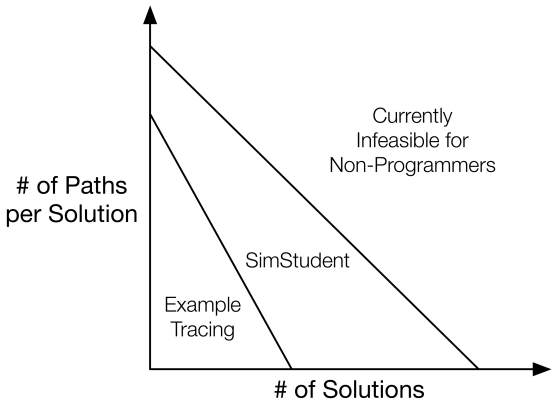

Fig. 4. How the space of solution space complexity is handled by existing non-programmer authoring approaches.

Overall our findings, when paired with those of previous work [6], suggest an interpretation depicted in Figure 4. In this figure the potential space of complexity is depicted in terms of number of unique solutions and number of paths per solution. The inner region denotes the area of the complexity space where we believe Example Tracing will maximize non-programmers' authoring utility. This region is skewed towards a higher number of paths, owing to Example Tracing's capacity to specify unordered actions. This portion of the complexity space contains many of the tutors that have already been built using Example Tracing [2]. As the complexity of a problem's solution space increases, Example Tracing becomes less practical (though still capable) and SimStudent becomes a more promising option, despite the caveats for using it. SimStudent's power of rule generalization gives it the ability to deal with more paths and unique solutions with less author effort, however, these capabilities come with the risk of producing incomplete models (without the author being aware).

Notably missing in the figure is any coverage of the upper right quadrant. This area would be a fruitful place to direct future work that supports non-programmers in authoring problems with many solutions with many paths. In particular, simulated learning systems might be extended to give non-programmers access to this portion of the space. One existing approach for dealing with highly complex solution spaces is to only model the aspects of the space that students are most likely to traverse. For example, work by Rivers and Koedinger [11] has explored the use of prior student solutions to seed a feedback model for introductory programming tasks. As it stands this area can only be reached using custom built approaches and would benefit from authoring tool research.

One limitation of our current approach is the assumption that there is a body of non-programmers that wants to build tutors for more complex problems. Our analysis here suggests that there is an open space for non-programming tools that support highly complex solution spaces, but it is less clear that authors have a desire to create tutors in this portion of the space. A survey of authors interested in building complex tutors without programming would help to shed light on what issues non-programmers are currently having in building their tutors. It is important that such a survey also include the perspective of those outside the normal ITS community to see if there are features preventing those who are interested from entering the space.

From a pedagogical point of view, it is unclear how much of the solution space needs to be modeled in a tutor. Waalkens et al. [16] have explored this topic by implementing three versions of an Algebra equation solving tutor, each with progressively more freedom in the number of paths that students can take to a correct solution. They found that the amount of freedom did not have an effect on students learning outcomes. However, there is evidence that the ability to use and decide between different strategies (i.e. solution paths) is linked with improved learning [14]. Further, subsequent work [15] has suggested that students only exhibit strategic variety if they are given problems that favor different strategies. Regardless of whether modeling the entire solution space is

pedagogically necessary, it is important that available tools support the ability to model complex spaces so that these research questions can be further explored.

5 Conclusion

The results of our analysis suggest that both the Example Tracing and SimStudent authoring approaches are promising methods for non-programmers to create tutors even for problems with many solutions with many paths. More specifically, we found that SimStudent was more efficient for authoring a tutor for experimental design, but authoring with SimStudent had a number of caveats related to ensuring that the authored model was complete. In contrast, Example Tracing was simple to use and it was clear that the authored models were complete. Overall, our analysis shows that Example Tracing is good for a wide range of problems that non-programmers might want to build tutors for (supported by its extensive use in the community [2]). However, the SimStudent approach shows great promise as an efficient authoring approach, especially when the solution space becomes complex. In any case, more research is needed to expand the frontier of non-programmers' abilities to author tutors with complex solution spaces.

Finally, this work demonstrates the feasibility and power of utilizing a simulated learning system (i.e., SimStudent) to facilitate the tutor authoring process. In particular authoring tutors with SimStudent took only 10% of the time that it took to author a tutor with Example-Tracing, a non-simulated learner approach. Educational technologies with increasingly complex solution spaces are growing in popularity (e.g. educational games and open-ended learning environments), but current approaches do not support non-programmers in authoring tutors for these technologies. Our results show that simulated learning systems are a promising tool for supporting these non-programmers. However, more work is needed to improve our understanding of how simulated learners can contribute to the authoring process and how the models learned by these systems can be evaluated.

6 Acknowledgements

We would like to thank Caitlin Tenison for her thoughtful comments and feedback on earlier drafts. 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. 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.) ITS '06. pp. 61–70. Springer (2006)
2. Aleven, V., McLaren, B.M., Sewall, J., Koedinger, K.R.: A New Paradigm for Intelligent Tutoring Systems: Example-Tracing Tutors. IJAIED 19(2), 105–154 (2009)
3. Chen, Z., Klahr, D.: All Other Things Being Equal: Acquisition and Transfer of the Control of Variables Strategy. Child Development 70(5), 1098–1120 (1999)
4. Gobert, J.D., Koedinger, K.R.: Using Model-Tracing to Conduct Performance Assessment of Students' Inquiry Skills within a Microworld. Society for Research on Educational Effectiveness (2011)
5. Klahr, D., Triona, L.M., Williams, C.: Hands on what? The relative effectiveness of physical versus virtual materials in an engineering design project by middle school children. Journal of Research in Science Teaching 44(1), 183–203 (Jan 2007)
6. 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.) ITS '14 (2014)
7. Mathan, S.A., Koedinger, K.R.: Fostering the Intelligent Novice: Learning From Errors With Metacognitive Tutoring. Educational Psychologist 40(4), 257–265 (2005), http://www.tandfonline.com/doi/abs/10.1207/s15326985ep4004_7
8. Matsuda, N., Cohen, W.W., Koedinger, K.R.: Teaching the Teacher: Tutoring SimStudent Leads to More Effective Cognitive Tutor Authoring. IJAIED 25(1), 1–34 (2014)
9. Matsuda, N., Yarzebinski, E., Keiser, V., Cohen, W.W., Koedinger, K.R.: Learning by Teaching SimStudent – An Initial Classroom Baseline Study Comparing with Cognitive Tutor. IJAIED (2011)
10. Pane, J.F., Griffin, B.A., McCaffrey, D.F., Karam, R.: Effectiveness of Cognitive Tutor Algebra I at Scale. Tech. rep., RAND Corporation, Santa Monica, CA (2013)
11. Rivers, K., Koedinger, K.R.: Automating Hint Generation with Solution Space Path Construction. In: ITS '14, pp. 329–339. Springer (2014)
12. Roll, I., Aleven, V., Koedinger, K.R.: The Invention Lab : Using a Hybrid of Model Tracing and Constraint-Based Modeling to Offer Intelligent Support in Inquiry Environments. In: ITS '10. pp. 115–124 (2010)
13. Sao Pedro, M.A., Gobert, J.D., Heffernan, N.T., Beck, J.E.: Comparing Pedagogical Approaches for Teaching the Control of Variables Strategy. In: Taatgen, N., van Rijn, H. (eds.) CogSci '09. pp. 1–6 (2009)
14. Schneider, M., Rittle-Johnson, B., Star, J.R.: Relations among conceptual knowledge, procedural knowledge, and procedural flexibility in two samples differing in prior knowledge. Developmental Psychology 47(6), 1525–1538 (2011)
15. Tenison, C., MacLellan, C.J.: Modeling Strategy Use in an Intelligent Tutoring System: Implications for Strategic Flexibility. In: ITS '14, pp. 466–475. Springer (2014)
16. Waalkens, M., Aleven, V., Taatgen, N.: Computers & Education. Computers & Education 60(1), 159–171 (Jan 2013)