In the Proceedings of the Seventh International Conference on Intelligent Tutoring Systems (ITS-2004)

Opening the Door to Non-Programmers: Authoring Intelligent Tutor Behavior by Demonstration

Kenneth R. Koedinger[1], Vincent Aleven[1], Neil Heffernan[2], Bruce McLaren[1], and Matthew Hockenberry [1]

[1] Human-Computer Interaction Institute, Carnegie Mellon University, Pgh, PA, 15213
{koedinger, aleven, bmclaren}@cs.cmu.edu , mch2@andrew.cmu.edu
[2] Computer Science Dept., Worcester Polytechnic Institute, Worcester, MA 01609-2280
nth@wpi.edu

Abstract. Intelligent tutoring systems are quite difficult and time intensive to develop. In this paper, we describe a method and set of software tools that ease the process of cognitive task analysis and tutor development by allowing the author to demonstrate, instead of programming, the behavior of an intelligent tutor. We focus on the subset of our tools that allow authors to create "Pseudo Tutors" that exhibit the behavior of intelligent tutors without requiring AI programming. Authors build user interfaces by direct manipulation and then use a Behavior Recorder tool to demonstrate alternative correct and incorrect actions. The resulting behavior graph is annotated with instructional messages and knowledge labels. We present some preliminary evidence of the effectiveness of this approach, both in terms of reduced development time and learning outcome. Pseudo Tutors have now been built for economics, analytic logic, mathematics, and language learning. Our data supports an estimate of about 25:1 ratio of development time to instruction time for Pseudo Tutors, which compares favorably to the 200:1 estimate for Intelligent Tutors, though we acknowledge and discuss limitations of such estimates.

1 Introduction

Intelligent Tutoring Systems have been successful in raising student achievement and have been disseminated widely. For instance, Cognitive Tutor Algebra is now in more than 1700 middle and high schools in the US [1] (www.carnegielearning.com). Despite this success, it is recognized that intelligent tutor development is costly and better development environments can help [2, 3]. Furthermore, well-designed development environments should not only ease implementation of tutors, but also improve the kind of cognitive task analysis and exploration of pedagogical content knowledge that has proven valuable in cognitively-based instructional design more generally [cf., 4, 5]. We have started to create a set of Cognitive Tutor Authoring Tools (CTAT) that support both objectives. In a previous paper, we discussed a number of stages of tutor development (e.g., production rule writing and debugging) and presented some preliminary evidence that the tools potentially lead to substantial

savings in the time needed to construct executable cognitive models [6]. In the current paper, we focus on the features of CTAT that allow developers to create intelligent tutor behavior without programming. We describe how these features have been used to create "Pseudo Tutors" for a variety of domains, including economics, LSAT preparation, mathematics, and language learning, and present data consistent with the hypothesis that these tools reduce the time to develop educational systems that provide intelligent tutor behavior.

A Pseudo Tutor is an educational system that emulates intelligent tutor behavior, but does so without using AI code to produce that behavior. (It would be more accurate, albeit more cumbersome, to call these "Pseudo Intelligent Tutors" to emphasize that it is the lack of an internal AI engine that makes them "pseudo," not any significant lack of intelligent behavior.) Part of our investigation in exploring the possibilities of Pseudo Tutors is to investigate the cost-benefit trade-offs in intelligent tutor development, that is, in what ways can we achieve the greatest instructional "bang" for the least development "buck." Two key features of Cognitive Tutors, and many intelligent tutoring systems more generally, are 1) helping students in constructing knowledge by getting feedback and instruction in the context of doing and 2) providing students with flexibility to explore alternative solution strategies and paths while learning by doing. Pseudo Tutors can provide these features, but with some limitations and trade-offs in development time. We describe some of these limitations and trade-offs. We also provide preliminary data on authoring of Pseudo Tutors, on student learning outcomes from Pseudo Tutors, and development time estimates as compared with estimates of full Intelligent Tutor development.

2 Pseudo Tutors Mimic Cognitive Tutors

Cognitive Tutors are a kind of "model-tracing" intelligent tutoring systems that are based on cognitive psychology theory [7], particularly the ACT-R theory [8]. Developing a Cognitive Tutor involves creating a cognitive model of student problem solving by writing production rules that characterize the variety of strategies and misconceptions students may acquire. Productions are written in a modular fashion so that they can apply to a goal and context independent of what led to that goal. Consider the following example of three productions from the domain of equation solving:

Strategy 1:	IF the goal is to solve a(bx+c) = d THEN rewrite this as bx + c = d/a
Strategy 2:	IF the goal is to solve a(bx+c) = d THEN rewrite this as abx + ac = d
Misconception:	IF the goal is to solve a(bx+c) = d THEN rewrite this as abx + c = d

The first two productions illustrate alternative correct strategies for the same goal. By representing alternative strategies, the cognitive tutor can follow different students down different problem solving paths. The third "buggy" production represents a common error students make when faced with this same goal. A Cognitive Tutor makes use of the cognitive model to follow students through their individual approaches to a problem. A technique called "model tracing" allows the tutor to provide

individualized assistance in the context of problem solving. Such assistance comes in the form of instructional message templates that are attached to the correct and buggy production rules. The cognitive model is also used to estimate students' knowledge growth across problem-solving activities using a technique known as "knowledge tracing" [9]. These estimates are used to adapt instruction to individual student needs.

The key behavioral features of Cognitive Tutors, as implemented by model tracing and knowledge tracing, are what we are trying to capture in Pseudo Tutor authoring. The Pseudo Tutor authoring process does not involve writing production rules, but instead involves demonstration of student behavior.

3 Authoring Pseudo Tutors in CTAT

Authoring a Pseudo Tutor involves several development steps that are summarized below and illustrated in more detail later in this section.

1. Create the graphical user interface (GUI) used by the student
2. Demonstrate alternative correct and incorrect solutions
3. Annotate solutions steps in the resulting "behavior graph" with hint messages, feedback messages, and labels for the associated concepts or skills
4. Inspect skill matrix and revise

The subset of the Cognitive Tutor Authoring Tools that support Pseudo Tutor development and associated cognitive task analysis are:

1. ***Tutor GUI Builder*** — used to create a graphical user interface (GUI) for student problem solving.
2. ***Behavior Recorder*** —records alternate solutions to problems as they are being demonstrated in the interface created with the Tutor GUI Builder. The author can annotate these "behavior graphs" (cf., [10]) with hints, feedback and knowledge labels. When used in "pseudo-tutor mode", the Behavior Recorder uses the annotated behavior graph to trace students' steps through a problem, analogous to the methods of model-tracing tutors.

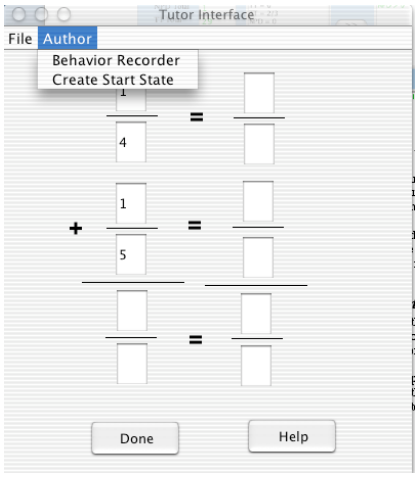

Fig. 1. The author creates the initial state of a problem that the learner will later see. Here, the initial state displays the fraction addition problem 1/4 + 1/5.

Create the Graphical User Interface. Figure 1 shows an interface for fraction addition created using a "recordable widget" palette we added to Java NetBeans, a shareware programming environment. To create this interface, the author clicks on the text field icon in the widget palette and uses the mouse to position text fields.

Typically in the tutor development process new ideas for interface design, particularly "scaffolding" techniques, may emerge (cf., [11]). The interface shown in Figure 1 provides scaffolding for converting the given fractions into equivalent fractions that have a common denominator. The GUI Builder tool can be used to create a number of kinds of scaffolding strategies for the same class of problems. For instance, story problems sometimes facilitate student performance and thus can serve as a potential scaffold. Consider this story problem: "Sue has 1/4 of a candy bar and Joe has 1/5 of a candy bar. How much of candy bar do they have altogether?" Adding such stories to the problem in Figure 1 is a minor interface change (simply add a text area widget). Another possible scaffold early in instruction is to provide students with the common denominator (e.g., 1/4 + 1/5 = __/20 + __/20). An author can create such a subgoal scaffold simply by entering the 20's before saving the problem start state. Both of these scaffolds can be easily implemented and have been shown to reduce student errors in learning fraction addition [12].

The interface widgets CTAT provides can be used to create interfaces that can scaffold a wide variety of reasoning and problem solving processes. A number of non-trivial widgets exist including a "Chooser" and "Composer" widget. The Chooser widget allows students to enter hypotheses (e.g., [13]). The Composer widget allows students to compose sentences by combining phrases from a series of menus (e.g., [14]).

Demonstrate Alternative Correct and Incorrect Solutions. Once an interface is created, the author can use it and the associated "Behavior Recorder" to author problems and demonstrate alternate solutions. Figure 1 shows the interface just after the author has entered 1, 4, 1, and 5 in the appropriate text fields. At this point, the author chooses "Create Start State" from the Author menu and begins interaction with the Behavior Recorder, shown on the left in Figure 2. After creating a problem start state, the author demonstrates alternate solutions as well as common errors that students tend to make. Each interaction with the interface (e.g., typing a

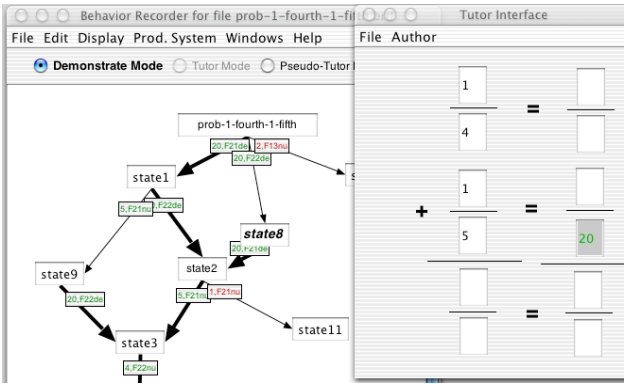

Fig. 2. The Behavior Recorder records authors' actions in any interface created with CTAT's recordable GUI widgets. The author demonstrates alternative correct and incorrect paths. Coming out of the start state (labeled "prob-1-fourth-1-fifth") are two correct paths ("20, F21den" and "20, F22den") and one incorrect path ("2, F13num"). Since *state8* is selected in the Behavior Recorder, the Tutor Interface displays that state, namely with the 20 entered in the second converted fraction.

20 in the cell to the right of the 4) produces a new action link and interface state node in the behavior graph displayed in the Behavior Recorder.

Figure 2 shows the Behavior Recorder after the author has demonstrated a complete solution and some alternatives. The link from the start state (prob-1-fourth-1-fifth) off to the left to state1 represents the action of entering 20. The links to the right from the start state represent either alternative solution strategies or common errors. Alternative solutions may involve reordering steps (e.g., putting the common denominator 20 across from the 5 before putting the 20 across from the 4), skipping steps (e.g., entering the final answer 9/20 without using the equivalent fraction scaffold on the right), or changing steps (e.g., using 40 instead of 20 as a common denominator). The common student errors shown in Figure 2 capture steps involved in adding the numerators and denominators of the given fractions without first converting them to a common denominator (e.g., entering 2/9 for 1/4 + 1/5).

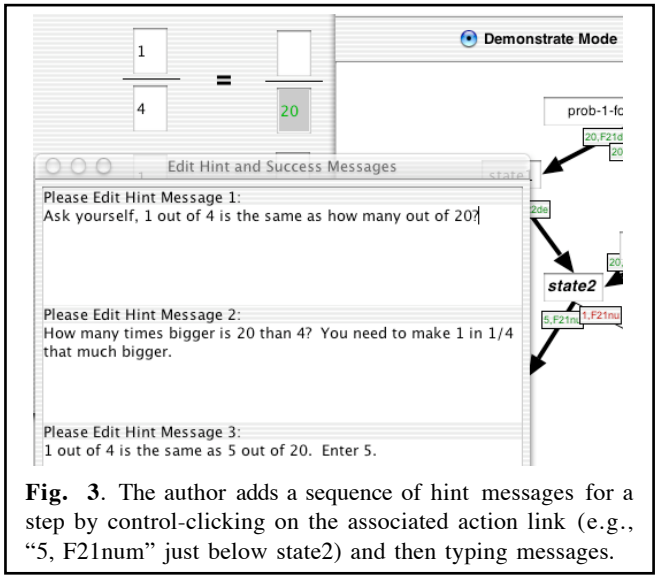

Fig. 3. The author adds a sequence of hint messages for a step by control-clicking on the associated action link (e.g., "5, F21num" just below state2) and then typing messages.

Label Behavior Graph with Hints and Feedback Messages. After demonstrating solutions, the author can annotate links on the behavior graph by adding hint messages to the correct links and error feedback messages to the incorrect links. Figure 3 shows an example of an author entering hint messages and Figure 4 shows an example of entering a feedback message. In Figure 3, the author has entered three layers of hint messages for finding the equivalent numerator in the blank cell in 1/4 = __/20.

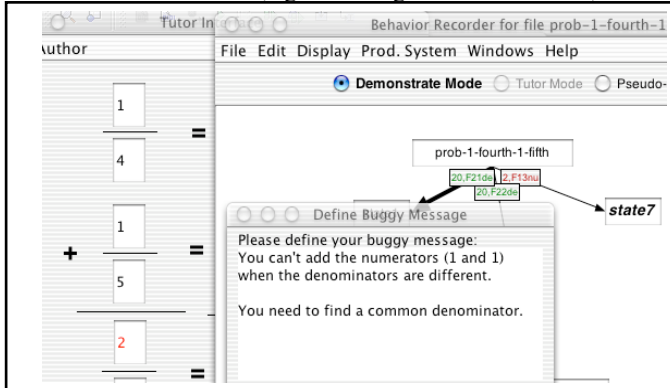

Fig. 4. The author enters an error feedback or "buggy" message by control-clicking on the link corresponding with the incorrect action (e.g.,"2, F13num" going to state7). The error of adding the numerators of fractions with unlike denominators is shown.

When a student requests a hint at this step in the resulting Pseudo Tutor, message 1 is presented and, only if further requests are made are the subsequent messages given.

When an author encounters a new step, in the same or different problem, in which a similar hint would make sense, this is a cue that that step draws on the same knowledge (concepts or skills) as the prior step. For instance, the hint sequence shown in Figure 3 can be re-used for the later step in this problem where the student needs to find the equivalent numerator to fill in the blank cell in 1/5 = __/20. The author need only substitute 5 for 4 and 4 for 5 in the message. Such similarity in the hint messages across different steps is an indication that learners can learn or use the same underlying knowledge in performing these steps. As described in the next section, the tools allow the author to annotate links in the behavior graph with knowledge labels that indicate commonalities in underlying knowledge requirements.

After demonstrating correct and incorrect solutions and adding hint and buggy messages, authors can have students use the Pseudo Tutor. The Pseudo Tutor provides feedback and context-sensitive error messages in response to students' problem-solving steps and provides context-sensitive hints at the students' request. Figure 5 shows a student receiving a hint message that may have been rewritten moments ago in response to observations of a prior learner using the Pseudo Tutor.

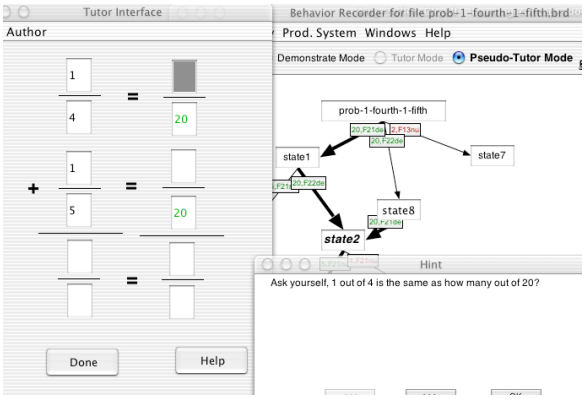

Fig. 5. The author begins testing tutor behavior by putting the Behavior Recorder in Pseudo-Tutor Mode (top right). She "plays student" by entering two correct steps (the 20's), which the Behavior Recorder has traced to follow the student to *state2*. She then clicks the Help button and the tutor highlights the relevant interface element (F21num cell) and displays the preferred hint (thicker line) out of *state2*.

Adding Knowledge Labels. Once the behavior graph has been completed the author can attach knowledge labels to links in the behavior graph to represent the knowledge behind these problem-solving steps, as illustrated in Figure 6. While these labels are referred to as "rules" in the tutor interface (reflecting a connection with the use of production rules in the ACT-R theory), the approach is neutral to the specific nature of the knowledge elements, whether they are concepts, skills, schemas, etc. One consequence of using knowledge labels is that it provides a way for the author to copy hint messages from one step to a similar step. In Figure 6 the author has labeled the step of entering 5 in 5/20 (i.e., the one between *state2* and *state3*) with *find-equivalent-numerator*. If the author believes the next step of entering the 4 in 4/20 (i.e., between *state3* and *state4*) draws upon the same knowledge, he or she can label it as *find-equivalent-numerator* as well. Doing so has the direct benefit that the hint that was written before will be

copied to this link. The author needs to make some modifications to the messages, in this case by changing 4 to 5 and 5 to 4 so that, for instance, message 1 in Figure 3 now becomes "Ask yourself, 1 out of 5 is the same as how many out of 20?" These steps of hint copying and editing push the author to make decisions about how to represent desired learner knowledge. When an author is tempted to copy a hint message from one link to another, they are implicitly hypothesizing that those links tap the same knowledge. When authors add knowledge labels to steps, they are performing cognitive task analysis. They are stating hypotheses about learning transfer and how repeated learning experiences will build on each other. Knowledge labels are also used by the Pseudo Tutor to do *knowledge tracing* whereby students' knowledge gaps can be assessed and the tutor can select subsequent activities to address those gaps. For instance, if a student is good at finding a common denominator, but is having difficulty finding equivalent fractions, the tutor can select a "scaffolded" problem, like "1/4 + 1/5 = __/20 + __/20", where the common denominator is provided and the student is focused on the *find-equivalent-numerator* steps.

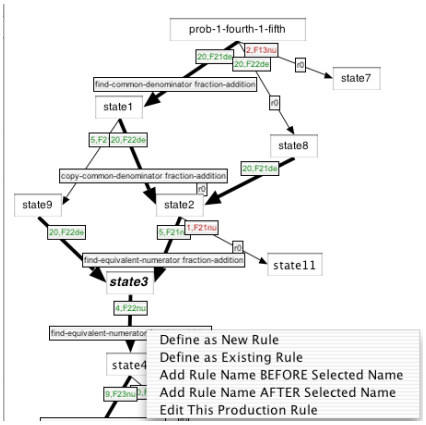

Fig. 6. The author labels links in the behavior graph to represent hypotheses about the knowledge needed to perform the corresponding step. Some steps are labeled the same, for instance, "find-equivalent-numerator" is on both the state2-state3 and state3-state4 link. The "Define as Existing Rule" option shown allows the selection of an existing label and copies associated hint messages.

Inspect Skill Matrix and Revise Tutor Design. Not only can knowledge labels be reused within problems, as illustrated above, they can also be reused across problems. Doing so facilitates the creation of a "skill matrix" as illustrated in Figure 7. The rows of the skill matrix indicate the problems the author has created and the columns are the knowledge elements required to solve each problem. The problems in the skill matrix are 1) prob-1-fourth-1-fifth described above, 2) prob-multiples is "1/3 + 1/6", 3) prob-same-denom is "2/5 + 1/5", and 4) prob-with-scaffold is "1/4 + 1/5 = __/20 + __/20" where the start state includes the common denominator already filled in. Inspecting the skill matrix, one can see how the problems grow in complexity from prob-same-denom, which only requires add-common-denominators and add-numerators, to prob-with-scaffold, which adds find-equivalent-numerators, to prob-1-fourth-1-fifth and prob-multiples which add more skills as shown in the matrix.

The skill matrix makes predictions about transfer. For instance, practice on problems like prob-with-scaffold should improve performance of the find-equivalent-numerator steps of problems like prob-1-fourth-1-fifth but should not improve performance of the find-common-denominator step. The author can reflect on the plausibility of these predictions and, better yet, use the Pseudo Tutor to collect student

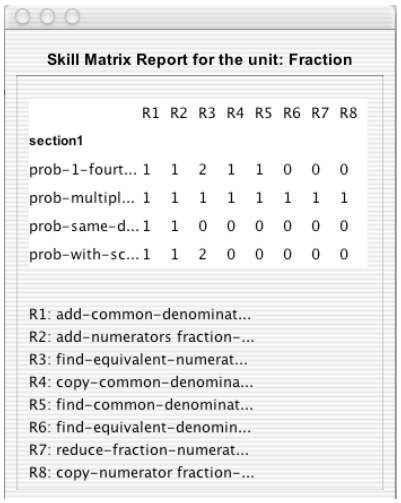

Figure 7. The skill matrix shows what knowledge elements (columns) are used in which problems (rows). For example, the 2 in the "prob-1-fourt" row means this problem requires two uses of knowledge element R3.

performance data to test these predictions. The cognitive task analysis and tutor can then be revised based on such reflection and data analysis.

4 Development Time and Use

The previous description illustrates that Pseudo Tutors are relatively easy to develop and do not require AI programming expertise. In this section we focus on the development time of Pseudo Tutors, comparing it to estimates for other types of computer-based learning environments. Estimates for the development time of intelligent tutoring systems have varied from 100-1000 hours of development per hour of instruction [2, 8, p. 254]. Estimates for the development of CAI vary even more widely [15 , p. 830]. One of our own estimates for the development of Cognitive Tutors, which comes from the initial 3-year project to create the Algebra Cognitive Tutor [16], is about 200 hours per one hour of instruction. We developed the original Cognitive Tutor Algebra in roughly 10,000 hours and this provided approximately 50 hours of instruction. While we have not, as yet, collected formal data on the development and instructional use of Pseudo Tutors, we do have some very encouraging informal data from 4 projects that have built Pseudo Tutors with our technology.

- *The Economics Project:* Part of the Open Learning Initiative (OLI) at Carnegie Mellon University, the Economics Project has a goal of supplementing an on-line introductory college-level microeconomics course with tutors.
- *The Math Assistments Project*: A four-year project funded by the Department of Education, the Assistments Project is intended to provide web-based assessments that provide instructional assistance while they assess.
- *The LSAT Project*: A small project aimed at improving the performance of students taking the law school entrance examination on analytic problems.
- *The Language Learning: Classroom Project*: Four students in a Language Technologies course at CMU used the Pseudo Tutor technology to each build two prototype Pseudo Tutors related to language learning.

In order to estimate the development time to instructional time ratio, we asked the authors on each project, after they had completed a set of Pseudo Tutors, to estimate the time spent on design and development tasks and the expected instructional time of the resulting Pseudo Tutors (see Table 1). Design time is the amount of time spent selecting and researching problems, and structuring those problems on paper. Development time is the amount of time spent with the tools, including creating a GUI, the

behavior diagrams, hints, and error messages. Instructional time is the time it would likely take a student, on average, to work through the resulting set of Pseudo Tutors. The final column is a ratio of the design and development time to instructional time for each project's Pseudo Tutors. The average Design/Development Time to Instructional Time ratio of about 23:1, though preliminary, compares favorably to the corresponding estimates for Cognitive Tutors (200:1) and other types of instructional technology given above. If this ratio stands up in a more formal evaluation, we can claim significant development savings using the Pseudo Tutor technology.

Table 1. Data on Pseudo Tutor Development and Instructional Use (in Minutes)

	# Of Pseudo Tutors	Design Time	Dev. Time	Instructional Time	Design/Dev to Instr.
Economics	11	3600	2190	180	**32.2**
Math Assistments	20	810	1170	98	**20.2**
LSAT	3	240	3000	180	**18.0**
Language Learning	8	210	575	50	**15.7**
Totals		*4860*	*6935*	*508*	***23.2***

Aside from the specific data collected in this experiment, this study also demonstrates how we are working with a variety of projects to deploy and test Pseudo Tutors. In addition to the projects mentioned above, the Pseudo Tutor authoring tools have been used in an annual summer school on Intelligent Tutoring Systems at CMU and courses at CMU and WPI. The study also illustrates the lower skill threshold needed to develop Pseudo-Tutors, compared to typical intelligent tutoring systems: None of the Pseudo Tutors mentioned were developed by experienced AI programmers. In the Language Learning Classroom Project, for instance, the students learned to build Pseudo Tutors quickly enough to make it worthwhile for a single homework assignment.

Preliminary empirical evidence for the instructional effectiveness of the Pseudo-Tutor technology comes from a small evaluation study with the LSAT Analytic Logic Tutor, involving 30 (mostly) pre-law students. A control group of 15 students was given 1 hour to work through a selection of sample problems in paper form. After 40 minutes, the correct answers were provided. The experimental group used the LSAT Analytic Logic Tutor for the same period of time. Both conditions presented the students with the same three "logic games." After their respective practice sessions, both groups were given a post-test comprised of an additional three logic games. The results indicate that students perform significantly better after using the LSAT Analytic Logic Tutor (12.1 ± 2.4 v. 10.3 ± 2.3, $t(28) = 2.06$, $p < .05$). Additionally, pre-questionnaire results indicate that neither group of students had a significant difference in relevant areas of background that influence LSAT test results. Thus, the study provides preliminary evidence that Pseudo Tutors are able to support student learning in complex tasks like analytic logic games.

5 Comparing Pseudo Tutors and full Cognitive Tutors

In principle, Pseudo Tutors and full Cognitive Tutors exhibit identical behavior in interaction with students. Both perform model tracing, provide context-sensitive instruction in the form of hints and error feedback messages, and are flexible to multiple possible solution strategies and paths. Authoring for this flexibility is different. In the case of a full Cognitive Tutor, such flexibility is modeled by a production system that generalizes across problem solving steps within and between problems. In a Pseudo Tutor, such flexibility is modeled by explicit demonstration of alternative paths in each problem. Authors face challenges in both cases. Writing production rules that work correctly across multiple situations requires significant skill and inevitable cycles of testing and debugging. On the other hand, demonstrating alternative solutions may become increasingly tedious as the number of problems increases and the complexity of alternative paths within problems increases.

To illustrate this contrast, we consider what it might take to re-implement a real Cognitive Tutor unit as a Pseudo Tutor. Consider the Angles Unit in the Geometry Cognitive Tutor [17]. This unit has about 75 problems. At first blush, the thought of developing 75 Pseudo Tutor behavior graphs may not seem daunting. It could be done by someone without AI programming expertise and might seem that it would take less time than developing the corresponding production rule model.

While alternative inputs can be handled by Pseudo Tutors, as described above, it can be time consuming to provide them, requiring separate links in the behavior diagram. For example, in the Angles unit of the Geometry Cognitive Tutor, students give reasons for their answers. Although there is always only a single correct solution for a numeric answer step, there may be different reasons for the step, at least in the more complex problems. Currently, those alternative correct reasons need to be represented with alternative links in a behavior diagram, which in itself is not a problem, except that the part of the diagram that is "downstream" from these links would have to be duplicated, leading to a potentially unwieldy diagram if there were multiple steps with alternative inputs. At minimum, a way of indicating alternative correct inputs for a given link would be useful. We are currently working on generalization features within Pseudo Tutor, one form of which is to allow authors write simple spreadsheet-like formulas to check student inputs.

While possible in principle, other behaviors are difficult in practice to replicate in Pseudo Tutors. For example, the Geometry Cognitive Tutor imposes some subtle constraints on the order in which students can go through the steps in a problem. These constraints are hard to express within Pseudo Tutors. To recreate this tutor's behavior, one would have to be able to (1) require students to complete a given answer-reason pair before moving on to the next answer-reason pair (i.e., if you give a numeric answer, the next thing you need to do is provide the corresponding reason - and vice versa) and (2) require students to complete a step only if the pre-requisites for that step have been completed (i.e., the quantities from which the step is derived). To implement these requirements with current Pseudo Tutor technology would require a huge behavior diagram. In practice, Pseudo Tutors often compromise on expressing such subtle constraints on the ordering of steps. Most of the Pseudo Tutors developed

so far have used a "commutative mode", in which the student can carry out the steps in any order. We are planning on implementing a "partial commutativity" feature, which would allow authors to express that certain groups of steps can be done in any order, whereas others need to be done in the order specified in the behavior graph.

Despite some limitations, Pseudo Tutors do seem capable of implementing useful interactions with students. As we are building more Pseudo Tutors, we are become more aware of their strengths and limitations. One might have thought that it would be an inconvenient limitation of Pseudo Tutors that the author must demonstrate all reasonable alternative paths through a problem. However, in practice, this has not been a problem. But, these questions would best be answered by re-implementing a Cognitive Tutor unit as a Pseudo Tutor. We plan to do so in the future.

6 Conclusions

We have described a method for authoring tutoring systems that exhibit intelligent behavior, but can be created without AI programming. Pseudo Tutor authoring opens the door to new developers who have limited programming skills. While the Pseudo Tutor development time estimates in Table 1 compare favorably to past estimates for intelligent tutor development, they must be considered with caution. Not only are the these estimates rough, there are differences in the quality of the tutors produced where most Pseudo Tutors to date have been ready for initial lab testing (alpha versions) and past Cognitive tutors have been ready for extensive classroom use (beta+ versions). On the other hand, our Pseudo Tutor authoring capabilities are still improving.

In addition to the goal of Pseudo Tutor authoring contributing to faster and easier creation of working tutoring systems, we also intend to encourage good design practices, like cognitive task analysis [5] and to facilitate fast prototyping of tutor design ideas that can be quickly tested in iterative development. If desired, full Intelligent Tutors can be created and it is a key goal that Pseudo Tutor creation is substantially "on path" to doing so. In other words, CTAT has been designed so that almost all of the work done in creating a Pseudo Tutor is on path to creating a Cognitive Tutor.

Pseudo Tutors can provide support for learning by doing and can also be flexible to alternative solutions. CTAT's approach to Pseudo-Tutor authoring has advantages over other authoring systems, like RIDES [3], that only allow a single solution path. Nevertheless, there are practical limits to this flexibility. Whether such limits have a significant affect on student learning or engagement is an open question. In future experiments, we will evaluate the effects of limited flexibility by contrasting student learning from a Pseudo Tutor with student learning from a full Cognitive Tutor. The Pseudo Tutor approach may be impractical for scaling to large intelligent tutoring systems where students are presented a great of number of problem variations. In full tutors, adding new problems is arguably less effort because only the machine-readable problem specification needs to be entered and the production rules take care of computing alternative solution paths. Adding new problems in Pseudo Tutors is arguably more costly because solution paths must be demonstrated anew. Future research should check these arguments and, more importantly, provide some guidance for when it might make sense to author an intelligent tutor rather than a Pseudo Tutor.

References

1. Corbett, A. T., Koedinger, K. R., & Hadley, W. H. (2001). Cognitive Tutors: From the research classroom to all classrooms. In Goodman, P. S. (Ed.) Technology Enhanced Learning: Opportunities for Change, (pp. 235-263). Mahwah, NJ: Lawrence Erlbaum.
2. Murray, T. (1999). Authoring intelligent tutoring systems: An analysis of the state of the art. International Journal of Artificial Intelligence in Education, 10, pp. 98-129.
3. Murray, T., Blessing, S., & Ainsworth, S. (Eds.) (2003). Authoring Tools for Advanced Technology Learning Environments: Towards cost-effective adaptive, interactive and intelligent educational software. Dordrecht, The Netherlands: Kluwer.
4. Lovett, M. C. (1998). Cognitive task analysis in service of intelligent tutoring system design: A case study in statistics. In Goettl, B. P., Halff, H. M., Redfield, C. L., & Shute, V. J. (Eds.) Intelligent Tutoring Systems, Proceedings of the Fourth Int'l Conference. (pp. 234-243). Lecture Notes in Comp. Science, 1452. Springer-Verlag.
5. Schraagen, J. M., Chipman, S. F., Shalin, V. L. (2000). *Cognitive Task Analysis*. Mawah, NJ: Lawrence Erlbaum Associates.
6. Koedinger, K. R., Aleven, V., & Heffernan, N. (2003). Toward a rapid development environment for Cognitive Tutors. In U. Hoppe, F. Verdejo, & J. Kay (Eds.), *Artificial Intelligence in Education, Proc. of AI-ED 2003* (pp. 455-457). Amsterdam, IOS Press.
7. Anderson, J. R., Corbett, A. T., Koedinger, K. R., & Pelletier, R. (1995). Cognitive tutors: Lessons learned. *The Journal of the Learning Sciences*, *4 (2)*, 167-207.
8. Anderson, J. R. (1993). *Rules of the Mind*. Mahwah, NJ: Lawrence Erlbaum.
9. Corbett, A.T. & Anderson, J.R. (1995). Knowledge tracing: Modeling the acquisition of procedural knowledge. *User modeling and user-adapted interaction*, *4*, 253-278.
10. Newell, A., & Simon, H. A. (1972). *Human problem solving*. Englewood Cliffs, NJ: Prentice-Hall.
11. Reiser, B. J., Tabak, I., Sandoval, W. A., Smith, B. K., Steinmuller, F., & Leone, A. J. (2001). BGuILE: Strategic and conceptual scaffolds for scientific inquiry in biology classrooms. In S. M. Carver & D. Klahr (Eds.), *Cognition and instruction: Twenty-five years of progress* (pp. 263-305). Mahwah, NJ: Erlbaum.
12. Rittle-Johnson, B. & Koedinger, K. R. (submitted). Context, concepts, and procedures: Contrasting the effects of different types of knowledge on mathematics problem solving. Submitted for peer review.
13. Lajoie, S. P., Azevedo, R., & Fleiszer, D. M. (1998). Cognitive tools for assessment and learning in a high information flow environment. *Journal of Educational Computing Research, 18*, 205-235.
14. Shute, V.J. & Glaser, R. (1990). A large-scale evaluation of an intelligent discovery world. *Interactive Learning Environments, 1*: p. 51-76.
15. Eberts, R. E. (1997). Computer-based instruction. In Helander, M. G., Landauer, T. K., & Prabhu, P. V. (Ed.s) *Handbook of Human-Computer Interaction*, (pp. 825-847). Amsterdam, The Netherlands: Elsevier Science B. V.
16. Koedinger, K. R., Anderson, J. R., Hadley, W. H., & Mark, M. A. (1997). Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education*, *8,* 30-43.
17. Aleven, V.A.W.M.M., & Koedinger, K. R. (2002). An effective metacognitive strategy: Learning by doing and explaining with a computer-based Cognitive Tutor. *Cognitive Science, 26*(2).