In the Proceedings of the 8th International Conference on Intelligent Tutoring Systems
Jhongli, Taiwan, June 26-30, 2006

The Cognitive Tutor Authoring Tools (CTAT): Preliminary Evaluation of Efficiency Gains

Vincent Aleven, Bruce M. McLaren, Jonathan Sewall, Kenneth R. Koedinger

Human-Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
aleven@cs.cmu.edu, bmclaren@cs.cmu.edu, sewall@cs.cmu.edu, koedinger@cmu.edu

Abstract Intelligent Tutoring Systems have been shown to be effective in a number of domains, but they remain hard to build, with estimates of 200-300 hours of development per hour of instruction. Two goals of the Cognitive Tutor Authoring Tools (CTAT) project are to (a) make tutor development more efficient for both programmers and non-programmers and (b) produce scientific evidence indicating which tool features lead to improved efficiency. CTAT supports development of two types of tutors, Cognitive Tutors and Example-Tracing Tutors, which represent different trade-offs in terms of ease of authoring and generality. In preliminary small-scale controlled experiments involving basic Cognitive Tutor development tasks, we found efficiency gains due to CTAT of 1.4 to 2 times faster. We expect that continued development of CTAT, informed by repeated evaluations involving increasingly complex authoring tasks, will lead to further efficiency gains.

Introduction

Intelligent Tutoring Systems can be very effective in improving student learning (e.g., [8, 18]). However, few ITSs are used regularly in real educational settings. E-Learning courses are created by the hundreds, but ITSs are seldom, if ever, seen as embedded components. A prime reason is that ITSs are typically hard to author. Estimates of development time have varied from 200-300 hours of authoring for one hour of instruction [4, 13, 19]. One way to make ITSs more widespread is to create authoring tools that speed up tutor development. A wide range of authoring tools have been built [1, 6, 13, 14, 17], and some of these have been used to build successful real-world systems [16]. Others have seen extensive evaluations focused on better understanding the authoring process and desired tool properties [1].

We report on an on-going project to create a set of authoring tools that supports the development of two types of tutors: Cognitive Tutors, which rely on a rule-based cognitive model and have been successful in improving students' math proficiency in American high schools [8], and Example-Tracing Tutors, a relatively novel type of tutors that provide the same core tutoring functionality as Cognitive Tutors but do not require any programming [6]. (Previously, these tutors were called Pseudo Tutors.) CTAT aims to increase the efficiency of authoring by means of an example-based approach to authoring. In this approach, an author demonstrates both correct and incorrect problem-solving behavior, which is recorded by the tool. The author then either generalizes and annotates the recorded examples, so they can serve as the basis for Example-Tracing Tutors, or uses them to guide the development and testing of a cognitive model for use in a Cognitive Tutor. CTAT aims to support a broad range of users: ITS/Ed Tech researchers, researchers in the field of the learning sciences interested in using ITSs as a vehicle for learning science experiments, on-line course developers, and computer-savvy college professors.

In evaluating the efficiency gains afforded by new tools, development time estimates derived from real-world tutor projects, such as those mentioned above, are helpful but are potentially subject to wide variability in terms of the experience of the developers, the subject matter for which tutors were built, and the scale of the project. Therefore, it is important that such estimates are supplemented with results from rigorous experiments that provide insight into the tool features most conducive to efficient authoring. We conducted such an experiment: a preliminary, small-scale ablation study in which we compared the authoring efficiency with the full CTAT tool suite to a version that had CTAT's novel tools taken out. To the best of our knowledge, this kind of evaluation focused on authoring efficiency has not been reported before in the ITS literature.

In this paper, which is meant to be a companion paper to an earlier paper focused on Example-Tracing Tutors [6], we present an overview of the CTAT tools used for developing Cognitive Tutors, illustrate hypothesized advantages of these tools in terms of authoring efficiency, and present the results from the small scale experiment.

Overview of CTAT

Cognitive Tutors and Example-Tracing Tutors, the two types of tutors supported by CTAT, represent different trade-offs between ease of authoring on the one hand and generality and flexibility of the resulting tutors on the other. Cognitive Tutors are rooted in the ACT-R theory of cognition and learning [4]. They interpret student problem-solving behavior using a cognitive model that captures, in the form of production rules, the skills that the student is expected to learn [8]. The tutor applies an algorithm called "model tracing" to monitor a student involved in a problem: it compares the students' actions against those that are appropriate according to the model. Developing a cognitive model for a Cognitive Tutor is a time-consuming task that requires AI programming. The upside is that the model works across a range of problems, and has flexibility, since it allows the tutor to recognize multiple student solution strategies and deal with subtle dependencies among solution steps.

Example-Tracing Tutors provide key elements of Cognitive Tutor behavior but are created "by demonstration" rather than by programming [6]. That is, an author demonstrates to the system how students are expected to solve each assigned problem and what errors they are expected to make. Compared to Cognitive Tutors, more per-problem-authoring is needed as solutions need to be demonstrated and annotated for each problem separately. However, a key advantage is that no AI programming is needed. We have seen repeatedly during workshops that people new to CTAT learn to build their first Example-Tracing Tutor in less than an afternoon.

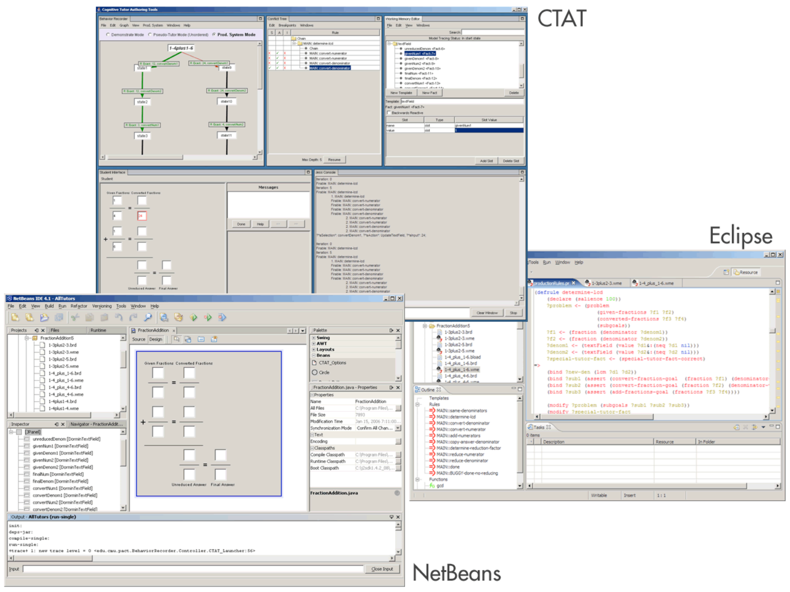


Figure 1: The Cognitive Tutor Authoring Tools

The Cognitive Tutor Authoring Tools, depicted in Figure 1, comprise three separate applications: an external GUI Builder (typically, NetBeans or Macromedia's Flash), a set of core tools for demonstration-based task analysis and for testing and debugging cognitive models, and an external editor for cognitive models (typically Eclipse). The following tools are used prominently when authoring Cognitive Tutors:

GUI Builder – used to create a **Student Interface,** a problem-solving environment in which the student interacts with the tutor. The GUI Builder supports interface building without programming: the author arranges interface widgets on a canvas by drag and drop techniques (see Figure 1, bottom left). The GUI Builder is an external, off-the-shelf tool enriched with "tutorable" widgets developed for CTAT. We have used both Macromedia's Flash and Java development environments such as NetBeans and IntelliJ. In projects focused on providing tutoring within an existing simulator or problem-solving environment, the Student Interface is replaced with the external environment, which typically can be done without extensive effort [3, 12]. The use of a GUI Builder in an ITS authoring tool is not novel, but to the best of our knowledge, it is novel to have plug-and-play compatibility with standard GUI Builders.

Behavior Recorder – a central tool with three key functions. First, it records examples of correct and incorrect behavior demonstrated by the author, in the Student Interface, in the form of a Behavior Graph. Second, it implements the example-tracing function. Third, it provides support for planning and testing of cognitive models.

Working Memory Editor – used for cognitive model development; allows an author to inspect and modify the contents of the cognitive model's "working memory," which is frequently needed during model development. The Jess rule engine we use does not itself come with such an editor.

Conflict Tree and Why Not Window – tools for debugging the cognitive model, which provide information about rule activations and partial activations explored by the model-tracing algorithm. The Conflict Tree is specific to model tracing, but the Why Not Window is useful for general production rule programming. However, we know of no existing production rule system that offers a tool like this.

Jess console – enables an author to interact directly with the Jess interpreter via the command line, which is helpful to carry out debugging strategies not directly supported by CTAT. It is similar to simple Jess tools such as JessWin.

External editor–used to edit the Jess rules for the cognitive model. The Jess plug-in for Eclipse (shown on the right in Figure 1) provides syntax checking and auto-completion features. Other editors can be used, since no tight link exists with CTAT.

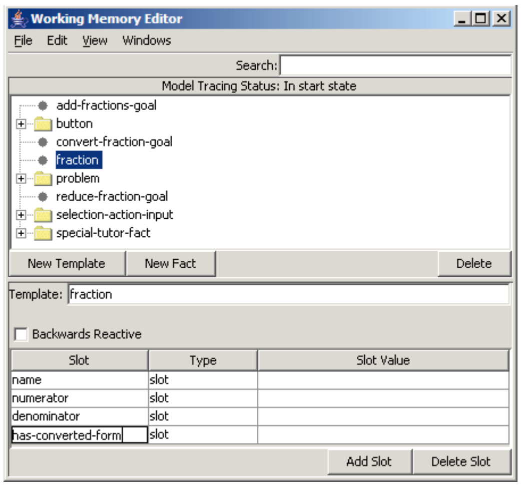
Hypothesized advantages of authoring with CTAT

Using fraction addition as an example domain, we illustrate four hypotheses about how CTAT facilitates cognitive model creation:

1. the Behavior Recorder supports the planning of a cognitive model;
2. CTAT auto-generates working memory content to facilitate modeling;
3. the Behavior Recorder facilitates testing, by providing automatic snap shots of all problem states and a facility for regression testing;
4. the Conflict Tree and Why Not Window facilitate error localization.

Prior to developing a cognitive model, the author creates a Student Interface suitable for solving fraction addition problems, using the GUI Builder shown at the bottom left of Figure 1. She also creates worked-out example problems with the Behavior Recorder. The examples will guide the model development efforts and serve as test cases. For example, our author may demonstrate two ways of solving the fraction addition problem 1/4 + 1/6: by converting the fractions to denominator 12, or by converting them to 24. Both are acceptable strategies that students are likely to employ. As the author demonstrates the steps, the Behavior Recorder records them in a "Behavior Graph," shown at the top left in Figure 1, with separate paths corresponding to each strategy. The author then labels each step in the recorded problem solutions with names for relevant skills. This activity is a form of cognitive task analysis, because the author determines how the overall problem-solving skill breaks down into smaller components. At the same time, it is a way of planning the cognitive model, since the author will later create production rules corresponding to each identified skill. This planning step could be done on paper, but doing it with CTAT has the advantage that the demonstrated examples are more likely to be complete and can later serve as semi-automated test cases for the cognitive model.

Having created a Student Interface and annotated examples, the next step for the author is to create a working memory representation for the cognitive model. In Jess, working memory is a collection of "facts" whose attributes (or "slots" in Jess terminology) must first be declared by means of "templates." CTAT helps by creating initial working memory content for the author. The structure generated by CTAT mirrors the Student Interface – it contains a fact for each element (i.e., widget) in the interface. This organization is useful in particular when the external representation of a problem (as captured in the interface) reflects its internal structure. Even if the



Model Tracing Status: In start state

Template: fraction

☐ Backwards Reactive

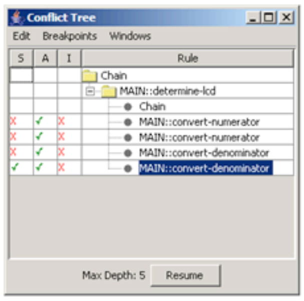
Slot	Type	Slot Value
name	slot	
numerator	slot	
denominator	slot	
has-converted-form	slot	

Figure 2: Augmenting Working Memory using CTAT's Working Memory Editor

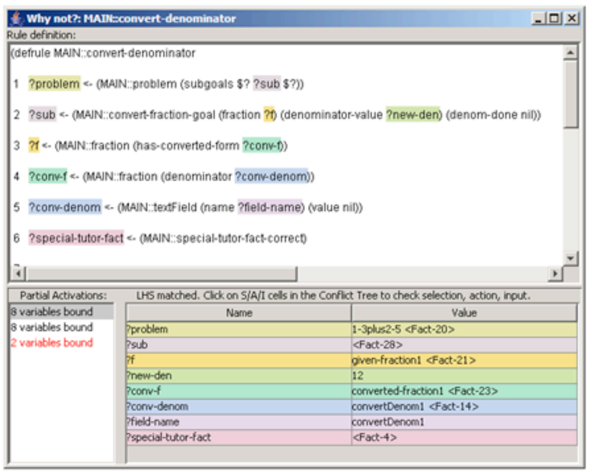
interface elements do not fully reflect the internal problem structure, facts corresponding to interface elements are still useful for model tracing. In the fraction addition example, the representation generated by CTAT contains one working memory fact per text field in the interface. This representation is a start, but the author must add the information that each text field represents a particular part of a particular fraction (numerator, denominator) and must also represent the role in the problem that each fraction plays (e.g., given fractions, converted fractions, sum fractions). In the process, she might create a new template to represent fractions, with slots for the numerator and denominator, using CTAT's Working Memory Editor (see Figure 2).

Next, the author needs to write the production rules for each skill in the problem. The actual editing of the production rules is done with a standard editor such as Eclipse. We plan to make the writing of production rules easier by means of structured editing techniques and automated rule stub generation – we implemented some of these facilities for the TDK production rule language [7] but not yet for Jess. We are also working on a machine learning approach to rule creation [9].

When it comes time to test the production rules, the Behavior Recorder is helpful in two ways. First, the Behavior Recorder essentially provides automated snapshots for all recorded problem states. That is, it can be used to move working memory and the Student Interface to any state recorded in the Behavior Graph, just by clicking on the state. This capability makes it easier to test rules involved in a particular step in the problem, since it saves the author from having to provide input in the Student Interface for all previous steps. This kind of manual input would be time-consuming, especially for more complex problems with many steps, and especially considering that it needs to be done for every edit-test-debug cycle. In addition, the Behavior Recorder supports semi-automated regression testing in which the cognitive model is tested against a full Behavior Graph as test case (i.e., as a specification of how the model should behave on the steps of the given problem). CTAT indicates by means of color coding whether the tutor (applying its model-tracing algorithm) produces the



S	A	I	Rule
			Chain
			MAIN::determine-lcd
			Chain
X	✓	X	MAIN::convert-numerator
X	✓	X	MAIN::convert-numerator
X	✓	X	MAIN::convert-denominator
✓	✓	X	MAIN::convert-denominator



```
(defrule MAIN::convert-denominator

1  ?problem <- (MAIN::problem (subgoals $? ?sub $?))

2  ?sub <- (MAIN::convert-fraction-goal (fraction ?f) (denominator-value ?new-den) (denom-done nil))

3  ?f <- (MAIN::fraction (has-converted-form ?conv-f))

4  ?conv-f <- (MAIN::fraction (denominator ?conv-denom))

5  ?conv-denom <- (MAIN::textField (name ?field-name) (value nil))

6  ?special-tutor-fact <- (MAIN::special-tutor-fact-correct)
```

8 variables bound
8 variables bound
2 variables bound

LHS matched. Click on S/A/I cells in the Conflict Tree to check selection, action, input.

Name	Value
?problem	1-3plus2-5 <Fact-20>
?sub	<Fact-28>
?f	given-fraction1 <Fact-21>
?new-den	12
?conv-f	converted-fraction1 <Fact-23>
?conv-denom	convertDenom1 <Fact-14>
?field-name	convertDenom1
?special-tutor-fact	<Fact-4>

Figure 3: The Conflict Tree and Why Not Window: debugging tools

expected result for each link in the graph. If not, then typically the cognitive model is to blame; one or more rules are not yet working as intended. This kind of testing is especially useful when rules that have been authored for an earlier tutor problem need to be modified for a later problem; such modifications sometimes introduce errors in problems on which previously the rule worked correctly.

To help localize errors in a cognitive model, the Conflict Tree window shows the space of rule activations explored in the process of model tracing (shown on the left in Figure 3). When a student submits a problem-solving step to the tutor, the model-tracing algorithm searches for a sequence of rule activations that produce the same action as the student. Showing the search space graphically, as a tree of rule activations, helps an author fully understand the model's behavior, which is often useful for debugging or for getting to know models built by others. The Why Not window (shown on the right in Figure 3) provides further detail about each search node depicted in the Conflict Tree. It shows both full and partial rule activations that were generated at any given node. This information is useful particularly when a rule that was expected to fire did not – hence the name "Why Not". Experienced modelers typically like these tools. Without them, an author would have to use the Jess command-line interface to extract the information in the Conflict Tree in piecemeal fashion. It is not possible to extract the information that is presented in the Why Not window using the Jess command-line. An author would have to resort to inserting print statements in rule conditions or other cleverness to infer this information.

Preliminary evaluation of efficiency gains

We conducted a small-scale experiment to test if the novel tools in CTAT lead to more efficient tutor development. The experiment was an ablation study, in which we compared the full CTAT suite ("Full Tool Set") to a version in which the novel tools

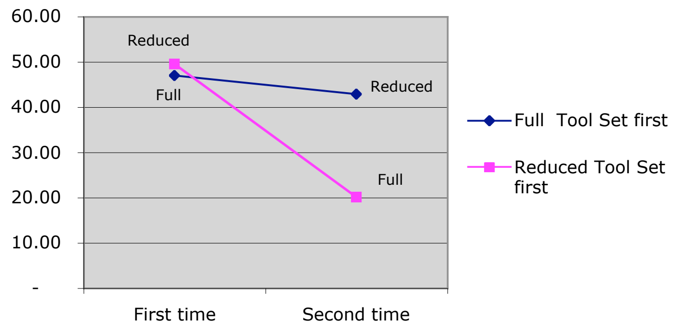


Figure 4: Comparison of the time to author production rules with CTAT versus an ablated version of CTAT from which its novel tools were removed

were removed, namely, the Behavior Recorder, the Conflict Tree, the Why Not window, and the Working Memory Editor. This "Reduced Tool Set" is essentially a standard Jess environment, akin for example to JessWin, but augmented with the model-tracing algorithm and a fully-integrated Student Interface. Thus, the control condition in this experiment begins at a point far ahead of an author implementing a model-tracing tutor from scratch. The goal of the experiment was to get an indication of CTAT's efficiency and to identify areas for improvement. In particular, we wanted to see whether the last three of the hypothesized advantages of CTAT would materialize. Due to the small number of subjects, the experiment does not allow for statistically significant results. Nonetheless, it is valuable as a formative evaluation.

The experiment focused on a simple, semi-realistic modeling activity, in which the participants were asked to create a cognitive model consisting of 6 rules, given detailed statements of the form of "If … then … " that mapped quite directly onto the rule conditions and actions to be implemented. Four subjects participated, all of whom were students at CMU who had used CTAT for a class project. The subjects thus had some experience but were not expert cognitive modelers. All subjects did the modeling task twice, two of them first with the Full Tool Set and then again, approximately a week later, with the Reduced Tool Set, the other two with the order reversed. Each task lasted 3.5 hours at most – less if the subject finished before that amount of time had elapsed. We measured the number of rules completed.

As shown in Figure 4, the decrease in time per rule was greater when the subjects switched from the Reduced Tool Set to the Full Tool Set than when they used the tool sets in the opposite order. Thus, the increase in efficiency between the first and second time the task was performed is not likely to be due solely to the subjects' greater familiarity with the task. There is an effect of the tools that suggests that the Full Tool Set improves the efficiency of modeling. Overall, the time per rule with the Full Tool Set was a factor of 1.4 faster than with the Reduced Tools. That effect is less than the 2x improvement that we reported previously for an experiment with a single participant [7]. That earlier experiment involved a similar ablation design as the current but involved CTAT tools that support modeling not in Jess, but in TDK, the modeling language used to develop the Algebra and Geometry Cognitive Tutors.

In the earlier experiment, the difference between the Full and Reduced Tool Sets was greater, since the CTAT tools to support TDK had features that have not yet been implemented in the CTAT/Jess tools, which may explain the higher efficiency gain.

In addition to evaluating the overall efficiency of rule creation, we undertook a detailed quantitative analysis of subjects' actions with the tools, recorded with Camtasia, to evaluate CTAT's hypothesized efficiency advantages. The experiment seemed to confirm that the auto-generation of working memory saves time, as measured by the amount of time spent editing the working memory content (7 mins on average with the Full Set, 14 mins with the Reduced Set). The time savings were very modest, but it is worth pointing out that the measure used does not include any savings in development time that may have resulted from starting with a solid working memory representation. In other words, the auto-generation may provide useful scaffolding for inexperienced tool users that is hard to quantify – and this may indeed be its main advantage. The results were more surprising with respect to the hypothesized time savings due to the Behavior Recorder's automated snap shot facility. Even though all subjects knew about this facility, three of the four hardly used it, with one subject accounting for almost 60% of its use. The limited use may reflect the simplicity of the modeling task that the subjects were given (a six-step problem), or it may be that when testing a cognitive model, it is more natural to work in the Student Interface than in the Behavior Graph. In the Student Interface the details of the current problem state are always clearly visible, whereas in the Behavior Graph only non-descriptive names for the states are shown, without details, which may hamper the intended nimble navigation among problem states. If the author's attention is naturally anchored in the Student Interface, then it makes sense that the commands for navigating the Behavior Graph should be issued from this window, for example, by means of arrow keys or "bookmarks". This solution would, we expect, retain any efficiency advantages due to the snapshots. Finally, there was evidence that the debugging tools (the Conflict Tree and the Why Not Window) were useful. The subjects used the Why Not window regularly (31.25 times on average). We do not know how often they used the Conflict Tree, since the tool is usually visible without requiring interaction by the author. Importantly, there was evidence of a higher number of edit-test-debug cycles with the Reduced Tool Set. While the time spent editing rules was about the same in each condition, the number of editing episodes was higher (81 v. 59 on average) with the Reduced Tool Set. Further, the subjects using the Reduced Tool Set spent more time testing, as distinct from debugging (26 mins v. 16 mins), and had many more testing actions (164 v. 93). While we cannot attribute these numbers solely to any greater diagnostic power of the CTAT debugging tools, they are certainly consistent with the notion that with better debugging tools one needs fewer edit-test-debug cycles.

In spite of the modest scale of the experiment, analyses such as those presented above are very useful in guiding future tool redesign and development efforts. They underscore the importance of getting the HCI right in designing interactive tools and lead to specific suggestions for improvement. Further qualitative analysis of the errors in the subjects' production rules will also help in that regard. In interpreting the efficiency results, it is important to keep in mind that the modeling tasks in these experiments were simple. The task involved only 6 straightforward rules of which detailed English versions were given, and thus was significantly less complex than a typical real-world modeling task. Viewed in that light, a 1.4-2x gain in efficiency is encouraging, even if our eventual goal is to achieve higher gains.

Conclusion

In CTAT, the authoring of Example-Tracing Tutors and Cognitive Tutors is organized around examples of demonstrated behavior. These examples include alternative strategies for solving problems and errors students are expected to make, and can be recorded conveniently with CTAT's Behavior Recorder tool. They can be used as the basis for Example-Tracing Tutors to provide guidance to students. The examples can also be used as planning cases and semi-automatic test cases for cognitive models, if an author is developing a Cognitive Tutor.

The preliminary experiment described in this paper suggests that authoring with CTAT is becoming more rapid. Holding ourselves to a high scientific standard of rigorous laboratory experimentation with a high-bar control condition (all but the newest features), we have shown a modest efficiency improvement estimate of 1.4 to 2 times faster, compared to standard tools for model tracing. We are aiming for higher overall efficiency gains, but it is nonetheless encouraging that a speed-up was attained on a small and easy task. An interesting finding was that CTAT seems to lower the number of edit-test-debug cycles needed to create a cognitive model. The preliminary experiment led to a number of ideas for tool improvement, focused on improving the HCI of the tools. To improve the efficiency of the tools, we are also developing semi-automated techniques, such as machine learning [9] and bootstrapping [10]. We expect that the advantages of CTAT will be more pronounced in a more complex modeling task and as we continue to improve CTAT.

So far, CTAT has been used by over 220 users in a number of workshops, graduate courses, summer schools, and tutorials. We estimate that 30-40% of these users were non-CMU people. CTAT is being used to develop a set of tutors for introductory college-level genetics (http://www.cs.cmu.edu/~genetics), which have been piloted in various colleges across the country, and has been used in learning science experiments in the domains of thermodynamics [3], stoichiometry [11], French culture [15], and Chinese character recognition. Clearly, these numbers indicate that there is a need for authoring tools and that CTAT is offering useful functionality that can be applied in a range of domains. CTAT is available free of charge for research and educational purposes (http://ctat.pact.cs.cmu.edu). It is our hope that through tool development and other efforts by the ITS community, intelligent tutoring systems will become more widespread and will one day be staples of on-line courses.

Acknowledgments

Brett Leber, Grace Lee Leonard, and the anonymous reviewers helped make this a better paper. The research is sponsored by ONR Award N000140410107, by NSF Award SBE0354420 to the Pittsburgh Sciences of Learning Center, and by a grant from the Grable Foundation. Their contributions are gratefully acknowledged. The views expressed in the paper may not be the official views of the funding agencies.

References

1. Ainsworth, S.E & Fleming, P.F. (2005) Evaluating a mixed-initiative authoring environment: is redeem for real? In *Proceedings of the 12th International Conference on Artificial Intelligence in Education* (pp. 9-16). Amsterdam, IOS Press.
2. Aleven, V., & Koedinger, K. (2002). An effective metacognitive strategy: learning by doing and explaining with a computer-based Cognitive Tutor. *Cognitive Science, 26*, 147-179.
3. Aleven, V., & Rosé, C. (2005). Authoring plug-in tutor agents by demonstration: Rapid, rapid tutor development. In *Proceedings of the 12th International Conference on Artificial Intelligence in Education, AIED 2005* (pp. 735-737). Amsterdam: IOS Press. (Poster).
4. Anderson, J. R. (1993). *Rules of the Mind*. Hillsdale, NJ: Erlbaum.
5. Friedman-Hill, E. (2003). *Jess in Action*. Manning Publications Co.
6. Koedinger, K. R., Aleven, V., Heffernan, N., McLaren, B., & Hockenberry, M. (2004). Opening the door to non-programmers: authoring intelligent tutor behavior by demonstration. In *Proceedings of Seventh International Conference on Intelligent Tutoring Systems, ITS 2004* (pp. 162-174). Berlin: Springer Verlag.
7. Koedinger, K. R., Aleven, V. & Heffernan, N. T. (2003). Toward a rapid development environment for Cognitive Tutors. In *Proceedings of the 11th International Conference on Artificial Intelligence in Education, AI-ED 2003* (pp. 455-457). Amsterdam: IOS Press.
8. 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.
9. Matsuda, N., Cohen, W. W., & Koedinger, K. R. (2005). Applying programming by demonstration in an intelligent authoring tool for Cognitive Tutors. In AAAI Workshop on Human Comprehensible Machine Learning (pp. 1-8). Menlo Park, CA: AAAI Association.
10. McLaren, B. M., Bollen, L., Walker, E., Harrer, A., & Sewall, J. (2005). Cognitive Tutoring of collaboration: developmental and empirical steps toward realization, *In Proceedings of the Conference on Computer Supported Collaborative Learning (CSCL-05),* Taipei, Taiwan.
11. McLaren, B. M., Lim, S., Gagnon, F., Yaron, D., & Koedinger, K. R. (in press). Studying the effects of personalized language and worked examples in the context of a web-based intelligent tutor. 8th International Conference on Intelligent Tutoring Systems, ITS 2006.
12. Murray, T. (1999). Authoring intelligent tutoring systems: An analysis of the state of the art. *International Journal of Artificial Intelligence in Education, 10,* 98-129.
13. Murray, T., Blessing, S., & Ainsworth S.E. (Eds.) (2003). *Tools for Advanced Technology Learning Environments*. Amsterdam: Kluwer Academic Publishers.
14. Nuzzo-Jones, G., Walonoski, J.A., Heffernan, N.T., Livak, T. (2005). The eXtensible Tutor Architecture: A new foundation for ITS. In *Proceedings of the 12th Artificial Intelligence In Education* (pp. 902-904). Amsterdam: IOS Press.
15. Ogan, A., Aleven, V., & Jones, C. (2005). Improving intercultural competence by predicting in French film. In *Proceedings of E-Learn 2005*. Norfolk, VA: AACE.
16. Razzaq, L., Feng, M., Nuzzo-Jones, G., Heffernan, N.T., Koedinger, K. R., et al. (2005). The Assistment Project: Blending Assessment and Assisting. In *Proceedings of the 12th Artificial Intelligence In Education* (pp. 555-562). Amsterdam: IOS Press.
17. Suraweera, P., Mitrovic, A. & Martin, B. (2005). A Knowledge Acquisition System for Constraint-based Intelligent Tutoring Systems. In *Proceedings of the 12th international conference on Artificial Intelligence in Education,* (pp. 638-645). Amsterdam, IOS Press.
18. VanLehn, K., Lynch, C., Schulze, K., Shapiro, J. A., Shelby, R., et al. (2005). The Andes physics tutoring system: five years of evaluations. In *Proceedings of the 12th international conference on Artificial Intelligence in Education*. Amsterdam, IOS Press.
19. Woolf, B. P., & Cunningham, P. (1987). Building a community memory for intelligent tutoring systems. AAAI 1987 (pp. 82-89).