The Behavior of Tutoring Systems

Kurt VanLehn, *LRDC, University of Pittsburgh, Pittsburgh, PA, USA*
VanLehn@pitt.edu

Abstract. Tutoring systems are described as having two loops. The outer loop executes once for each task, where a task usually consists of solving a complex, multi-step problem. The inner loop executes once for each step taken by the student in the solution of a task. The inner loop can give feedback and hints on each step. The inner loop can also assess the student's evolving competence and update a student model, which is used by the outer loop to select a next task that is appropriate for the student. For those who know little about tutoring systems, this description is meant as a demystifying introduction. For tutoring system experts, this description illustrates that although tutoring systems differ widely in their task domains, user interfaces, software structures, knowledge bases, etc., their behaviors are in fact quite similar.

Keywords. Intelligent tutoring systems, knowledge components, learning events, tutoring

INTRODUCTION

Human tutoring is widely believed to be the most effective form of instruction on the planet, and experimental work confirms that expert human tutors can produce extremely large learning gains (Bloom, 1984). Ever since computers were invented, they seemed capable of becoming untiring and economical alternatives to expert human tutors. This dream has proved difficult to achieve, but significant progress has been made.

This focuses mostly on intelligent tutoring systems (ITS), a particularly effective eductional technology that began in the late 1960s and is just now emerging from laboratories into the field. Although ITS tutorials (Beck, Stern, & Haugsjaa, 1996; Corbett, Koedinger, & Anderson, 1997; Rickel, 1989; Shute & Psotka, 1996; Woolf, 1992) emphasized their structure (i.e., the software modules and knowledge sources), this description emphasizes their behavior. That is, it discusses what tutoring systems tend to do, and some important design options in the space of tutorial behaviors. Although a great deal is known about which design options are effective under which circumstances, it would take a book to cover the empirical work. Thus, this description will only present options and not evaluate them.

This description was written for two audiences. For readers who know little about ITSs, this will serve as a non-technical introduction. For readers who know a great deal about ITSs, this serves as a proposed standardization of the descriptions of ITS behavior. The underlying claim is that despite vast differences in their internal structure, ITSs behave in fundamentally similar ways. Although only a few pedagogical features have been invented, different tutoring systems offer different combinations of features and they tend to use different terminology for the same concepts. This has led to the impression that ITSs are such complex technology that only those with advanced degrees in Artificial Intelligence can understand and develop them. The occurrence of "intelligent" in "ITS" bolsters this unfortunate impression. This description is intended to counteract this impression by demonstrating that ITSs are simple to understand if one sticks to just one name for each concept and focuses on their behavior instead of their internal structures.

Table 1
Terminology

Term	**Brief definition**
Task domain	The information and skills being taught by the tutor.
Task	A multi-minute activity that can be skipped or interchanged with other tasks.
Step	Completing a task consists of multiple steps. Each is a user interface event.
Knowledge component	A task domain concept, principle, fact, etc. Any fragment of the persistent, domain-specific information that should be used to accomplish tasks.
Learning event	A mental event; the construction or application of a knowledge component, often while trying to achieve a task.
Outer loop	An ITS behaves as if it had an outer loop (iteration) over tasks.
Inner loop	An ITS behaves as if it had an inner loop over steps.
Incorrect	Inconsistent with the instructional objectives of the tutor.

TERMINOLOGY

In order to capture the similarities among the behaviors of tutoring systems, it is essential to introduce some technical terms. The terms are listed in Table 1 and discussed in this section. In order to illustrate them and other concepts introduced later, an appendix describes 6 tutoring systems.

The competence that the tutoring system tries to get students to acquire is called the *task domain*. For the 6 systems in the appendix, the tasks domains are:

- Algebra Cognitive Tutor: Problem solving in a high-school algebra course.
- Andes: Quantitative problem solving in introductory college physics.
- AutoTutor: Several, including qualitative problem solving in college physics.
- Sherlock: Troubleshooting a large piece of avionic electronic equipment.
- SQL-Tutor: Constructing queries to relational databases in the language SQL.
- Steve: Operating complex equipment, such as a room-sized air compressor.

Except for Sherlock, these systems were designed to be used as part of a course. That is, they assume the student has a textbook, an instructor and perhaps other resources as well. Sherlock was intended for on-the-job training.

Most tutoring consists of a chronological sequence of *tasks* that the student does with the aid of the tutor. For the 6 illustrative tutors, a task is:

- Algebra Cognitive Tutor: Solving a complex, multi-part algebra problem. A single problem might require filling in a table, plotting points on a graph and solving an equation.
- Andes: Solving a standard quantitative physics problem. Such a problem's solution typically involves drawing a vector diagram, writing several equations, and solving the system of equations algebraically.
- AutoTutor: Answering qualitative physics questions, such as "A massive truck and a lightweight car have a head-on collision. Assuming the vehicles had the same initial velocity, which suffers the greater impact? Explain your reasoning."
- Sherlock: Finding a fault in the equipment (which is simulated) and repairing it. This involves taking electrical measurements, setting switches and dials and replacing parts.
- SQL-Tutor: Writing an SQL query that will retrieve data from a given database. A description of the data to be retrieved is given.
- Steve: Executing a particular procedure under specified conditions, such as starting up a naval diesel engine (simulated) given that the #2 heat exchanger is down for repairs.

A task usually takes several minutes to an hour or so. Most tutors assume that the student is working alone on a task, but some (e.g., Zachary et al., 1999) have the student work as one member of a multi-student team.

The tasks and the tutor's user interface are usually designed so that completing a task requires multiple *steps*, where a step is a user interface action that the student takes in order to achieve a task. For instance, suppose the task is to solve the equation 2.34*x-19.01=23.44 and that the student has entered three lines:

2.3*x = 23.44 +19.01
2.3*x = 42.45
x = 18.46

Each line is a step, so the student solved the problem in three steps. Not all user interface actions are steps. For instance, clicking on the "Save" button would not count as a step because it doesn't contribute directly to solving the equation. Clicking on a "Help" button also does not count as a step, even though the resulting helpful information from the tutor may well be crucial for this student's solving of the problem. Some more examples of a single step are:

- Andes: Defining "mc" to be the mass of the car.
- Andes: Drawing a Cartesian coordinate system rotated so that the x-axis is parallel to an inclined plane.
- Andes: Typing in the equation "mc = 20 kg."
- Sherlock: Measuring the voltage of pin 21 on cable jack B.
- Sherlock: Turning off the power supply.
- Sherlock: Replacing the A1A2A5 printed circuit card.
- AutoTutor: In response to the tutor's question, "Can you say more?", typing in the response: "It's because of one of Newton's laws. I forget which one. Third, maybe?"
- Steve: Pulling out the dipstick and checking the oil level.
- SQL-Tutor: Filling in the WHERE field with "aawon>1 and year>=1988."

Let us use *knowledge* for the information possessed by students that determines their behavior on task. Let us also assume that knowledge can be decomposed, and let us use the term *knowledge component* for the units into which it is decomposed. A knowledge component can be a principle, a concept, a rule, a procedure, a fact, an association or any other fragment of task-specific information. Despite the connotations of "knowledge", a knowledge component can be incorrect, in that instructors would rather that students not apply this knowledge component while achieving a task.

A *learning event* is the construction or application of a knowledge component, often while trying to achieve the task. Learning events are mental events, whereas steps are physical events. A learning event occurs in the mind of the student where it cannot be observed, whereas a step occurs on the user interface and the computer can observe it. In the algebra example mentioned earlier, the step x=18.46 can be considered to result from three learning events: 2.3*x=42.45 → x=42.45/2.3 → x=18.4565… → x=18.46. Although some tutoring systems (e.g., Steve) insist on a one-to-one relationship between steps and learning events, many tutoring systems (e.g., Andes, Algebra Cognitive Tutor) assume multiple learning events per step.

The following examples illustrate the terms "learning event" and "knowledge component":

- Andes: Suppose the problem is "A Porsche starts at rest and achieves a speed of 10 m/s in 3 seconds. Assuming its acceleration is constant and that it travels in a straight line, how much distance does it cover in that time?" After defining d_p as the displacement, t_p as the time and vf_p is the final velocity of the dragster, the student enters "d_p=(vf_p/2)*t_p." This step results from the following 3 knowledge components, each of which produces a learning event (a problem-specific equation, in this case) when applied to this problem:

  - An object that starts at rest has an initial velocity of 0, so vi_p = 0.
  - An object that accelerates constantly has an average velocity that is the average of its initial and final velocities, so v_avg_p = (vi_p+vf_p)/2.
  - An object that travels in a straight line has a displacement equal to its average velocity times the duration of travel, so d_p = v_avg_p*t_p.

  The step results from algebraically combining the 3 learning events, d_p=v_avg_p*t_p, v_avg_p=(vi_p+vf_p)/2 and vi_p=0. Andes does not represent the algebraic manipulations themselves as learning events because it does not tutor them.

- Steve: Suppose the student begins an engine start-up procedure by pushing the "lamp test" button beneath the "low oil" light, causing the light to illuminate. The student has applied the knowledge component that "Before starting the engine, the lamps of all indicators that have a lamp-test button should be checked by pushing the button and confirming that the lamp lights." The particular learning event is that "because the low-oil light has a lamp test button beneath it, and I have not yet checked it, I should push the lamp test button."
- AutoTutor: Suppose the tutor asks, "What can you say about the keys' motion?" and the student answers, "The keys are in freefall because the only force acting on them is gravity." It uses statistical language understanding techniques to extract two learning events from this step: "The keys are in freefall" and "The only force acting on the keys is the force due to gravity." These learning events follow from the following knowledge components: "If the only force acting on an object is the force due to gravity, then the object is in freefall." "If an object is falling and air resistance can be neglected, then the only force acting on it is the force due to gravity."
- SQL-Tutor: Given the problem shown in Figure 5, suppose that the student's step is filling in the FROM field with "movie." The student has used the knowledge that "If there is an attribute in the SELECT clause, then the name of the corresponding table should be given in the FROM clause." This knowledge component produced the learning event, "Because the problem asks for the titles and numbers of some movies, and these fields appear in the 'movie' table, I should put 'movie' in the FROM field."

Human reasoning can be analyzed at many grain sizes, from the level of neuronal firings on up. For instance, instead of assigning just three learning events to the x=18.46 step (2.3*x=42.45 $\rightarrow$ x=42.45/2.3 $\rightarrow$ x=18.4565… $\rightarrow$ x=18.46), one could assign just one: 2.3*x=42.45 $\rightarrow$ x=18.46. Moreover, even at the same grain size, there might be several plausible ways to divide up a line of reasoning into learning events, or a body of knowledge into knowledge components. How can one determine which decompositions to use?

There are empirical techniques for establishing the knowledge components that students themselves use. One technique involves modeling transfer: given levels of mastery on a set of knowledge components, predict the performance on a new problem (Singley & Anderson, 1989). For instance, suppose a student has mastered 15 of the 20 knowledge components required to do a task. This predicts the student's behavior on the task—where the student will ask for help, how long it will take to do the task, which errors occur, etc. However, these predictions can be inaccurate if the assumed knowledge components are not accurate reflections of how the student actually understands the task domain. Thus, performance on transfer experiments can be used to determine a tutoring system's representation of knowledge components.

A second empirical technique for determining students' knowledge components involves predicting learning curves (Croteau, Heffernan, & Koedinger, 2004). The learning curve of a knowledge component graphs the durations of its learning events (or their probability of error). According to theory, a learning curve should be a smoothly descending power-law or exponential curve. For instance, the first learning event for a knowledge component may take 50 seconds, because the student is constructing the knowledge component by referring to the textbook and asking the tutoring system for help. The next learning event might take only 25 seconds because the student is reconstructing the knowledge component. On the third learning event, the student recalls the knowledge component after a brief struggle, so the event takes 12 seconds. The fourth event takes 6 seconds, the fifth takes 3 seconds, and so on. However, if the representation of knowledge is inaccurate, a knowledge component's learning curve may have a huge jump in the middle or be quite jagged. Learning curves can be used to determine the tutoring system's representation of knowledge components.

Because tutoring system developers typically cannot afford to do the empirical studies necessary to determine the knowledge components that most accurately represent students' cognition, the choice of grain size is usually determined by the instructional objectives of the designers. For instance, consider the decision about whether to model the algebra step mentioned earlier with a single learning event (2.3*x=42.45 → x=18.46) or a sequence of 3 learning events (2.3*x=42.45 → x=42.45/2.3 → x=18.4565… → x=18.46). If the designers of the tutoring system want to detect rounding errors and give error specific advice, then they would need to use the latter, fine-grained analysis, and also include the incorrect learning event x=42.45/2.3 → x=18.45 associated with some error-specific feedback such as "Incorrect. You may have made a mistake when rounding the answer off to 2 decimal places. Try again." On the other hand, if the designers assume students can and should diagnose rounding errors themselves, then they can use the coarse-grained analysis of this step. Thus, if the student enters x=18.45, the tutoring system will give its default response, such as "Incorrect. See if you can find your error and fix it." Thus, the designers' decisions about what learning events should be detected and advised by the tutor determine what knowledge components are included in the tutor's representation of knowledge.

By analyzing tutoring as a set of tasks each of which can be solved by multiple steps, we have divided the duties of a tutoring system into two loops: an *outer loop* over tasks, and an *inner loop* over steps. The main duty of the outer loop is to select a task for the student that will help the student learn. The inner loop is responsible for both eliciting the right steps from the student and helping the student to learn. As an illustration, Table 2 shows the two loops for a particularly simple tutoring system. The outer loop executes once per task, while the inner loop executes once per step. This tutoring system always gives feedback after a student step, but it may or may not give the student a hint on a step just before the student attempts it. In real tutoring systems, the inner loop may be more complex that the one shown in Table 2. Here are some examples:

- Andes: After each step, the student gets immediate feedback just as shown in Table 2. The step is colored red if it is incorrect, and green if it is correct. However, Andes never gives unsolicited hints. It gives a hint only if the student clicks on a hint button, in which case Andes picks a step that the student has not yet done, and gives the student a hint on how to do that step.
- Algebra Cognitive Tutor: The inner loop gives immediate feedback after each student step, just like Andes. It also gives hints when asked, just like Andes. However, it also gives a hint when the student has just taken two incorrect steps in a row.
- SQL-Tutor: The inner loop offers neither feedback nor hints as the student enters steps. It is silent until the student clicks on the Submit Answer button. Then the tutor analyzes all the

Table 2
The two loops of an immediate feedback tutoring system

```
Until tutoring is done, do:
    • Tutor poses a task
    • Until the task is achieved, do:
        o Tutor may give a hint
        o Student does a step
        o Tutor gives feedback on the step
    • Student submits the solution to the task
```

steps entered so far and decides what kind of help to give. The default policy is to start with minimal feedback and go upwards, giving more information on each subsequent submission. However, at any submission, students can override this default policy by selecting the level of feedback they want from a menu.

- Sherlock: As the student troubleshoots the equipment, Sherlock gives feedback only if the student has just taken a step that would severely damage the equipment or the student. It will not permit such safety violations. After the student has finished solving the troubleshooting problem, Sherlock offers various forms of reflective activities. For example, it displays a list of the student's steps alongside a list of an expert's steps. The student can click on steps to get feedback.

There are many ways to organize an inner loop because hints and feedback can be immediate or delayed, and can be solicited (given in response to a request by the student) or unsolicited. A later section will discuss this part of the design space in detail. However, the one constant in all inner loops is that they deal with steps, as opposed to tasks.

Lastly, there is the contentious term "*incorrect*." It is widely used in the literature of tutoring systems, so many educators reject tutoring systems because they reject the whole idea of an absolute standard of correctness. However, the term really has a benign technical meaning. A step, learning event or knowledge component is incorrect *only if the tutoring system should say something about it in order to meet its current instructional objectives*. That is, correctness is not absolute, but relative to the tutor's current goals for the student. In general, a step that is correct for one student at one time might be incorrect for a different student or for the same student at a different time. However, in most tutoring systems, including all those in the Appendix, correctness does not vary over time or students. Nonetheless, it is still defined pedagogically. For instance, one pedagogical issue is what to do about a step that is on a solution path that is much longer than the ideal solution path. Should the steps of the long solution be considered incorrect or correct? Typically, such questions can only be answered by course instructors, who should be part of the tutoring system's development team.

Because correctness is relative to instructional objectives, it is often important to avoid feedback messages that portray the tutor as an absolute authority. In Sherlock's case, for instance, instructors at one Air Force base disagreed with the correctness classifications of instructors at another Air Force base because the two bases placed different values on troubleshooting time and replacement part costs. Because correctness is relative, Sherlock's feedback messages do not say, "Action X is wrong" but instead say, "Action X saves time but raises the cost of the solution; some experts would do Y instead."

It is appropriate to end this discussion of terminology with the term "tutor." Given the prevalence of constructivist ideas and terminology, the old-fashion, authoritarian connotations of the term "tutor" are an impediment to the widespread dissemination of the technology. However, a tutoring system does not have to replace a teacher or run an after-school remediation session. Its role in the student's learning ecology can be anything—a smart piece of paper; an encouraging coach; a stimulating peer, etc. The technology is quite neutral. It is the pedagogical policies of the instructors and the developers that determine how the "tutor" behaves and the role it plays. For instance, tutoring systems have been built that do not teach their task domain at all, but instead encourage students to discover its principles (e.g., Shute & Glaser, 1990).

As a more detailed illustration, consider the role of the Algebra Cognitive Tutor, which is currently the most widely used ITS in the world. The Algebra Cognitive Tutor is typically used two days a week in high school algebra classes; the other three days are devoted to group work, projects, and other activities that do not involve the tutoring system (Koedinger, Anderson, Hadley, & Mark, 1997). An ethnographic study (Schofield, 1995; Schofield & Evans-Rhodes, 1989) found on days that the class used the tutoring system (actually, they used a predecessor that taught geometry), a vibrant culture sprang up that any constructivist teacher would be proud of. As the students worked with the tutor, they often talked with their neighbors—seeking help, giving help, comparing progress or just socializing. When both the tutor's help and their neighbor's help failed, students raised their hands to get individualized help from the teacher. The teachers preferred this, as it let them concentrate on discussing complex issues with students who really needed such a discussion. On the other hand, some students opted out of this noisy collaborative work, and preferred to work privately, sharing their errors and help requests only with the tutoring system and the teacher. The ethnographic study suggests that almost all students were enthusiastic about working on the tutor. For urban high school mathematics classes, attendance is usually higher on days when the class is working with the tutor than on days when it is not.

The remainder of this document is structured around the two loops. The first section discusses the outer loop. The remaining sections discuss the inner loop.

Some tutoring systems lack an inner loop. Instead, the system assigns the student a task, collects the student's answer, gives the student feedback on the answer, and continues, either by giving the student another chance to answer the problem correctly or by going on to a new task. Systems that lack an inner loop are generally called Computer-Aided Instruction (CAI), Computer-Based Training (CBT) or Web-Based Homework (WBH). Systems that do have an inner loop are called Intelligent Tutoring Systems (ITS). All the comments below about the outer loop pertain to both kinds of systems. The comments about the inner loop pertain only to ITS.

THE OUTER LOOP

The main responsibility of the outer loop is to decide which task the student should do next. Its other responsibilities, such as presenting the task to the student, are more mundane and will not be mentioned again. The main design issues are (1) selecting a task intelligently and (2) obtaining a rich set of tasks to select from.

Task selection

Four basic methods have been used for selecting tasks for the student. They are listed in order of simplicity, with the simplest method first.

(1) The outer loop displays a menu and lets the student select the next task. For instance, Andes displays a hierarchical menu of tasks, where each task is a physics problem for the student to solve. The student chooses which problem to do next. This design is appropriate because Andes is used as a "homework helper" for students in a normal physics class. That is, students attend lectures, labs and recitations as normal; they use Andes only to do their homework assignments. Each section of the course is taught by a different instructor, and instructors prefer to assign different homework problems. Moreover, they sometimes change their assignments as they see how the course is progressing. It is simpler to have instructors email students the homework assignments rather than reconfigure Andes itself. Moreover, some students may want to do extra problems. During office hours, the instructor and student may select an unassigned problem to work on together. Having students select tasks simplifies the logistics of running a large course with multiple instructors teaching multiple sections.

(2) The outer loop assigns tasks in a fixed sequence. For instance, suppose the tutoring system has the student first read a short text, then observe the tutor as it demonstrates how to solve a problem, then solve 10 problems with tutorial help, then take a test consisting of 3 problems without help. This is a fixed sequence of 15 (=1+1+10+3) tasks. AutoTutor and many other tutoring systems have used this method. The main advantage is that students must finish a task before moving to the next, which increases the chance that the students have mastered the prerequisites of a task before starting it.

(3) The outer loop implements a pedagogy called *mastery learning* (Bloom, 1984). Sherlock and many other tutors have used this method. The curriculum is structured as a sequence of units or a sequence of difficulty levels. When a student is working on a unit (or level of difficulty), the tutoring system keeps assigning tasks from that unit until the student has mastered the unit's knowledge. Only then does it allow the student to proceed to the next unit. Thus, some students finish the curriculum having done fewer tasks than other students. This design for the outer loop is mostly used in *self-paced* courses. It is seldom used for a *class-paced* course, where it is important that all students stay together as they move through the curriculum. A common mistake is to develop a tutoring system with a fancy outer loop, then discover that instructors cannot use its features due to the class-paced nature of the course.

Mastery learning requires that the tutoring system has some way to assess the student's mastery of knowledge taught by a unit. Ideally, this assessment is done by the inner loop as it helps the student work on each task, but if that is not feasible, then the tutoring system must assign assessment tasks (e.g., tests) as well as instructional tasks.

(4) Among four common designs for outer loops, the most complex is based on a pedagogy called *macroadaptation* (Corbett & Anderson, 1995; Shute, 1993). The Algebra Cognitive Tutor uses this method, for instance. For each task that the tutoring system can assign, it knows which knowledge components are exercised by the task. For each knowledge component, the tutor maintains an estimate of the student's degree of mastery of that knowledge component. When a student has completed a task and the tutor needs to select the next one, it chooses one based on the overlap between the tasks' knowledge components and the student's mastered knowledge components. For example, it might assign a task that requires many knowledge components that are already mastered by the student and just two components that are not yet mastered.

Some tutoring systems take this a step further. In addition to representing the correct knowledge components that students should learn, they also represent incorrect knowledge components. If the tutor infers that the student has a particular incorrect knowledge component, then it will select a task that is known to remedy that misconception.

Some tutoring systems represent not only correct and incorrect knowledge components, but also other stable traits of students. They might represent learning styles and preferences, such as a preference for visual or verbal explanations, so they can choose tasks that are marked as compatible with the student's style or preference.

For the outer loop to function correctly across multiple tasks and sessions, the information about the student must be stored on a server or on the student's computer's disk. This persistent information is often called a *student model.* Exactly what it contains depends on the type of outer loop. For an outer loop that marches down an ordered list of tasks, the student model could simply be a pointer to the next task in the list. For a macroadaptive outer loop, the student model will list attribute-value pairs, where the attribute is the name of the trait (i.e., a knowledge component, a learning style, etc.), and the value is the tutor's estimate that the student possesses that trait. In addition to containing information that the tutor has inferred about the student, the student model may contain other information about the student such as standardized test scores, the student's college major, the student's grade point average, etc. It may also contain information about which tasks have been accomplished, how long they took, the number of help requests or errors, and other easily obtained data on performance. All this information can be used by the task selection process to decide which task to assign next. Although "student model" is a rather strange name for such a heterogeneous collection of data, it is the traditional term.

In addition to selecting a task, some tutoring systems also select a mode for the task. For instance, Steve can either (1) demonstrate how to do the task by taking each step itself and explaining it, or (2) hint each step before the student attempts it, or (3) let the student try to solve the task without such unsolicited hints. Each of these is a mode. For tutoring systems with modes, the outer loop must select both a task and a mode. A typical policy for mode selection is start with a mode that provides a great deal of help to students (e.g., mode 1 in the list above) and gradually reduce the amount of help (e.g., move to mode 2 then to mode 3). This is an instance of a more general policy, called "fading the scaffolding" (Collins, Brown, & Newman, 1989).

Some tutoring systems (e.g., http://www.cmu.edu/oli/courses/enter_logic.html) are intended to almost completely replace a course, so they have tasks of many different kinds, and not just problem-solving tasks. Such a tutor might, for instance, start the instruction on a unit by requiring the student to read several pages of hypertext. This counts as one task. Then it might play a video of a problem being solved. It might then assign several problem solving tasks. Finally, it might require the student to go to a testing center, show identification to the proctor, logon to the tutor, and take an exam. This would also count as a task.

When a course is complex, it almost always has a hierarchical structure similar to the chapter-section-subsection structure of a textbook. Tasks are the leaves of such a hierarchy. In such a course, one can view task selection as comprised of several decisions, one for each level of the hierarchy. Different levels of the hierarchy may have different selection methods. For instance, suppose the tutoring system lets students select which chapter they want to work on. That is, the tutoring system uses the first selection method in the list mentioned earlier (and repeated in the next paragraph) for the top level of the hierarchy. Furthermore, suppose that once students have selected a chapter, they must complete all sections of the chapter in order. That is, the tutoring system employs the second selection method for the second level of the hierarchy. Lastly, suppose that tasks are selected from a section until the student has mastered the section's instructional objectives. That is, the tutoring system employs the third selection method for the lowest level of the hierarchy.

To summarize, there are four common types of outer loops:

- The student selects the task from a menu of all tasks.
- The tutor assigns tasks in a predetermined sequence.
- Mastery learning: The tutor assigns tasks from a unit's pool of tasks until the student has mastered the knowledge taught by the unit.

- Macroadaptive learning: The tutor tracks traits, including both unchanging traits such as learning styles, and changing traits, such as correct and incorrect knowledge components. It chooses a task based on the match between the task's traits and the student's traits.

Outer loops that employ more sophisticated technology have been proposed and sometimes implemented (e.g., Macmillan, Emme, & Berkowitz, 1988; Peachy & McCalla, 1986; Zinn, Moore, & Core, 2002). For instance, the tutoring system could plan a specific sequence of tasks based on the student model, then start executing the sequence. Halfway through the sequence, it might discover that the student has a serious misconception, and so revise its planned sequence. Plan generation, monitoring and replanning are part of a well-developed subfield of Artificial Intelligence and Operations Research called Planning (Russell & Norvig, 2003). From that subfield's perspective, even the macroadaptive outer loop is just a simple "reactive agent." Nonetheless, more complex, plan-based agents have not yet become common in tutoring systems.

Obtaining a rich set of tasks to select from

In addition to providing some kind of mechanism for selecting tasks, the designer must provide a set of tasks for the outer loop to select from. It is often surprising how many problems are necessary in order to field a tutoring system. For a 13 week semester at 10 problems per week, the tutoring system needs 130 problems at minimum. If the instructors want freedom to choose their own problems, or if the outer loop implements macroadaptation or mastery learning, the tutoring system may need 500 problems for a semester-long course. Instructors are often surprised by how quickly a complex task is "ruined" because students who have gone through it brag about their experiences to their friends who have not yet gone through it. For instance, such bragging might reveal faults that Sherlock wants the students to find.

Ideally, the tutor can generate its own tasks when given a specification of the desired characteristics. For instance, given a specification of a knowledge component to be taught, the SQL-Tutor (Martin & Mitrovic, 2002) generates a database query, written in SQL, that involves the knowledge component. A human author then writes text for a problem that has this database query as its solution. Using this technique, a human author was able to create 200 problems in about 3 hours—enough for a 6 week instructional module in an SQL course. Such programs are called *problem generators*.

Most often, the tasks are created by human authors, so designers often build authoring tools that facilitate creation of new tasks (Murray, 1999). A good authoring tool does as much of the work as possible. Many systems have enough domain knowledge in them that the authoring tool can solve the problem all by itself. For instance, the problem authoring tool for Andes has the author describe a physics problem formally as well as with the text and graphics that the student sees. The tool then solves the formalized problem itself. If it cannot, it complains to the author.

For some task domains, the tools cannot solve a problem, but when shown a solution by the author, the tool can generalize the solution. For instance, Steve expects the author to enter a long sequence of virtual actions (opening valves, pushing buttons, etc.). The system generalizes the sequence into a procedure for achieving the given task. To do so, it must ask the author many questions, such as which effects are intentional and which are not. It also experiments with variants of the demonstration, asking the author to verify their correctness.

To summarize, the outer loop of a generic tutoring system is all about selecting tasks for the student to do. The designer confronts two major problems: (1) providing a set of instructional tasks and (2) designing an algorithm that decides which task (and mode) to assign next.

THE INNER LOOP

Whereas the outer loop is about tasks, the inner loop is about steps within a task. Recall that a step is a user interface action that is part of completing a task (e.g., solving a problem). Because tutoring systems vary considerably in how they support learning as students generate and review steps, let us simplify by assuming that they offer the students *services* as they work. In particular, the following are probably the most common services:

- Minimal feedback on a step. In most cases, this means indicating only whether the step is correct or incorrect.
- Error-specific feedback on an incorrect step. This information is intended to help the student understand why a particular error is wrong and how to avoid making it again.
- Hints on the next step.
- Assessment of knowledge.
- Review of the solution.

The next five sections cover each of these services in detail. However, there are many other services that are not covered. As an example of one, consider the Transom tutoring system (Roberts, Pioch, & Ferguson, 2000). Transom teaches students how to drive a remotely operated underwater vehicle. Due to the limited sensory input of the vehicle (video and sonar), students often get confused about the orientation and position of the vehicle. When the student asks, Transom shows the simulated vehicle from a helpful viewpoint, such as that of a virtual observer standing on the sea floor several meters from the vehicle. This instantly removes the student's disorientation. It is an example of a service that makes sense only for certain task domains. The five services covered in detail are much more widely useful.

Although the services are designed to help students learn, the students may not use them as intended. For instance, when the tutor provides minimal feedback, students may rapidly guess steps until the tutor indicates that their step is correct. Misuse of services probably occurs with all educational activities, including classroom instruction and one-on-one human tutoring. For instance, when I was a volunteer tutor in college, I sometimes suspected that my tutees were playing dumb in order to elicit a solution from me. As the services below are described, types of misuse that are known to occur will be mentioned. However, students can probably figure out a way to abuse any service no matter how well designed it is. Ultimately, it is the student who controls the learning process. As the old American saying goes, "you can lead a horse to water, but you can't make it drink."

In order to simplify the exposition, let us assume that tutoring systems have two major internal functions: *step generation* and *step analysis*. Both functions can access the entire state of the tutoring system. When the step *generator* is called, it returns a step that the student should do next. The step generator is used when giving students hints on what step to do next, which is one of the services listed above. The step *analyzer* is used for the other services. When it is given the student's step, it returns a classification used by minimal feedback, such as "correct" or "incorrect." If the student's step is correct, the step analyzer may also return the knowledge components that the student probably used to derive the step—this is a key part of the assessment service. If the student's step is incorrect, the step analyzer may return a description of the specific error made by the student, which is necessary for the error-specific feedback service.

The designs of the step analyzer and the step generator vary considerably from one tutoring system to another. Although the appendix has some brief descriptions of them for its 6 tutoring systems, it would take another paper to adequately describe step analysis and generation in general. Thus, this article will treat them as black boxes. That is, it will specify what they need to output without describing how they can compute such outputs.

MINIMAL FEEDBACK

This section covers a common service of tutoring systems, which is to offer minimal feedback on a step taken by the student. Minimal feedback usually indicates whether a step is correct or incorrect, although other categories are sometimes used as well. Here are the kinds of minimal feedback used in the 6 illustrative tutors:

- Andes colors a correct step green and an incorrect step red, and the Algebra Cognitive Tutor also changes the appearance of steps to flag incorrect ones. On some incorrect steps, the tutors may pop up a message such as "Forgot to put units on a number." Such pop-up error messages counts as error-specific feedback and not as minimal feedback.
- Steve shakes his head "yes" or "no."
- AutoTutor gives minimal feedback by prefacing the tutor's turn with different openers. There are five levels of feedback: negative "No"; negative neutral "Well, almost"; neutral "Okay"; neutral positive "Yeah"; and positive "Great job!"
- When students click on the Submit button of the SQL-Tutor the first time, it tells them whether the whole solution is correct or incorrect. If they click on it again, it highlights an incorrect step, but doesn't say anything about it.
- As the student solves a troubleshooting problem with Sherlock, it only gives feedback when a safety violation occurs, and the feedback is non-minimal because it points out the error and why it is wrong. When the student has finished solving the problem, Sherlock's default method for reviewing solutions is to display the student's steps in one column and an expert's steps in an adjacent column. This does not explicitly identify any student step as incorrect or correct, so it is not minimal feedback either. However, some of Sherlock's other methods for reviewing solutions do provide minimal feedback.

This section covers only feedback on *steps*, as opposed to "feedback" on the student's competence, which is usually called assessment and is covered in another section. A known problem with feedback is that students can interpret it as assessment, which can lead them to become discouraged, to stop trying, etc. (Kluger & DeNisi, 1996).

Categories of minimal feedback

Although most tutoring systems use just two categories, correct and incorrect, it is possible to use others. This section discusses a few options.

Suppose a step is not part of an ideal solution to the problem, but it might be part of a non-ideal solution. The instructors might wish to consider this as a third category for minimal feedback—correct but non-optimal. For instance, when students solve physics problems on Andes, they can enter an equation that is true of the given physical situation but is not necessary for solving the problem. Because Andes colors correct steps green and incorrect steps red, the instructors considered using yellow for correct but non-optimal steps, but ultimately decided against it on grounds of parsimony.

If instructors sometimes disagree on what makes for the best solution to the problem, it might be wise to subcategorize "non-optimal." For instance, because some Sherlock instructors emphasize reducing time and others emphasize reducing costs, Sherlock could subcategorize non-optimal steps as "wastes time" or "wastes parts." In short, the categories used for minimal feedback should reflect the pedagogical policies of the instructors.

If a step cannot be classified as into one of the minimal feedback categories (correct, incorrect, non-optimal, etc.), then it lands in an "unrecognizable" classification. There are basically just three ways to treat such unrecognizable steps. (1) Some tutoring systems, such as the Algebra Cognitive Tutor, assume that they can recognize all correct steps, so they treat unrecognizable steps as incorrect. (2) Other tutoring systems, such as the SQL-Tutor, assume that students will sometimes produce novel, correct solutions, so they treat unrecognized steps as correct. (3) Lastly, the tutoring system could simply tell the student that the step is unrecognizable. In this case, "unrecognizable" is yet another category for minimal feedback.

When to give minimal feedback

Tutoring systems vary on their policies about *when* to give feedback. As illustrations, here are the feedback policies of our 6 example systems.

- *Immediate feedback*: Andes, Algebra Cognitive Tutor, Steve and AutoTutor give feedback immediately after each student step. They vary considerably in how the feedback is presented.
- *Delayed feedback*: Only if the step violates a safety regulation will Sherlock give immediate feedback as the student solves a problem. However, after the problem is solved, the student can request a reviewing mode where minimal feedback is given. In particular, as Sherlock replays the student's solution step-by-step, it indicates which steps did not contribute any diagnostic information.
- *Demand feedback*: The SQL-Tutor gives feedback only when the student clicks on the Submit button. Because the student can continue working on their solution after receiving such feedback, this does not count as delayed feedback.

In addition to these 3 simple policies (immediate, delayed and demand), more complex policies are possible. A tutoring system can even decide after each step whether to give minimal feedback.

The feedback policy can be a function of the student's competence. For instance, if the student is nearing mastery, then the feedback is delayed whereas a less competent student would be given the same task with immediate feedback. This policy has been observed in human tutoring, and is one instance of "fading the scaffolding," (Collins et al., 1989), because feedback is a kind of scaffolding (pedagogically useful help).

To summarize this section on minimal feedback, one can envision a tutoring system as calling the step analyzer every time the student has entered a step. The step analyzer returns one of the minimal feedback categories, e.g., correct, incorrect, non-optimal, unrecognizable, etc. The tutor then decides whether to display this categorization to the student immediately, later or never. The tutor can make this decision based on fixed pedagogical policies or more adaptively.

NEXT-STEP HINTS

Students often get stuck and would like help on determining what step to do next. There are three important design issues involved in giving hints about next steps, each discussed in its own subsection:

1. *When* should the tutor give a hint about the next step? Should it wait for the student to ask? Should it give unsolicited hints when it detects guessing or floundering?

2. *What* step should the tutor suggest? For instance, if there are multiple paths to a solution, and the student appears to be following a long complex one, should the tutor suggest starting over?
3. *How* can the tutor give hints that maximize learning, keep frustration under control, and allow the student to finish the problem?

A tutoring system that gives minimal feedback should also have the ability to hint next steps. Without minimal feedback, students can proceed in blissful ignorance past their errors all the way to an incorrect solution, which they submit. With minimal feedback but no next-step hints, students often guess repeatedly then give up in frustration. Thus, next-step hinting is a very common service for tutoring systems that employ minimal feedback.

When to hint

The common wisdom is that a tutor should give a hint on the next step only if the student really needs it. If the student can enter a correct step by thinking hard enough, then the system should refuse to give a hint even if the student asks for one. On the other hand, if the student is likely to waste time and get frustrated by trying in vain to enter a correct step, or if the student is making repeated guesses instead of trying to figure out the next step, then the tutoring system should probably give a hint even if the student doesn't ask for one.

DT Tutor (Murray, VanLehn, & Mostow, 2004) implements a version of this ideal tutoring policy. The tutor applies decision theory to make its choice about whether to give a hint. For each tutor action it can make (e.g., to give a hint, and which kind of hint), it uses a probabilistic model of the student to predict all possible student reactions to the tutor's action and their probability. The predicted student state includes the likelihood of learning, of becoming frustrated, of entering the next step correctly, etc. DT Tutor evaluates the utility of each of the predicted student states, multiplies the state's utility by the state's probability, and eventually produces the expected utility of each proposed tutor action. It then takes the tutor action with the highest expected utility. Although advances in probabilistic reasoning make it feasible for DT Tutor to perform this calculation in real time, considerable data from human students is needed in order to set the parameters in its model of student learning. Nonetheless, this is clearly an appropriate, albeit data-intensive way for tutoring systems to make decisions about whether to give help or not.

A much simpler policy, used by Andes, Sherlock and many other systems, is to give a hint only if the student explicitly asks for one by clicking on the appropriate button. However, hint buttons have two well-known problems (Aleven & Koedinger, 2000):

- *Help abuse*: Some students ask for help even when they don't need it. In the extreme cases, some students ask for help on every step.
- *Help refusal*: Some students refuse to ask for help even when they need it. They enter a long series of incorrect steps, which may be guesses, instead of clicking on the help button.

In order to discourage help abuse, some systems keep score and deduct points for each help request. In order to discourage help refusal, some systems give help whenever the student has entered three incorrect steps in a row or have paused a long time.

Although most tutoring systems give help in reaction to a help request or a sign of floundering, a few tutoring systems have experimented with "proactive" help, which is given before the student even attempts to do a step. For instance, if DT Tutor estimates that letting the student do a step without a hint will lead to a failed attempt and frustration, then it says something like, "The next step is tricky. Remember that the chain rule has the form dx/dz = dx/dy*dy/dz." Steve used an even more extreme form of proactive help. If the next step had never been done or observed by the student, then Steve said something like, "The next step is new. Let me show you how to do it," and executed the step itself, instead of letting the student attempt the step.

In summary, the general problem about deciding when to give hints on the next step is complex. Although fixed policies are simple to implement and to explain to students, tutoring systems that can accurately predict the outcome of the student's next attempt to enter a step may have pedagogical advantages over fixed help-giving policies.

Which step to hint

This section discusses the step generator. Given access to the complete state of the tutoring system, it returns a step to be hinted. Several criteria determine which step the tutor should hint:

- The step must be correct.
- The hinted step should not have been done already by the student.
- Instructors' preferences should be honored.
- If the student has a plan for solving the problem or even for just the next step, then the tutor should hint the step that the student is trying to complete.

The first two criteria require no discussion, but the others do. The third criterion is that the to-be-hinted step should conform to instructors' preferences. For instance, if a problem has multiple solutions, the instructor may prefer only some of them. If the task domain allows students to skip steps, the instructors may prefer that a student who needs a hint not skip any steps. For instance, suppose the student is trying to enter the step $2-5x=12 \rightarrow x=-2$ and instead enters $x=2$. The student has skipped several steps, and that might have contributed to the mistake, so the instructor might prefer to hint $-5x=12-2$ rather than $x=-2$.

If these first three criteria do not uniquely specify a step to hint, then the tutoring system may consider the student's preferences. To do so, it must infer what step the student is trying to enter. Although this is hard to do in general, there are a few simple cases where the student's intentions can be determined without too much computation.

The first simple case is where a problem has multiple solutions, and all the student's steps belong to just one of the solutions. Clearly, the student intends to continue pursuing that solution, so the to-be-hinted step should be the next step in that solution.

Another simple case occurs when the user interface consists of a set of graphical "widgets" such as buttons, pull-down menus and type-in boxes. Some of these are actuated by first clicking on the widget, then either typing or gesturing with the mouse. For instance, clicking on a box might drop down a menu from which the user selects an item. If the student first clicks on such a widget then asks for a hint, the tutor can be quite confident that the student wants a hint on how to finish actuating that widget. For instance, if the student clicks on a cell in a spreadsheet of the Algebra Cognitive Tutor and then asks for a hint, the tutor assumes the student wants a hint on how to fill the cell in. It might give such a hint, or it might advise the student to try a different step altogether.

A somewhat more complex case occurs when the student has just made an incorrect step and received minimal negative feedback. It is likely that the student wants a hint on how to do that step correctly. If the tutor can determine which correct step corresponds to the incorrect step entered by the student, then it can safely hint that correct step. On the other hand, if the tutor cannot determine which correct step corresponds to the student's step, the tutor may ask the student what step they were trying to enter. Andes does this for unrecognizable equations that the student wants help on.

If none of these simple cases apply, then the tutoring system has to somehow figure out which of the possible next steps is most likely to be part of the solution that the student is trying to pursue. In AI, this is called the plan recognition problem (Russell & Norvig, 2003). The general idea is to match the sequence of steps leading up to the most recent student step against step sequences that are either generated by a planner or pulled from a plan library. Once the tutor has determined the plan that the student seems to be following, then it can easily determine what the next step in that plan is.

We have suggested that the step generator first determine the set of steps preferred by the instructor, then use the student's preferences to select among them. If the tutoring system has a reliable method for determining the student's preferences, then it might make sense to reverse this procedure. That is, the tutoring system first determines what step the student wants to do next, then checks whether that step is part of the ideal (i.e., preferred by instructors) solution. If not, the tutoring system can say something like, "Although I could help you correct that step, it is actually not a good way to solve the problem. I advise you to try something else."

If inferring the student's intentions for next steps is too difficult or unreliable, a tutoring system can try to asking the student directly. Some systems, such as Steve and Sherlock, list possible next steps and let the student choose. Andes identifies the student's top-level plan by eliciting the top level goal ("What quantity is the problem asking you to determine?") and the method for achieving it ("What principle should you use to determine that quantity? Hint: when the principle's equation is finally written, it will contain the sought quantity."). Students answer these two questions via hierarchical menus, and they get immediate feedback if their answers are incorrect.

Research is needed in order to determine the pedagogical benefits of using the student's plans and preferences during step generation. If a student needs a hint, then they may have no coherent plan and would be grateful for a hint on any step at all. Thus, a tutoring system that only uses the instructors' preferences, and not the student's, might be just as effective. This is a topic for empirical study.

How to hint the next step

Once a tutoring system has decided to hint a certain step, it must actually give the hints. Perhaps the most common policy is to construct a short sequence of hints for the next step. The first hints are weak—they divulge little information so that students are encouraged to do most of the thinking themselves. Later hints are stronger.

If the step requires only one knowledge component, then a standard hint sequence is Point, Teach and Bottom-out (Hume, Michael, Rovick, & Evens, 1996). Pointing hints mention problem conditions that should remind the student of the knowledge component's relevance. Teaching hints describe the knowledge component briefly and show how to apply it. Bottom-out hints tell the student the step. For instance, here is a sequence of hints that Andes might give:

1. "Notice that the elevator is slowing down." This is a pointing hint.
2. "When an object is slowing down, its acceleration is in the opposite direction from its velocity." This is a teaching hint. It consists of a brief statement of the knowledge component in its generic form.
3. "Because the elevator is slowing down, its acceleration is directed in the opposite direction of its velocity. Its velocity is downward, so its acceleration is upward." This is another teaching hint. It indicates how the student should apply the knowledge component to this problem.
4. "Draw the acceleration vector straight up." This is the bottom-out hint.

If the tutoring system represents knowledge components explicitly, then it is often convenient to associate a sequence of text templates with each knowledge component. When the step generator returns a step to hint, it also returns the knowledge component application that was used to derive the step. The hint sequence can then be generated by filling slots in the templates with text from the problem, such as "the elevator" in the above illustration.

If more than one knowledge component is involved in deriving the to-be-hinted step, then it can be difficult to generate a pedagogically effective hint sequence. A complex hint sequence might be worse than none. That is, it might be better to just give a bottom-out hint.

Hint sequences can sometimes be simplified by keeping track of hint sequences that were given earlier. A simple case occurs when the tutor and the student have earlier discussed a goal or plan, and that goal or plan is still relevant. The tutor can refer to this "common ground" more briefly. For instance, Andes sometimes elicits the top level goal and plan from a student via a menu-based dialogue, as mentioned earlier. If that goal and plan are still relevant for a subsequent hint sequence, Andes does not go through the whole menu-based dialogue again. Instead, it says something like, "I assume you are still applying Newton's second law along the y-axis in order to determine the acceleration of the elevator." That is, it *tells* the student the relevant plan and goal, rather than eliciting it from the student. Linguistics has discovered many ways that communication can be made more efficient (Jurafsky & Martin, 2000), but significant research is needed in order to transfer them from natural language dialogues to the tutoring systems, which mix short intervals of dense, tutor-controlled communication (e.g., hint sequences) with long intervals of sparse, student-controlled communication (e.g., entering steps and getting only minimal feedback).

Hint sequences are usually delivered in a stylized dialogue. The first hint is presented along with buttons such as "Done," "🡐" and "🡒." Clicking on "🡒" prints the next hint in the sequence; clicking on "🡐" goes back to the previous hint; clicking on "Done" closes the tutorial dialogue so that the student can resume working on the problem. Students sometimes click "Done," try to enter the next step, fail, and ask for help again. In this case, it would violate Gricean principles of communication (Jurafsky & Martin, 2000) to show the student the hints that were seen already, so the tutoring system starts with the first hint in the sequence that the student has not yet seen.

Another policy, called contingent tutoring, was inspired by studies of human tutors. Wood, Wood and Middleton (1978) found that when effective human tutors react to a student's step, they don't always start with the weakest hint, as the hint-sequence tutoring systems do. Instead, the initial level of help is contingent on the students' recent performance. If the last few attempts at giving help were not immediately successful, then the tutor starts with a more specific level of help than it did last time. For instance, suppose a contingent tutoring system, such as Quadratic (H. A. Wood & Wood, 1999), has hint sequences that are 5 hints long. Suppose that the tutor starts with hint 2 but the student proceeds all the way to hint 4 before successfully entering a step. The next time the tutoring system gives hints, it will start with hint 3. On the other hand, if the hint 2 had worked, then the tutor will start with hint 1 next time.

Hints are often presented as hypertext. For instance, links are used beneath technical terms; clicking on the term pops up its definition.

A common form of help abuse is for students to click rapidly on the "🡒" button until they reach the bottom-out hint (Aleven & Koedinger, 2000). In order to discourage this, tutoring systems can insert a delay between the time that a bottom-out hint is requested and the time that it is displayed (Murray & VanLehn, 2005).

Although hint sequences mimic the behavior of expert human tutors (Hume et al., 1996) and are almost universally used in tutoring systems, there has been little research on alternative methods of hinting. For instance, suppose the tutor only gives one hint, namely, the bottom-out hint, but it demands that the student explain why the step is correct. Eliciting self-explanations causes learning gains in other settings (e.g., Chi, Bassok, Lewis, Reimann, & Glaser, 1989; Renkl, 2002), so it might work as a hinting tactic as well. The field would benefit from having a greater variety of hinting tactics to choose from, and from empirical studies comparing their effectiveness. If the hinting tactics were better, then help abuse might be much less common.

To summarize this section on next-step hints, recall that the main issues are when to give hints, which step to hint and how to hint the step. The decision about when to give a hint is generally decided by a policy, such as only giving a hint if the student asks for one. The decision about which step to hint is complex and involves finding a step that is correct, that is not redundant with existing steps, and that is preferred by instructors and perhaps even by the student. Hints are usually given in a sequence, starting from the weakest hint and terminating in a bottom-out hint. Although hinting is more computationally complex than minimal feedback, it is probably more pedagogically important. Hints are given when students get stuck, and being stuck may be a powerful motivation for learning. Thus, getting hinting right may significantly increase student learning.

ERROR-SPECIFIC FEEDBACK

Suppose that an algebra student has entered the incorrect step 2+3*x=20 ➔ 5*x=20. To an experienced human tutor, this is a familiar error. The student obtained 5 by adding the 2 and the 3, thus violating operator precedence. A possible hint sequence is:

1. You seem to have added 2+3. Is that really appropriate?
2. You seem to have grouped 2+3*x as (2+3)*x. Is that legal?
3. Because multiplication has a higher precedence than addition, you should group 2+3*x as 2+(3*x). If you want to review precedence, click here.
4. You need to isolate x, and it is embedded in two expressions. The outer expression is "2+❑" where ❑ is an expression containing x. You need to strip off the outer expression so that you have an equation whose left side is ❑. If you want to review the procedure for solving single-variable equations, click here.
5. In order to strip "2+…" off "2+3*x", you need to subtract 2 from both sides of the equation.
6. You should enter 3*x=20-2.

Although this isn't a real tutor's hint sequence, it does illustrate two kinds of hints. Hints 1, 2 and 3 discuss the incorrect learning event that led to the error. Hints 4, 5 and 6 discuss the step the student should enter next. If the tutoring system can only determine that the step is incorrect but not what caused it to be incorrect, then it can only give hints 4, 5 and 6. However, if it can also determine the incorrect learning event that led to the incorrect step, then it can give hints 1, 2 and 3 as well.

This illustrates a new service that a tutoring system can provide to learners. The service consists of analyzing an incorrect step in order to determine the incorrect learning event(s) that led to it, then giving instruction that is aimed at preventing that incorrect learning event(s) from occurring again. In order to reduce the length of this article, we will assume that the first phase, diagnosing the incorrect step, is carried out in one of our black boxes, the step analyzer.

In particular, when given an incorrect step, we will assume that the step analyser may return an *error description*. The error description indicates the type of error (e.g., operator precedence) and information about the student's error that is needed for generating instruction. For instance, in order to generate the hint sequence shown a moment ago, the error description would need to include (1) the high precedence operator: multiplication, (2) the low precedence operation: addition, (3) the unknown variable: x, (4) the outer expression containing the unknown: 2+❑, (5) the side of the equation containing the unknown: 2+3*x, (6) the original equation: 2+3*x=20, and (7) the new equation: 3*x=20-2. In order to generate the hint sequence shown earlier, the tutoring system fills slots in the text templates with these 7 strings. The text templates come from the error type, operator precedence in this case.

There are many techniques for diagnosing errors. Most require that authors observe student errors, figure out what is causing a common error, write an appropriate error type for it, and implement some way to recognize such errors. It is easy to get carried away by the challenge of this process and to try to build recognizers for more and more of student errors. However, for the purposes of helping students learn (as opposed to assessing their knowledge, which will be discussed later), there is little point in recognizing an error if there is nothing pedagogically useful that can be said about it. For instance, suppose that there are 5 common error types that occur when students expand binomials, such as (2x+1)(3y-5). However, in all 5 cases, the instructors prefer that the tutor teach the correct method (i.e., "You need to apply FOIL: First, Outer, Inner and Last…"). There is no point in differentiating these 5 error types, since it will not affect the tutor's response to them.

How to give error-specific feedback

The main purpose of error-specific feedback is to get the student to change their knowledge so that they will not make this mistake again. Correcting one's knowledge is sometimes compared to debugging a piece of software (Ohlsson, 1996). A programmer must first find evidence that the bug exists, then find out what the bug is, and then fix the bug. When tutees are not working with a tutor, they must do the whole self-debugging process themselves. They must first notice that a step is incorrect, then find the flaw in their reasoning and knowledge, then figure out what the correct learning events and knowledge components should be. Although the tutor cannot replace the self-debugging process (because only the student can modify the student's knowledge) it can help the self-debugging process along via well-designed error-specific feedback. The more self-debugging the student does with the tutor, the better the student should be at self-debugging when the tutor is not available.

Many tutors present feedback as a sequence of hints that are loosely associated with the stages of self-debugging. When the student enters an incorrect step, the tutor begins the sequence by simply giving minimal feedback. This is like providing the student with evidence of a knowledge bug but giving no further help toward identifying the bug or correcting it.

Typically, the next few hints help the student identify the bug. For instance, if only part of a step is incorrect, then the tutor can help the student diagnose their error by pointing out the incorrect part. For instance, if an Andes student enters a complex equation that has an incorrect cosine in it, then Andes might say, "Check your trigonometry." Another kind of hint alludes to the relevant knowledge component. For instance, Andes might hint a sin/cos error by saying, "You seem to have applied the right-triangle rule inappropriately. Check your opposite side, adjacent side and hypotenuse."

Further hints in the sequence are often intended to push the student further and further along the self-debugging process. At some point, the hints stop referring to the incorrect step, learning events and knowledge components, and start referring to the *correct* step, learning events and knowledge components. For instance, consider again the algebra student who entered the incorrect step 2+3*x=20 ➔ 5*x=20 which elicited this hint sequence:

1. The step is incorrect. [This is minimal feedback. It may be conveyed by color, etc.]
2. You seem to have added 2+3. Is that really appropriate?
3. You seem to have grouped 2+3*x as (2+3)*x. Is that legal?
4. Because multiplication has a higher precedence than addition, you should group 2+3*x as 2+(3*x). If you want to review precedence, click here.
5. You need to isolate x, and it is embedded in two expressions. The outer expression is "2+❑" where ❑ is an expression containing x. You need to strip off the outer expression so that you have an equation whose left side is ❑. If you want to review the procedure for solving single-variable equations, click here.
6. In order to strip "2+…" off "2+3*x", you need to subtract 2 from both sides of the equation.
7. You should enter 3*x=20-2.

The first few hints focus on the incorrect step and the last few hints focus on the correct step that should be entered next. The transition can be subtle, as it is often hard to distinguish a hint about what is wrong (feedback) from a hint about what should be done next (hinting the next step). Hint 4 is ambiguous; it could be considered a discussion of either the student's incorrect step or what step to enter next. However, from the developers' point of view, hint 4 goes with hints 2 and 3 because all of them require determining that the incorrect step is due to a common confusion about operator precedence.

When to give error-specific feedback

Once the tutor has assigned an error description to a step and has a hint sequence ready to display, it may or may not be wise to present it. Although this decision can be made dynamically, on an error-by-error basis, most tutoring systems use a fixed policy. This section mentions a few of the more popular policies.

As mentioned earlier, the Algebra Cognitive Tutor gives only minimal feedback, unless the student has tried in vain to enter the same step twice before. Then it will start with the first hint of its hint sequence for that step. However, if the student's incorrect entry at the time the hint sequence starts is an instance of an error type, then the error-specific hints are given before the next-step hints.

As mentioned earlier, Andes' general policy on hinting is to provide immediate minimal feedback without being asked and to provide hints only when students click on a hint button. It also gives the student control over what kind of hints they get by providing two hint buttons. One provides next-step hints, and the other, labeled "what's wrong with this?" provides error-specific feedback on whatever incorrect step is currently selected.

Students sometimes make careless errors, called "slips" in psychology (Norman, 1981), but fail to notice that they have made them, which can cause them to waste time looking for deep, potential misconceptions. To ameliorate this, Andes divides error types into slips and potential misunderstandings. Slips are often made by the experts, including the instructors. A potential misunderstanding is an error type that could be due to many factors, including incorrect beliefs, but it is almost never seen in expert's work. When a student enters an incorrect step that is classified as a slip, then Andes gives the error specific remediation immediately, e.g., "You forgot to include a unit on a number." If the error is classified as a potential misunderstanding, then Andes merely turns the incorrect step red, and lets the student ask for an error-specific hint if they want one.

Some student steps contain two or more errors. In order to keep the communication simple, the tutoring system should probably respond to only one of them. Students typically fix that error only, so the tutoring system can then mention the second error. For instance, suppose an Andes student enters "vf = 5.2" and the correct step is "vf = -5.2 m/s". The student has left off both the negative sign and the units. Andes responds to the units error, and the student enters "vf = 5.2 m/s". This step still has the sign error, so Andes alerts the student by giving minimal negative feedback.

To summarize, our discussion of the inner loop has so far included three services: minimal feedback, next-step hints and error-specific feedback. If a tutoring system provides all three services, as do Andes, the Algebra Cognitive Tutor, Steve and many others, then the implementation blends all three of them. However, each requires a different kind of expertise from the tutoring system. A tutoring system that can only distinguish correct from incorrect steps can give minimal feedback but cannot give any hints. The ability to provide next-step hints requires knowledge about how to select an appropriate next step. The ability to provide error-specific feedback requires knowledge about what error types occur and how to give pedagogically useful hints on those error types. Although conceptually independent, the first "hint" for both next-step hinting and error-specific feedback is usually minimal feedback, so that service is a prerequisite for the other two. Thus, the logically possible combinations are: (1) minimal feedback, (2) minimal feedback + next-step hints, (3) minimal feedback + error-specific feedback, and (4) all three services.

ASSESSING KNOWLEDGE

The inner loop is often called upon to assess the student's knowledge and other persistent student traits. This assessment service is conceptually independent of the three services discussed so far, although its implementation is interwoven with theirs if the tutoring system has them. It is common practice to use "assessment" to refer to an analysis of the *student's* competence, and "evaluation" to refer to an analysis of the *tutoring system's* competence (Lesgold, 1994; Mark & Greer, 1993; Regian & Shute, 1994; Shute & Regian, 1993; Siemer & Angelides, 1998). This section covers assessment only.

What kind of assessment is required?

Three clients may use assessments: the instructor, the student and the tutoring system itself. Designers need to design an assessment that fits all their needs. For instance, instructors often do not want the tutoring system to assign course grades to students because that leaves them in the awkward position of defending such a grade if the student questions it. However, instructors often do want to have failing students pointed out to them early in the semester so that the instructor can work with them. Students often want the same kind of information as instructors so that they can manage their own learning. Tutoring systems may need assessments to drive their decision making. For instance, if the outer loop uses mastery learning, then it needs to know when the student's competence exceeds the mastery threshold so it can advance the student to the next curriculum unit or difficulty level.

A fundamental issue is the grain-size or granularity of the assessment. The granularity of an assessment refers to the degree of aggregation over the task domain's knowledge. An assessment is fine-grained if it provides, for instance, a probability of mastery for each knowledge component in the task domain. An assessment is coarse-grained if it provides a single number that indicates the student's overall competence.

Assessments are decision aids, and as a general heuristic, the bigger the decision, the coarser the required assessment. If a decision affects a whole semester, e.g., whether the student needs to retake a required course, then the assessment should cover the whole semester and lead ultimately to a yes-no decision, so a single number per student is appropriate. If a decision affects only, say, whether the tutor starts at the beginning of a hint sequence or in the middle, then only a small part of a fine-grained assessment is relevant. The general idea is that if the decision is small, in that it affects only a small amount of instructional time, then only a small amount of the domain's knowledge can be accessed during that time and thus the relevant decision aid is the student's competence on just that knowledge. These are just heuristic guidelines. They warn developers to think twice before developing an elaborate fine-grained assessment to aid big decisions. For instance, instructors probably have little use for an assessment comprised of 500 numbers, one for each knowledge component.

Tutors make many small decisions, so they are the main customers for fine-grained assessments. As an example, consider the outer loop. A macro-adaptive tutoring system tries to assign a task that exercises a few unmastered knowledge components; thus it must have a fine-grained assessment that provides a probability of mastery for each of the relevant knowledge components. However, tutoring systems do not always need fine-grained assessments. A system that does only mastery learning needs just a single number that estimates the student's mastery of the whole unit's knowledge. A tutoring system that lets the student choose the next task may not need an assessment at all.

Coarse-grained assessment

If the required assessment is coarse grained, such as a single number reporting the student's competence on the current unit, then it is usually computed from several measures such as:

- A measure of progress and coverage, such as the number of problems solved or the number of correct steps.
- A measure of the amount of help given, such as the number of hint sequences started and the proportion that ended with a bottom-out hint.
- Some measure of competence, such as the frequency of incorrect initial steps, the time required to enter a correct step, or the number of attempts at a step before it is entered correctly.

These raw measures need to be combined into an overall assessment. Psychometrics is the field that studies how to do this for conventional tests (e.g., multiple choice tests). One of their main tools is item-response theory (IRT), which assumes that every test item has a difficulty, and different items have different difficulties (Embretson & Reise, 2000). Unfortunately, IRT generally assumes that test items are conditionally independent given the student's competence. This is seldom true of the raw measures collected at the step level by tutoring systems. Although IRT has powerful features, such as calibration algorithms that empirically determine item difficulties and other parameters, considerable work is needed before it can be applied to tutoring systems.

Instructors need to be fully involved in designing the coarse-grained assessment. They must determine the raw measures, such as the ones listed above. Until IRT is extended, they will also need to be involved in designing algorithms to combine the raw measures into an overall measure.

If instructors are uncertain or unavailable for designing the assessment, an alternative is to implement a fine-grained assessment (discussed in the next section), then combine the 500 or so numbers from the fine-grained assessment using some principled algorithm (Martin & VanLehn, 1995). For instance, if the fine-grained assessment delivers a probability of mastery of each knowledge component in the task domain, then the probability of correctly answering a multiple-choice test item is roughly the product of the probabilities of mastery of the knowledge components needed to answer the test item correctly. Given some additional assumptions about guessing, one can use a fine-grained assessment to predict the student's score on a multiple-choice final exam. This is just one principled method for generating a coarse-grained assessment from a fine-grained one. It has the advantage that it can be easily explained to instructors and students. However, an instructor-designed assessment would probably be more acceptable to the instructors if they are the ones making the decisions.

Fine-grained assessment: Counting learning events

Recall that we defined steps to be user interface events. We also assumed that a step is the product of one or more learning events, and defined a learning event to be a mental event based on a knowledge component. Learning events and knowledge components are not directly observable, but steps are. This section discusses how to assess mastery of individual knowledge components from data on steps. Such a fine-grained assessment is provided by many tutoring systems, include the Algebra Cognitive Tutor, the SQL-Tutor and Sherlock.

"Mastery" really means the probability that a knowledge component will be applied when it should be applied. If the student's competence was frozen instead of constantly changing due to learning and forgetting, then mastery could be estimated by counting the number of times a knowledge component was applied and dividing by the number of times it should have been applied. Thus, we need to discuss three issues: (1) How to detect applications of a knowledge component, (2) how to detect times when a knowledge component should have been applied, and (3) how to adjust for instruction, learning and forgetting.

There are many methods for analyzing steps in order to determine the knowledge components used to derive them. In order to keep this article brief, we will assume that one of our black boxes, the step analyzer, is responsible. That is, given a correct student step, it returns the set of knowledge components that were used by the student to derive the step.

Actually, this is too simple an assumption, because in some cases, the interpretation of a step can be ambiguous. That is, a single step can correspond to one of several sets of learning events. For instance, suppose an algebra equation solving tutor asks the student to solve 3*(x-3)=3 and the student writes just x = 4. There are several possible derivations that the student could have done mentally:

1. 3(x-3)=3
2. 3x-9=3
3. 3x=12
4. x=4

or

1. 3(x-3)=3
2. x-3=1
3. x=4

or

1. Try x=1: $3(1-3) = -6 \neq 3$
2. Try x=2: $3(2-3) = -3 \neq 3$
3. Try x=3: $3(3-3) = 0 \neq 3$
4. Try x=4: $3(4-3) = 3$

All solutions are correct, but they employ different learning events. If the tutoring system could tell which learning events the student made, it could increment the counters on the corresponding knowledge components. This kind of ambiguity of interpretation is known as the *assignment of credit* problem. Bayesian networks and other techniques from the Uncertainty subfield of Artificial Intelligence (UAI) have been used to address the assignment of credit and blame problems (Jameson, 1995).

Fine-grained assessment: Counting failures

Counting only the successful applications of a knowledge component is not enough; we need to know how many times the student failed to apply it as well. For instance, suppose a knowledge component has been applied 5 times. If the student had exactly 5 opportunities to apply it, then a count of 5 would suggest high mastery. If the student had 50 opportunities to apply it, then a count of 5 indicates low mastery. Thus, in order to assess knowledge, the tutoring system should count not only how many times the knowledge component was applied, but also how many times the student *failed* to apply it.

One approach is to detect failed attempts at *steps*. Suppose for simplicity that there is a one-to-one correspondence between a step and a learning event. If the student fails to make the step, then the student must lack the knowledge that underlies the learning event. Thus, if the system could detect a failure to make the step, then it could increment the failure counter on the corresponding knowledge component.

Detecting a failed attempt at a step is simple if the tutoring systems' user interface has certain properties. If the step must be located at a *specific place* (e.g., the FROM field of the SQL query) or must be done at a *specific time* (e.g., because Steve just asked, "Why don't you test the alarm panel now."), then an error at that place or time can be interpreted as a failed attempt to make the step. Many tutoring systems use this basic idea.

However, other tutoring systems cannot use this basic idea because they give the students the freedom to make steps at any place and any time. For instance, Andes lets students enter equations in any equation box on the screen and in any order. Thus, when a student asks for help or enters an incorrect equation, Andes has no idea which equation the student was trying to enter (unless an error type recognizer recognizes the incorrect equation, which is not guaranteed). In general, there is a tradeoff between the freedom that the tutoring system gives to the student and its ability to diagnose the student's failures (VanLehn & Niu, 2001).

We assumed for simplicity that there was one-to-one correspondence between the step and the learning event. Suppose instead that the step required a bit more thinking, so that it corresponds to a conjunction of 3 learning events. If the student does the step correctly, then the tutoring system should increment the success counters on the knowledge associated with all three learning events because they were all involved in generating the step. On the other hand, if the student makes a failed attempt at the step, the tutoring system should *not* increment the failure counters on all the knowledge components. Only one component needs to be missing in order to block the step. This is a classic problem in the ITS literature, called the *assignment of blame* problem. UAI techniques can be used to deal with it (Jameson, 1995).

Fine-grained assessment: Other issues

There are many other issues involved with assessment. Here are a few:

- There is a big difference between little evidence and mixed evidence. If one merely takes the frequency of success relative to opportunities, then 0.5 can mean both 1 success and 1 failure (little evidence) or 20 successes and 20 failures (mixed evidence).
- Students should be learning, so their knowledge should be changing. When should old evidence be ignored?
- Should help given by the tutoring system be counted as evidence of learning or of lack of knowledge?

- Are all learning events equally difficult? If not, then item response theory (IRT) can be used so that success on easy learning events provides less evidence of competence than success on difficult learning events.
- If the tutoring system includes error-specific feedback based on error types, should it somehow include the frequency of occurrence of the error types in its assessments?
- Prior probabilities of mastery are not independent. If a student has not mastered a knowledge component that is taught early in the course, then it is less likely that the student has mastered.
- Knowledge components that appear later.

As many authors have noted (e.g., Self, 1990), it is easy to get carried away by the sheer intellectual challenge of assessment and to overbuild this part of the tutoring system. Designers must be clear about why their tutoring system needs an assessment, and should design no more than necessary.

REVIEWING THE WHOLE SOLUTION

Many tutoring systems give students feedback and hints only after the student has submitted a solution. Sometimes this is done because giving feedback and hints during problem solving would be disruptive. For instance, SCoT (Pon-Barry, Clark, Schultz, Bratt, & Peters, 2004; Schultz et al., 2003) and DC-Train (Bulitko & Wilkins, 1999) work together to teach naval Damage Control Officers to fight fires, floods and other disasters. DC-Train watches them handle a simulated emergency, and as it watches, it says nothing, as this would disrupt the exercise. When the student has finished saving the ship (or not), SCoT goes over the student's mistakes during the exercise. This kind of delayed feedback is also used on other tutoring systems that teach real-time skills such as tactical air combat (Ritter & Feurzeig, 1988), steering a large naval ship in close quarters (Roberts, 2001) and making critical decisions in a combat information center (Zachary et al., 1999).

Even in non-real-time task domains, delayed feedback and hints are sometimes used purely for pedagogical reasons. When feedback is given during problem solving, students don't have to check their own work because the tutor will point out errors to them. They also don't have to deal with getting lost and starting over, because they can ask the tutor for a hint when they feel lost. Thus, certain meta-cognitive skills may not be exercised during problem solving because the tutor makes them unnecessary. When students get only delayed feedback, then they must practice these meta-cognitive skills themselves. This may prepare them for learning from problem solving when the tutoring system is no longer present. As mentioned earlier, the outer loop can "fade the scaffolding" by initially assigning tasks with immediate feedback then later assigning tasks with delayed feedback.

Delaying feedback adds a new design issue for developers. When it is time to give feedback, the solution may have many things wrong with it. Which should it discuss? This issue is addressed in the next section, followed by a few comments linking delayed feedback back to immediate feedback.

Plans and policies for reviewing flaws in the student's solution

When the student has submitted a solution and the tutoring system has finished analyzing its steps, the system now knows which steps need to be mentioned because they are incorrect. It may also have noticed that some steps' interpretations are ambiguous, and some of those may need to be discussed as well. The incorrect and ambiguous steps comprise a set of tutorial dialogue goals.

Although a tutor could simply give a short lecture on every goal, a more pedagogically effective dialogue may arise from considering the following issues:

- Which tutorial dialogue goals should *not* be mentioned? If there are many tutorial dialogue goals to discuss, then it might be better to focus on just one or two.
- In what order should the tutorial dialogue goals be discussed? Sometimes it may be necessary to achieve one goal before another can be discussed.
- How deeply should each goal be discussed?
- How can the student's dialogue goals, such as requests for clarification, be accommodated?

Some tutoring systems have used tutorial dialogue managers based on classic techniques for generating and executing plans (e.g., Carberry, 1990; Schultz et al., 2003; Zinn et al., 2002). However, most tutoring systems have used simpler techniques. A few will be described here to illustrate the variety.

SQL-Tutor has a default policy for managing the tutorial dialogue, but provides the student with a way to override it. The tutor has the following levels of feedback:

1. Simple: The number of mistakes, if any.
2. Error flag: The clause where an error has occurred (see Figure 5).
3. Hint: A general description of the error.
4. Detailed hint: More information about the error.
5. All errors: A list containing a description of every error.
6. Partial solution: The correct version of the clause where an error appeared.
7. Complete solution: The ideal solution to the problem.

On the initial submission of an answer, the tutor gives level 1 feedback. With each submission of an answer to the problem, the feedback level is increased by 1 to a maximum of 5. However, the student can select a feedback level from a menu. The menu includes levels 6 and 7, which function somewhat like bottom-out hints. Thus, SQL-Tutor's dialogue management policy is a mixture of tutor control and student control.

In contrast, Cove gives the student total control over the review session (Roberts, 2001). It displays a matrix of chronological periods (columns) and performance measures (rows). For instance, one row reports whether the Conning Officer (the student) said the commands properly. Another row indicates whether the student gazed at the appropriate features. (The student's gaze is monitored via a head-mounted eyetracker.) The top row, not surprisingly, reports whether the student's ship hit anything it wasn't supposed to hit. The buttons are colored to indicate how well the student performed (a type of minimal feedback). Clicking on a button causes the tutor to display a chart, a map, a text or some other explanation of what the student did and what should have been done better. Some of these explanations have buttons which cause more details to be displayed. These analyses of the student's performance are a kind of error-specific feedback combined with next-step help. Thus, the student has complete control over which learning events are discussed and the depth of the discussion.

As these examples illustrated, a key issue is how to share control over the dialogue between the student and the tutor. For instance, the decision about what learning events to discuss may belong to the student (Cove), the tutor, or both (SQL-Tutor).

Studies of human tutors suggest that they often give help both during and after a student's performance (Katz, Connelly, & Allbritton, 2003). Moreover, the help they give after the performance often follows up on help given during the performance. For instance, during the performance, the tutor might merely flag an error, and if that does not suffice to elicit a correction, the tutor just gives the bottom-out hint. However, when reviewing the students' performance afterwards, the tutor would mention the error in order to discuss why it was wrong. This capability would be an interesting addition to tutoring systems.

Reviewing solutions: Other issues

Most of the issues that were discussed in the sections about immediate feedback, error-specific feedback and hints are also issues for delayed feedback. Just like an immediate feedback tutoring system, when a delayed feedback tutor decides to give a hint, it can use hint sequences or dialogue, and if it uses a hint sequence, it can start with the first hint or use a contingent hinting policy. If the student's solution has an incorrect step, the delayed feedback tutor may or may not recognize the error. If so, it must decide whether the error's analysis is worth discussing with the student. If the student's solution is incomplete, and there are multiple alternative ways to complete it, the tutor must decide which one to hint.

To summarize, reviewing the student's solutions brings up many of the same issues as giving feedback and hints in between steps during problem solving. However, because there are usually multiple incorrect steps in the student's solution that need discussion, post-problem reviewing has an additional design problem, which is organizing the discussion to maximize learning. Some systems let the student control the discussion by selecting the steps to be discussed and the depth of the discussion. Other systems try to share control over the discussion with the student. In addition, having a discussion period after problem solving creates new pedagogical opportunities such as splitting the discussion of incorrect steps into immediate and delayed portions, or discussing the student's meta-cognitive strategies, or comparing this solution to earlier problems' solutions, etc.

FINAL REMARKS

The focus of this article has been on what tutoring systems do, as opposed to how they are structured. In particular, it focuses on the differences between the outer loop, where the major responsibility is selecting a task for the student to do next, and the inner loop, where the major responsibilities are analyzing the steps, giving pedagogically effective help and assessing knowledge. These terms, "inner loop" and "outer loop," should be understood as behavioral descriptions rather than software structures, for the software organizations of tutoring systems are far too complex for the simple two-loop structure (Devedzic & Harrer, 2005). For instance, an interesting software structure is to build software objects that represent both graphical widgets on the screen (e.g., a dipstick) but also everything that the tutoring system knows about that object, including methods of manipulating it, conditions for such manipulations being correct vs. incorrect, hint sequences for various conditions, error-types and error-specific feedback. That is, the software object represents a miniature tutoring system for that particular widget.

As mentioned earlier, many tutoring systems do not have an inner loop. For instance, there are physics tutoring systems where the student solves a problem on scratch paper for several minutes then enters a numerical answer into the computer. If the number is correct, the tutor says so and assigns the next problem. If the answer is incorrect and happens to be a number that the authors know can only be generated by making one particular mistake, then the tutoring system points out the mistake and asks the student to try again. These single-loop systems are like the ones described here, except that they get a single piece of evidence per task, namely a numerical answer. Despite the paucity of evidence, they try to analyze it in terms of learning events. When they succeed, they give help based on the learning events they found. Even though these systems lack inner loops, they still try to tutor at the level of learning events. This suggests that the descriptive framework defined here for tutoring systems may be even more general. Indeed, it might cover human tutoring as well as computer tutoring.

Tutoring systems that do not have an inner loop can be quite easy to write. For each task, the designers record the correct answer and perhaps a few incorrect ones. For each incorrect answer, the author writes text intended to repair the learning event and the knowledge behind it. In terms of the amount of student learning per unit of development time, a single-loop system might be a good choice. The policy of "fading the scaffolding" suggests starting with a double-loop system then moving to a single-loop system. More empirical work is necessary in order to compare the single- and double-loop system's effectiveness. This may help developers determine what student and task domain conditions make an ITS worth the added development costs.

Several important topics have not been covered but are essential reading for any tutoring system developer. One is how to implement the step analyzer and step generator. Another is how to evaluate a tutoring system both during its development and at the end (Shute & Psotka, 1996). A third issue is how to involve instructors in the design and evaluation process (Dick & Carey, 1990). A fourth topic is designing appropriate activities for students and appropriate user interfaces. The fourth topic is especially important and far too complex to cover in a single paper. It covers simple issues, such as when to present hints as text versus speech (Clark & Mayer, 2002), and more complex issues, such as whether to assign complex, well-defined problems or open-ended, inquiry-style problems.

In principle, the dream of having tutoring systems for every student in every course is within our grasp. Although human-computer interaction has limitations, especially in understanding spoken language and visual input, and these limitations restrict the ability of computer tutors to interact with students, even within these limitations, many more tutoring systems could and should exist than do now. As this introduction suggests, a new tutoring system can start simple with more complex features added as experience suggests that they are needed.

Perhaps the biggest impediment to wider adoption of tutoring systems is the unfortunate choice of terms. "Tutor" implies that the software replaces teachers instead of helping them. "Correct" implies that the software is only applicable to drill-and-practice on well-defined tasks. The lack of distinction between observables (steps) and unobservables (learning events, knowledge components) discourages participation by those who see student cognition as essentially impenetrable. For some, the term "student model" is considered particularly offensive because it implies that students can actually *be* modeled by a computer. In fact, the student model is nothing more than a fancy grade book. Hopefully, this article will both dispel some of the bad connotations of the technical terms of the field and encourage creative educators and developers to consider how the technology might help them achieve their dreams for students.

ACKNOWLEDGEMENTS

I greatly appreciate feedback on the article from Min Chi, Art Graesser, Sandy Katz, Ken Koedinger, Tanja Mitrovic and Chas Murray. Any errors that remain were probably introduced after they read it! The ONR Cognitive Science program supported the development of Andes. Continued support for Andes is provided by the Pittsburgh Science of Learning Center under grant 0354420 from NSF.

REFERENCES

Aleven, V., & Koedinger, K. R. (2000). Limitations of student control: Do students know when they need help? In G. Gauthier, C. Frasson & K. VanLehn (Eds.) *Intelligent Tutoring Systems: 5th International Conference, ITS 2000* (pp. 292-303). Berlin: Springer.

Anderson, J. R., Corbett, A. T., Koedinger, K. R., & Pelletier, R. (1995). Cognitive Tutors: Lessons Learned. *Journal of the Learning Sciences,* 4(2), 167-207.

Beck, J., Stern, M., & Haugsjaa, E. (1996). Applications of AI in Education. *ACM Crossroads*, 3.1.

Bloom, B. S. (1984). The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher*, 13, 4-16.

Bulitko, V. V., & Wilkins, D. C. (1999). Automated instructor assistant for ship damage control. In *Proceedings of AAAI-99.*

Carberry, S. (1990). *Plan Recognition in Natural Language Dialogue*. Cambridge, MA: MIT Press.

Chi, M. T. H., Bassok, M., Lewis, M., Reimann, P., & Glaser, R. (1989). Self-explanations: How students study and use examples in learning to solve problems. *Cognitive Science,* 15, 145-182.

Clark, R. C., & Mayer, R. E. (2002). *E-learning and the Science of Instruction*. Hoboken, NJ: John Wiley.

Collins, A., Brown, J. S., & Newman, S. E. (1989). Cognitive apprenticeship: Teaching the craft of reading, writing and mathematics. In L. B. Resnick (Ed.) *Knowing, learning and instruction: Essays in honor of Robert Glaser* (pp. 543-494). Hillsdale, NJ: Lawrence Erlbaum Associates.

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

Corbett, A., Koedinger, K. R., & Anderson, J. R. (1997). Intelligent Tutoring Systems. In M. Helander, T. K. Landauer & P. Prahu (Eds.) *Handbook of Human-Computer Interaction, Second Edition* (pp. 849-874). Amsterdam: Elsevier Science.

Croteau, E. A., Heffernan, N. T., & Koedinger, K. R. (2004). Why are algebra word problems difficult? Using tutorial log files and the power law of learning to select the best fitting cognitive model. In J. C. Lester, R. M. Vicari & F. Paraguaçu (Eds.) *Intelligent Tutoring Systems: 7th International Conference, ITS 2004* (pp. 240-250). Berlin: Springer.

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

Dick, W., & Carey, S. (1990). *The Systematic Design of Instruction: Third edition*. New York: Scott-Foresman.

Embretson, S. E., & Reise, S. P. (2000). Item response theory for psychologists. In L. L. Harlow (Ed.) *Multivariate Applications*. Mahwah, NJ: Erlbaum.

Graesser, A. C., Lu, S., Jackson, G. T., Mitchell, H. H., Ventura, M., Olney, A., et al. (2004). AutoTutor: A tutor with dialogue in natural language. *Behavioral Research Methods, Instruments and Computers,* 36, 180-193.

Graesser, A. C., Moreno, K., Marineau, J., Adcock, A., Olney, A., & Person, N. (2003). AutoTutor improves deep learning of computer literacy: Is it the dialog or the talking head? In U. Hoppe, F. Verdejo & J. Kay (Eds.) *Proceedings of Artificial Intelligence in Education* (pp. 47-54). Amsterdam: IOS.

Graesser, A. C., Wiemer-Hastings, K., Wiemer-Hastings, P., & Kreuz, R. (1999). AutoTutor: A simulation of a human tutor. *Journal of Cognitive Systems Research,* 1, 15-25.

Hume, G., Michael, J., Rovick, A., & Evens, M. (1996). Hinting as a tactic in one-on-one tutoring. *Journal of the Learning Sciences,* 5(1), 23-49.

Jameson, A. (1995). Numerical uncertainty management in user and student modeling: An overview of systems and issues. *User Modeling and User-Adapted Interactions,* 5(3/4), 193-251.

Johnson, W. L., Rickel, J., Stiles, R., & Munro, A. (1998). Integrating pedagogical agents into virtual environments. *Presence: Teleoperators and Virtual Environments*, 7(6), 523-546.

Jurafsky, D., & Martin, J. H. (2000). *Speech and Language Processing: An introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*. Upper Saddle River, NJ: Prentice-Hall.

Katz, S., Connelly, J., & Allbritton, D. (2003). Going beyond the problem given: How human tutors use post-solution discussions to support transfer. *International Journal of Artificial Intelligence in Education,* 13, 79-116.

Katz, S., Lesgold, A., Hughes, E., Peters, D., Eggan, G., Gordin, M., et al. (1998). Sherlock 2: An intelligent tutoring system built on the LRDC framework. In C. P. Bloom & R. B. Loftin (Eds.) *Facilitating the development and use of interactive learning environments* (pp. 227-258). Hillsdale, NJ: Erlbaum.

Kluger, A. N., & DeNisi, A. (1996). The effects of feedback intervention on performance: A historical review, a meta-analysis and a preliminary feedback intervention theory. *Psychological Bulletin,* 112(2), 254-284.

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.

Lesgold, A. (1994). Assessment of intelligent training technology. In E. L. Baker & H. F. O'Neil (Eds.) *Technology Assessment in Education and Training* (pp. 97-116). Mahwah, NJ: Erlbaum.

Macmillan, S. A., Emme, D., & Berkowitz, M. (1988). Instructional Planners: Lessons Learned. In J. Psotka, L. E. Massey & S. A. Mutter (Eds.) *Intelligent Tutoring Systems: Lessons Learned* (pp. 229-256). Hillsdale, NJ: Lawrence Erlbaum Associates.

Mark, M. A., & Greer, J. (1993). Evaluation methodologies for intelligent tutoring systems. *Journal of Artificial Intelligence in Education*, 4(2/3), 129-153.

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

Martin, J., & VanLehn, K. (1995). Student assessment using Bayesian nets. *International Journal of Human-Computer Studies*, 42, 575-591.

Mitrovic, A. (2003). An intelligent SQL tutor on the web. *International Journal of Artificial Intelligence in Education,* 13(2-4), 197-243.

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.

Murray, C., VanLehn, K., & Mostow, J. (2004). Looking ahead to select tutorial actions: A decision-theoretic approach. *International Journal of Artificial Intelligence in Education,* 14(3-4), 235-278.

Murray, R. C., & VanLehn, K. (2005). Effects of dissuading unnecessary help requests while providing proactive help. In G. I. McCalla & C.-K. Looi (Eds.) *Proceedings of Artificial Intelligence in Education*. Amsterdam: IOS Press.

Murray, T. (1999). Authoring Intelligent Computer Systems: An analysis of the state of the art. *International Journal of Artificial Intelligence in Education,* 10, 98-129.

Norman, D. A. (1981). Categorization of action slips. *Psychological Review,* 88(1), 1-15.

Ohlsson, S. (1996). Learning from performance errors. *Psychological Review*, 103(2), 241-262.

Peachy, D. R., & McCalla, G. I. (1986). Using planning techniques in intelligent tutoring systems. *International Journal of Man-Machine Studies*, 24, 77-98.

Person, N., Graesser, A. C., Bautista, L., Mathews, E. C., & TRG. (2001). Evaluating student learning gains in two versions of AutoTutor. In J. D. Moore, C. Redfield & W. L. Johnson (Eds.) *Artificial Intelligence in Education: AI-ED in the Wired and Wireless future* (pp. 268-293). Amsterdam: IOS.

Pon-Barry, H., Clark, B., Schultz, K., Bratt, E. O., & Peters, S. (2004). Advantages of spoken language interaction in dialogue-based intelligent tutoring systerms. In J. C. Lester, R. M. Vicari & F. Paraguaçu (Eds.) *Intelligent Tutoring Systems: 7th International Conference, ITS 2004* (pp. 390-400). Berlin: Springer-Verlag.

Regian, J. W., & Shute, V. J. (1994). Evaluating intelligent tutoring systems. In E. L. Baker & H. F. O'Neil (Eds.) *Technology Assessment in Education and Training* (pp. 79-96). Mahwah, NJ: Erlbaum.

Renkl, A. (2002). Worked-out examples: Instructional explanations support learning by self-explanations. *Learning and Instruction,* 12, 529-556.

Rickel, J. (1989). Intelligent computer-aided instruction: A survey organized around system components. *IEEE Transactions on Systems, Man and Cybernetics,* 19(1), 40-57.

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, F., & Feurzeig, W. (1988). Teaching real-time tactical thinking. In J. Psotka, L. D. Massey & S. A. Mutter (Eds.) *Intelligent Tutoring Systems: Lessons Learned* (pp. 285-301). Hillsdale, NJ: Erlbaum.

Ritter, S., Blessing, S., & Wheeler, L. (2003). User modeling and problem-space representation in the tutor runtime engine. In P. Brusilovsky, A. Corbett & F. de Rosis (Eds.) *9th International Conference, UM 2003*

(Vol. LNAI 2702, pp. 333-336). Johnstown, PA: Springer.

Roberts, B. (2001). *COVE-A shiphandling trainer with an attitude.* Paper presented at the Interservice/Industry Training, simulation and Education Conference.

Roberts, B., Pioch, N., & Ferguson, W. (2000). Verbal coaching during a real-time task. *International Journal of Artificial Intelligence in Education,* 11. 377-388.

Russell, S., & Norvig, P. (2003). *Artificial Intelligence: A Modern Approach* (Second ed.). Upper Saddle River, NJ: Prentice Hall.

Schofield, J. W. (1995). *Computers, classroom culture and change*. Cambridge, UK: Cambridge University Press.

Schofield, J. W., & Evans-Rhodes, D. (1989). Artifical intelligence in the classroom. In D. Bierman, J. Breuker & J. Sandberg (Eds.) *Artificial Intelligence and Education: Synthesis and Reflection* (pp. 238-243). Springfield, VA: IOS Press.

Schultz, K., Bratt, E. O., Clark, B., Peters, S., Pon-Barry, H., & Treeratpituk, P. (2003). A Scalable, Reusable Spoken Conversational Tutor: SCoT. In V. Aleven, U. Hoppe, J. Kay, R. Mizoguchi, H. Pain, F. Verdejo & K. Yacef (Eds.) *AIED 2003 Supplementary Proceedings* (pp. 367-377).

Self, J. A. (1990). Bypassing the intractable problem of student modelling. In C. Frasson & G. Gauthier (Eds.) *Intelligent Tutoring Systems: At the crossroads of AI and Education* (pp. 107-123). Norwood, NJ: Ablex.

Shapiro, J. A. (2005). Algebra subsystem for an intelligent tutoring system. *International Journal of Artificial Intelligence in Education*, 15(3), 205-228.

Shute, V. J. (1993). A macroadaptive approach to tutoring. *Journal of Artificial Intelligence in Education,* 4(1), 61-93.

Shute, V. J., & Glaser, R. (1990). A large-scale evaluation of an intelligent discovery world. *Interactive Learning Environments,* 1, 51-76.

Shute, V. J., & Psotka, J. (1996). Intelligent tutoring systems: Past, Present and Future. In D. Jonassen (Ed.) *Handbook of Research on Educational Communications and Technology*: Scholastic Publications.

Shute, V. J., & Regian, J. W. (1993). Principles for evaluating intelligent tutoring systems. *Journal of Artificial Intelligence in Education,* 4(2/3), 245-272.

Siemer, J., & Angelides, M. C. (1998). A comprehensive method for the evaluation of complete intelligent tutoring systems. *Decision Support Systems,* 22(1), 85-102.

Singley, M. K., & Anderson, J. R. (1989). *The transfer of cognitive skill.* Cambridge, MA: Harvard University Press.

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

VanLehn, K., Lynch, C., Taylor, L., Weinstein, A., Shelby, R. H., Schulze, K. G., et al. (2002). Minimally invasive tutoring of complex physics problem solving. In S. A. Cerri, G. Gouardères & F. Paraguaçu (Eds.) *Intelligent Tutoring Systems, 2002, 6th International Conference* (pp. 367-376). Berlin: Springer.

VanLehn, K., & Niu, Z. (2001). Bayesian student modeling, user interfaces and feedback: A sensitivity analysis. *International Journal of Artificial Intelligence in Education*, 12(2), 154-184.

Wood, D. J., Wood, H., & Middleton, D. (1978). An experimental evaluation of four face-to-face teaching strategies. *International Journal of Behavioral Development*, 1, 131-147.

Wood, H. A., & Wood, D. J. (1999). Help seeking, learning, and contingent tutoring. *Computers and Education*, 33, 153-169.

Woolf, B. P. (1992). AI in Education. In *Encyclopedia of Artificial Intelligence (Second Ed.)* (pp. 434-444). New York: Wiley.

Zachary, W., Cannon-Bowers, J., Bilazarian, P., Krecker, D., Lardieri, P., & Burns, J. (1999). The Advanced Embedded Training System (AETS): An intelligent embedded tutoring system for tactical team training. *International Journal of Artificial Intelligence in Education*, 10, 257-277.

Zinn, C., Moore, J. D., & Core, M. G. (2002). A 3-tier planning architecture for managing tutorial dialogue. In S. A. Cerri, G. Gouardères & F. Paraguaçu (Eds.) *Proceedings of the Intelligent Tutoring Systems Sixth International Conference* (pp. 574-584). Berlin: Springer.

APPENDIX: EXAMPLES OF ITS

Steve

Outer loop: Steve is a tutoring system that teaches hierarchical, multi-step procedures, such as how to start a large air compressor (Johnson, Rickel, Stiles, & Munro, 1998; Rickel & Johnson, 1999). The equipment is simulated, and Steve is displayed as an agent in a virtual reality (Figure 1).

Inner loop: Students take steps by manipulating graphical widgets, such as clicking on a valve icon to open it or on a dipstick to check the oil level. Steve can give immediate feedback (e.g., "You let the sea water in too fast. The cold water damaged the hot heat exchanger."). Students can ask for hints. Steve can give proactive hints (e.g., "Go ahead with initializing the heat exchanger now, and don't forget that sea water is very cold."). Steve can also execute a step for the student. In fact, it can demonstrate the whole procedure for a student, explaining each step as it goes. This allows it to collaborate with the student by initially doing all the steps, then letting the student do more and more steps while getting less and less proactive hinting.

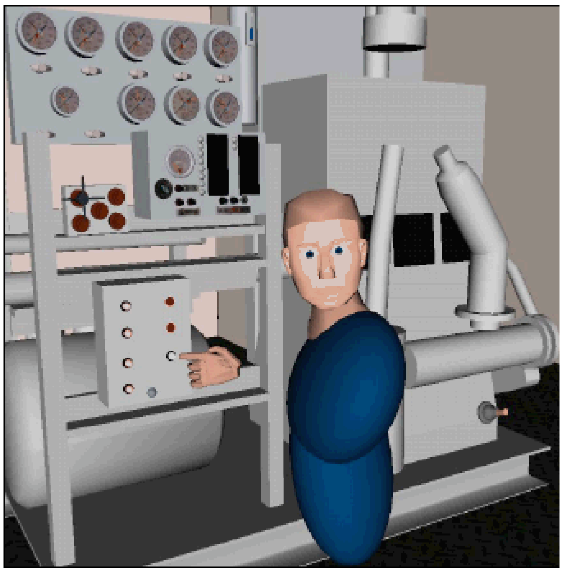


Fig.1. Steve describing a power light.

Step analysis: Steve interprets the student's step by matching it to a set of anticipated steps. In particular, after each student step, Steve computes all possible correct next steps (usually there is just one). It stores with each anticipated step the learning events used to generate it. If the student's next step matches any of the anticipated steps, then an interpretation has been found. For instance, suppose the student starts up a heat exchanger by opening the seawater valve to position 10 on a 100 point scale. (Actually, this illustration was not taken from the Steve literature, but invented to compactly illustrate several points.) Position 10 falls within certain thresholds, so the student's step matches the anticipated step Crack_open_seawater_inlet. Since this anticipated step is correct, Steve gives positive feedback by shaking his head "yes."

Steve uses this same basic framework to compute anticipated incorrect steps as well as correct steps. For instance, when it generates Crack_open_seawater_inlet, it might also generate the incorrect step, Fully_open_seawater_inlet. If the trainee's step corresponds to that, the tutoring system can produce a pedagogically useful error message, such as "When you open the seawater inlet valve all the way, the cold seawater strikes the hot metal of the heat exchanger, causing thermal stress that could crack the metal. As a general principle, if you are letting coolant into a potentially hot container, such as a heat exchanger or an engine, always let it in gradually in order to avoid thermal fatigue."

Let us map this illustration onto the main concepts introduced in the main body of the article:

- The correct step: Crack_open_seawater_inlet.
- The knowledge component: "If one is letting coolant into a potentially hot container, always let it in gradually in order to avoid thermal fatigue."

- The learning event: "Because the heat exchanger on this air compressor is potentially hot, and the seawater acts as a coolant, I should open the seawater valve only a little bit so that the seawater fills the heat exchanger gradually."

Notice that there is a one-to-one relationship between the step, the learning event and the knowledge component. This is a major contributor to the simplicity of Steve's analysis.

Step generation: The student's most recent correct step is, by definition, part of the procedure being taught. Steve uses immediate feedback to block incorrect steps, so Steve always knows where in the procedure the student is. When it needs to give hints on what to do next, it merely picks the next step. If there are multiple steps that could follow the student's most recent correct step, then Steve lists them and lets the student choose. In general, the more procedural the knowledge being taught, the easier it is to determine what step the student should do next.

Algebra Cognitive Tutor

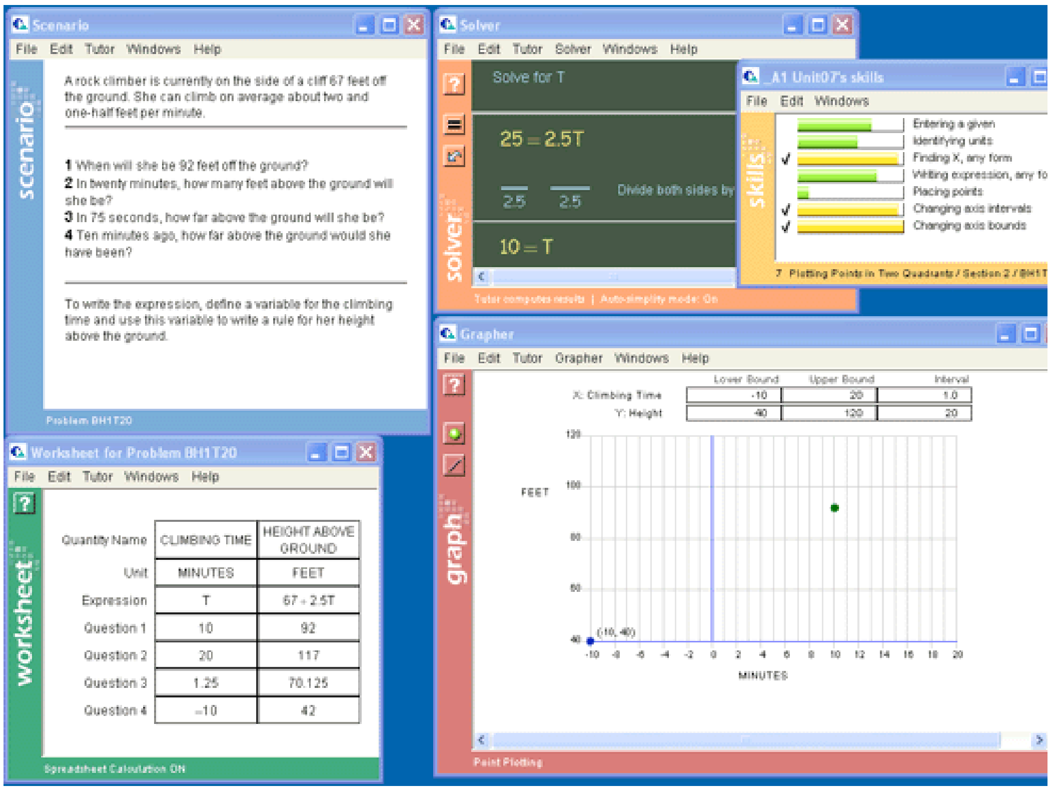
Figure 2 shows the screen of the Algebra 1 Algebra Cognitive Tutor, which was originally developed by Anderson's group (Anderson, Corbett, Koedinger, & Pelletier, 1995), and has been extended and marketed by Carnegie Learning (www.carnegielearning.com). Because it is so widely used and has undergone so many positive evaluations, it is arguably the most successful ITS in the world at this time.

Inner loop: The inner loop monitors the student's steps while solving an algebra problem. Although most problems, including the one in Figure 2, involve using multiple representational tools (graphs, tables, etc.) to analyze a problem scenario, some problems focus only on specific tools, such as the equation solver. In Figure 2, the problem to be solved is shown in the upper left window. This problem has four parts, labeled 1 through 4. When the student began, the cells of the table in the worksheet window at the lower left were all empty. The student has filled every cell with a number, text or an algebraic formula. In the process of figuring out what to put in the cells, the student used the solver window (upper right) and the graphing window (lower right). Each time the student filled a table cell, plotted a point on the graph, entered an equation in the solver window, etc, the tutor gave immediate feedback that told the student whether the step was correct or incorrect.

The Algebra Cognitive Tutor also lets students ask for help at any time by clicking on any of the "?" buttons. It will also offer help if the student makes two or more attempts at the same step.

Knowledge components: The tutor conducts a fine-grained assessment of the student. The probabilities of mastery for each of the knowledge components being taught in the current unit of the curriculum are shown in a bar graph (see the upper right window).

Outer loop: The outer loop in the Algebra I Algebra Cognitive Tutor selects an algebra problem for the student to do and makes sure that the student submits a solution. The tutor uses its fine-grained assessment to select a task that exercises a few knowledge components that the student has not yet mastered. When the student has mastered all the knowledge components in a unit (all the bars turn gold), the student is advanced to the next unit in the algebra curriculum.



A rock climber is currently on the side of a cliff 67 feet off the ground. She can climb on average about two and one-half feet per minute.

1 When will she be 92 feet off the ground?
2 In twenty minutes, how many feet above the ground will she be?
3 In 75 seconds, how far above the ground will she be?
4 Ten minutes ago, how far above the ground would she have been?

To write the expression, define a variable for the climbing time and use this variable to write a rule for her height above the ground.

Problem BH1T20

Quantity Name	CLIMBING TIME	HEIGHT ABOVE GROUND
Unit	MINUTES	FEET
Expression	T	67 + 2.5T
Question 1	10	92
Question 2	20	117
Question 3	1.25	70.125
Question 4	-10	42

Spreadsheet Calculation ON

Solve for T

$25 = 2.5T$

$\overline{2.5}$ $\overline{2.5}$ Divide both sides by

$10 = T$

Tutor computes results | Auto-simplify mode: On

	Skill
	Entering a given
	Identifying units
✓	Finding X, any form
	Writing expression, any fo
	Placing points
✓	Changing axis intervals
✓	Changing axis bounds

7 Plotting Points in Two Quadrants / Section 2 / BH1T

	Lower Bound	Upper Bound	Interval
X: Climbing Time	-10	20	1.0
Y: Height	40	120	20

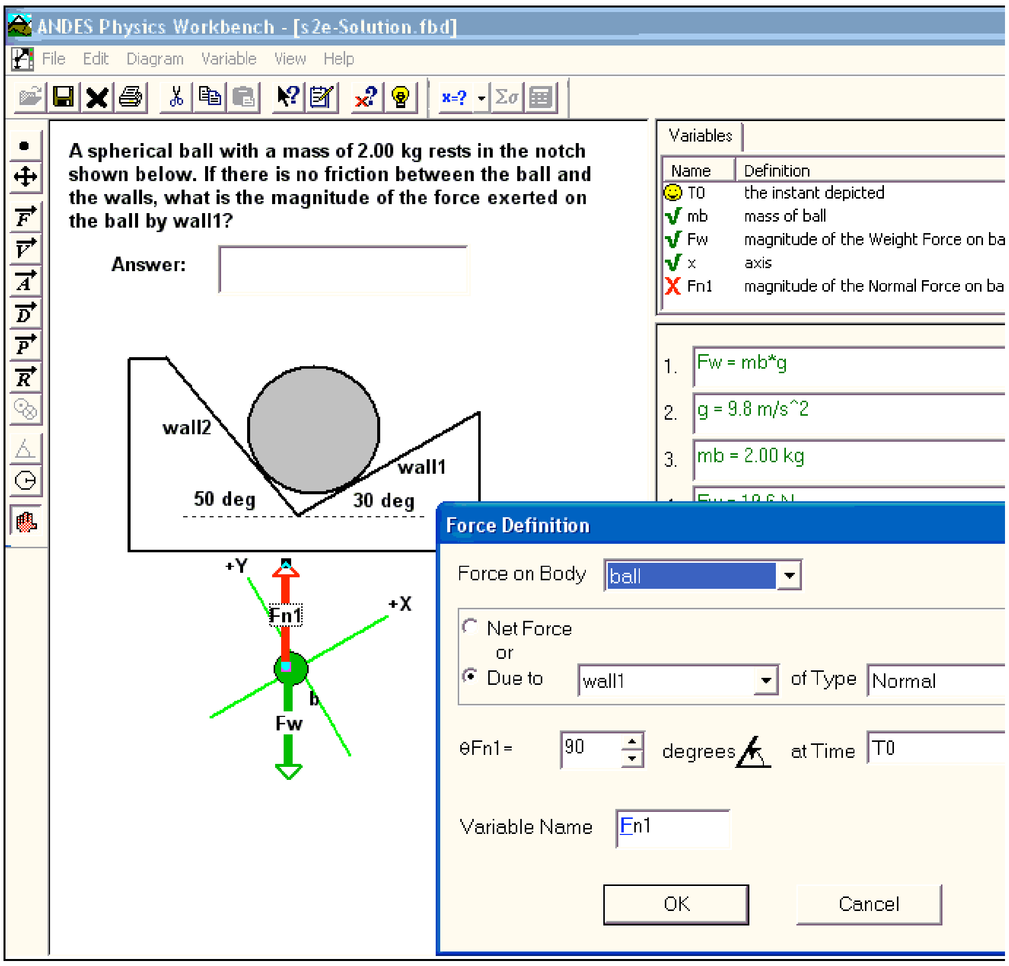
Fig.2. Screen of the Algebra I Cognitive tutor.

Step analysis: The tutor analyzes each student step in terms of a set of anticipated steps (S. Ritter, Blessing, & Wheeler, 2003). The set of anticipated steps for a problem is precomputed by solving the problem in all acceptable ways by running a rule-based problem solver. The rules are written to correspond to knowledge components. Each step is associated with the rules that were used to generate it during the precomputation. During tutoring, the student's step is matched against these anticipated steps. For some kinds of steps (e.g., menu selections), the matching algorithm is simple. For other kinds of steps (e.g., equations, points in a graph), the matching algorithm is complex. When a student's step matches an anticipated step, the student is credited in the assessment with having applied the associated knowledge components. In order to give error-specific feedback, some of the anticipated steps are generated by incorrect rules (called buggy rules). Text templates for error specific feedback are associated with buggy rules.

Step generation: When the student needs hints on what step to do next, the tutor selects a step using policies described in section "Which step to hint" A hint sequence is composed from text templates associated with the rules used to generate the step.

Andes

Outer loop: The Andes physics tutoring system (http://www.andes.pitt.edu/) helps students learn how to solve physics problems (K. VanLehn et al., 2005; K. VanLehn et al., 2002). Figure 3 shows its user interface. A physics problem is displayed in the upper left window. Students select the problem



ANDES Physics Workbench - [s2e-Solution.fbd]

File Edit Diagram Variable View Help

A spherical ball with a mass of 2.00 kg rests in the notch shown below. If there is no friction between the ball and the walls, what is the magnitude of the force exerted on the ball by wall1?

Answer:

Name	Definition
T0	the instant depicted
✓ mb	mass of ball
✓ Fw	magnitude of the Weight Force on ba
✓ x	axis
✗ Fn1	magnitude of the Normal Force on ba

1. Fw = mb*g
2. g = 9.8 m/s^2
3. mb = 2.00 kg

Force Definition

Force on Body: ball

Net Force or [✓ Due to] wall1 of Type Normal

θFn1 = 90 degrees at Time T0

Variable Name: Fn1

OK Cancel

Fig.3. Andes' graphical user interface.

they are to solve from a hierarchical menu.

Inner loop: The student solves the problem by making steps similar to the ones they would make if solving the problem with pencil and paper. One kind of step is to type an equation into one of the numbered boxes in the right window. Another kind of step is to draw a Cartesian coordinate system, such as the one showing in the lower left. A third kind of step is to sketch a vector, then fill out a dialogue box that defines it. A vector-drawing operation was in progress at the time the screen shot was taken, so its dialogue box covers part of the screen.

Every time the student makes a step, Andes gives immediate feedback. For most types of steps, Andes merely colors the step green if it is correct and red if it is incorrect. In Figure 3, the vector Fn1 is incorrect, so the student is in the process of modifying its definition.

Andes has two help buttons, both located in the top menu bar. The light bulb button initiates a hint sequence on what step to do next. The "x?" button analyzes the currently selected step. If it is incorrect and Andes can determine what the student's error type is, it gives error specific feedback in the form of a hint sequence.

Step analysis: Like the Algebra Cognitive Tutor, Andes analyzes non-equation steps by pre-computing anticipated steps and matching the student's step against them. However, this method does not work for the equation steps because there are too many anticipated equations to generate. For instance, for the problem in Figure 3, two correct equations are Fn1_y+Fn2_y+W_y=0 and W_y = –W. However, their algebraic combination, Fn1_y+Fn2_y –W =0, is also correct. If there are N primitive equations (i.e., generated directly from the application of a physics principle), then there are of the order of 2^N possible non-primitive equations (i.e., generated via algebraic combination). For all but the simplest physics problems, it is infeasible to list all anticipated equations. This is a problem not only for Andes, but for many other equation-based problem solving tutors in mathematics, science and engineering. See VanLehn et al. (2005) for a brief description of Andes' solution, and Shapiro (2005) for a complete description.

Sherlock

Outer loop: Sherlock (Katz et al., 1998) tutors the troubleshooting of a large piece of simulated electrical equipment, an avionics test station. It provides many views on the equipment and its schematics, so no screenshots are included here. Sherlock installs a fault in the simulated equipment, and lets the student find and repair it. Each troubleshooting problem is assigned a difficulty level, and Sherlock gradually increases the difficulty of the problems as the student practices.

Inner loop: Troubleshooting the equipment requires taking many steps, such as measuring voltages and replacing suspect parts. Sherlock gives unsolicited feedback only if a step is unsafe, that is, if the step would cause serious damage to the equipment or the student if it were done in the real world. However, students can ask for several different kinds of hints during problem solving.

Sherlock also makes extensive feedback available *after* the student has solved the problem. Sherlock's default post-solution feedback is to display a side-by-side summary of the student's solution and Sherlock's ideal solution. Students can click on steps in order to receive detailed feedback. Notice that Sherlock's post-problem discussion is conducted at the level of steps. One can think of Sherlock as having two inner loops: one that loops over steps *during* problem solving, and the other that loops over steps *after* problem solving. However, both inner loops execute once per step.

AutoTutor

Outer loop: AutoTutor (http://demo.autotutor.org/) teaches by engaging students in a natural language (English) dialogue (Graesser et al., 2004; Graesser et al., 2003; Graesser, Wiemer-Hastings, Wiemer-Hastings, & Kreuz, 1999; Person, Graesser, Bautista, Mathews, & TRG, 2001) For AutoTutor, a task corresponds to a single question, such as the one shown in the upper right of Figure 4, that has a complex answer. Its outer loop consists of selecting such a question and working with the student to get it completely answered.

Inner loop: The inner loop starts with the student typing in an initial answer to the top level question (see Figure 4; the student types into the lower right window; the whole dialogue is displayed in the lower left window). Typically, the answer is brief and vague, such as "The keys fall toward the earth." AutoTutor might respond, "Um, what else can you say about that?" The student might reply, "They stop when they hit the floor of the elevator." AutoTutor might give negative feedback by saying, "Well, the elevator and the keys are both falling, and the keys never catch up to the elevator's floor." AutoTutor next selects a target concept and tries to elicit it from the student with a question, such as "What can you say about the acceleration of the keys and the elevator?" This kind of dialogue continues until AutoTutor has got the student to articulate all the concepts required for a complete answer to the top level question.

AutoTutor has been used to compare output modalities. Figure 4 shows one of its talking heads. In this version, the tutor not only prints its response, it has the talking head speak it and gesture while doing so.

An AutoTutor dialogue is composed of tutor turns alternating with student turns. On most of the student turns, the student makes a small contribution toward completing the whole task. Those student turns count as steps, because they are a user interface event that contributes to a solution of the whole task. However, the student can also ask a question, such as "What is freefall?" Such a question is a kind of help request, and is not counted as a step.

Actually, it is convenient for analysis to include in the step not just the student's turn, but also the tutor question that preceded the student's turn and the tutor's minimal feedback, which is expressed as the first word or two of the tutor turn that follows the student step. For instance, in the dialogue shown in Figure 4, two steps are visible:

1. Tutor: What else can you say about it?
   Student: The keys and the boy have equal acceleration due to gravity.
   Tutor: Right.
2. Tutor: What about the acceleration of the objects involved?
   Student: It is equal.
   Tutor: Great job!

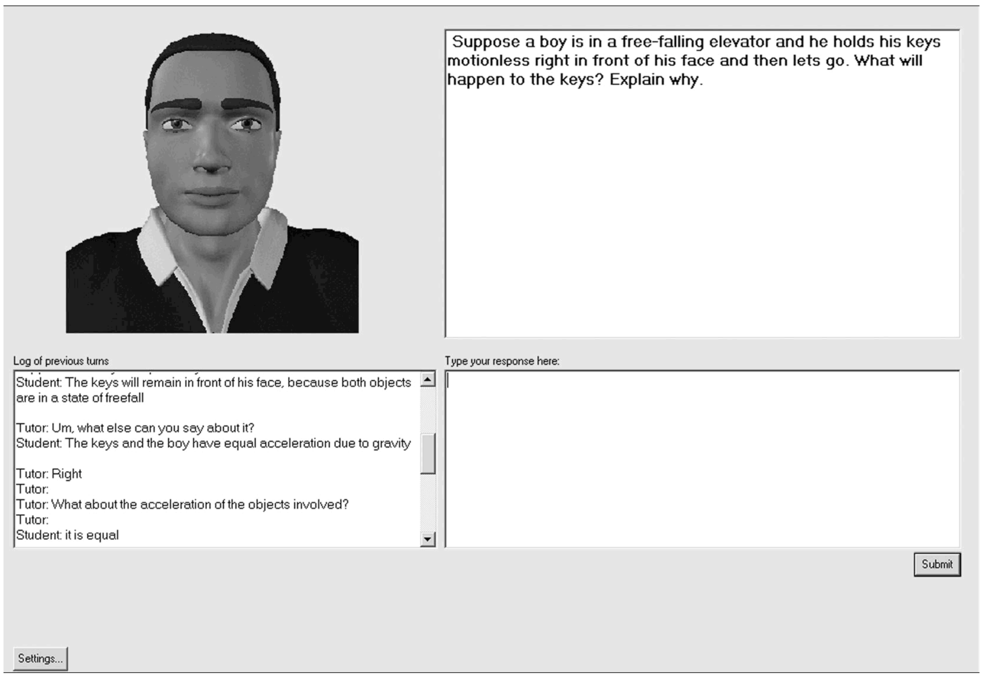
Step analysis: In order to illustrate AutoTutor's analysis of steps, let us use step 1 above. For each task, AutoTutor has a list of correct and incorrect task-specific pieces of knowledge that it expects to see in student answers, and the step analyzer indicates which ones of these are present in the student's answer. For instance, the step analyzer determines that step 1 contains the following:

- The keys' acceleration is g, the acceleration due to gravity.
- The boy's acceleration is g, the acceleration due to gravity.
- The boy and the keys have the same acceleration.

These are conclusions that are produced by applying knowledge components. For instance, the first two items above correspond to distinct learning events, wherein the student has applied the same knowledge component, "When an object is in freefall, its acceleration is g, the acceleration due to gravity," to two different objects, the keys and the boy. Let us use the term "learning event" to refer to one of AutoTutor's task-specific pieces of knowledge, even though it is more like the outcome of a learning event rather than the event itself.

In addition to having a list of all anticipated *correct* learning events, such as the ones mentioned above, AutoTutor has a list of several of the most important incorrect learning events. For instance, an important incorrect learning event in this problem is that the keys fall to the floor of the elevator. By listing such incorrect learning events when the problem is authored, the author can associate error-specific remediation with them.

To find out which learning events underlie the student's step, AutoTutor measures the semantic similarity between the text of the learning event and the text of the step. It uses a measure called Latent Semantic Analysis (LSA). Roughly speaking, a learning event is considered to be included in the student's step if the degree of semantic similarity is above a certain threshold. If the student's initial answer has all the correct learning events and none of the incorrect ones, then AutoTutor considers the task to be completed successfully and it moves on to the next one. If it detects that the student's answer has one of the incorrect learning events, it explains why that learning event is unjustified. If it detects that a correct learning event is missing, it tries to elicit it from the student by giving the student a sequence of increasingly specific hints. Thus, the learning events determine the tutorial dialogue that AutoTutor has with the student.


Suppose a boy is in a free-falling elevator and he holds his keys motionless right in front of his face and then lets go. What will happen to the keys? Explain why.

Log of previous turns

Student: The keys will remain in front of his face, because both objects are in a state of freefall

Tutor: Um, what else can you say about it?
Student: The keys and the boy have equal acceleration due to gravity

Tutor: Right
Tutor:
Tutor: What about the acceleration of the objects involved?
Tutor:
Student: it is equal

Type your response here:

Fig.4. AutoTutor's screen.

SQL-Tutor

Outer loop: SQL-Tutor (http://www.aw-bc.com/databaseplacedemo/sqltutor.html) teaches students how to write a query to a relational database (B. Martin & Mitrovic, 2002; Mitrovic, 2003; Mitrovic & Ohlsson, 1999). Each task consists of a database and information to be retrieved from it. In Figure 5, for instance, the student has been given a database of movie information and has been asked to write a query that will "List the titles and numbers of all movies that have won at least one Academy Award and have been made in or after 1988."

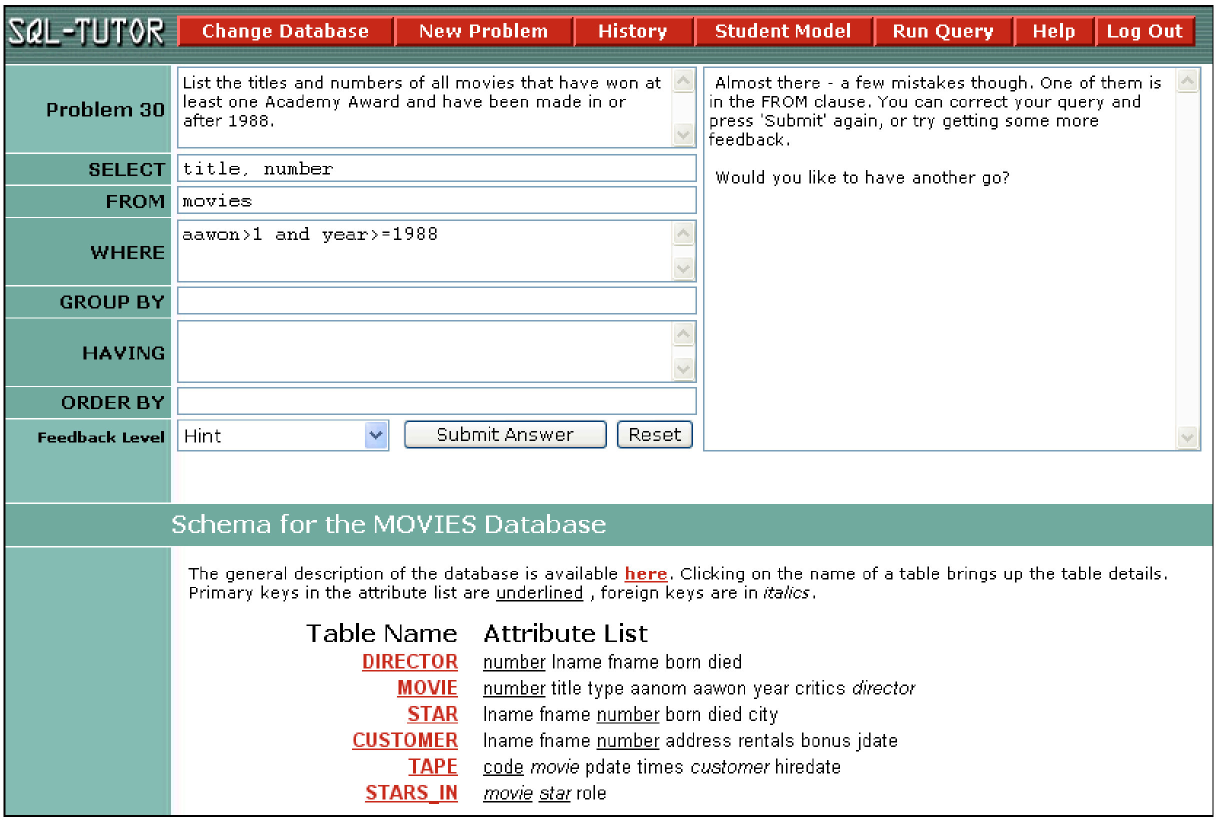
Inner loop: Students write a query in the SQL language by clicking on buttons and filling in blanks. This may take several minutes. At any point, they can press the Submit Answer button. The tutor, which has been completely silent up until now, analyzes the student's query to find its flaws. It gives a variety of levels of feedback and hints. For instance, under some conditions, it may select the pedagogically most important flaw and display a short text describing the error and how to fix it. Students can also select the level of feedback they would like to receive. After receiving a hint, students edit their SQL query and again submit it.

One way to think of SQL-Tutor's inner loop is that the student takes multiple steps, each comprised of filling in a blank in the query. Unlike tutors that give feedback as soon as a student had taken a step, SQL-Tutor delays its feedback until the student requests it. Although students can request feedback after each step, many prefer to fill in all the blanks before requesting feedback. However, regardless of when the feedback is given, it is essentially at the level of a single step. For instance, the tutor locates all the incorrect steps, picks one that is worth discussing and generates a hint for it.

Step analysis: In order to analyze steps, SQL-Tutor has a set of *constraints*, where a constraint consists of a relevance condition and a satisfaction condition. If the relevance condition is false of the students' step, then the constraint is irrelevant, so the tutor says nothing about it. If the constraint has a true relevance condition and a true satisfaction condition, then the constraint is satisfied and the tutor says nothing about it. If the relevance condition is true, and the satisfaction condition is false, then the student's step violates the constraint and the tutor has identified a topic worth talking about. In particular, every constraint has two messages. Depending on the feedback level selected by the tutor or the student, one of them may be presented to the student when the constraint is violated. One message describes the constraint and its violation briefly. The other presents more details.

Although the constraints are task independent, many of them refer to a correct solution of the problem, which is stored in the tutoring system. For instance, one constraint might check that for the FROM clause in the query, the student's value equals the value in the correct solution. In the case of Figure 5, the student's value "movies" does not equal the correct value, "movie", so the satisfaction condition of the constraint is not met.

The relationship between steps, learning events and constraints is quite simple in the SQL-Tutor. Each constraint corresponds to a knowledge component. For instance, the constraint that checks the value of the FROM clause represents the knowledge that the value of the FROM clause should be the name of a table that contains the fields mentioned in the SELECT clause. A learning event is the application of a constraint to a particular task. For the constraint just mentioned, the corresponding learning event is, "Because the SELECT clause has *title* and *number* as its value, the FROM clause should have *movie* as its value because the movie table contains the title and number fields." The step is just entering the fields and values of the SQL query that are determined by this constraint's application.



Problem 30: List the titles and numbers of all movies that have won at least one Academy Award and have been made in or after 1988.

SELECT	title, number
FROM	movies
WHERE	aawon>1 and year>=1988
GROUP BY	
HAVING	
ORDER BY	
Feedback Level	Hint

Almost there - a few mistakes though. One of them is in the FROM clause. You can correct your query and press 'Submit' again, or try getting some more feedback.

Would you like to have another go?

Schema for the MOVIES Database

The general description of the database is available here. Clicking on the name of a table brings up the table details. Primary keys in the attribute list are underlined , foreign keys are in *italics*.

Table Name	Attribute List
DIRECTOR	number lname fname born died
MOVIE	number title type aanom aawon year critics *director*
STAR	lname fname number born died city
CUSTOMER	lname fname number address rentals bonus jdate
TAPE	code *movie* pdate times *customer* hiredate
STARS_IN	*movie* *star* role

Fig.5. SQL-Tutor screen.