Aleven, V., McLaren, B.M., Sewall, J., & Koedinger, K.R. (2009). A new paradigm for intelligent tutoring systems: Example-tracing tutors. *International Journal of Artificial Intelligence in Education*, 19(2), 105-154.

A New Paradigm for Intelligent Tutoring Systems: Example-Tracing Tutors

Vincent Aleven, Bruce M. McLaren, Jonathan Sewall, Kenneth R. Koedinger
Human-Computer Interaction Institute, Carnegie Mellon University, Pittsburgh, PA, US
{aleven, bmclaren, sewall}@cs.cmu.edu, koedinger@cmu.edu

Abstract. The Cognitive Tutor Authoring Tools (CTAT) support creation of a novel type of tutors called example-tracing tutors. Unlike other types of ITSs (e.g., model-tracing tutors, constraint-based tutors), example-tracing tutors evaluate student behavior by flexibly comparing it against generalized examples of problem-solving behavior. Example-tracing tutors are capable of sophisticated tutoring behaviors; they provide step-by-step guidance on complex problems while recognizing multiple student strategies and (where needed) maintaining multiple interpretations of student behavior. They therefore go well beyond VanLehn's (2006) minimum criterion for ITS status, namely, that the system has an inner loop (i.e., provides within-problem guidance, not just end-of-problem feedback).

Using CTAT, example-tracing tutors can be created without programming. An author creates a tutor interface through drag-and-drop techniques, and then demonstrates the problem-solving behaviors to be tutored. These behaviors are recorded in a "behavior graph," which can be easily edited and generalized. Compared to other approaches to programming by demonstration for ITS development, CTAT implements a simpler method (no machine learning is used) that is currently more pragmatic and proven for widespread, real-world use by non-programmers.

Development time estimates from a large number of real-world ITS projects that have used CTAT suggest that example-tracing tutors reduce development cost by a factor of 4 to 8, compared to "historical" estimates of ITS development time and cost. The main contributions of the work are a novel ITS technology, based on the use of generalized behavioral examples to guide students in problem-solving exercises, as well as a suite of mature and robust tools for efficiently building real-world ITSs without programming.

Keywords. authoring tools, example-tracing tutors, cognitive tutors, ITS architectures, behavior of tutoring systems, programming by demonstration

INTRODUCTION

Intelligent tutoring systems (ITSs) have been shown to lead to impressive improvement in student learning in a range of domains, using a variety of different approaches (Beal, Walles, Arroyo, & Woolf, 2007; Graesser, Chipman, Haynes, & Olney, 2005; Koedinger, Anderson, Hadley, & Mark, 1997; Martin & Mitrovic, 2002; Mostow & Beck, 2007; Rickel & Johnson, 1999; VanLehn, et al., 2005; Mitrovic, et al., 2008). However, they have traditionally been difficult to build (e.g., Murray, 2003; Anderson, 1993). Many ideas have been offered to make ITSs easier to build: ITS design patterns (Harrer, Pinkwart, McLaren, & Scheuer, 2008; Devedzic & Harrer, 2005), reusable components and learning objects (Koedinger, Suthers, & Forbus, 1999; Ritter, Blessing, & Wheeler, 2003; Ritter & Koedinger, 1997), community authoring (Aleahmad, Aleven, & Kraut, 2008), use of off-the-shelf tools as an integrated part of the ITS development process (e.g., Aleven, Sewall, McLaren, & Koedinger, 2006), and authoring tools (e.g., Murray, Blessing & Ainsworth, 2003). In our

view, all of these developments and approaches are likely to contribute to making ITS technology widespread in the years ahead. We expect however that authoring tools will turn out to be the centerpiece of that endeavor.

Many ITS authoring systems have been developed, over 25 by most counts, since the earliest days of intelligent tutoring systems (Murray, 2003). Typically, each authoring tool focuses on a particular kind of ITS, such as constraint-based tutors (Mitrovic et al., 2006) or model-tracing tutors (Blessing, Gilbert, Ourada, & Ritter, 2007). In addition to trying to make tutors easier to build, there have been a variety of objectives behind ITS authoring systems, including helping authors transfer knowledge more accurately into the resulting tutoring system, supporting good (and consistent) design principles, and enabling rapid prototyping of ITSs. Different tools have focused on different aspects of tutoring, such as support of different tutoring strategies (Ainsworth et al., 2003), tutoring within a simulation context (Munro, 2003), supporting multiple knowledge types (Halff et al., 2003), and use of hypermedia (Brusilovsky, 2003).

Our holy grail is to create cost-effective tools that *non-programmers* can use to create and deliver sophisticated tutors for real-world use, a goal we have taken very significant steps toward achieving. We have come to believe that efficient authoring tools for non-programmers are an important condition for making ITSs widespread. Making ITS development possible for non-programmers helps bring down the development cost, makes it easier for users to get started with the tools (important to ensure adoption of the tool by companies and other organizations who create e-learning and other types of curricular materials), and helps broaden the potential user base. In particular, professional instructional designers and developers are typically computer- and multimedia-savvy but tend not to be programmers, and thus are much more likely to start using an ITS authoring tool if the tool does not require programming.

We report on our 7-plus years of experience with a suite of authoring tools called the *Cognitive Tutor Authoring Tools* (CTAT). In these 7 years, CTAT has grown into a mature set of authoring tools that are relatively easy to learn and that have been effective in the hands of a wide range of authors. Over 400 authors have used CTAT to build a diverse set of tutors for various purposes (see for example Appendix A or the many figures in this paper). A substantial number of these tutors have been used in real educational settings (i.e., are not demo or laboratory systems). CTAT-built tutors have been used in as many as 26 research studies, most of them carried out in actual classrooms or courses. In a number of these projects, the main authors who used CTAT to create tutors were non-programmers. Our own experience is not just as tool developers, but also as users of our own tools, as consultants to learning science researchers who build tutors as a platform for learning science experiments, as consultants to developers of tutors for real-world use, as instructors of students learning about intelligent tutoring systems through practice with our tools, and, finally, as academic advisors to graduate students using the tools for their research projects. As much as possible, in extending CTAT, we were driven by the needs and requirements of authors who were using the tools for real-world projects. It is our strong belief that this *use-driven* design approach has contributed greatly to the successful transition from a proof-of-principle system to real-world tools.

Although CTAT is not the only current authoring tool that non-programmers can use to develop and deploy real-world intelligent tutoring systems, CTAT is perhaps furthest down this path. The Assistments authoring tool (Razzaq et al., 2005; 2009) and REDEEM (Ainsworth et al., 2003) also support tutor authoring without programming, but of simpler tutors than CTAT. Further, using ASPIRE (Mitrovic et al., 2006; this issue), constraint-based tutors can be built without programming, including a text-based user interface, a domain ontology, and a set of constraints used to evaluate

student solutions, generated from author-provided solution examples using machine learning techniques. A study showed that ASPIRE's induction methods could reproduce 90% of the constraint base used in existing constraint-based tutors (Mitrovic et al., 2006; this issue). CTAT supports different types of tutors, as discussed below, and provides an author with more advanced options for building tutor interfaces without needing to do any programming.

CTAT supports development of two types of ITSs, Cognitive Tutors (Anderson, Corbett, Koedinger, & Pelletier, 1995; Koedinger & Aleven, 2007; Koedinger & Corbett, 2006) and a newer type of tutors called *example-tracing tutors* (Koedinger, Aleven, Heffernan, McLaren, & Hockenberry, 2004). Cognitive Tutors rely on cognitive theory and cognitive modeling; they require sophisticated AI programming skills to develop. They have a long history and are being used extensively in U.S. schools (in more than 2,600 of them, at the time of this writing). Cognitive Tutors for high-school mathematics have proven to improve student learning in many studies (Koedinger & Aleven, 2007; Koedinger & Corbett, 2006).[1] Example-tracing tutors, the second type of tutors built with CTAT, are often (though not always) behaviorally indistinguishable from Cognitive Tutors, but are much easier to build than Cognitive Tutors. They can be built without programming. In this paper, we focus on example-tracing tutors.

The main idea behind example-tracing tutors is straightforward. Whereas model-tracing tutors, Cognitive Tutors included (Koedinger et al., 1997; VanLehn et al., 2005), interpret and assess student behavior with reference to a *cognitive model* that can solve problems in the way that competent students can, and whereas constraint-based tutors (Mitrovic & Ohlsson, 1999) interpret and assess student work with respect to *a set of constraints* that all student problem solutions should satisfy, example-tracing tutors interpret and assess student behavior with reference to *generalized examples* of problem-solving behavior. Such generalized examples tend to be easier to create than general production rules or constraints. Performing a task, even with the requirement that multiple solution paths must be captured, tends to be much easier than formally specifying procedures or constraints. A key novel aspect of the work presented here, therefore, is a tutoring technology that relies on generalized examples to provide guidance during problem-solving practice. The use of examples in CTAT tutors is quite different from that in other ITSs that use examples, including systems that use case-based reasoning techniques for purposes of student modeling (e.g., Weber & Brusilovsky, 2001), or systems that present examples to students (e.g., Conati & VanLehn, 2000; McLaren, Lim & Koedinger, 2008a; 2008b; Salden, Aleven, Renkl, & Schwonke, 2009; Schwonke et al., 2009).

The generalized examples used in example-tracing tutors comprise acceptable solution paths for a given tutor problem, that is, they are "behavior graphs" in the sense of Newell and Simon (1972). They are created without programming, by (a) creating a tutor interface for the targeted problem type through drag-and-drop techniques, (b) demonstrating, within the tutor interface, how problems can be solved, and (c) editing, annotating, and generalizing the resulting behavior graph with a graphical tool (called the "Behavior Recorder"). Example-tracing tutors were originally conceived as a tool for doing cognitive task analysis and for rapidly prototyping tutor behavior en route to developing a Cognitive Tutor (and they are still useful for these purposes). However, a number of extensions to the basic idea have transformed it into a viable methodology for building tutors in its own right. In particular, we added a number of techniques for *generalizing* the behavioral examples used by the tutor, so that a tutor can recognize a wider range of correct student behavior than only the solution steps that the

[1] The Cognitive Tutors for high-school and middle-school mathematics that were built in our lab pre-date CTAT. They formed the inspiration for developing a set of authoring tools.

author demonstrated, in exactly the order they were demonstrated. We also added a template-based authoring method (called "Mass Production") that greatly facilitates the creation of large numbers of isomorphic or near-isomorphic problem instances.

Programming-by-demonstration systems enable users to create software without actual coding, by demonstrating behaviors that the software should implement. The software then mimics these behaviors, or even, learns to generalize them. Programming by demonstration has been applied to a number of application areas, both outside the domain of ITSs (Lieberman, 2001; Myers, McDaniel, & Kosbie, 1993) and within, in systems such as DEMONSTR8 (Blessing, 2003), DISCIPLE (Tecuci & Keeling, 1999), and SimStudent (Matsuda, Cohen, Sewall, Lacerda, & Koedinger, 2007; 2008). Finally, the ActiveMath learning environment (Melis et al., 2001) has visual authoring tools that can be used to create tutor exercises through a simple form of programming by demonstration (Libbrecht & Groß, 2006; Goguadze & Tsigler, 2007). These tools bear some resemblance to CTAT, but we cannot judge how the tutoring systems built with them compare to example-tracing tutors. The novelty in the CTAT (and its tools for creating example-tracing tutors) is not in applying programming by demonstration per se, but in how it applies it. It does so in a way that is simpler but more powerful than other systems, at least until induction approaches, ones that typically leverage a combination of human intelligence and machine intelligence such as DEMONSTR8 and SimStudent, have matured further.

DEMONSTR8 (Blessing, 2003) is an early ITS authoring system that, to the best of our knowledge, pioneered programming by demonstration techniques for ITS development. CTAT can be viewed as the "next generation" of DEMONSTR8, which pre-dates example-tracing tutors; it focused on rule-based cognitive modeling and model-tracing tutors. Using DEMONSTR8, an author created example solutions for a small number of problems by creating a tutor interface through drag-and-drop techniques and demonstrating solutions on the interface. (CTAT follows squarely in DEMONSTR8's footsteps in this regard.) An author then generalized the examples into production rules through carefully designed interaction, helped by machine learning. To the best of our knowledge, DEMONSTR8 was not used to create any real-world ITS, or to create tutors for multiple domains. Another early authoring system prototype that pioneered the idea of developing tutors through programming-by-demonstration is DISCIPLE (Tecuci & Keeling, 1999). This tool used a variety of interactive machine learning techniques to learn from examples and explanations provided by an author. DISCIPLE was used to create a tutor for history problem solving that has been evaluated in an actual middle-school classroom, but we are not aware that DISCIPLE has seen much use since. Finally, SimStudent, a separate CTAT module not discussed in this paper, induces rule-based cognitive models from a combination of author-provided examples, author-provided "background" knowledge, and author feedback on its (SimStudent's) attempts to solve new problems using the cognitive-model-being-learned (Matsuda et al., 2007; 2008). Its machine learning algorithm, based on inductive logic programming, can generalize a solution structure (i.e., can generate solutions with a different combination or order of skills). At the time of this writing, SimStudent has not been used to create a real-world ITS, although unlike DEMONSTR8 and DISCIPLE, it has been used in multiple domains. In a study with data from the Algebra Cognitive Tutor, SimStudent's learning mechanisms were able to account for student learning data with about 80% accuracy (Matsuda et al., 2007). (This study did not directly evaluate the efficacy of SimStudent as an authoring tool. It may well be that with more examples to learn from, as would be appropriate in a study on authoring, SimStudent could reproduce the cognitive model of the Algebra Cognitive Tutor with greater or even complete accuracy.)

All three systems (DEMONSTR8, DISCIPLE, and SimStudent) were highly significant advances in the state-of-the-art of ITS authoring tools. Their creators clearly saw the urgent need to make ITS authoring simpler and to reduce the involvement of highly-skilled programmers. Yet is fair to say that the systems they created were research prototypes, with perhaps the exception of DISCIPLE. None of these tools have been used by authors other than their original creators to create real-world tutors in a range of application domains, and only one was used in multiple domains. It is important to recognize how much more technically ambitious these approaches are compared to CTAT; creating techniques for automated or semi-automated domain-general knowledge acquisition based on a very small number of examples (keeping the number of examples small is necessary since they are created by demonstration) is a notoriously difficult problem for AI, one that AI researchers have long been pursuing with only limited success in specific areas (cf. Wolf & Martin, 2005; Rodner & Denzler, 2009).

Example-tracing tutors represent a simpler approach. They provide tutoring using (generalized) behavioral examples, captured in an editable behavior graph, without the use of machine learning to induce general knowledge structures. One of CTAT's contributions lies in the fact that it shows how surprisingly effective simpler techniques can be, and how they can lead to faster real-world success. It may well be that ITS tools that induce general knowledge structures will one day be the preferred choice over CTAT, and perhaps that day is not far away. Then again, it is possible that for a long time to come, tools that induce general knowledge structures from examples will not fully shield authors from having to hand-edit the knowledge structures that the tool generates. This would mean that these tools make tutor authoring more efficient, but not for non-programmers. Be that as it may, for now, CTAT represents a more pragmatic option. We hasten to add that while CTAT simplifies authoring, compared to building tutors from scratch, the tutors built with CTAT are by no means simple. As discussed further below, example-tracing tutors implement many of the behaviors typical of ITSs (VanLehn, 2006). In fact, they go well beyond VanLehn's minimum requirement that in order to be considered an ITS, a system must have an "inner loop," that is, provide *step-by-step guidance* within problem-solving activities. We consider the main contributions of the work presented in this paper to be both the example-tracing tutor technology (i.e., the use of generalized behavioral examples to provide guidance during problem-solving practice) and mature and robust tools for building these kinds of tutors without programming.

In this paper, we describe the behavior and inner workings of example-tracing tutors, illustrate the process of creating an example-tracing tutor with CTAT, and describe CTAT's modular architecture. We also provide evidence of CTAT's effectiveness by documenting its widespread use ("vote-with-your-feet evidence") and by presenting cost-effectiveness data from a range of projects that have used CTAT. Finally, we offer a set of six general requirements for ITS authoring tools, distilled from our experience developing CTAT and assisting CTAT users, and discuss how CTAT and its tools for creating example-tracing tutors stack up with respect to these requirements.

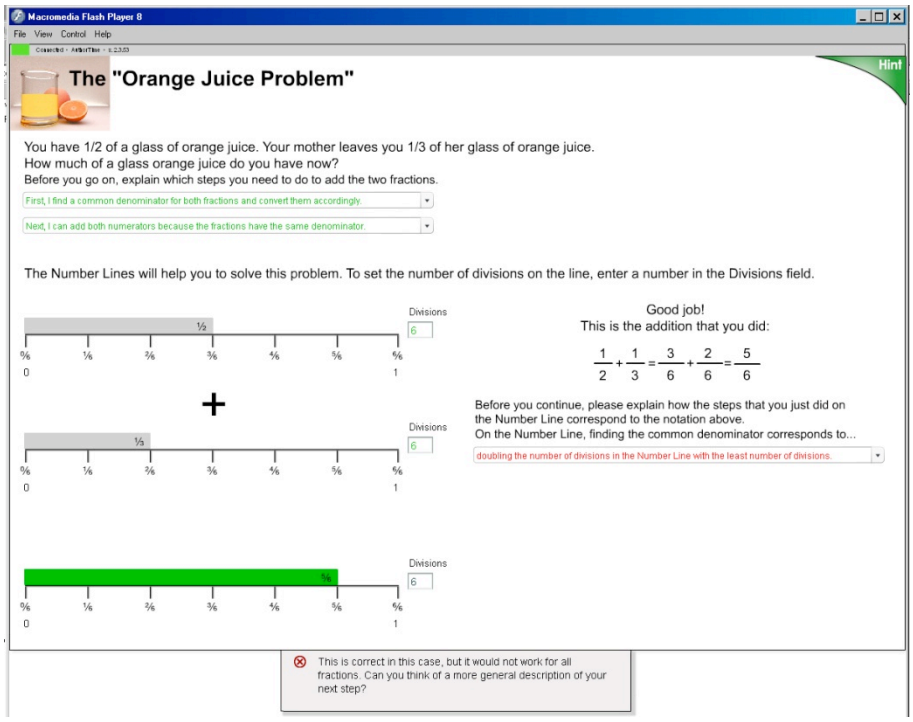


Fig. 1. An example-tracing tutor for 6th-grade fractions learning (Rau, Aleven, & Rummel, 2009). This tutor was used in a study in a middle school with 130 students, who used it for 2.5 hours.

THE BEHAVIOR OF EXAMPLE-TRACING TUTORS

Like many ITSs, example-tracing tutors help students acquire a complex cognitive skill through guided practice (but see du Boulay (2006) for a broader notion of ITSs). In the current section, we establish that example-tracing tutors are capable of a large subset of the behaviors catalogued in Kurt VanLehn's (2006) seminal paper "The Behavior of Tutoring Systems." We consider this set of behaviors to be a good starting point for comparing ITS authoring tools. Example-tracing tutors cover the same subset of these behaviors as Cognitive Tutors do. This is not to say that there are no behavioral differences between example-tracing tutors and Cognitive Tutors. Ultimately, Cognitive Tutors are more flexible. However, this flexibility is not always needed, and comes at the considerable expense of having to create a rule-based cognitive model. We briefly return to this topic in a later section. In the next section, we show further that example-tracing tutors are capable of sophisticated behaviors not differentiated in VanLehn's categories.

As is typical of ITSs, CTAT-built tutors provide an interface that makes thinking visible (e.g., Anderson et al.,1995), in the sense that they lay out (or prompt students to enter) intermediate steps in the solution of a problem (e.g., Figures 1 and 2). Further, example-tracing tutors provide various types of guidance that are typical of ITSs, divided by VanLehn (2006) into an "inner loop" and "outer loop" (see Table 1). The inner loop denotes the support that the tutor provides a student within a single problem, including step-by-step guidance while the student solves the problem and assistance with

Table 1: CTAT's coverage of the categories of tutoring behavior identified by VanLehn (2006) – the table reflects CTAT features that existed in the spring of 2009

Inner loop (within-problem guidance)

+	Minimal feedback on steps - classified as correct, incorrect, or suboptimal
+	Immediate feedback
+/–	Delayed feedback – not built-in, but certain forms can be authored
–	Demand feedback
+	Error-specific feedback
+	Hints on the next step
+	Assessment of knowledge
–	End-of-problem review of the solution

Outer loop (problem selection options)

–	Student picks
+	Fixed sequence
–	Mastery learning
(+)	Macroadaptation

Legend	
+	CTAT supports it
(+)	CTAT will soon support it
+/–	CTAT supports a limited form of it
–	CTAT does not support it

end-of-problem reflection. The outer loop pertains to the selection of appropriate problems for a student to solve. We review the inner and outer loop behaviors that CTAT supports.

With respect to the inner-loop behaviors identified by VanLehn (2006), example-tracing tutors provide correctness feedback on all problem-solving steps by the student, and may provide specific feedback messages for commonly occurring errors. This feedback is given immediately after each step. The CTAT example-tracing algorithm does not support delayed feedback, although some specific ways of delaying feedback can be authored within CTAT, as illustrated below with a demo tutor for factoring quadratics (see also Roll et al., 2006; Roll, Aleven, McLaren, & Koedinger, 2007). Second, example-tracing tutors provide next-step hints at the student's request. Third, they assess student knowledge. Following the ACT-R theory of cognition and learning (Anderson & Lebière, 1998), and following the way Cognitive Tutors operate (e.g., Koedinger & Aleven, 2007; Koedinger & Corbett, 2006), example–tracing tutors are based on the notion that a complex cognitive skill is composed of many fine-grained "knowledge components" that can be acquired and strengthened separately. In its inner loop, an example-tracing tutor interprets student problem-solving behavior in terms of knowledge components, based on a mapping between problem steps and knowledge components provided by the author. Finally, CTAT does not have built-in facilities to support students in reviewing and reflecting on their solution at the end of a problem.

With respect to the outer loop (i.e., the way in which problems are selected for students to solve), CTAT currently supports fixed problem sequences only. CTAT will soon be extended so that it supports a form of individualized problem selection based on Corbett and Anderson's (1995) Bayesian "knowledge tracing" algorithm. Using this algorithm, CTAT-based tutors will track how well the student masters each knowledge component targeted in a particular unit of instruction, and will select problems that involve unmastered knowledge components. VanLehn (2006) refers to this kind of task

selection based on fine-grained knowledge assessment as "macroadaptation," although in the Cognitive Tutor literature, it is referred to as "cognitive mastery learning" (Corbett & Anderson, 1995). CTAT does not provide facilities for what VanLehn calls "mastery learning," which under his definition is a less-sophisticated form of individualized problem selection based on more aggregate or coarse-grained measures of student performance.

VanLehn (2006) seems to imply that the existence of an inner loop is what defines ITSs *vis-à-vis* other forms of computer-based instruction. However, in our opinion, this definition puts the bar too low, as it encompasses very simple forms of computer-based instruction (e.g., simple tutors for two-step problems that do not allow for multiple different answers) that belie the historic roots of ITSs in artificial intelligence, cognitive science, and investigations into the epistemology of knowledge (e.g., Wenger, 1987). We propose that example-tracing tutors should be regarded as ITS not just because they have an inner loop, but also (and primarily) because of the flexibility with which they assess student behavior within the inner loop. They are capable of providing guidance with respect to *multiple strategies* for solving a given problem, regardless of which strategy the student decides to take. Furthermore, example-tracing tutors are capable of entertaining *multiple interpretations* of student behavior, when a student action can be interpreted in multiple ways. These properties are illustrated in the next section.

HOW EXAMPLE-TRACING TUTORS WORK

In this section, we focus on the workings of the inner loop of example-tracing tutors, the way in which they provide step-by-step guidance to students for a given problem. As mentioned, example-tracing tutors interpret a student's problem-solving behavior with respect to a generalized solution graph for the given problem, which, following Newell and Simon (1972), we call a "behavior graph." As discussed in the next section, these graphs can be created, edited, annotated, and generalized without programming. We review the properties of behavior graphs, the process of interpreting student behavior against such a graph (called "example tracing," by analogy to model tracing), and the mechanisms in CTAT for flexibly matching student behavior against a behavior graph: the ability to handle multiple paths and multiple interpretations of student behavior.

A behavior graph is a directed, acyclic graph that represents acceptable ways of solving a problem. The links in the graph represent problem-solving actions, and the nodes represent problem-solving states. A behavior graph may contain multiple paths corresponding to different ways of solving a problem. It may also contain links that represent incorrect behavior, marked as such by the author who created the graph. Let us consider, for example, the stoichiometry problem shown in Figure 2. The stoichiometry tutor is an example-tracing tutor developed using CTAT that has been used in a series of classroom studies (McLaren, et al., 2008a; 2008b, McLaren, Lim, Yaron, & Koedinger, 2007, McLaren, Lim, Gagnon, Yaron, & Koedinger, 2006). Stoichiometry uses basic mathematics to solve elementary chemistry problems; solving problems in this domain requires applying concepts, such as unit conversions and molecular weight, in solving equations. The behavior graph of Figure 3 represents a section of the solution to the stoichiometry problem in Figure 2.

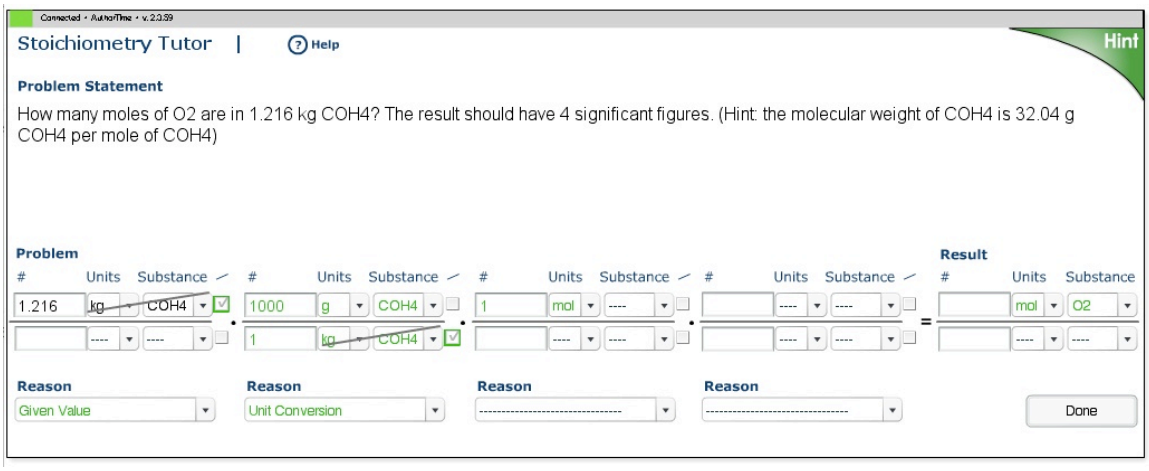


Fig. 2. An example-tracing tutor for stoichiometry[2]. Approximately 200 students have used this tutor for an average of approximately 1¼ hours (McLaren, et al., 2008a).

The optimal solution to this problem, according to a chemistry instructor, is:

$$1.216 \text{ kg } COH_4 \times (1000 \text{ g } COH_4 / 1 \text{ kg } COH_4) \times (1 \text{ mol } COH_4 / 32.04 \text{ g } COH_4) \times (1 \text{ mol } O_2 / 2 \text{ mol } COH_4) = 18.98 \text{ mol } O_2$$

That is, the student is expected to take the given value (1.216 kg COH_4) and multiply it by three terms in order to calculate the final answer of 18.98 mol O_2: first by a term that represents a unit conversion (1000 g COH_4 / 1 kg COH_4, as has already been done in Figure 2), then by a term that represents the molecular weight of COH_4 (32.04 g COH_4 / mol COH_4), and finally by a term that represents a stoichiometric relationship (1 mol O_2 / 2 mol COH_4). The substances and units in the numerators and denominators of the terms cancel each other out, leading to a non-ratio as the final answer. There are other ways of solving the problem, because the order of the multiplicative terms in the solution does not matter. It is perfectly valid, for example, to swap the third and fourth terms:

$$1.216 \text{ kg } COH_4 \times (1000 \text{ g } COH_4 / 1 \text{ kg } COH_4) \times (1 \text{ mol } O_2 / 2 \text{ mol } COH_4) \times (1 \text{ mol } COH_4 / 32.04 \text{ g } COH_4) = 18.98 \text{ mol } O_2$$

[2] There is no subscript notation in the stoichiometry tutor, as one of the reviewers remarked. Chemistry teachers who have worked with us advise that the notation used in this tutor is standard in other software tools and does not distract students. We are working to upgrade CTAT's facilities to handle a wider range of mathematical notation, which would be useful across a range of tutors.

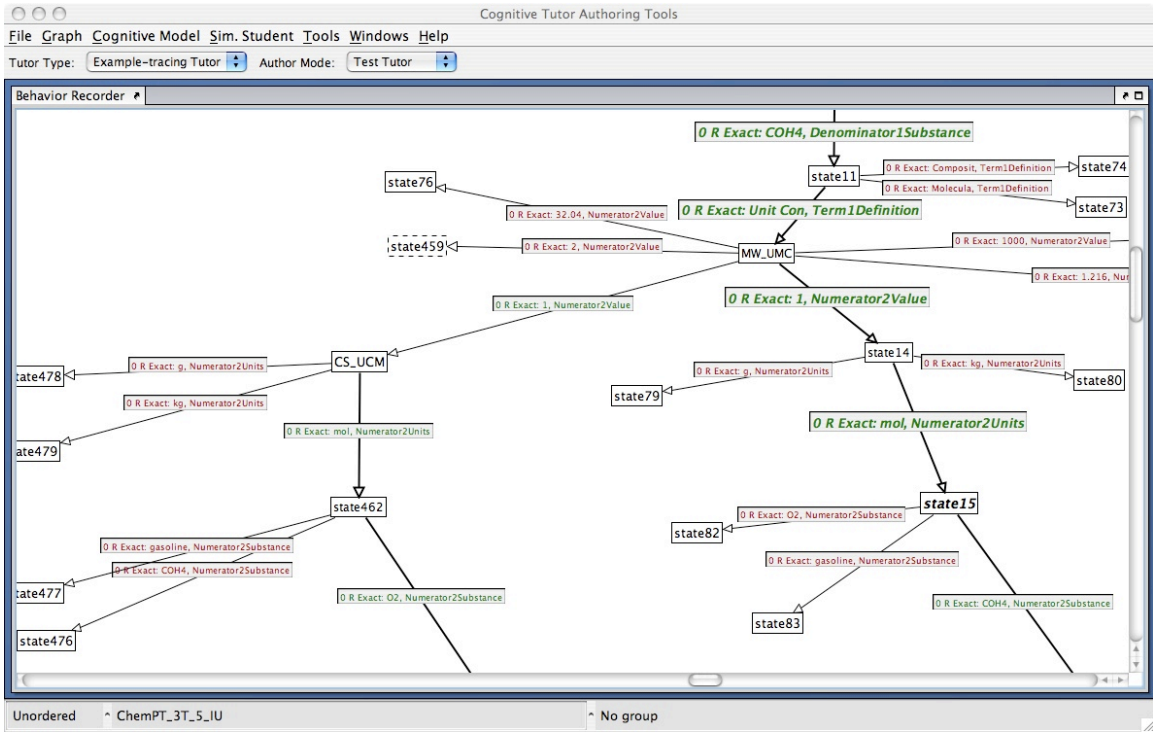


Fig.3. Part of an example behavior graph for the stoichiometry tutor.

Chemistry instructors, however, recommend the first strategy – to apply terms in an order that allows cancellation of parts (e.g., units and substance) of *prior* terms. For instance, the unit conversion (1000 g COH_4 / 1 kg COH_4, i.e., the second term in each of the equations shown above) is the best initial multiplier, since it allows the cancellation of parts of the first term (i.e., the given value), as shown by the "strike outs" in the interface of Figure 2. The instructor-recommended solution and the alternative solution are both represented in the behavior graph for this problem (see Figures 3 and 4 – the links representing correct actions are indicated with green text on the label, but this is not visible in the print version of this article). The instructor-recommended path is shown on the right in Figure 3. The boldface arrows indicate that the author has designated this path as representing the *preferred* way of solving this problem. The alternative solution path described above is encoded in the behavior graph as the path on the left.

Let us consider in more detail how the example tracer uses the graph to monitor student problem solving and provide feedback. In the terminology of VanLehn (2006), the example tracer serves as *step analyzer*. It classifies each student action in the tutor's interface as correct, incorrect or suboptimal by flexibly comparing it against the steps stored in the graph. It also keeps track of the student's problem state by recording which links in the graph the student has "visited." Specifically, it keeps track of the *viable paths* through the graph, meaning start-to-finish paths that are consistent with the student's observed behavior thus far.[3] In order for a student action to be accepted as correct, it must

[3] More specifically, a path is viable if all student actions that have been deemed correct so far correspond to links on this path. It is viable in the sense that it is a viable *interpretation* of the student behavior so far.

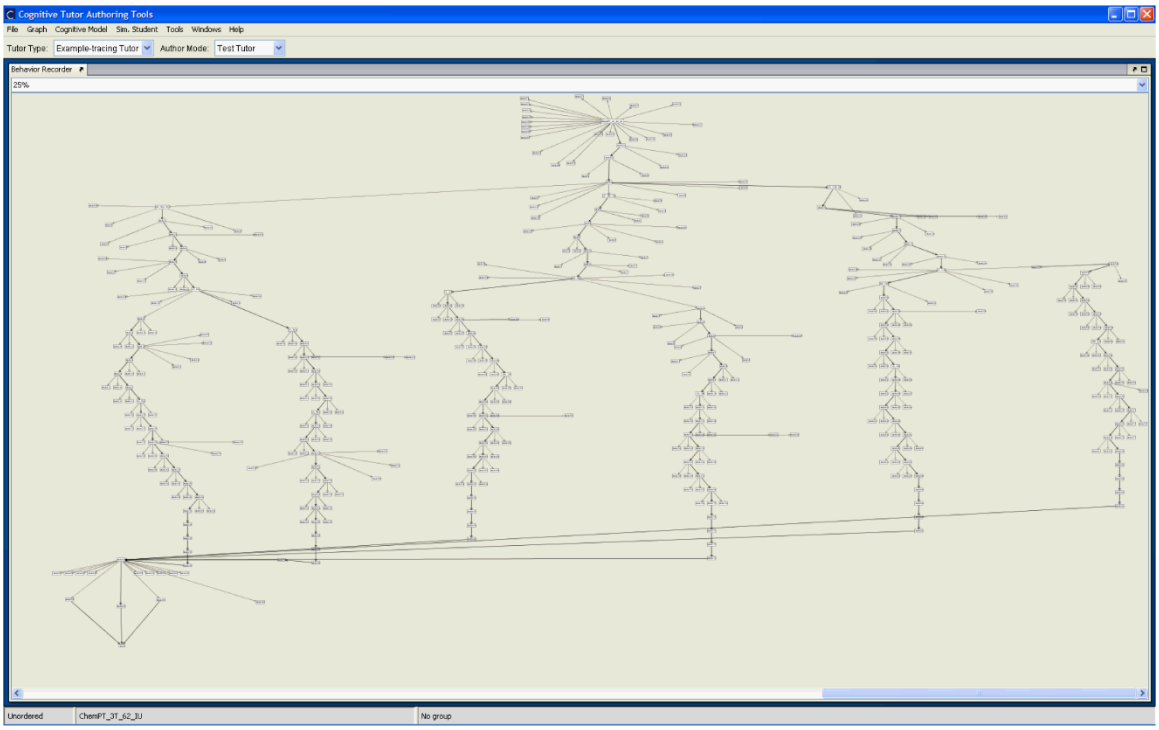


Fig.4. Complete stoichiometry behavior graph (zoomed-out view); this behavior graph is at the upper end of complexity of what has been achieved with CTAT.

correspond to an as-yet unvisited link on a viable path, and that link must represent correct behavior. (We use the term *viable link* to refer to such links.) Thus, once on a viable path, the student must stay on that path. If the student has visited a complete path through the graph, she has completed the problem.

As one measure of the flexibility of the example-tracing process, the example tracer is capable of following a student through a problem even if the graph contains multiple paths, regardless of which of them the student actually takes. For example, suppose the student, working on the stoichiometry problem discussed above, has entered the second term, and is about to enter the third term. That is, the student is at the point in the problem where the expert and non-expert solutions paths diverge (represented by state MW_UMC in the graph, shown in Figure 3). At this point, both the start-to-finish path through the left branch, as well as that through the right branch, are viable paths, since all student actions that have been accepted as correct occur on both. Assume that the student enters the third term of the expert solution (1 mol COH_4 / 32.04 g COH_4). The six required actions all appear on the right branch (the expert branch), but not all are included on the left branch. Thus, after the tutor accepts these actions, the right branch is a viable path, but the left branch has been ruled out as a viable path. Subsequently, only actions on the right path will be accepted as correct actions. Conversely, if the student had gone down the left path (representing the non-expert solution), the right path would have been ruled out as a viable path, and only actions on the left path would be accepted.

As another measure of the flexibility of example-tracing tutors, these tutors are capable of following the student in the face of ambiguity as to how a student action should be interpreted. This ambiguity occurs when a student action matches multiple viable links in the graph. In such situations, the example tracer will entertain multiple possible *interpretations* of the student's problem-solving behavior. This capability enables it to follow along smoothly with what the student is doing. For instance, this kind of ambiguity occurs at the point in problem solving illustrated in Figure 2, where the student has completed the second term, and has entered "1" and "mol" as part of the third term. The student could either be multiplying by (a) the molecular weight (with "1 mol COH_4" placed in the numerator), corresponding to the expert solution path (right branch of the behavior graph), or (b) a stoichiometric relationship (with "1 mol O_2" placed in the numerator), corresponding to the non-preferred solution (left branch). The CTAT example tracer maintains *both* interpretations as possibilities at this point, meaning that both the paths through the left branch and through the right branch are viable paths. The example tracer is therefore in a position to accept as correct student behavior, either "COH_4" or "O_2" as the substance in the numerator, regardless of which one the student decides to enter. By contrast, if the example tracer had committed to one interpretation or the other at the moment that the ambiguous student input occurred, it would not have been able to accept, with equal smoothness, all valid subsequent student input. The ability to entertain multiple interpretations of student behavior sets example-tracing tutors apart from Cognitive Tutors, which (although they can follow students along multiple strategies) do not at any point in time entertain multiple interpretations in parallel. Rather, they disambiguate on the spot (see e.g., the example in Anderson, 1993).

In addition to providing "generic" correctness feedback (i.e., "yes/no" feedback), example-tracing tutors can provide specific feedback on common student errors. A behavior graph may contain links that represent incorrect actions (marked with red font on the label, although the color may not be visible in the print version of Figure 3). For instance, the links that lead to nodes with no children at the top of Figure 3 (e.g., states 76 and 459) represent incorrect actions. When a student action matches no viable link in a graph, but does match an incorrect action link emanating from a viable path, an error message associated with that incorrect action link is shown to the student. If a student input value does not match any viable link in the graph, or any suboptimal or incorrect action link attached to a viable path, the tutor provides unspecific feedback indicating only that the student action is incorrect.

In addition to using their behavior graphs to generate feedback, example-tracing tutors use the behavior graph to generate hints as to what the student might do next. Essentially, CTAT's *step generator,* in the terminology of VanLehn (2006), works by finding an appropriate unvisited link in the behavior graph, and then displaying a hint message associated with that link. (Typically, all links in the graph have hint messages attached to them.) A key question is what hint to give (e.g., Van Lehn, 2006, pp. 242-243). When there are multiple interpretations of the student behavior, one is designated as the reportable interpretation – the one that most closely matches the preferred solution path through the problem. The first unvisited link in this viable path is chosen and the hints associated with this link are displayed to the student. Thus, in our stoichiometry example, if the student is in the (ambiguous) situation shown in Figure 2, the tutor's hints will focus on the expert solution (i.e., the right branch). If the student ignores the hint and follows the alternative strategy (i.e., the left branch), the right branch is no longer viable and the next hint will focus on the left branch.

Table 2. Developing an example-tracing tutor with CTAT

1. Define instructional objectives
2. Identify problem categories and problems for which to provide tutoring
3. Perform Cognitive Task Analysis (e.g., think-alouds and difficulty factors analysis)
4. Design and develop tutor
    a. Design and create interface
    b. For each problem (category)
        i. Demonstrate correct and incorrect behavior (i.e., create a behavior graph)
        ii. Generalize and annotate the behavior graph
        iii. (Optional) Use template-based Mass Production to create multiple problems with isomorphic behavior graphs
    c. Organize curriculum and create curriculum files
5. Deploy prototype version
6. Run pilot test
7. Iterate
8. Deliver final version

In sum, example-tracing tutors exhibit a number of desirable properties of the inner loop not enumerated explicitly by VanLehn (2006), although perhaps assumed. They are able to deal with multiple strategies for solving a given problem, and are able to maintain multiple interpretations of student behavior. In addition, as discussed further below, an author can *generalize* a behavior graph in a number of ways, expanding the range of student behavior that matches any particular behavior graph. In our opinion, this level of flexibility is required in many ITS applications.

CREATING EXAMPLE-TRACING TUTORS WITH CTAT

In this section, we describe how an author uses CTAT's main tools for creating example-tracing tutors. (As mentioned, CTAT also provides tools for building Cognitive Tutors, including tools for creating cognitive models. Those tools however are outside the scope of the current paper.) The use of the CTAT tools is part of a more-encompassing process aimed at tutor creation, outlined in Table 2. We focus on the steps in which CTAT is used, primarily steps 4-6.). The other steps are outside the scope of this paper. Some of them are described by Baker, Corbett, and Koedinger (2007).

Creating the User Interface for a Tutor

After doing cognitive task analysis to learn how students solve, or fail to solve, problems of the targeted type, an author must design and create a tutor interface (specific to the targeted problem). When using CTAT (see Figure 5), an author uses an interface builder, in drag-and-drop fashion, to create a tutor interface that lays out the problem-solving steps that the student will go through (step 4.a in Table 2). The author has the choice between creating a Java-based interface and a Flash-based interface. Both can be created using an off-the-shelf Interactive Development Environment (IDE), either for Java development (e.g., Netbeans) or for Flash. For instance, the stoichiometry and fractions tutors, discussed above, were developed using Flash.

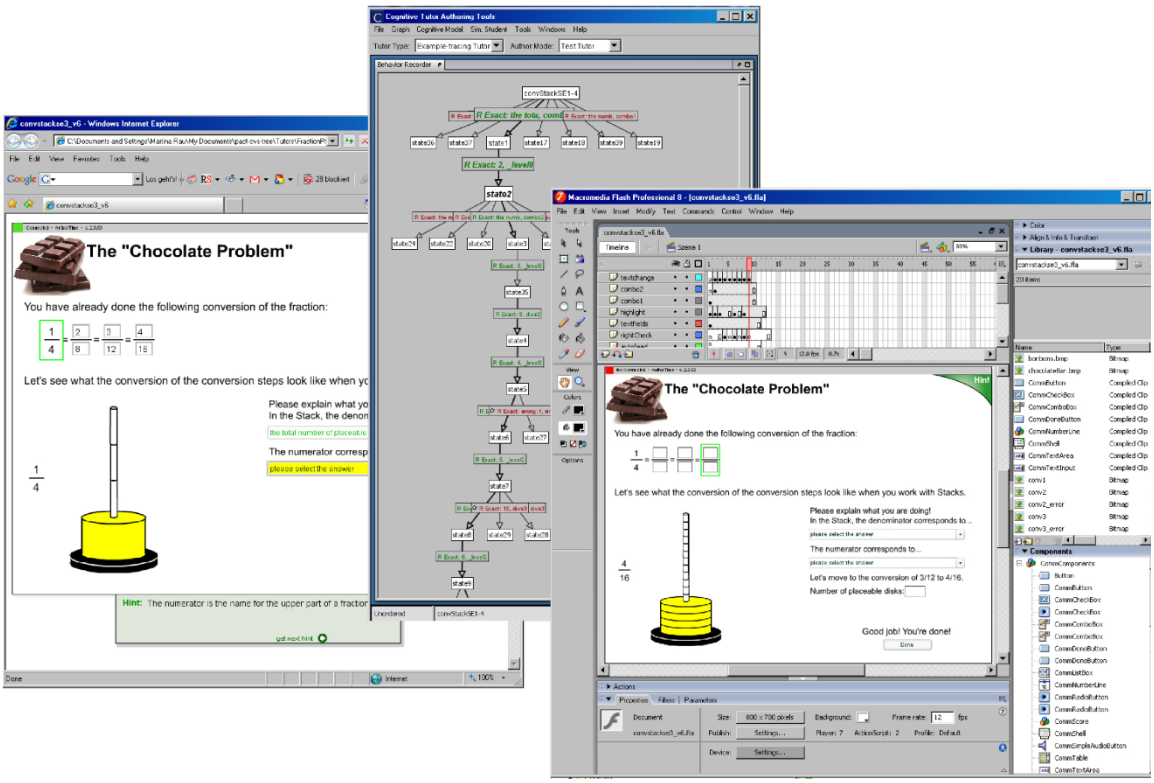


Fig. 5. Creating an example-tracing tutor with CTAT. An author creates an interface through drag-and-drop techniques in the Flash development environment (shown on the right), runs the interface (shown on the left, in a browser), and uses CTAT's Behavior Recorder (middle) to create, generalize, and annotate a behavior graph. The tutor in this example is for 6th-grade fractions.

For each of these environments, Java and Flash, we have created a set of interface widgets that are "CTAT-enabled" in the sense that they communicate with the rest of the tools. These sets contain CTAT-enabled versions of standard widgets such as buttons, combo boxes, text fields, text areas, labels, and lists. In addition, both sets contain composite widgets, such as a table widget, and (in the Java set only), a "Composer" used to build sentences from a sequence of menus (see Figure 6). Various widgets support often-used combinations of interface elements, such as a question followed by a menu to select the answer, or a question followed by multiple possible answers. The set of tutor-enabled Flash widgets (or "components" in the Flash terminology) contains a number of media-enabled components (see Figure 7). Finally, as an example of a domain-specific widget, an interactive number line was created for the fractions tutor illustrated in Figures 1 and 10.

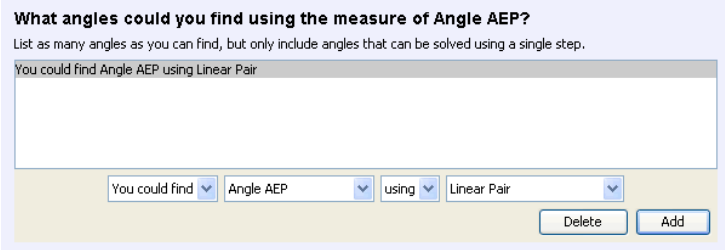


Fig. 6. Composer widget (in Java) for building sentences from a sequence of menus.

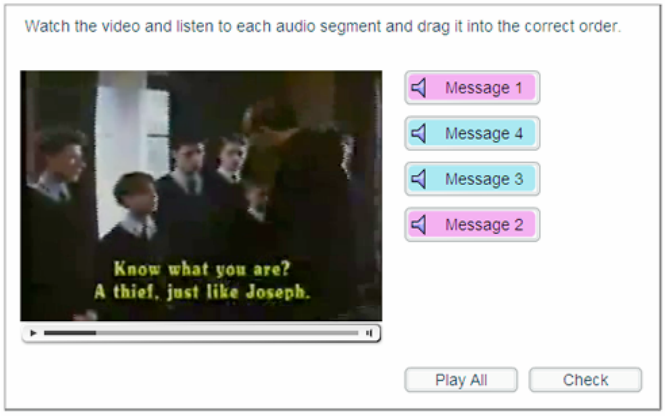


Fig. 7. Multiple selection widget (in Flash) with a video prompt and audio responses.

Using these widgets, an author can construct interfaces that support complex reasoning by students, such as the tutor for introductory (college- and high-school level) genetics shown in Figure 8 and the stoichiometry tutor shown above. These tutor interfaces are made up entirely of standard CTAT-enabled widgets (with a small amount of custom programming in the case of the stoichiometry tutor, to show strike-out of terms in a stoichometry equation that cancel each other out). One of the media widgets was used in a tutor for intercultural competence (Ogan, Aleven, & Jones, 2008; see Figure 9). To the extent that the pre-defined widget sets are sufficient, tutor interfaces can be built entirely without programming, using the drag-and-drop GUI builders built into standard Java or Flash development environments. Programming is necessary only for tutor applications that require new widgets.

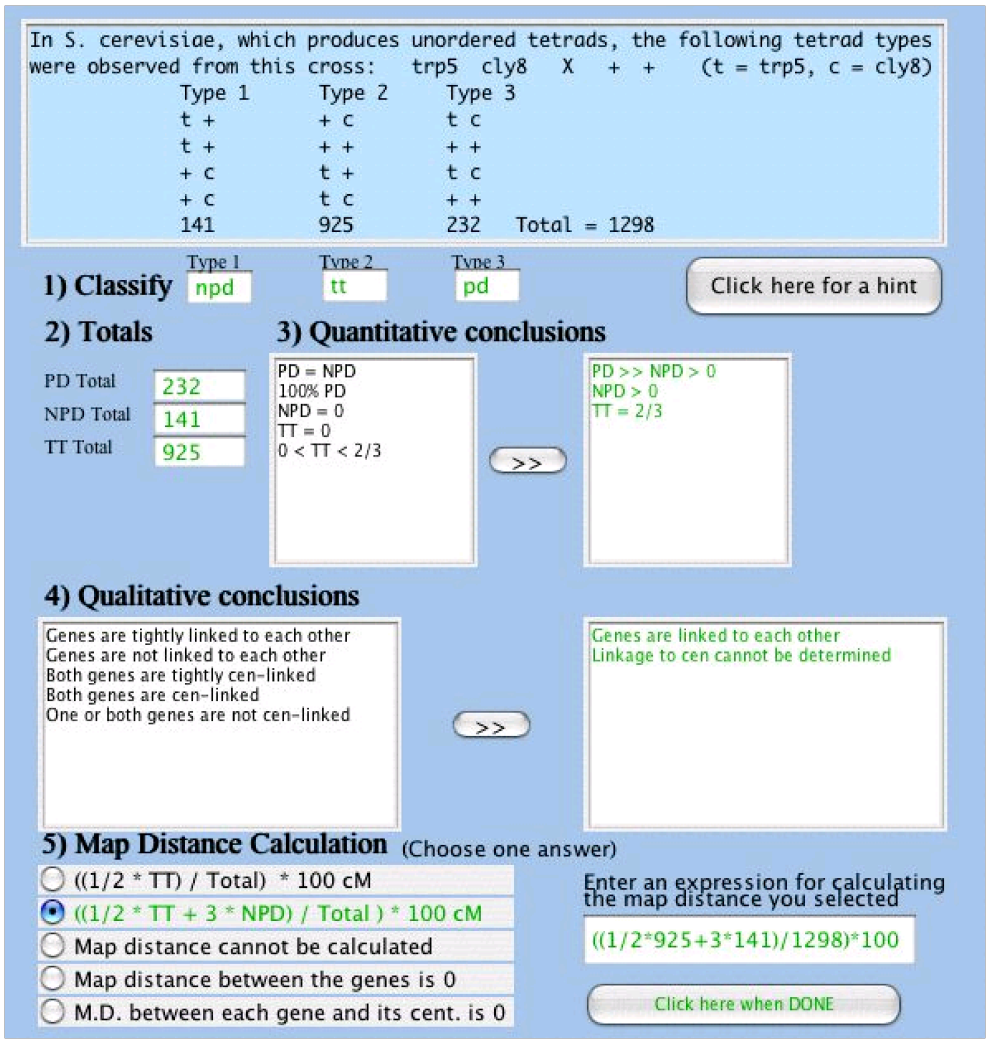


Fig. 8. Tutor for genetics (with Java interface) built with CTAT. The Genetics Problem Solving Tutor was implemented with CTAT by Albert Corbett and colleagues. It covers a wide range of genetics topics, and has been deployed and evaluated at 14 colleges and universities around the country. It is primarily a Cognitive Tutor, although for one of the units in the tutor curriculum, example-tracing tutors have been created.

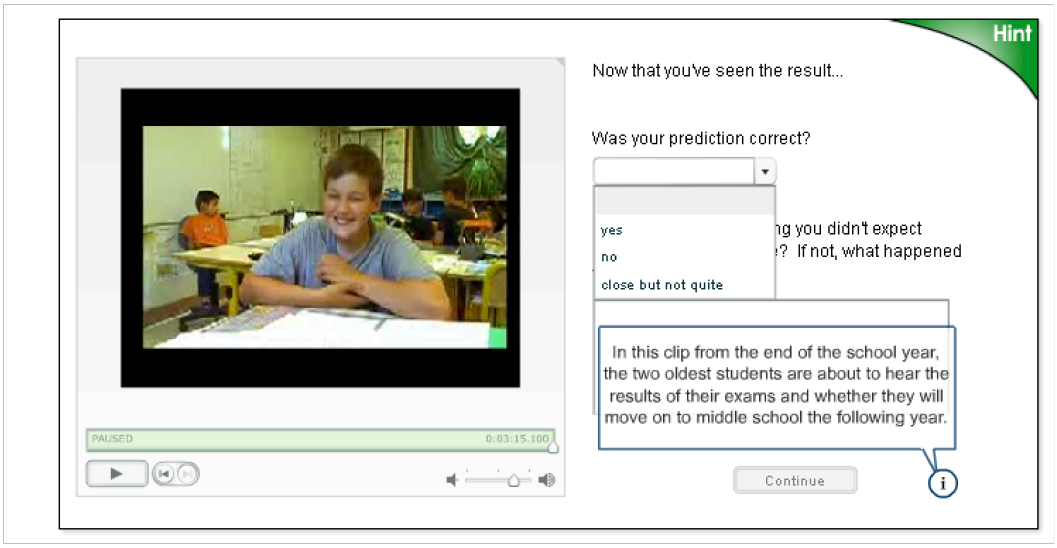


Fig. 9. A tutor for French culture makes use of CTAT's video widget.

Dynamic Interfaces

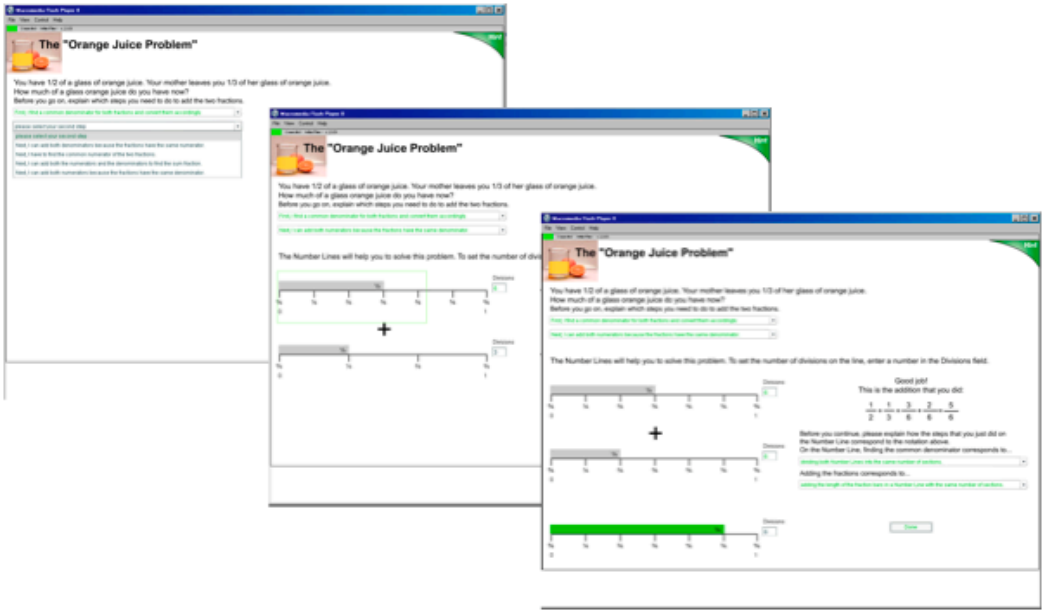


Fig. 10. Sequence of screens in a tutor for fractions; this kind of dynamic interface can be authored without programming.

CTAT also offers the capability to create *dynamic tutor interfaces,* meaning that the interface changes in response to student actions, for example by rearranging the screen layout, adding or subtracting widgets, or changing the content of existing widgets (see Figure 10). Using CTAT,

dynamic interfaces can be created without programming. They serve many purposes, and CTAT users have requested this feature since day one. One purpose is to make effective use of limited screen real estate, which is important especially for tutors that run in a browser, and for tutors that (e.g., in less-affluent school districts) run on computers with low-resolution screens. The facility can also be used to provide dynamic scaffolding: if a student is struggling, the tutor may add steps in the interface to make thinking visible at a finer-grained level (e.g., Razzaq & Heffernan, 2006). As another example, the steps in the tutor's user interface could be revealed one-by-one, as the student solves the problem, as a way of leading the student through these steps in order (as illustrated in Figure 10). Alternatively, the dynamic interface facility could be used to implement dynamically-linked representations in tutors that display multiple representations of a problem (Moore, 1992; Reed, 2005). Also, the facility can be used for the tutor to communicate certain consequences of his or her choice to the student. Finally, as a relatively simple use of this facility, the system may take over some of the steps that are necessary to solve a problem but that do not address learning objectives (e.g., Brown, 1985).

In order to create dynamic interfaces, an author can insert links in a behavior graph that represent "tutor-performed actions." Among the possible tutor-performed actions are showing and hiding of widgets, and changing the content of a widget. For interfaces that consist of a sequence of different screens, it may be more convenient for an author to make use of Flash's "frame" facility, which makes it possible to create multiple screens within a single interface. The transition between frames can be controlled by the tutor, through the use of tutor-performed actions.

Creating Behavior Graphs

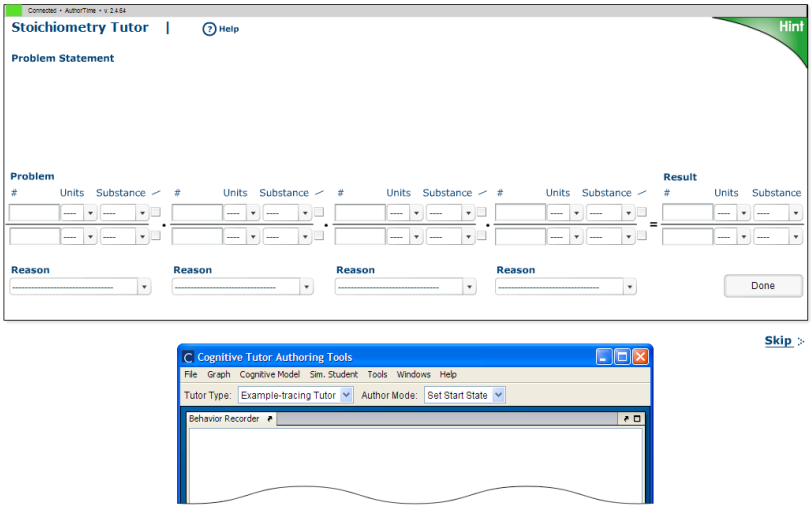


Fig. 11. Empty interface and behavior graph, prior to entering a "start state."

An author must create behavior graphs for all problems-to-be-tutored, although a template-based "Mass Production" process (described below) substantially reduces the amount of problem-specific authoring needed to create multiple problems with the same behavior graph structure. CTAT's Behavior Recorder can be used for recording, editing, annotating, and generalizing behavior graphs.

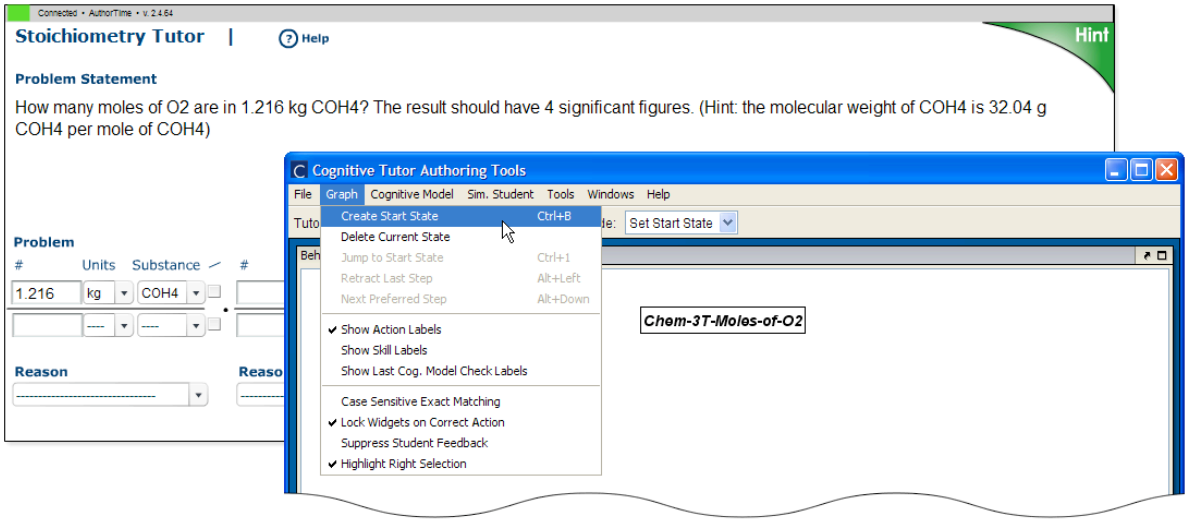


Fig. 12. Interface after entering the start state information.

The process of creating a behavior graph starts with defining a "start state" for the given problem. The author enters problem-specific information into the "empty" interface (see Figure 11), typically, a problem statement, values in some of the widgets, problem-specific graphical elements, and so on (see Figure 12). Next, the author selects "Create Start State" in the Behavior Recorder (Figure 12). CTAT prompts the author to enter a name for the problem, and the initial, solitary node of the behavior graph appears in the Behavior Recorder, that is, the "start state."

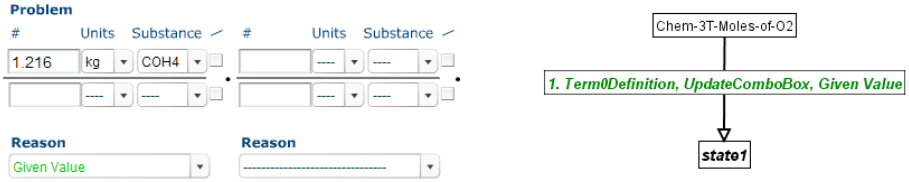


Fig. 13. Interface and behavior graph after demonstrating the first step of the expert solution.

After these preliminaries, the author puts the Behavior Recorder in *Demonstrate mode* and demonstrates expected problem-solving behavior in the interface. The author's problem-solving steps are recorded in a behavior graph, one link per step, where a step is typically a single entry in an interface widget. For instance, in Figure 13, the author has just selected "Given Value" from the pull-down menu in the lower left of the stoichiometry user interface as the reason for the initial value. This step has been captured by the Behavior Recorder as the first link of the behavior graph, with a new node at the end of the link to represent the new problem-solving state. The label on the link shows details about the specific action it represents. A green label (not visible in the print version) indicates that the link represents correct behavior.

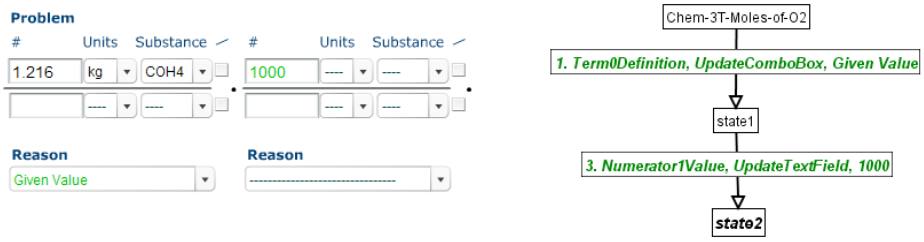


Fig. 14. After demonstrating the second step of the expert solution.

As the author demonstrates the second step of the solution (e.g., entering "1000" as the numerator value of the second term), that step is recorded as well (Figure 14), and so on (Figure 15), until a complete solution path is captured in the Behavior Recorder.

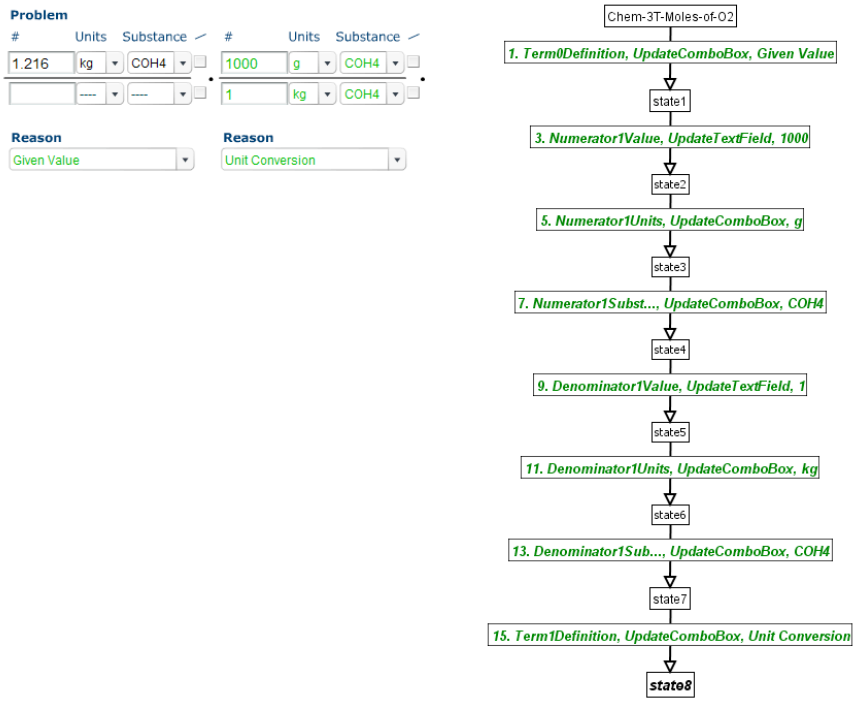


Fig. 15. After demonstrating steps through the completion of the second term.

When creating a behavior graph for a given problem, it is important to record all reasonable (or pedagogically desirable) solution paths that students may take. Otherwise, there is a risk that the tutor will reject valid student input. An author can create alternative paths in the graph by backing up to a problem-solving state that has been recorded previously (when the authors clicks on a state, the interface reverts to that state), and demonstrating a different way of solving the problem. The new solution steps will be recorded in a new branch off of the selected state. For example, as discussed earlier, in our stoichiometry problem, a student can enter the terms of the stoichiometric equation in

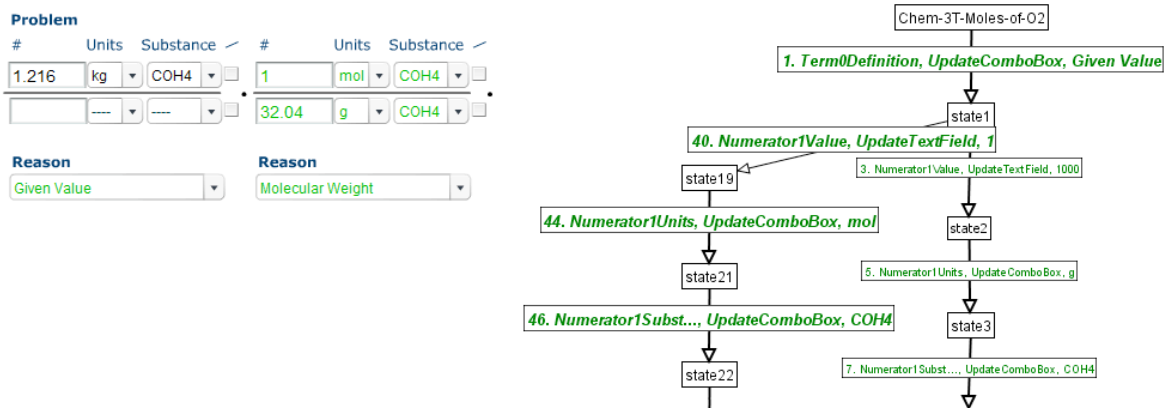


Fig. 16. Demonstrating the first steps of an alternative, correct solution.

any order, leading to alternative options for the first, second, and third term. After the initial step of entering "Given Value" as the explanation for the first term, therefore, valid next steps are (in addition to the already-recorded expert solution) to provide either the molecular weight of COH_4 (1 mol COH_4 / 32.04 g COH_4) or the relevant stoichiometric relationship (1 mol O_2 / 2 mol COH_4). Our author can record these strategies as separate paths in the manner described above (Figure 16).

To enable the tutor-being-built to generate error feedback messages for commonly-occurring errors, an author must demonstrate these errors, mark the corresponding links in the behavior graph as representing incorrect behavior, and attach a feedback message. For instance, in Figure 17, the author has just demonstrated an incorrect action by entering "32.04" in the numerator of the second term from the left. This number is a relevant value to the problem (i.e., the molecular weight of COH_4), but it does not appear in the numerator of any term for any of the possible correct solutions. After marking the corresponding behavior graph link as representing incorrect behavior (see Figure 17), the author is prompted to provide a specific error message (not shown).

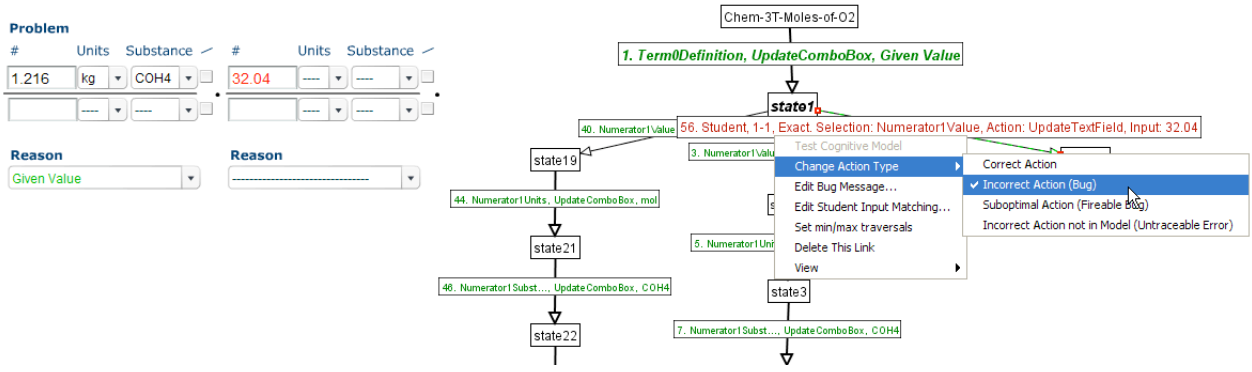


Fig. 17. Demonstrating an incorrect step and marking it as an "incorrect action."

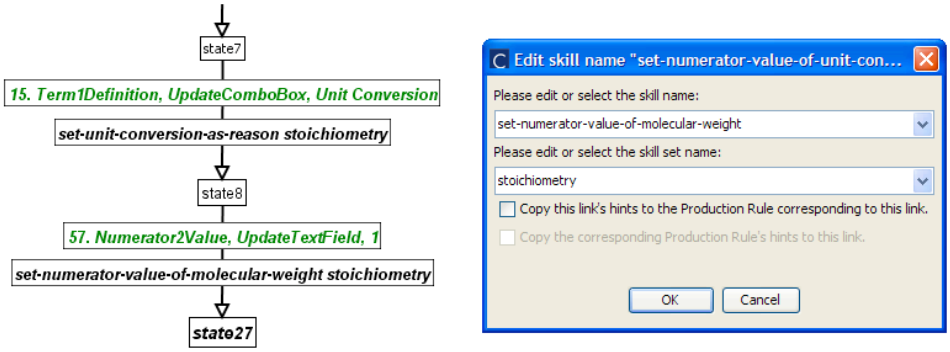


Fig. 18. Labeling a step in the behavior graph with a knowledge component; the second (lower) label on each link indicates the knowledge component.

Annotating a Behavior Graph

After a behavior graph has been created, the author's next step is to annotate the links in the graph with hints and with knowledge component labels. It is in this step that the roots of example-tracing tutors in the Cognitive Tutor methodology and the ACT-R theory of cognition and learning are most clearly visible. Example-tracing tutors share, with Cognitive Tutors, the notion that a complex cognitive skill can be decomposed into small "knowledge components" that can be acquired and strengthened separately through practice (Anderson & Lebière, 1998). Thus, the learning objectives targeted by these tutoring systems are (often) defined in terms of knowledge components. In Cognitive Tutors, the knowledge components are expressed as production rules. In example-tracing tutors, by contrast, the knowledge components are defined extensionally, by labeling steps in a behavior graph with the name(s) of the knowledge component(s) that are (hypothesized to be) exercised in solving the step (see Figure 18). The knowledge component names are displayed in the behavior graph in separate link labels. The labeling of steps in terms of knowledge components is an important form of cognitive task analysis. By labeling multiple steps in a behavior graph (or across behavior graphs) with the same knowledge component, an author makes a prediction that practice on the one step will lead to improved performance on the other (see Koedinger et al., 2004 for a detailed example).

For example, in our stoichiometry task, there are (the authors of that tutor hypothesize) different knowledge components for applying a molecular weight term versus a unit conversion term versus a stoichiometric relation term. The knowledge components are subdivided further by the 6 different elements of each term (value, unit, and substance of the numerator and denominator), and by whether the step is an explanation step. Thus, as illustrated in Figure 18, the stoichiometry problem described above involves knowledge components for explaining unit conversion steps ("set-unit-conversion-as-reason") and for entering the numerator value of a molecular weight term ("set-numerator-value-of-molecular-weight").

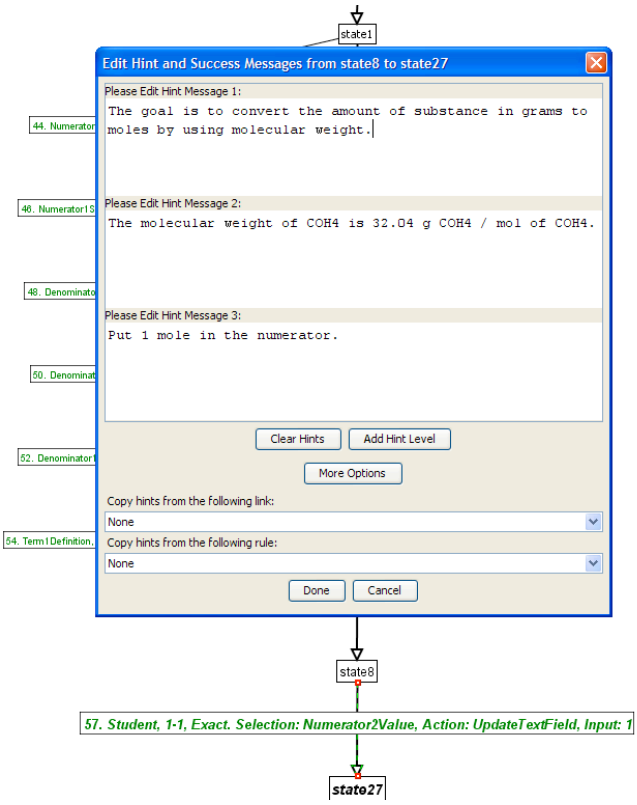


Fig. 19. Defining a hint sequence.

The knowledge component labeling also helps in designing a curriculum and problem set for the tutor. It helps an author keep track of whether the problem set she is creating provides sufficient practice opportunities with each of the targeted knowledge components. CTAT automatically generates a *skill matrix* to support this analysis; this report shows which problems involve which knowledge components. Also, in the near future, the knowledge component labels will enable the tutor to assess student knowledge, one of the key inner loop functions of ITSs (VanLehn, 2006). We are in the process of adding the Bayesian knowledge-tracing algorithm created by Corbett and Anderson (1995) to CTAT, so that it can track individual students' knowledge growth, and select problems accordingly.

In addition to knowledge component labels, an author annotates behavior graph links with hint sequences – these hints will be displayed when the student requests help from the tutor on the given step (see Figure 19). Typically, multiple levels of principle-based help are provided, going from broader hints (e.g., what goal or step to work on, what problem-solving principle to apply), to more specific advice (e.g., how does the problem-solving principle apply, what calculations need to be done), and finally, to a *bottom-out hint* that provides the resulting answer, or something close to it. The hints should be consistent with the knowledge component labels: steps that involve the same knowledge component should have similar hint sequences.

Generalizing Behavior Graphs

As mentioned, an author can generalize a given behavior graph in a number of ways to extend the range of student behavior that the example tracer will recognize as matching the graph, beyond recognizing only the exact same steps as were demonstrated by the author, in the exact same order as they were demonstrated. This flexibility has been key in making example-tracing tutors into a viable ITS paradigm. Using CTAT, an author can:

- specify constraints on the ordering of steps: the author can define arbitrarily nested groups of steps that should be "ordered" or "unordered";
- define a range of student input that matches a particular link; for example, an author can specify a numeric range, a set of values, or a regular expression;
- specify how one step depends on other steps, by attaching Excel-like formulas to the links in a behavior graph; and
- specify that links are "optional" or "re-usable," by setting a lower or upper limit on the number of times it can be "traversed" by the student (by default, both limits are set to 1, meaning that the link must be traversed exactly once in order to complete the given path through the problem).

We illustrate a number of these mechanisms, starting with the one aimed at promoting flexibility with respect to the order in which the student can carry out the steps needed to solve a problem. In many situations, the specific order of (some or all of) the steps in the problem does not matter. In such situations, a tutor should allow students the freedom to perform the steps in any order they may choose. Otherwise, the tutor would reject correct steps for reasons that are likely to be viewed as arbitrary or confusing by students. On the other hand, where order *does* matter, the tutor should help students learn the right order, meaning that it should reject steps when they are out of order, even if they are otherwise correct, preferably with a message explaining a better order of steps. CTAT gives an author fine-grained control over the order in which problem steps will be accepted by the tutor. An author can group links in a behavior graph and specify for each group whether it is "ordered" or "unordered." Groups can be nested within groups.

Consider, for example, a tutor unit dealing with decimal addition and subtraction (see Figure 20; this tutor is part of a project, led by the authors of this paper, to create a comprehensive website with tutors for middle-school math, Aleven, McLaren, & Sewall, 2009). In this tutor, students, after reading the problem statement, must determine whether they are dealing with an addition problem or a subtraction problem (by selecting the operation to be performed, "+" or "–"), copy the two numbers presented in the problem statement into the interface, and add or subtract to find the final answer. There is probably no good reason to require that students enter the digits of the two numbers in any particular order. On the other hand, it makes sense that students first determine whether they are dealing with an addition or subtraction problem, before doing anything else. Also, it is pedagogically desirable that they follow a systematic addition or subtraction strategy, processing columns in a right-to-left manner. With respect to borrowing/regrouping (in the case of subtraction), a small amount of flexibility is desirable: it does not matter whether the "borrower column" or the "lender column" is updated first (in the "regrouping row," see Figure 20), or whether the result in a column is written before or after the regrouping has been completed. An author can implement such constraints in CTAT using its "grouping" mechanism.

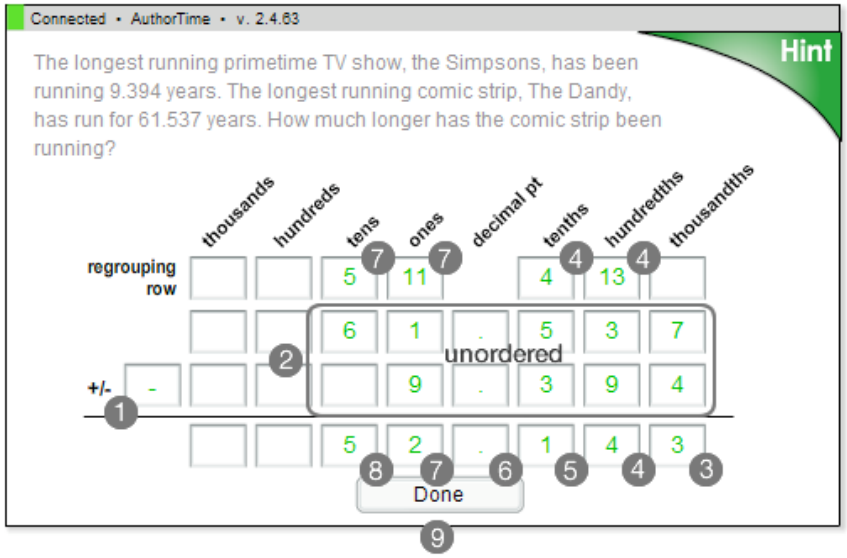


Fig. 20. A tutor for decimal addition and subtraction, annotated to show the desired step order; steps with the same number should be viewed as an "unordered group."

Specifically, an author would specify, after recording all the steps in a behavior graph, that overall, the graph should be viewed as ordered, and would then create three unordered groups of links: one for the steps involved in entering the two given numbers (group 2 in Figure 20), and one each related to the borrowing that occurs in the hundredths and ones columns (groups 4 and 7 in Figure 20), respectively. As illustrated in Figure 21, an author can use CTAT's Group Editor to create and modify link groups, and to specify for each group whether it should be treated as ordered or unordered. In Figure 21, the author has created two groups (shown in the Group Editor at the top left) and is about to create a third group, consisting of three links selected in the behavior graph; she is prompted to specify whether the new group is ordered or unordered.

As a second way in which authors can generalize a behavior graph, they can broaden the matching criterion for a link by specifying a numerical range, a regular expression, or a formula. We illustrate the first and third of these options with a demo tutor for factoring quadratic expressions (see Figure 22). This tutor supports a guess-and-test strategy, reflecting the way that factoring is often done in real life – bypassing the quadratic formula.[4] In this tutor, the students define the problem they want to solve by entering the coefficients a and b of a quadratic expression $x^2 + ax + b$. They must then factor the expression into the form $(x + N_1)(x + N_2)$. The tutor lets the student venture three guesses for the number pair N_1 and N_2. For each, the student must correctly enter the sum $N_1 + N_2$ and product $N_1 * N_2$. If the sum and product are equal to the coefficients a and b, the expression has been factored successfully, and the tutor will accept a click on the "Done" button. Otherwise, the tutor only accepts a click on the "Next Try" button, and adds input fields for the next

[4] The point is to illustrate what is possible with CTAT – we make no claim that the tutor for factoring quadratic expressions is effective in real life.

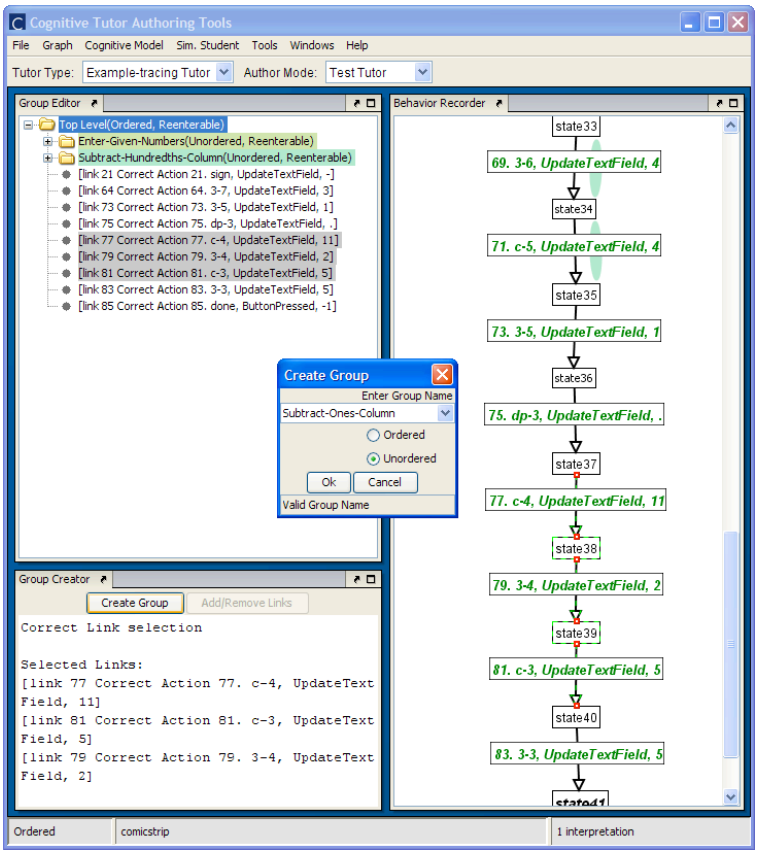


Fig. 21. Defining an unordered group of links.

attempt to find the coefficients (through CTAT's dynamic interface facility). Thus, returning to a point discussed above, the tutor supports a specific form of delayed feedback: the correctness of the student's guesses for N_1 and N_2 is not revealed until later steps.

This tutor makes use of CTAT's range mechanism, one of the three ways in which an author can broaden the matching criterion for a link. Using this facility, an author can specify that a link in the graph can be matched by input within a predefined numeric range. In this tutor, range matches are used on the steps where the student enters the coefficients *a* and *b* for the quadratic-expression-to-be-factored and the steps where students enter their guesses for N_1 and N_2.

The tutor also relies on the third of the generalization mechanisms mentioned above, CTAT's formula mechanism. It does so in two places. First, as one might expect, formulas are used on the steps where the student enters the sum and product of N_1 and N_2. Because N_1 and N_2 are not known in advance, the tutor needs to compute the sum and product on the fly. Second, the formula mechanism is used on the steps where the student enters the values for N_1 and N_2, to check whether these values are correct. The behavior graph for this tutor (see Figure 22) has separate paths for correct and incorrect values, the left and right paths off state "EverythingVisibleForTry2," respectively. On the steps for N_1 and N_2 in the left branch, the author used a CTAT formula based on the quadratic formula. (To accommodate the fact that N_1 and N_2 are interchangeable, the left path splits and then joins again.) The

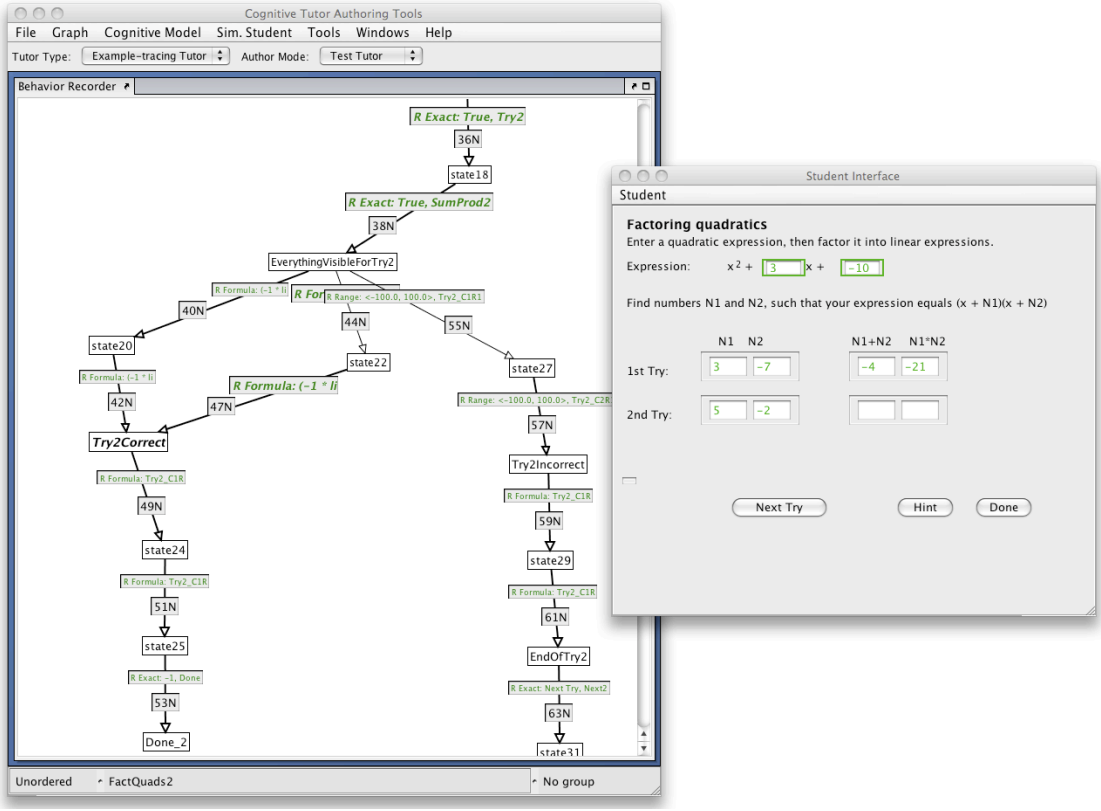


Fig. 22. A demo tutor for factoring quadratic expressions that supports a guess-and-test strategy, and part of its behavior graph. After the student's first attempt at factoring $x^2+3x-10$ as $(x+3)(x-7)$, which is incorrect, the tutor accepts a click on the "Next Try" button but will reject a click on the "Done" button. The student's second try, $(x+5)(x-2)$, is correct. After the student fills out the sum and product of 5 and -2, the tutor will accept a click on the "Done" button.

left path then continues with steps for the sum and product of N_1 and N_2, and finally ends with the Done button. The right path matches *any* values for N_1 and N_2, and this condition is specified by means of a range match. The path continues with steps for the sum and product, followed by the TryNext button, but not the Done button. Therefore, only when the student entered correct values for N_1 and N_2 (i.e., goes down the left path) will the tutor accept the Done button without first requiring another try for N_1 and N_2.

Figure 23 shows the window in which an author enters formulas, displaying a formula used on a link in the left branch to express how the value in the N_2 field depends on other values. The formula refers to values accepted on other links (e.g., "link1.input") and also includes mathematical functions (e.g., "sqrt" for the square root). A formula wizard (not shown) lets an author browse the set of functions that can be included in formulas. They include the Java string and mathematics libraries and a growing CTAT-specific library. It is straightforward to write Java code to extend the function library.

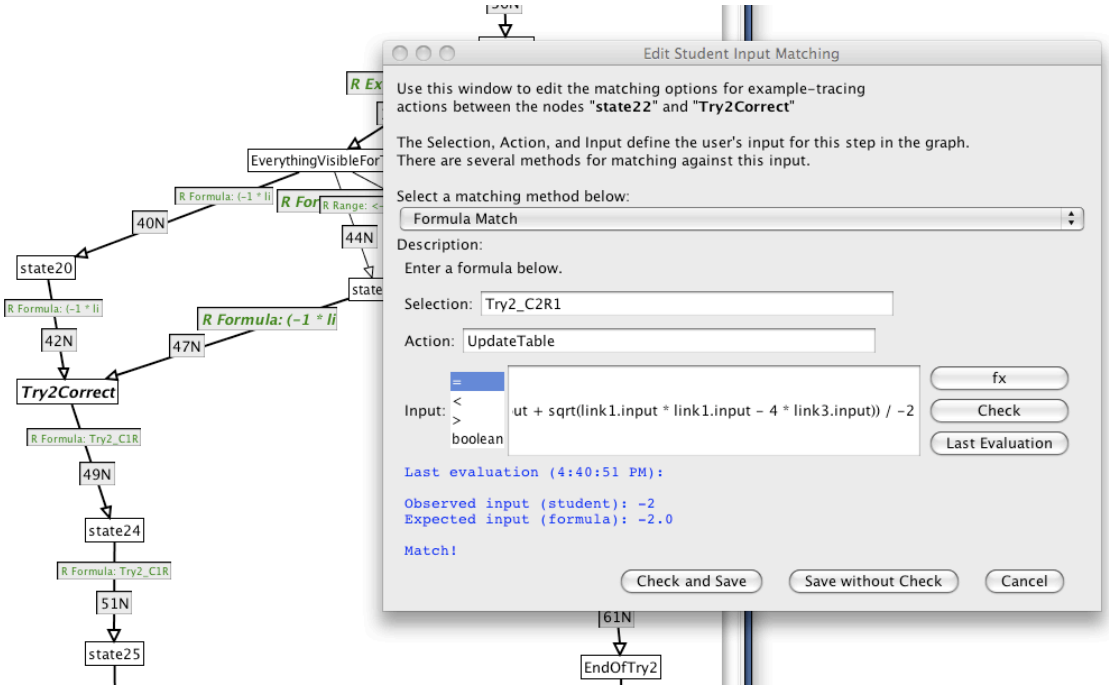
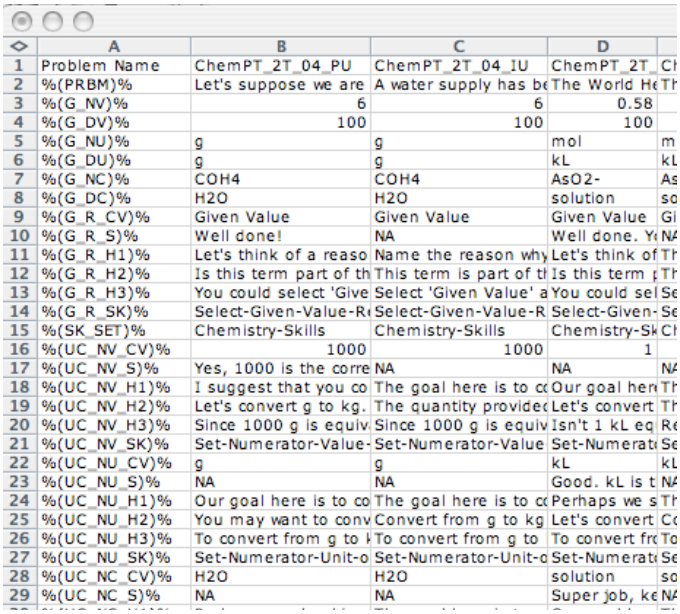


Fig. 23. Dialog box for attaching a formula to a link; the formula expresses how the value on the link depends on values on other links.

Without the range and formula mechanisms, it would have been virtually impossible to build an example-tracing tutor that supports (1) students' choosing the coefficients for the quadratic-expression-to-be-factored and (2) a guess-and-test strategy where the tutor can respond to a wide range of student input that cannot be anticipated in advance. Without the two mechanisms, the behavior graphs would need to contain an intractable number of paths. Thus, the example illustrates how CTAT's capabilities for generalizing demonstrated examples lead to more powerful and flexible tutors than would otherwise be possible.

Mass Production of Behavior Graphs

To facilitate the creation of many *isomorphic* or *near-isomorphic* tutored problems, CTAT provides a facility for template-based *Mass Production*. The idea behind Mass Production is to help an author generate many problems of the same structure without having to demonstrate solutions to each and every one of them. Rather, Mass Production lets an author take advantage of the (ample) reusable information contained in a behavior graph – all of the problem-solving states and links and possibly many of the hint messages and error feedback. The need for (near-)isomorphic problems arises in many task domains, given that ITSs tend to focus on recurring problem-solving tasks. For example, in fraction addition, a student typically completes multiple practice problems with the same underlying structure. As another example, consider the stoichiometry problem discussed earlier:



	A	B	C	D	
1	Problem Name	ChemPT_2T_04_PU	ChemPT_2T_04_IU	ChemPT_2T_	Cl
2	%(PRBM)%	Let's suppose we are	A water supply has be	The World He	Th
3	%(G_NV)%	6	6	0.58	
4	%(G_DV)%	100	100	100	
5	%(G_NU)%	g	g	mol	m
6	%(G_DU)%	g	g	kL	kL
7	%(G_NC)%	COH4	COH4	AsO2-	As
8	%(G_DC)%	H2O	H2O	solution	so
9	%(G_R_CV)%	Given Value	Given Value	Given Value	Gi
10	%(G_R_S)%	Well done!	NA	Well done. Y	NA
11	%(G_R_H1)%	Let's think of a reaso	Name the reason why	Let's think of	Th
12	%(G_R_H2)%	Is this term part of th	This term is part of th	Is this term p	Th
13	%(G_R_H3)%	You could select 'Give	Select 'Given Value' a	You could sel	Se
14	%(G_R_SK)%	Select-Given-Value-R	Select-Given-Value-R	Select-Given-	Se
15	%(SK_SET)%	Chemistry-Skills	Chemistry-Skills	Chemistry-Sk	Ch
16	%(UC_NV_CV)%	1000	1000	1	
17	%(UC_NV_S)%	Yes, 1000 is the corre	NA	NA	NA
18	%(UC_NV_H1)%	I suggest that you co	The goal here is to co	Our goal here	Th
19	%(UC_NV_H2)%	Let's convert g to kg.	The quantity provided	Let's convert	Th
20	%(UC_NV_H3)%	Since 1000 g is equiv	Since 1000 g is equiv	Isn't 1 kL eq	Re
21	%(UC_NV_SK)%	Set-Numerator-Value-	Set-Numerator-Value	Set-Numerati	Se
22	%(UC_NU_CV)%	g	g	kL	kL
23	%(UC_NU_S)%	NA	NA	Good. kL is t	NA
24	%(UC_NU_H1)%	Our goal here is to co	The goal here is to co	Perhaps we s	Th
25	%(UC_NU_H2)%	You may want to conv	Convert from g to kg	Let's convert	Co
26	%(UC_NU_H3)%	To convert from g to k	To convert from g to	To convert fro	To
27	%(UC_NU_SK)%	Set-Numerator-Unit-o	Set-Numerator-Unit-o	Set-Numerati	Se
28	%(UC_NC_CV)%	H2O	H2O	solution	so
29	%(UC_NC_S)%	NA	NA	Super job, ke	NA

Fig. 24. Part of a Problems Table for the Stoichiometry Tutors; the rows represent variables, the columns represent problems; the cells contain problem-specific values for the variables.

$$1.216 \text{ kg } COH_4 \times (1000 \text{ g } COH_4 / 1 \text{ kg } COH_4) \times (1 \text{ mol } COH_4 / 32.04 \text{ g } COH_4) \times (1 \text{ mol } O_2 / 2 \text{ mol } COH_4) = 18.98 \text{ mol } O_2 \quad \textbf{(1)}$$

Another problem provided in the stoichiometry tutor is the following:

$$1 \text{ g } Fe_2O_3 \times (1 \text{ mol } Fe_2O_3 / 159.692 \text{ g } Fe_2O_3) \times (2 \text{ mol Fe} / 1 \text{ mol } Fe_2O_3) \times (55.847 \text{ g Fe} / 1 \text{ mol Fe}) = 0.69943 \text{ g Fe} \quad \textbf{(2)}$$

While the specific terms used to solve these two problems are different (e.g., the first multiplier of equation (1) is a unit conversion term, as discussed earlier, while the first multiplier in equation (2) is a molecular weight term) the *structure* across the problems is identical. More specifically, the given value (i.e., the first term) is a single, non-ratio number multiplied by three ratios (i.e., 6-part terms of the kind discussed before) leading to a non-ratio solution. In CTAT we take advantage of this very common situation by providing a template-based approach to creating behavior graphs.

The Mass Production process is straightforward. After the author has demonstrated and edited (and, ideally, debugged and pilot-tested) a behavior graph for a single problem, she turns this behavior graph into a template by introducing *variables* to given values and demonstrated input, effectively

removing problem-specific information from the graph. The author may also insert the newly defined variables into hints and error messages, as appropriate, or may introduce variables that stand for entire hint or error messages. Once all variables have been introduced into the template, the author edits a "problems table" in Microsoft Excel. For each problem for which a behavior graph is to be created, the author provides values for each of the variables in the template, as shown in Figure 24, which has all values filled in for a range of problems, including (1) and (2) above. The final step is for the author to merge the behavior graph template file with the problems table, in a fashion similar to the way mail merge works in Microsoft Word. The merge step yields multiple instances of the behavior graph, corresponding to individual problems in the problems file (shown as different columns in Figure 24).

Thus, once an initial behavior graph has been created and debugged, tutors of similar structure can be created very efficiently. In the case of the stoichiometry tutors, we generated a total of 35 tutored problems, similar to problems (1) and (2) above, from three basic templates: a one-term template, a two-term template, and a three-term template. We have used Mass Production in a number of other projects as well. In one project for elementary-school whole-number division, over a thousand behavior graphs were created in this manner. In our experience, however, Mass Production is worthwhile even with very small numbers of isomorphic tutors (e.g., 4). A recent extension of CTAT makes it possible to Mass Produce behavior graphs that are *near isomorphs* but not full isomorphs. Using CTAT's facility for specifying an upper limit to the number of traversals for any link in a behavior graph, an author can render a link inactive by setting its maximum number of traversals to 0. Such links are essentially ignored by the example tracer. Therefore, an author can generate near-isomorphic problems from a single Mass Production template by (a) introducing (into the template) variables for the max number of traversals for one or more links in the graph, and (b) varying, problem by problem, which of these links are active and which are inactive, by setting these variables in the problems table.

Not only does the Mass Production process make it easier to produce many example-tracing tutors that share similar structure, it also supports *content consistency* and *maintenance.* It is much easier to look across a row of a table, such as that shown in Figure 24, to check for consistent values and text, than it is to open separate behavior graphs and search for consistency. With respect to maintenance, we have found that many changes involve a simple global replace in the Excel problems table. Without Mass Production, individual behavior graphs would have to be opened and edited; a time-consuming process highly prone to errors. The Mass Production facility illustrates how an authoring tool can be combined synergistically with off-the-shelf software.

CTAT'S ARCHITECTURE

Over the years of developing and using CTAT, we have come to realize that it is important to provide an architecture that is modular. A modular architecture supports integration of existing ITS components with relative ease. Ease of integration is important, because the wholesale plugging-in of new components may be the easiest way to customize tutors. A modular architecture also adds to the versatility of an authoring tool, especially if multiple options are available for (some of) the modules, enabling an author to select the tools or modules appropriate to their particular situation and application. In this section, we briefly describe the main components in CTAT architecture, and illustrate ways in which it is modular and flexible.

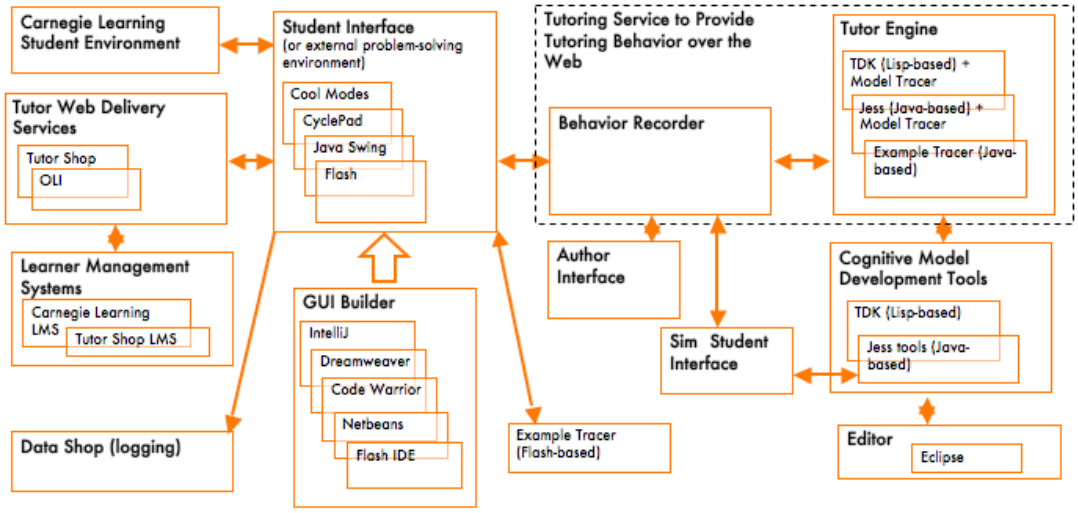


Fig. 25. CTAT's modular architecture.

CTAT's open architecture (see Figure 25) supports the functions described above: building an interface, creating, editing, annotating, and generalizing a behavior graph, and using a behavior graph to provide tutoring. In addition, it supports the following functions:

- building cognitive models for use in Cognitive Tutors;
- delivering tutors and tutoring behaviors over the web and by other means;
- managing student functions such as setting up student accounts, authentication, and teacher reports; and
- logging student-tutor interactions for research purposes. All CTAT- tutors have the built-in option to write and transmit detailed logs without requiring any extra effort from the author. Flash tutors, in addition, can log student-recorded audio clips to Flash's media server.

We already discussed the GUI Builder, Student Interface, Behavior Recorder, and Example Tracer, but have not yet fully discussed the following components:

- The ***Tutor Engine*** takes care of a tutor's inner loop (VanLehn, 2006), that is, it provides step-by-step guidance within a given problem. CTAT provides three different tutor engines: the example tracer described above, as well as two model-tracing engines for cognitive models in Jess (Friedman-Hill, 2003) and TDK (Anderson & Pelletier, 1991).
- The ***Tutoring Service*** provides access to inner-loop tutoring behavior over the web by running a tutor engine on a server machine. It reflects our desire to have a single example tracer compatible with different types of student interfaces running on the web (e.g., Flash, Java).[5] Currently, the Tutoring Service supports example-tracing tutors only.

[5] It was easier to create a server-side tutor engine than to figure out how to integrate the tutor engine and student interfaces locally on the client, although recent developments in Java-Flash integration appear to make that option feasible.

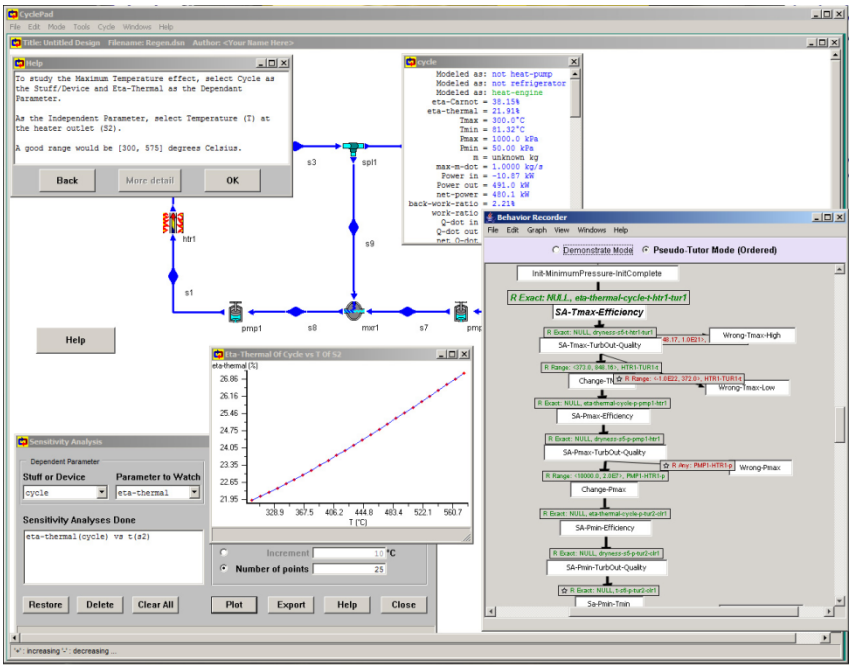


Fig. 26. CTAT was used to add tutoring to the CyclePad thermodynamics simulator; the Behavior Recorder is show at the bottom right; a hint button and hint window were added to the simulator (top left).

- The ***Tutor Web Delivery Services*** serve up tutors or tutor interfaces embedded in web pages. They include authentication services and accommodate such functions as gaining students' consent for participating in an experiment. These services integrate with web servers to also provide (non-tutoring) multi-media material interleaved with tutor problems.
- The ***Learner Management System*** provides services such as creation of class lists, student accounts, and teacher reports. It also provides outer loop functionality (VanLehn, 2006) in that it keeps track of each student's position in the curriculum and provides problem selection options (e.g., macroadaptation). We use an industrial-strength LMS system created in our lab and substantially perfected by Carnegie Learning, a company that markets Cognitive Tutor mathematics courses.
- The ***DataShop*** is a repository for student-tutor interaction data, primarily for research purposes (Koedinger, Cunningham, Skogsholm, & Leber, 2008). It also provides analysis tools for researchers, such as tools to display and analyze "learning curves" (VanLehn et al., 2007), and an option to export log data to Excel or statistical packages for further analysis. The DataShop is not part of CTAT proper, but is a crucial service for CTAT tutors used in research experiments.
- The ***Cognitive Model Development Tools*** are used to create and debug rule-based cognitive models. As they are outside the scope of the current paper, we refer the reader to Aleven, McLaren, Sewall, & Koedinger (2006).

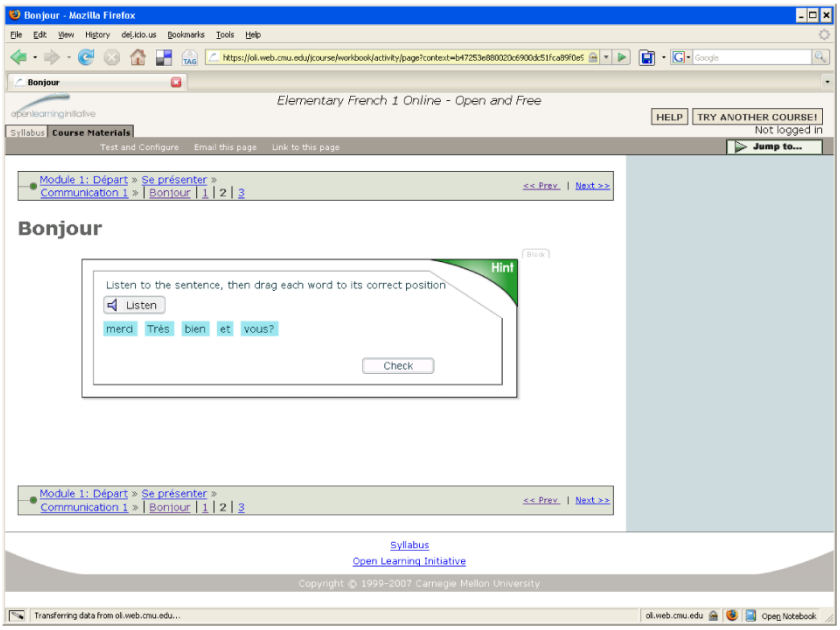


Fig. 27. A simple Example-Tracing tutor embedded in an on-line introductory French course within the Open Learning Initiative. This tutor uses the Flash Jumble widget, which lets a student drag a scrambled set of items (here, words in a sentence) into the desired order.

As evidence of desirable modularity in the CTAT architecture, a number of components in the CTAT architecture have multiple instantiations, indicated in the architecture diagram (Figure 25) with "stacked" rectangles, which represent configuration alternatives. We discuss each in turn.

Alternatives for the student interface: CTAT provides support for creating student interfaces in Flash and Java, as described above. Providing both Java and Flash options makes CTAT more versatile (e.g., Flash may be better for the web, Java is better for object-oriented software engineering; and there are many more factors affecting the choice). Furthermore, the modular separation of the student interface and tutor engine has permitted CTAT users to connect their own interactive simulators or user interfaces to our tutor engines, to create tutored versions of the original system. A simulator for thermodynamics (Rosé, Kumar, Aleven, Robinson, & Wu, 2006) (see Figure 26), an interface for student collaboration (Harrer, McLaren, Walker, Bollen & Sewall, 2006), and a chemistry simulator have been hooked up to CTAT in this way (cf. Blessing et al., 2007; Ritter & Koedinger, 1997). As another aspect of CTAT's modularity, the sets of interface components that can be used in CTAT tutors are easily extensible, because the API for these widgets is well-developed. At least one outside author, with our help, has created a custom component (Wylie, 2007), and we hope that others will contribute components as well, as we make the necessary sources and documentation available.

Alternatives for the tutor engine: The interface options available in CTAT can be combined in mix-and-match fashion with the three tutor engines mentioned above. So far, in real educational settings, we have seen example-tracing tutors with Flash interfaces and Java interfaces, a Jess-based

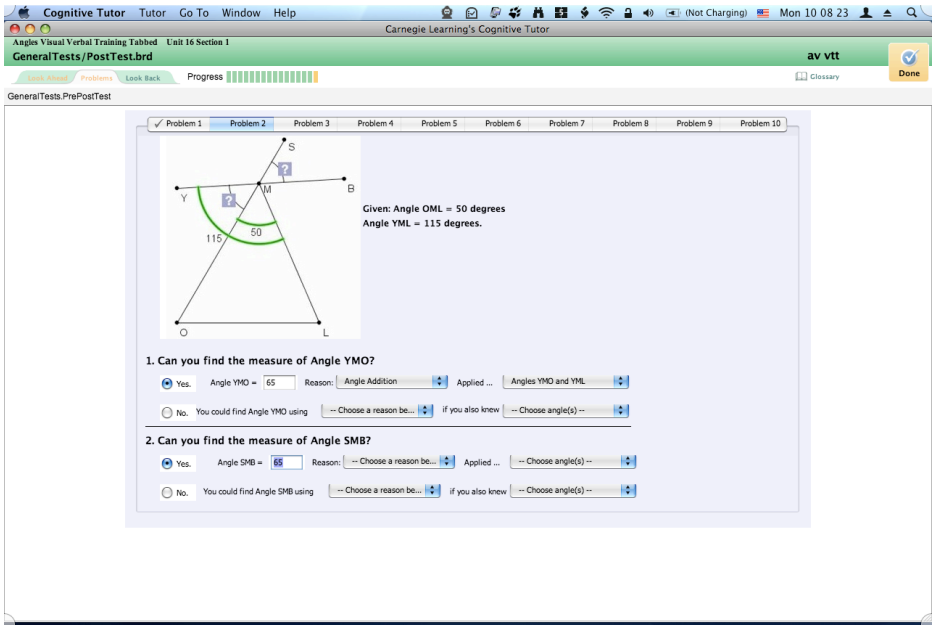


Fig. 28. An on-line assessment authored with CTAT embedded in a Carnegie Learning interface (Butcher & Aleven, 2008).

Cognitive Tutor with a Java interface, an example-tracing tutor with a simulator, and TDK-based cognitive tutors (of the "traditional" kind) and with a CTAT-built interface. Jess-based cognitive tutors are used heavily in the SimStudent project, which uses machine learning to infer a rule-based cognitive model from behavior graphs (Matsuda et al., 2007; 2008). The ability to mix-and-match interface options and tutor engines hinges on the message protocol between interface and tutor engine. This message protocol (http://ctat.pact.cs.cmu.edu/index.php?id=tool-tutor) is derived from Ritter and Koedinger's (1997) architecture for plug-in tutor agents with its careful separation of tool/interface functionality and tutor functionality.

Alternatives for the delivery environment: As another main driver of a flexible architecture, we have encountered the need to deliver tutors in a variety of technical contexts – reflecting the experience that CTAT tutors (and ITS more generally) are often embedded in a larger framework for e-Learning or other forms of technology-enhanced learning. For example, CTAT tutors have been used in the following contexts:

- within an existing e-Learning framework, namely, on-line courses created as part of CMU's Open Learning Initiative (OLI) (see Figure 27);
- web-based but separate from any existing e-Learning framework, with the tutor engine running either on the client or the server (as part of the Tutoring Service) (The stoichiometry and fractions tutors described above belong in this category.);
- integrated with Carnegie Learning's Cognitive Tutors, so that CTAT and CL problems are interleaved (see Figure 28);
- installed desktop applications; although there are many advantages to running tutors on the web, there are situations (e.g., schools with poor network connectivity) where it is an advantage to not have to rely on the web and to install tutors on student machines.

EVIDENCE OF USE AND IMPROVED AUTHORING AFFORDABILITY

CTAT has been used extensively to create example-tracing tutors and on-line assessments. We know of approximately 400 people who have used CTAT to create working tutors and assessments, including people who have used CTAT in research and development projects, and people who have used CTAT in courses and summer schools at Carnegie Mellon, Worcester Polytechnic Institute, and the University of Edinburgh. In addition, there may be others who have used CTAT that we do not know about, since it is freely available on the web. In the past two years alone, CTAT was downloaded over 4,300 times and the CTAT website (http://ctat.pact.cs.cmu.edu) drew over 1.5 million hits from over 100,000 unique visitors. We regularly receive technical questions and requests for support via e-mail.

As shown in Appendix A, CTAT has been used to create tutors for mathematics (elementary, middle-school, and high-school, including whole-number division, fractions, algebra, and geometry), chemistry, genetics, thermodynamics, French language learning and culture, Chinese language learning, and English as a Second Language (ESL). This list includes only tutors that have seen actual use in real educational settings, either as part of classroom research, or as part of regular instruction. As the appendix indicates, CTAT-built tutors and assessments have been used in 26 research studies in real educational settings. Further, CTAT is being used extensively in a project to create a comprehensive website for middle-school mathematics (Aleven et al., 2009). To the best of our knowledge, this makes CTAT the most widely used ITS authoring tool in existence.

In order to get a sense for whether CTAT improves the cost-effectiveness of authoring ITSs, we compare against "historical" estimates, reported in the literature, of overall development time and cost versus the project output (e.g., hours of instruction produced). This method provides some confidence that true gains are being achieved, but it is important not to lose sight of its limitations. The overall project cost-output estimates run the risk of comparing apples and oranges, for example, in that historical projects often have had time to produce high quality output, which may require more iteration and maintenance, whereas new projects may underestimate costs for such iteration and maintenance. On the other hand, historical projects often stretched out over longer periods of time, so that project development teams had time to become familiar with the tools and the technology, to hone their development skills, establish good working procedures, select the best tools for the job, re-use work done in earlier phases, and so on. In other words, within these projects, there usually was sufficient time to achieve an economy of scale. Smaller projects typically do not reach that point. Further, the estimates of development time tend to be very rough, since they are often based on retrospective accounting of how time was spent, and they tend to be somewhat inconsistent across projects, since it is not easy to separate tutor development from other project activities. Finally, different projects used CTAT at different stages of its development – for instance, the stoichiometry tutors were built before functions were available, functionality that would have made their development much easier – which also introduces variability. The main point is that the development time estimates should be treated as rough estimates.

As our baseline, we use development time estimates reported in the literature, which for ITSs range from 200:1 (Woolf & Cunningham, 1987) to 300:1 (Murray, 2003). The "historical" development time estimates for the successful Algebra and Geometry Cognitive Tutors (see, e.g., Koedinger et al., 1997) are in line with these estimates. A rough estimate puts the development time for these two Cognitive Tutors at 200 hours for each hour of instruction. It is important to note that these tutors *pre-date CTAT.* They were in fact an important source of inspiration for CTAT.

Against these historical estimates we compare the development times for CTAT-built tutors that have seen actual use in real educational settings listed in Appendix A. With the exception of one project (stoichiometry), these tutors were built by project staff or students not directly associated with CTAT, who typically had little prior experience developing tutors. In most of these projects, the CTAT staff had a consulting role, but did not do substantial tutor development. In order to provide a fair comparison against historical estimates, the development time estimates listed in the table include all tutor development activities, from the earliest conception of the tutor to the cognitive task analysis activities, tutor implementation, testing, pilot testing (sometimes), and debugging. However, they do not include general CTAT extensions implemented (by CTAT personnel) in the course of these projects, since we are interested in finding out how much the tool contributes to the affordability of authoring. Appendix A also lists, for those projects for which this information could be ascertained, the amount of instructional time covered by the CTAT tutors that were developed. This information was obtained from the lead researchers involved in each project. Since these projects were research studies in real educational settings, the amount of time covered by the tutors is known quite accurately, as arrangements needed to be made with collaborating teachers regarding the number of class periods that the study would cover.

The development time picture that emerges is very interesting. If we distinguish between small projects (at most one hour of instruction created) and larger projects (more than one hour), we see that small projects do not "beat" the 200:1 and 300:1 ratio reported in the literature. On the other hand, the larger projects consistently do better than that, with the Whole-Number Division project "weighing in" at a 107:1 ratio, an Algebra tutoring project at 87:1, the Stoichiometry project a 85:1 ratio, and the fractions project at 48:1. These estimates are higher than those reported in an early paper on CTAT (Koedinger et al., 2004), which no doubt reflects the tougher standard used in the current paper, in particular, the requirement that the tutor must have been used in an actual educational setting. Even so, the larger projects that have used CTAT point to considerable savings in development time, with development time estimates 3-4 times better than the historical averages. The fact that the small projects have not consistently done better than the historical averages is likely due to the many factors discussed above; in these projects there simply may not have been enough time to recoup the start-up costs. The savings are even greater when we factor in personnel cost. In many of the projects listed above, the CTAT tools were used by non-programmers, so if one estimates that PhD level AI programmers cost about 1.5-2 times as much as non-programmers, then our very rough estimate indicates that CTAT leads to 4-8 times more cost-effective development of ITSs than the "historical" results reported in the literature. We reiterate that there is a risk of comparing apples and oranges: the historical estimates were based on larger-scale projects that may have created higher-quality tutors through multiple iterations of use and revision, although we should emphasize that we included in our sample only tutors that were good enough for real educational settings, which is a high standard. Further, the historical projects were typically much larger than even the largest of the projects being reviewed here, and therefore may have developed a within-project economy of scale not possible for the shorter-duration projects being reviewed here.

The greater cost-effectiveness of example-tracing tutors may be due in the first place to lower upfront implementation costs for each new problem type for which tutoring is to be provided, and to a lesser degree to lower implementation costs per tutored problem. Creating a new problem type for a Cognitive Tutor requires programming extensions to the cognitive model, in the form of production rules, and often requires interface programming. With example-tracing tutors, on the other hand, creating a new problem type requires creating an interface and Mass Production template, both of

which can be done without programming and therefore typically take less time. Once the upfront work for a new problem type has been done, adding a new problem is rather easy both for Cognitive Tutors and example-tracing tutors, in particular when Mass Production is used to create the example-tracing tutors.

DISCUSSION

In this section, we review how CTAT stacks up against six requirements for authoring tools that have emerged from the experience of developing CTAT, and of assisting users of CTAT. Our conclusion is that CTAT address all of these requirements to a substantial degree, fully meeting most of them. An authoring tool suite should:

1. support authoring of effective, intelligent computer-based tutors;
2. facilitate the development of tutors across a range of application domains;
3. support cost-effective tutor development (i.e., minimize time and money, compared to using different tools/methodologies);
4. support delivery of tutors in a wide variety of technical contexts characterized by different interface technologies, content management systems, e-learning frameworks, student management systems, and web technologies;
5. create tutors that are easy to maintain (e.g., easy to modify when requirements change);
6. support research use of tutors through such functionality as logging of student-tutor interactions, application of different treatments to different students, and on-line assessment.

These requirements represent some of the lessons learned in the CTAT project. We do not expect them to be controversial, although it is perhaps surprising that only one of the six requirements focuses on tutoring behavior (namely, requirement #1). During the CTAT project, we shifted from an early focus on affordable authoring of tutor behavior to the more encompassing problems of easy deployment, delivery, and maintenance of tutors. This shift was driven largely by the needs of CTAT users who were creating tutors for use in real educational settings.

CTAT supports the authoring of effective tutors (criterion 1). Example-tracing tutors implement much of the behavior that is considered typical of ITSs, described by VanLehn (2006), focused on step-by-step guidance to students as they solve complex problems, primarily through feedback and hints. However, example-tracing tutors go well beyond VanLehn's basic criteria. First, they follow students along multiple solution paths within a given problem, regardless of which one the student decides to pick. This capability sets CTAT apart from some other authoring tools, including Assistments (Razzaq et al., 2009) though not from others, such as ASPIRE (Mitrovic et al., 2006). This capability matters, because many recurring problem-solving tasks naturally give rise to multiple solution paths (e.g., stoichiometry, fraction addition). Further, CTAT's capbilities for *generalizing* behavior graphs considerably enhances the utility of the example-tracing technology. It enables CTAT to recognize a range of student behavior as correct, well beyond the exact steps captured in the graph in the exact order that they were demonstrated by the author, greatly enhancing the flexibility of CTAT tutors. A second way in which example-tracing tutors go beyond VanLehn's basic criterion is

in their ability to maintain *multiple interpretations* of student behavior. This capability obviates the need for disambiguating student intent "on the spot," as, for example, Cognitive Tutors do. Delayed disambiguation enables the tutor to follow students "wherever they go" in the given problem, as long as they stay within the behavior graph, without interrupting them with disambiguation questions and without running the risk of steering them down a different solution path than they had in mind.

While example-tracing tutors support an interesting range of tutoring behaviors, there remains a class of applications for which Cognitive Tutors are the preferred choice, at least when personnel with the requisite rule-based programming skill are available. For example, Cognitive Tutors are often preferred for problems with variable or unpredictable subgoals, problems in which the relation between actions and subgoals is unclear, problems whose solutions constitute complex structures (e.g., the Lisp Tutor, Anderson, Conrad, & Corbett, 1989), or problems with subtle ordering constraints among steps. We anticipate that we will have more to say about the difference between example-tracing tutors and Cognitive Tutors as we gain experience using CTAT's new formula mechanism, which significantly narrows the gap.

Example-tracing tutors have proven their effectiveness on measures of learning. For instance, the stoichiometry tutors discussed earlier led to statistically significant learning gains from pre to posttest in *all* conditions of three separate studies (McLaren, Lim, & Koedinger, 2008a).

Example-tracing tutors have been built for a range of application domains (criterion 2). In general, CTAT is useful for building tutoring systems that support multi-step problem solving. CTAT has been applied not only in quantitative domains such as mathematics and science, in which ITSs have long had a strong track record, but also in language learning and even cultural learning. The specific domains for which CTAT tutors have been built (and used in real educational settings) include (see Appendix A): mathematics (at the elementary school, middle-school, and high-school level), chemistry, genetics, thermodynamics, Chinese, French, and English as a Second Language. CTAT has also been used to provide tutoring within simulators for thermodynamics and chemistry.

On the other hand, the example-tracing technology seems unnecessary for applications in which there is a single question and answer. For example, CTAT may not be particularly useful for teaching vocabulary items or historical facts, (although it may well be used to create a system for historical analysis). But even for such simpler applications, many components of the CTAT architecture may be useful (e.g., web delivery, logging, student management). CTAT proper is not geared toward supporting the authoring of tutorial interactions in natural language. Finally, CTAT is not geared towards building systems with large domain ontologies (e.g., Crowley & Medvedeva, 2006) or that draw on large stores of factual or conceptual knowledge (e.g., question generation systems).

Example-tracing tutors provide a cost-effective way of developing tutors (i.e., minimize time and money) (criterion 3). One way in which CTAT helps bring down the cost of creating ITSs is that it supports ITS authoring without programming. (ASPIRE (Mitrovic *et al*., 2006) is another such tool.) In many of the CTAT projects reported in Appendix A, non-programmers used CTAT to create tutors or on-line assessments, including many authors who did not have extensive CTAT experience. Of course, authoring tools do not obviate the need for careful cognitive task analysis, interface design, or use of instructional design or learning sciences principles. Thus, one still needs personnel skilled in these areas. However, those same personnel may use the authoring tools directly, if the tool does not require programming, a significant advantage. Although CTAT authors do not have to be programmers, so far they typically have been tech-savvy people, not intimidated by relatively complex

environments such as the Flash IDE, the Netbeans IDE, and CTAT proper. Finally, although the sets of interface-enabled widgets that CTAT provides are often sufficient to create interfaces that support complex reasoning tasks, for certain interactions new widgets are no doubt necessary. These new widgets will require programming.

Development time estimates from projects that have used CTAT over the past years indicate an improvement of as much as 4-8 times greater cost-effectiveness, due to CTAT. It should be kept in mind that the reliability of such estimates is limited because of the many sources of variability (e.g., different personnel, different task domain, different quality of resulting tutors, different time scale of projects, and different opportunity to develop within-project economy of scale). It appears that the savings may be due in the first place to lower upfront implementation costs for each new problem type for which tutoring is to be provided. One way to take advantage of the relative ease with which new problem types and tutor interfaces can be created with CTAT might be to increase problem variability (e.g., Paas & Van Merrienboer, 1994), which has been shown to lead to increased learning.

A number of developments are underway that may further improve the cost-effectiveness of authoring with CTAT, and the quality of the resulting tutors: the SimStudent module of CTAT (Matsuda et al., 2005; 2007), which uses machine learning to create rule-based cognitive models, and a novel "bootstrapping" facility that lets an author create behavior graphs directly from log data of students. The bootstrapping facility is likely to lead to tutors with greater ability to recognize actual student strategies and errors, especially in domains with somewhat more open-ended activities such as the use of simulators (Harrer *et al.*, 2006; McLaren, Koedinger, Schneider, Harrer, & Bollen, 2004).

CTAT supports delivery of tutors in a wide variety of technical contexts (criterion 4). Given the rise in Internet applications in present times, it is key that CTAT makes the process of building and delivering web-based tutors relatively easy. Being able to deliver in multiple contexts adds considerably to the complexity of CTAT (and the amount of testing that needs to be done for each CTAT release), but CTAT's modular architecture helps achieve this goal.

CTAT supports ease-of-maintenance of example-tracing tutors in a number of ways (criterion 5). When tutors are built on a larger scale, and are used over longer periods of time, by successive cohorts of students, maintenance becomes an important concern. How easy or hard is it to modify tutors when requirements change? Although at this point in time, we have limited experience with regular maintenance of CTAT-built tutors over extended periods of time, we can point out that, serendipitously, many of the mechanisms in CTAT that enhance the ease of authoring, scaling up of tutored problems, and the flexibility of tutors also result in tutors that are easier to maintain. In particular, the Mass Production facility, designed to reduce repetitious authoring tasks and to support the production of many tutored problems, has the welcome side effect of making tutors easier to maintain. (It is easier to make consistent changes in a column in an Excel file than to have to open a large number of behavior graph files and scroll to the right link in the graph in each.) Similarly, many of the mechanisms that were designed to make example-tracing tutors more flexible have the effect of making them easier to maintain. Mechanisms such as unordered mode, range matches, and formulas effectively serve to "collapse" multiple behavior graph paths into a single one, which should significantly enhance maintainability. We note that our views on maintainability of CTAT tutors are still developing. Few of the tutors built in CTAT have had a multi-year lifespan, and only a few of them have been updated regularly, a real test of maintainability. So far, however, our experience indicates that it is manageable to maintain a set of example-tracing tutors. Not surprisingly, perhaps, it

is clear that maintaining a certain "discipline" in the authoring process is key (e.g., to rely on Mass Production as much as possible, and avoid minor variations in tutors and interfaces if they preclude re-using the same interface or Mass Production template).

Part of the motivation for CTAT has been from the start that *it supports research use of tutors (criterion 6)*. In particular, CTAT supports research into how people learn with ITSs, and what features make ITSs effective. In a typical research study, a researcher compares student learning with two tutor versions that differ only with respect to a single feature hypothesized to be a key influence on student learning. Many (but not all) of the tutors presented in this paper were built for such research purposes. For instance, the stoichiometry tutors were used to investigate the effect of polite problem statements, feedback, and hints (McLaren et al., 2007). The fractions tutors were used to investigate the effect of having students work with multiple graphical representations of fractions and of prompts for students to self-explain the representations (Rau, et al., 2009). CTAT supports research in two main ways: first, by lowering the skill threshold for tutor building, it makes it easier for researchers from non-technical disciplines (e.g., the learning sciences or educational psychology) to create and do experiments with CTAT-built tutors. Second, a number of key features are geared toward research use of tutors. Most importantly, CTAT tutors write detailed logs of all student-tutor interactions, which allow researchers to study students' learning trajectories in great detail (VanLehn et al., 2007). CTAT is tightly integrated with the DataShop (Koedinger et al., 2008), described above, a web-based repository and facility for analyzing tutor data. CTAT also makes it possible to create on-line assessments, essentially tutors with the tutoring turned off. These assessments can be used as automatically-graded on-line pre-and post-tests in experiments. Future versions of CTAT may include facilities for more easily assigning students to conditions, keeping track of what condition students are in, and making tutors adjust to the condition that students are in as a way of implementing the different conditions in an experiment. Interestingly, results from experiments involving ITSs, such as those described above, are beginning to have an influence on CTAT. Given results showing that adding worked examples to ITSs increases their effectiveness (McLaren et al., 2008a; 2008b; Salden et al., 2009), we have added a simple mechanism for the "backward fading" of examples.

CONCLUSION

In this paper, we report on our on-going project to develop the Cognitive Tutor Authoring Tools, highlighting the progress made in the first 7 years of the project. CTAT supports the development of example-tracing tutors, a novel way of building ITSs. Unlike other types of ITSs, example-tracing tutors evaluate students problem-solving actions by flexibly comparing them against generalized examples of demonstrated behavior. Using CTAT, example-tracing tutors can be created without programming, through programming by demonstration, in the form of an *editable behavior graph.* Example-tracing tutors meet VanLehn's (2006) minimal criterion for being regarded as ITSs, namely, that the system has an inner loop (i.e., it provides within-problem guidance and not just feedback on the final answer to each problem). In fact, they go well beyond VanLehn's criterion. They can follow students with respect to multiple solutions paths, whichever path they go down, even in the face of ambiguity of what path the student is actually following. Importantly, CTAT offers a number of mechanisms for generalizing examples, so that example-tracing tutors can recognize a wider range of student behavior than just the exact same steps that the author demonstrated in the exact same order. A

template-based Mass Production facility greatly facilitates the creation of multiple instances of the same problem type.

As we developed CTAT, frequently assisting authors using CTAT in creating real-world tutors, we gradually came to realize that an ITS authoring tool should do more than support cost-effective authoring of effective (and sophisticated) tutoring systems in a range of application domains. The tool should also support easy deployment and delivery of tutors in a variety of technical contexts. We have tried to address this requirement by providing a modular architecture, and by providing multiple options for several key components within this architecture. As an additional requirement for ITS authoring tools, tutors created with the tool should be easily maintainable (like all other software). Many of the features in CTAT that make tutors easier to create also make them easier to maintain (see also Aleven et al., 2009), but we add that we are still learning about the maintainability of example-tracing tutors. A final requirement is that an ITS authoring tool should (ideally) facilitate the use of tutors in empirical experiments, such as experiments testing the effect on student learning of particular ITS features. Such experiments will contribute substantially to further developing the science of learning with ITSs. Lessons learned in such experiments may also help shape the next versions of ITS authoring tools. Key research-related functionality in CTAT is the logging of student-tutor interactions. Also helpful is the fact that CTAT can be used to create on-line assessments.

Evidence of CTAT's effectiveness is its widespread and continued use ("vote-with-your-feet evidence"). At least 400 people have used CTAT, perhaps many more. Tutors have been built with CTAT for both real classroom and experimental use, across a wide range of domains, often by authors who were not directly associated with the CTAT project and had little prior experience in tutor development. During our courses and workshops, we frequently observe that it is relatively easy to get started with CTAT. Development time estimates from a range of projects indicate that CTAT greatly reduces the cost-effectiveness of building ITSs, compared to prior estimates of development time (Murray, 2003), which have ranged from 200-300 hours of development time per one hour of instructional time. CTAT has managed to lower this ratio to 50-100 hours. If one factors in the lower cost of personnel, then the savings may be as much as 4-8 times.

The CTAT authoring suite has now persisted through over 7 years of development. Although we do not feel we are done yet, we do feel that CTAT has reached a mature state where it is a viable and easy-to-learn tool for non-programmers. Other ITS authoring tools also have reached or are reaching a mature state where they support development of real-world tutors. It is our strong belief that in the years to come, authoring tools that non-programmers can use (e.g., CTAT, ASPIRE, the Assistments Builder) will contribute greatly to making ITSs widespread.

ACKNOWLEDGEMENTS

Brett Leber and Martina Rau contributed to this paper. We also thank Tanja Mitrovic and Noboru Matsuda for helpful comments. The paper also benefited greatly from the comments of the anonymous reviewers, who pushed us hard to state precisely what was claimed to be novel. The CTAT project was supported by grants from ONR Award No: N000140310220, the Grable Foundation, and the National Science Foundation under Award No. SBE0354420. Their contributions are gratefully acknowledged. The opinions expressed in the current paper do not represent any official position of the funding agencies.

REFERENCES

Ainsworth, S., Major, N., Grimshaw, S., Hayes, M., Underwood, J., Williams, B., & Wood, D. (2003). REDEEM: Simple intelligent tutoring systems from usable tools. In T. Murray, S. Blessing, & S. Ainsworth (Eds.) *Authoring Tools for Advanced Learning Environments* (pp. 205-232). Dordrecht, the Netherlands: Kluwer Academic Publishers.

Aleahmad, T., Aleven, V., & Kraut, R. (2008). Open community authoring of targeted worked example problems. In B. Woolf, E. Aimeur, R. Nkambou, & S. Lajoie (Eds) *Proceedings of the 9th International Conference on Intelligent Tutoring Systems* (pp. 216-227). Berlin: Springer.

Aleven, V., McLaren, B. M., & Sewall, J. (2009). Scaling up programming by demonstration for intelligent tutoring systems development: An open-access website for middle-school mathematics learning. *IEEE Transactions on Learning Technologies,* 2(2), 64-78.

Aleven, V., McLaren, B.M., Sewall, J., & Koedinger, K. R. (2006). The Cognitive Tutor Authoring Tools (CTAT): Preliminary evaluation of efficiency gains. In M. Ikeda, K. D. Ashley, & T. W. Chan (Eds.) *Proceedings of the 8th International Conference on Intelligent Tutoring Systems* (pp. 61-70). Berlin: Springer Verlag.

Aleven, V., Sewall, J., McLaren, B. M., & Koedinger, K. R. (2006). Rapid authoring of intelligent tutors for real-world and experimental use. In Kinshuk, R. Koper, P. Kommers, P. Kirschner, D. G. Sampson, & W. Didderen (Eds.) *Proceedings of the 6th IEEE International Conference on Advanced Learning Technologies* (pp. 847-851). Los Alamitos, CA: IEEE Computer Society.

Anderson, J. R. (1993). *Rules of the Mind.* Mahwah, NJ: Lawrence Erlbaum.

Anderson, J. R., Conrad, F. G., & Corbett, A. T. (1989). Skill acquisition and the LISP Tutor. *Cognitive Science,* 13, 467-506.

Anderson, J. R., Corbett, A. T., Koedinger, K. R., & Pelletier, R. (1995). Cognitive Tutors: Lessons learned. *The Journal of the Learning Sciences,* 4, 167-207.

Anderson, J. R., & Lebière, C. (1998). *The Atomic Components of Thought.* Mahwah, NJ: Lawrence Erlbaum.

Anderson, J.R., & Pelletier, R. (1991). A development system for model-tracing tutors. In L. Birnbaum (Ed.), *Proceedings of the International Conference of the Learning Sciences* (pp. 1-8). Charlottesville, VA: Association for the Advancement of Computing in Education.

Baker, R.S.J.d., Corbett, A.T., & Koedinger, K.R. (2007). The difficulty factors approach to the design of lessons in intelligent tutor curricula. *International Journal of Artificial Intelligence in Education,* 17(4), 341-369.

Beal, C. R., Walles, R., Arroyo, I., & Woolf, B. P. (2007). Online tutoring for math achievement: A controlled evaluation. *Journal of Interactive Online Learning,* 6, 43-55.

Blessing, S., Gilbert, S., Ourada, S., & Ritter, S. (2007). Lowering the bar for creating model-tracing intelligent tutoring systems. In R. Luckin, K. Koedinger, & J. Greer (Eds.), *Proceedings of the 13th International Conference on Artificial Intelligence in Education* (pp. 443–450). Amsterdam, the Netherlands: IOS Press.

Blessing, S.B. (2003). A programming by demonstration authoring tool for model-tracing tutors. In T. Murray, S. Blessing, & S. Ainsworth (Eds.) *Authoring Tools for Advanced Learning Environments* (pp. 93-119). Dordrecht, the Netherlands: Kluwer Academic Publishers.

Booth, J.L., Koedinger, K.R., & Siegler, R.S. (2007). The effect of prior conceptual knowledge on procedural performance and learning in algebra. In D.S. McNamara & J.G. Trafton (Eds.) *Proceedings of the 29th Annual Cognitive Science Society* (pp. 137-142). Austin, TX: Cognitive Science Society.

Brown, J. S. (1985). Idea amplifiers: New kinds of electronic learning environments. *Educational Horizons,* 63, 108–112.

Brusilovsky, P. (2003). Developing adaptive educational hypermedia systems: from design models to authoring tools. In T. Murray, S. Blessing & S. Ainsworth (Eds.) *Authoring Tools for Advanced Learning Environments* (pp. 377-409). Dordrecht, the Netherlands: Kluwer Academic Publishers.

Butcher, K., & Aleven, V. (2008). Diagram interaction during intelligent tutoring in geometry: Support for knowledge retention and deep transfer. In C. Schunn (Ed.) *Proceedings of the 30th Annual Meeting of the Cognitive Science Society, CogSci 2008* (pp. 1736-1741). New York, NY: Lawrence Erlbaum.

Conati, C., & VanLehn, K. (2000). Toward computer-based support of meta-cognitive skills: A computational framework to coach self-explanation. *International Journal of Artificial Intelligence in Education,* 11(4), 389-415.

Corbett, A. T., & Anderson, J. R. (1995). Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modeling and User-Adapted Interaction,* 4, 253-278.

Crowley R. S., & Medvedeva, O. (2006). An intelligent tutoring system for visual classification problem solving. *Artificial Intelligence in Medicine,* 36(1), 85-117.

Devedzic, V. & Harrer, A. (2005). Software patterns in ITS architectures. *International Journal of Artificial Intelligence in Education*, 15(2), 63-94.

du Boulay, B. (2006). Commentary on Kurt VanLehn's "The Behavior of Tutoring Systems." *International Journal of Artificial Intelligence in Education,* 16(3), 267-270.

Friedman-Hill, E. (2003). *Jess in Action.* Greenwich, CT: Manning Publications Co.

Goguadze, G. & Tsigler, I. (2007). Authoring interactive exercises in ActiveMath. In: *Proceedings of the Mathematical User-Interfaces Workshop (MathUI-2007)* at the 6th Mathematical Knowledge Management Conference (MKM-2007), Linz, Austria, June 27, Online-Proceedings, 2007.

Graesser, A.C., Chipman, P., Haynes, B.C., & Olney, A. (2005). AutoTutor: An intelligent tutoring system with mixed-initiative dialogue. *IEEE Transactions in Education,* 48, 612-618.

Halff, H.M., Hsieh, P.Y., Wenzel, B.M., Chudanov, T.J., Dirnberger, M.T., Gibson, E.G., & Redfield, C.L. (2003). Requiem for a development system: reflections on knowledge-based, generative instruction. In T. Murray, S. Blessing & S. Ainsworth (Eds.) *Authoring Tools for Advanced Learning Environments* (pp. 33-59). Dordrecht, the Netherlands: Kluwer Academic Publishers.

Harrer, A., McLaren, B.M., Walker, E., Bollen, L., & Sewall, J. (2006). Creating Cognitive Tutors for collaborative learning: steps toward realization. *User Modeling and User-Adapted Interaction: The Journal of Personalization Research* (UMUAI), 16, 175-209.

Harrer, A., Pinkwart, N., McLaren, B.M., & Scheuer, O. (2008). The Scalable Adapter design pattern: Enabling interoperability between educational software tools. *IEEE Transactions on Learning Technologies*, 1(2), 131-143.

Koedinger, K.R., & Aleven, V. (2007). Exploring the Assistance Dilemma in experiments with Cognitive Tutors. *Educational Psychology Review*, 19(3), 239-264.

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 J. C. Lester, R. M. Vicario & F. Paraguaçu (Eds.) *Proceedings of the Seventh International Conference on Intelligent Tutoring Systems* (pp. 162-174). Berlin: Springer Verlag.

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(1), 30-43.

Koedinger, K. R., & Corbett, A. T. (2006). Cognitive Tutors: Technology bringing learning science to the classroom. In K. Sawyer (Ed.) *The Cambridge Handbook of the Learning Sciences* (pp. 61-78). New York: Cambridge University Press.

Koedinger, K., Cunningham, K., Skogsholm A., & Leber, B. (2008). An open repository and analysis tools for fine-grained, longitudinal learner data. In R.S.J.d. Baker, T. Barnes, & J.E. Beck (Eds.) *Educational Data Mining 2008: 1st International Conference on Educational Data Mining, Proceedings* (pp. 157-166). Montreal, Quebec, Canada. June 20-21, 2008. http://www.educationaldatamining.org/EDM2008/index.php?page=proceedings

Koedinger, K. R., Suthers, D. D., & Forbus, K. D. (1999). Component-based construction of a science learning space. *International Journal of Artificial Intelligence in Education*, 10, 292-313.

Kumar, R., Rosé, C. P., Wang, Y. C., Joshi, M., & Robinson, A. (2007). Tutorial dialogue as adaptive collaborative learning support. In R. Luckin, K. Koedinger & J. Greer (Eds.) *Proceedings of the 13th International Conference on Artificial Intelligence in Education* (pp. 383-390). Amsterdam: IOS Press.

Libbrecht, P. & Groß, C. (2006). Experience report writing LeActiveMath Calculus. In J. Borwein & W. Farmer (Eds.) *Proceedings of Mathematical Knowledge Management 2006* (pp. 251–265). Berlin: Springer-Verlag.

Lieberman, H. (Ed.) (2001). *Your wish is my command: Programming by example.* San Francisco, CA: Morgan Kaufmann.

Martin, B., & Mitrovic, A. (2002). Automatic problem generation in constraint-based tutors. In S.A, Cerri, G. Gouarderes & F. Paraguacu (Eds.) *Intelligent Tutoring Systems: 6th International Conference* (ITS-2002). (pp. 388-398). Berlin: Springer.

Matsuda, N., Cohen, W. W., Sewall, J., Lacerda, G., & Koedinger, K. R. (2008). Why tutored problem solving may be better than example study: Theoretical implications from a simulated-student study. In B. P. Woolf, E. Aimeur, R. Nkambou & S. Lajoie (Eds.) *Proceedings of the International Conference on Intelligent Tutoring Systems* (pp. 111-121). Berlin: Springer.

Matsuda, N., Cohen, W. W., Sewall, J., Lacerda, G., & Koedinger, K. R. (2007). Evaluating a simulated student using real students data for training and testing. In C. Conati, K. McCoy & G. Paliouras (Eds.) *Proceedings of the International Conference on User Modeling* (pp. 107-116). Berlin: Springer.

McLaren, B.M., Lim, S., & Koedinger, K.R. (2008a). When and how often should worked examples be given to students? New results and a summary of the current state of research. In B. C. Love, K. McRae & V. M. Sloutsky (Eds.) *Proceedings of the 30th Annual Conference of the Cognitive Science Society* (pp. 2176-2181). Austin, TX: Cognitive Science Society.

McLaren, B.M., Lim, S., & Koedinger, K.R. (2008b). When is assistance helpful to learning? Results in combining worked examples and intelligent tutoring. In B. Woolf, E. Aimeur, R. Nkambou, & S. Lajoie (Eds) *Proceedings of the 9th International Conference on Intelligent Tutoring Systems* (pp. 677-680). Berlin: Springer Verlag.

McLaren, B.M., Lim, S., Yaron, D., & Koedinger, K.R. (2007). Can a polite intelligent tutoring system lead to improved learning outside of the lab? In R. Luckin, K. Koedinger & J. Greer (Eds.) *Proceedings of the 13th International Conference on Artificial Intelligence in Education* (pp. 433-440). Amsterdam: IOS Press.

McLaren, B.M., Lim, S., Gagnon, F., Yaron, D., & Koedinger, K.R. (2006). Studying the effects of personalized language and worked examples in the context of a web-based intelligent tutor. In M. Ikeda, K. D. Ashley, & T. W. Chan (Eds.) *Proceedings of the 8th International Conference on Intelligent Tutoring Systems* (pp. 318-328) Berlin: Springer Verlag.

McLaren, B.M., Koedinger, K.R., Schneider, M., Harrer, A., & Bollen, L. (2004). Toward Cognitive Tutoring in a collaborative, web-based environment. In *Engineering Advanced Web Applications,* From the

Proceedings in Connection with the 4th International Conference on Web Engineering (ICWE 2004) (pp. 167-179). Munich: Rinton Press.

Melis, E., Andrès, E., Büdenbender, J., Frischauf, A., Goguadze, G., Libbrecht, P., Pollet, M., & Ullrich, C., (2001). ActiveMath: A generic and adaptive web-based learning environment. *International Journal of Artificial Intelligence in Education,* 12, 385-407.

Mitrovic, A., Martin, B., Suraweera, P., Zakharov, K., Milik, N., Holland, J., & McGuigan, N. (this issue). ASPIRE: an authoring system and deployment environment for constraint-based tutors. *International Journal of Artificial Intelligence in Education.*

Mitrovic, A., McGuigan, N., Martin, B., Suraweera, P., Milik, N., Holland, J. (2008). Authoring constraint-based tutors in ASPIRE: A case study of a capital investment tutor. In *Proceedings of theWorld Conference on Educational Multimedia, Hypermedia & Telecommunications (ED-MEDIA 2008)* (pp. 4607-4616). Chesapeake, VA: AACE.

Mitrovic, A., & Ohlsson, S. (1999). Evaluation of a constraint-based tutor for a database language. *International Journal of Artificial Intelligence in Education,* 10, 238-256.

Mitrovic, A., Suraweera, P., Martin, B., Zakharov, K., Milik, N., & Holland, J. (2006). Authoring Constraint-Based Tutors in ASPIRE. In M. Ikeda, K. D. Ashley & T. W. Chan (Eds.) *Proceedings of the 8th International Conference on Intelligent Tutoring Systems* (pp. 41-50). Berlin: Springer Verlag.

Moore, J. L. (1992). Winch simulations: Multiple, linked representations of linear functions. In C. Frasson, G. Gauthier & G. I. McCalla (Eds.) *Proceedings of the Second International Conference on Intelligent Tutoring Systems* (pp. 111-115). Berlin: Springer-Verlag.

Mostow, J., & Beck, J. (2007). When the rubber meets the road: Lessons from the in-school adventures of an automated Reading Tutor that listens. In B. Schneider & S.-K. McDonald (Eds*.) Conceptualizing Scale-Up: Multidisciplinary Perspectives* (pp. 183-200). Lanham, MD: Rowman & Littlefield.

Munro, A. (2003). Authoring simulation-centered learning environments with Rides and Vivids. In T. Murray, S. Blessing & S. Ainsworth (Eds.) *Authoring Tools for Advanced Learning Environments* (pp. 61-91). Dordrecht, the Netherlands: Kluwer Academic Publishers.

Murray, T. (2003). An overview of intelligent tutoring system authoring tools: Updated analysis of the state of the art. In T. Murray, S. Blessing & S. Ainsworth (Eds.) *Authoring Tools for Advanced Learning Environments* (pp. 491-544). Dordrecht, the Netherlands: Kluwer Academic Publishers.

Murray, T., Blessing, S., & Ainsworth, S. (Eds.) (2003). *Authoring Tools for Advanced Learning Environments: Toward Cost-Effective Adaptive, Interactive and Intelligent Educational Software*. Dordrecht, the Netherlands: Kluwer Academic Publishers.

Myers, B.A., McDaniel, R.G., & Kosbie, D.S. (1993). Marquise: Creating complete user interfaces by demonstration. In S. Ashlund, A. Henderson, E. Hollnagel, K. Mullet, T. White (Eds.), *Proceedings of INTERCHI'93: Human Factors in Computing Systems* (pp. 293-300). Amsterdam: IOS Press.

Newell, A., & Simon, H. A. (1972). *Human Problem Solving.* Englewood Cliffs, NJ: Prentice-Hall.

Ogan, A., Aleven, V., & Jones, C. (2008). Pause, predict, and ponder: Use of narrative videos to improve cultural discussion and learning. In M. Czerwinski, A.M. Lund & D.S. Tan (Eds.), *Proceedings of the Twenty-Sixth Annual SIGCHI Conference on Human Factors in Computing Systems* (pp. 155-162). New York, NY: ACM.

Paas, F., & Van Merrienboer, J. (1994). Variability of worked examples and transfer of geometry problem-solving skills: A cognitive-load approach. *Journal of Educational Psychology,* 86, 122-133.

Rau, M., Aleven, V., & Rummel, N. (2009). Intelligent tutoring systems with multiple representations and self-explanation prompts support learning of fractions. In V. Dimitrova, R. Mizoguchi, B. du Boulay, & A.

Graesser (Eds.) *Proceedings of the 14th International Conference on Artificial Intelligence in Education, AIED 2009* (pp. 441-448). Amsterdam: IOS Press.

Razzaq, L., & Heffernan, N.T. (2006). Scaffolding vs. hints in the Assistment System. In M. Ikeda, K. D. Ashley, & T. W. Chan (Eds.) *Proceedings of the 8th International Conference on Intelligent Tutoring Systems (ITS 2006)* (pp. 635-644). Berlin: Springer Verlag.

Razzaq, L., Feng, M., Nuzzo-Jones, G., Heffernan, N.T., Koedinger, K. R., Junker, B., Ritter, S., Knight, A., Aniszczyk, C., Choksey, S., Livak, T., Mercado, E., Turner, T.E., Upalekar. R., Walonoski, J.A., Macasek, M.A., Rasmussen, K.P. (2005). The Assistment project: Blending assessment and assisting. In C.K. Looi, G. McCalla, B. Bredeweg & J. Breuker (Eds.) *Proceedings of the 12th International Conference on Artificial Intelligence In Education* (pp. 555-562). Amsterdam: IOS Press.

Razzaq, L., Patvarczki, J., Almeida, S. F., Vartak, M., Feng, M., Heffernan, N. T., & Koedinger, K. R. (2009). The Assistment Builder: Supporting the life cycle of tutoring system content creation. *IEEE Transactions on Learning Technologies,* 2(2), 157-166.

Reed, S. K. (2005). From research to practice and back: The Animation Tutor project. *Educational Psychology Review,* 17(1), 55-82.

Rickel, J., & Johnson, W.L. (1999). Animated agents for procedural training in virtual reality: Perception, Cognition, and Motor Control. *Applied Artificial Intelligence*, 13, 343-382.

Ritter, S., Blessing, S., & Wheeler, L. (2003). Tools for component-based learning environments. In T. Murray, S. Blessing & S. Ainsworth (Eds.) *Authoring Tools for Advanced Learning Environments* (pp. 467-489). Dordrecht, the Netherlands: Kluwer Academic Publishers.

Ritter, S., & Koedinger, K. R. (1997). An architecture for plug-in tutoring agents. *International Journal of Artificial Intelligence in Education*, 7 (3/4), 315-347.

Rodner, E., & Denzler, J. (2009). Theory of learning with few examples and object localization. In J. Denzler & M. Koch (Eds.), *Computer Vision in Camera Networks for Analyzing Complex Dynamic Natural Scenes*, Dagstuhl Event Proceedings 08422. Dagstuhl, Germany: Leibniz-Zentrum für Informatik. http://drops.dagstuhl.de/opus/volltexte/2009/1861

Roll, I., Ryu, E., Sewall, J., Leber, B., McLaren, B. M., Aleven, V., & Koedinger, K. R. (2006). Towards teaching metacognition: Supporting spontaneous self-assessment. In M. Ikeda, K. D. Ashley & T. W. Chan (Eds.) *Proceedings of the 8th International Conference on Intelligent Tutoring Systems* (pp. 738-740). Berlin: Springer Verlag.

Roll, I., Aleven, V., McLaren, B., & Koedinger, K. (2007). Can help seeking be tutored? Searching for the secret sauce of metacognitive tutoring. In R. Luckin, K. Koedinger, & J. Greer (Eds.) *Proceedings of the 13th International Conference on Artificial Intelligence in Education* (pp. 203–210). Amsterdam, the Netherlands: IOS Press.

Rosé, C. P., Kumar, R., Aleven, V., Robinson, A., & Wu, C. (2006). CycleTalk: Data driven design of support for simulation based learning. *International Journal of Artificial Intelligence and Education,* 16, 195-223.

Salden, R., Aleven, V., Renkl, A., & Schwonke, R. (2009). Worked examples and tutored problem solving: Redundant or synergistic forms of support? *Topics in Cognitive Science,* 1(1), 203-213.

Schwonke, R., Renkl, A., Krieg, C., Wittwer, J., Aleven, V., & Salden, R. (2009). The worked-example effect: Not an artefact of lousy control conditions. *Computers in Human Behavior,* 25(2), 258-266.

Tecuci, G., & Keeling, H. (1999). Developing an intelligent educational agent with Disciple. *International Journal of Artificial Intelligence in Education,* 10, 221-237

Towne, D. (2003). Automated knowledge acquisition for intelligent support of diagnostic reasoning. In T. Murray, S. Blessing & S. Ainsworth (Eds.) *Authoring Tools for Advanced Learning Environments* (pp. 121-147). Dordrecht, the Netherlands: Kluwer Academic Publishers.

VanLehn, K., Koedinger, K.R., Skogsholm, A., Nwaigwe, A., Hausmann, R.G.M., Weinstein, A., & Billings, B. (2007). What's in a step? Toward general, abstract representations of tutoring system log data. In C. Conati, K. McCoy, & G. Paliouras (Eds.) *Proceedings of the 11th International Conference on User Modeling 2007* (pp. 455-459). *Berlin: Springer Verlag.*

VanLehn, K. (2006). The behavior of tutoring systems. *International Journal of Artificial Intelligence in Education,* 16(3), 227-265.

VanLehn, K., Lynch, C., Schultz, K., Shapiro, J.A., Shelby, R.H., Taylor, L., Treacy, D., Weinstein, A., & Wintersgill, M. (2005). The Andes physics tutoring system: Lessons learned. *International Journal of Artificial Intelligence in Education*, 15(3), 147-204.

Weber, G., & Brusilovsky, P. (2001). ELM-ART: An adaptive versatile system for web-based instruction. *International Journal of Artificial Intelligence in Education,* 12(4), 351-384.

Wenger, E. (1987). *Artificial Intelligence and Tutoring Systems: Computational and Cognitive Approaches to the Communication of Knowledge.* Los Altos, CA: Morgan Kaufmann.

Wolf, L., & Martin, I. (2005). Robust boosting for learning from few examples. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '05)* (pp. 359-364). Los Alamitos, CA: IEEE Computer Society.

Woolf, B. P., & Cunningham, P. (1987). Building a community memory for intelligent tutoring systems. In K. Forbus & H. Shrobe (Eds.), *Proceedings of the Sixth National Conference on Artificial Intelligence* (pp. 82-89). Menlo Park, CA: AAAI Press.

Wylie, R. (2007) Are we asking the right questions? Understanding which tasks lead to robust learning of the English article system. In R. Luckin, K. Koedinger & J. Greer (Eds.) *Proceedings of the 13th International Conference on Artificial Intelligence in Education* (pp. 709-710). Amsterdam, the Netherlands: IOS Press.

Yaron, D., Evans, K., & Karabinos, M. (2003). Scenes and Labs Supporting Online Chemistry. Paper presented at the 83rd Annual AERA National Conference, April 2003.