Instructional Science 17: 281–307 (1988)
© Kluwer Academic Publishers, Dordrecht – Printed in the Netherlands

Skill-oriented task sequencing in an intelligent tutor for basic algebra

DAVID McARTHUR, CATHY STASZ, JOHN HOTTA, ORLI PETER & CHRISTOPHER BURDORF
The RAND Corporation, 1700 Main Street, PO Box 2138, Santa Monica, CA 90406–2138, U.S.A.

Abstract. As part of a project to develop an intelligent computer tutor for basic algebra, we have been investigating task sequencing. In this paper we present an approach to task sequencing that is based on a component-skills view of intelligence and learning. We postulate that tutors use inferences about past and present student performance to determine a current skill set that will be the new target for learning. The skill set is then used as a basis for generating tasks that should elicit those skills. Current skill sets are modified slowly over time so that lessons appear coherent and well-planned. We first describe the approach at a general level, where it can be viewed as a cognitive model of human task sequencing. Then we discuss the implementation of the model in our intelligent algebra tutoring system.

Introduction

One of the most important skills effective tutors possess is task sequencing, the ability to generate an intelligent sequence of tasks for the student. To be effective, the sequence of tasks generated must adhere to several constraints. Locally, each task must be at the right cognitive level for the student – neither too simple nor too difficult. Globally, the sequence as a whole must be coherent: that is, successive tasks should deal with the same or related concept sets.

How do tutors generate tasks that satisfy these constraints? In this paper we present an approach we have been pursuing in the context of building an intelligent computer-based tutor for algebra. The approach is based on a component-skills view of intelligence and learning. First, we will describe the model at a general level, where it can be viewed as a cognitive model of human task sequencing. Next we discuss the implementation of the model in our algebra tutor, where it can be viewed as a component of an intelligent tutoring system.

The role of task sequencing in learning

The ability to produce coherent and appropriate sequences of tasks is a key feature of many learning environments (Collins, Brown and Newman, 1987). For example, in algebra tutoring, a tutor who fails to select questions that are at the "edge" of the student's current competence may not succeed in helping the student acquire new knowledge. Students may flounder on tasks that are too difficult or breeze mindlessly through ones that are too easy. Perhaps less obviously, task sequencing is important in learning environments that do not fit the question-and-answer mold. For example, Lave's (cited in Collins, Brown and Newman, 1987) analysis of apprenticeship learning shows that that masters reason carefully about the sequencing of assignments for novices. Apprentice tailors invariably practice putting together pieces of a precut garment before they learn to cut the pieces.

Reasoning about task sequences can also play a role in exploratory learning environments, or "microworlds" (see e.g., Lawler and Yazdani, 1987; Shute and Glaser, 1986). In such situations the student is typically free to choose the task to work on, since there is no tutor to reason about which kind of task would provide useful learning opportunities for the student. However, the students' *own* selection of tasks is often crucial in determining whether their explorations are informative or not. For instance, students using ARK (Smith, 1986) to explore the effect of gravity on the motion of objects may not learn about the independence of mass and velocity under gravity, unless their experiments are sequenced so that the motion of different objects is examined under identical configurations of position and gravity. In a more mundane context, students in self-paced concept acquisition experiments acquire concepts more effectively when the examples they examine vary systematically the features on which instances are defined.

A component-skill approach to task sequencing

In our view, tutors base task sequencing on a component-skills understanding of the students' learning needs. In the component-skills perspective, learning any complex body of knowledge entails acquiring a repertoire of interconnected skills. This perspective has its roots in diverse sources. In cognitive psychology, task analyses of many subjects have revealed that experts must learn complex networks of "rules", "scripts" or "schemas" (see e.g., Anderson, 1982; Newell and Simon, 1972; Schank and Abelson, 1977). In educational psychology there have been many attempts to analyze the components of mathematical competence (see e.g., Bell, Costello and Kuchemann, 1983), and to distinguish different component skills of mathematics (see e.g., Gagné and Briggs, 1974). In cognitive science, Goldstein (1982), among others, using Piaget as an inspiration, suggests a general theory of learning based on the evolution of component skills.

Many subjects are amenable to a componential analysis. In cognitive domains, such as algebra, learning frequently decomposes into acquiring interrelated concepts – how to collect terms, how to isolate a variable, etc. Complex physical skills decompose analogously, as the apprenticeship example above suggests. Apprentices acquire a global competence at tailoring not by practicing the skill in its whole and final form, but by first perfecting different component skills in their natural contexts of use. A component-skills approach can be applied to learning even in areas often regarded as more conceptual than procedural. For example, students learning concepts and laws of physics (either using ARK, as above, or in more traditional classroom settings), usually acquire mastery by learning simple component concepts in problem contexts, later graduating to more complex ones, or compound concepts.

We extend the component-skills approach to the analysis of tutoring expertise. According to this view, regardless of the domain, tutors use knowledge of interconnected networks of skills to chart a learning course for the student, generating sequences of tasks that help students learn efficiently and effectively.

A model of task sequencing

Figure 1 displays our model of task sequencing for tutoring based on the component-skills approach. As the figure suggests, we assume that a tutor begins a lesson by focusing on a set of component skills necessary to solve tasks in the chosen domain. Next, the tutor generates tasks that will tap those skills. Depending on the student's performance on the task (and on inferences about student performance maintained in a student model), the tutor will first update the target set of component skills, then generate a new task consistent with the modified skill set. If the current skill set is appropriate for the student, lesson tasks will be neither too difficult nor too easy. Assuming tasks are generated using an incrementally modified skill set, the task sequence should be coherent as a whole.

The figure is not intended as a complete model of tutoring, and parts not directly relevant to task sequencing are excluded. For example, we do not discuss *teaching tactics*, by which Ohlsson (1986) means the expertise needed to guide the student through local difficulties (e.g., remedial tactics used by tutors to deal with local student errors). In addition, several *strategic* aspects of tutoring are also not discussed. Strategic reasoning concerns tutorial decision-making of greater scope than local interventions. We regard task sequencing as one aspect of teaching strategy. Other aspects not described include plans for presenting concepts (e.g., Ohlsson, 1986) and techniques for opportunistically introducing issues (e.g., Burton and Brown, 1982). Some of these facets of a more complete tutoring system will be discussed tangentially in subsequent sections on the implementation of task sequencing in a tutor for algebra.

The model includes three kinds of components:

- *Active memory* components encode data structures that the tutor computes during a tutoring session. Often this information is transitory: for example, the current student performance changes from task to task. However, many aspects of the student model may endure across several tutoring sessions.

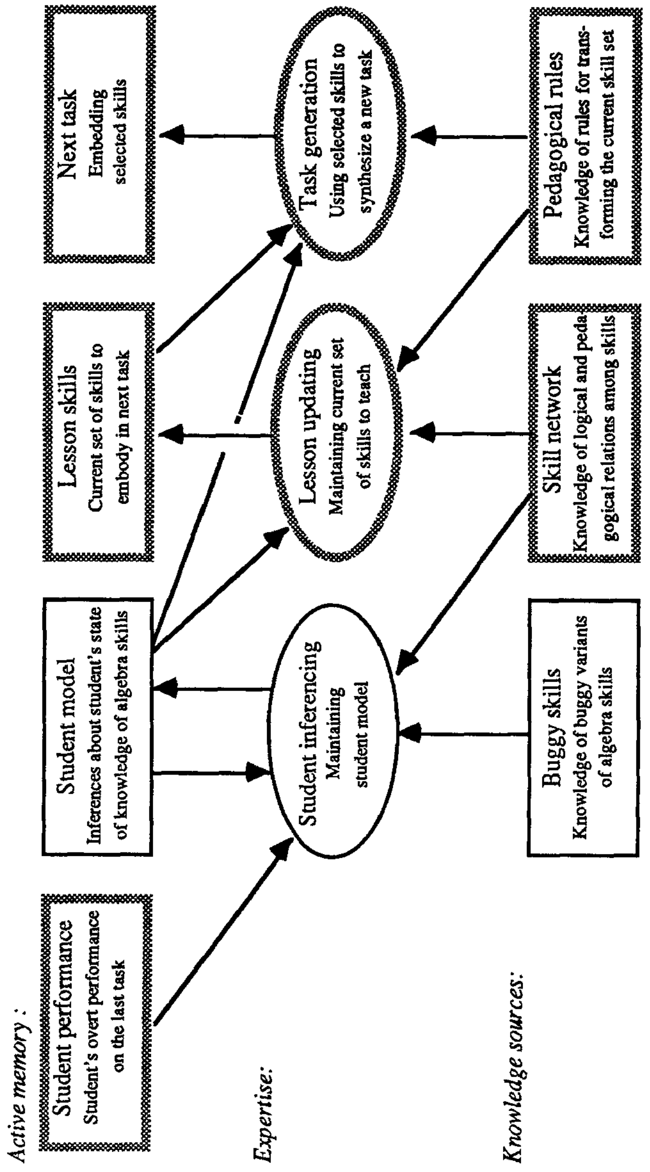

Active memory :
Student performance
Student's overt performance on the last task
Student model
Inferences about student's state of knowledge of algebra skills
Lesson skills
Current set of skills to embody in next task
Next task
Embedding selected skills
Expertise:
Student inferencing
Maintaining student model
Lesson updating
Maintaining current set of skills to teach
Task generation
Using selected skills to synthesize a new task
Knowledge sources:
Buggy skills
Knowledge of buggy variants of algebra skills
Skill network
Knowledge of logical and pedagogical relations among skills
Pedagogical rules
Knowledge of rules for transforming the current skill set

Figure 1. Overview of the task sequencing components of a skilled tutor

Active memory components encode local computations and choices made by the tutor during a particular lesson with a specific student. Knowledge sources refer to different kinds of long-term knowledge that the tutor must bring to bear to make lesson control decisions. Expertise comprise the inferential skills that use long-term knowledge and information in active memory to compute tutorial decisions. Versions of the shaded components have been implemented in our intelligent tutor for algebra (see section on implementing task sequencing).

- *Expertise* components perform the computations and inferences that result in active memory structures. In order to make intelligent computations, expertise exploits various kinds of knowledge sources.
- *Knowledge sources* encode kinds of general information about students, teaching, and subject matter. This information is distinct from data about specific students, teaching situations, and tasks, represented in active memory. Knowledge sources are often referred to as long-term memory, since their contents change little over time. [1]

The following paragraphs briefly describe the components shown in Figure 1.

Student performance refers to the "uninterpreted" behavior of the student, generated when he or she solves a task. In algebra, for example, overt behaviors include transformations of algebraic expressions (e.g., $2x+1 = 5 \rightarrow 2x = 5+1$).

Student model is a database that stores inferences made on the basis of student performance that concern the student's knowledge or lack of knowledge about skills that underlie overt performance. For example, given the above overt behavior, one facet in a model for an algebra student might be an inference that the student believes a buggy version of a rule for moving a term from one side of an equation to another – the student does not appear to understand that the inverse of the moved term should be added to the other side. The role of student modeling in tutoring recently has been a topic of extensive investigation (e.g., Anderson, Boyle and Yost, 1985; Brown and Burton, 1978; Matz, 1982; Ohlsson, 1986; Sleeman and Smith, 1981).

Student inferencing represents the tutorial expertise that is responsible for modifying a set of inferences in the student model in light of new student input. In algebra, for example, new student input can be interpreted in terms of algebra skills by an inductive process that matches the behavior against that predicted by skills and buggy versions of skills. Thus, student inferencing relies on knowledge sources that describe *buggy skills*, and the visible behavior they would predict. Existing inferences in the student model may control its performance as well. For example, before integrating a new inference based on current student performance into the model, student inferencing expertise may consult the existing model to determine whether the proposed inference is consistent with current inferences.

Generally, we view student inferencing as cumulative: inferences are maintained over time. This contrasts with the usual treatment in modeling (Ohlsson, 1986), where student performance at a given point in time strictly controls the inference made and inferences are discarded once used. Another novel aspect of student inferencing in our model is its overall role in tutoring. Unlike most intelligent tutoring systems (ITS) (e.g., Anderson, Boyle and Yost, 1985; Ohlsson, 1986), we do not use the student model just to compute the feedback to give a student when he or she has made a mistake. Instead, we suggest that a main use of student inferences is to contribute to decisions about the skills to focus on in a given lesson.

Lesson skills represent the skills the tutor is currently focusing on in a lesson. These skills are the basis for tasks that the tutor generates. Looked at another way, they represent the knowledge goals the tutor chooses for the student at any point during learning. Current lesson skills are typically limited to one or two; it rarely makes sense to teach more than one skill at a time. Thus, the current lesson skills usually comprise a small subset of the larger set of skills represented in the *skill network* or in the current student model.

In a subject of any depth, many kinds of skills can be the target of tutoring. For example, in algebra, a competent problem solver must learn low-level axioms of algebra (e.g., commutativity of addition), and strategy-level heuristics for applying the axioms (e.g., attracting instances of a variable to the same side of an equation). Less obvious, but equally important, are even higher-level "meta-cognitive" skills (Collins, Brown and Newman, 1987), such as the ability to debug or fix incorrect solutions or monitor one's own progress. Schoenfeld (1985) adds to this an even more abstract kind of understanding he refers to as (mathematical) belief systems. Collins and Brown (Brown, Collins and Harris, 1978; Collins, Brown and Newman, 1987) have demonstrated that these different knowledge levels are important in learning such diverse subjects as reading, writing, mathematics, and electronic troubleshooting. In our discussions we focus mainly on skills at the level of local heuristics, since careful analyses of higher-level skills are not yet available. Generalizing our model of task sequencing to these kinds of skills will depend on whether we find a natural decomposition for skills at these levels.

Lesson updating is the process that decides how to modify the current lesson skills in light of changes to the student model. It is the heart of our view of task sequencing. One main activity in lesson updating is to decide how to incrementally transform skills in the set of current lesson skills. If changes to the student model indicate the student has mastered a skill which is currently the focus of the lesson, updating must decide which new skill is a logical next target, and replace the old skill with the new one in the lesson skills. Similarly, if the student model indicates the student has failed to master a target skill, lesson updating must decide to keep the target skill, or even "back off", replacing it with a logically or pedagogically simpler one. To determine skills that are logically simpler (or more complex) replacements for a current lesson skill, lesson updating relies on the skill network, a long-term knowledge structure that records logical and conceptual relations between algebraic skills.

A second activity of lesson updating involves the addition of totally new skills to the lesson. In completing a task, the student may make a mistake that is unrelated to the current lesson skills. This may be reflected in the student model as a new inference about student misconceptions. The tutor may consider adding this skill to the agenda of lesson skills. However, care must be taken in such decisions. For example, it does not make sense to add a new skill to the lesson if there are already several focused skills. High-level knowledge about when to add skills, as well as what skills to add, is supplied by *pedagogical rules*. In general, they are responsible for determining which skill transformations or additions in the lesson set are pedagogically reasonable, while the skill network is merely a lower-level record of skill relations.

Task generation is a process that creates new problems or challenges for the student. Our skill-oriented view of task sequencing suggests that tasks are generated to embed a skill that is in the current set of lesson skills. For example, in algebra, if a current skill is "moving a term from one side of the equation to another" (focusing on the additive inverse) the task generator might compose a question whose solution will necessarily involve exercising that skill (e.g., $2x-4 = 10$). In addition, however, task generation must adhere to an important pedagogical constraint: a task should elicit the target skills and a minimum of others. Consequently, task generation is not an isolated process, but exploits information in the student model and is driven by pedagogical rules which characterize the skills, not in the current skill set, that may be included in a task.

In a complex subject, just as there may be several different levels of skills or knowledge to learn, there is often a variety of different kinds of tasks the tutor can construct to get its point across. Our model of task sequencing in tutoring is not limited to a question-and-answer style of teaching only. In algebra, for example, in addition to generating equations for the student to solve, the tutor could decide to model for the student by solving a problem himself or herself (Collins, Brown and Newman, 1987); give the student an incorrect solution and ask him or her to debug it; ask the student to "re-represent" an equation as a word problem; and so on. In learning environments with several students (e.g., apprenticeship), the tasks generated by the tutor(s) might be divided into several pieces for different students.

Not all of the current lesson skills necessarily act to constrain the generation of new tasks for the student. For example, we envision that one skill may serve to generate a new task, while a few others will just be kept "active" while the student solves the task. As the student progresses, if the opportunity arises, the tutor may highlight those additional skills. For example, in algebra, the tutor's main goal might be to help the student learn how to solve equations with implicit coefficients (e.g., $3x-x = 10$). At the same time, the tutor may also want to encourage the student to learn "debugging" skills. Consequently, if the student solves a task incorrectly, the tutor will not only focus on illustrating the correct technique for adding implicit coefficients, but also may emphasize debugging concepts, such as going back and testing the validity of each problem-solving step.

To summarize, we believe task sequencing aims at generating tasks with "a point." In our view, the point of a task must always be a skill that the tutor believes the student needs to acquire or improve. Several sources of information are jointly consulted to determine what the point of a task should be. New skills in the student model suggest plausible points in terms of the student's perceived strengths and weaknesses. This set of alternatives is further reduced by continuity and pedagogical considerations until only one or a few alternatives remain. Thus we achieve coherence in the set of tasks we pose the student, without invoking any explicit rigid lesson plan.

Related research

In this section we briefly discuss other work on task sequencing, including research from cognitive psychology, and computer-based instruction.

Research in cognitive psychology

Collins and Stevens (1981a, 1981b) have been developing a cognitive theory of interactive teaching from a different perspective. While some of their data has come from classroom settings, they have examined one-on-one Socratic tutoring most carefully. They describe the teaching goals, together with strategies for communicating the goals, and an overall control structure for interactive teaching. Some aspects of their view have analogues in ours. The teaching goals they mention are similar to the target skills that are the focus of task sequencing, and, like our task generation module, many of their teaching strategies take a given goal and translate it into a specific case for the student to work on.

In related work, Collins, Brown and Newman (1987) describe a cognitive apprenticeship approach to learning and tutoring. In this important paper they describe some general ideas for task sequencing which we consider later . In addition, they refer to teaching methods, including modeling, coaching, scaffolding, fading, articulation, and reflection. These comprise general principles of teaching. For example, scaffolding and fading suggest that the tutor should first help the student in solving tasks, but as the student gets better, it should gradually reduce its intervention. We regard these principles as important constraints on how tasks should be managed, once created, and on the roles that the student and tutor should take in executing the tasks. However, the principles themselves do not describe the reasoning involved in actually creating tasks. Similarly, their idea of cognitive apprenticeship is not a task sequencing technique, but rather can be seen as a global philosophy of learning and tutoring. As we noted above, and will discuss again below, we believe that specific task sequencing policies we create can be consistent with this general view.

VanLehn (1983, 1987) discusses STEP theory, mentioning various "felicity conditions" that describe pedagogical principles for sequencing concepts (e.g., "introduce one disjunct per lesson"). Although such conditions are discussed in the context of inductive learning of procedures from examples, it is obvious that similar kinds of conditions or rules apply to a variety of learning environments, including ones where tutors generate tasks for students. The modular component skills we describe in this paper can be viewed as the disjuncts, or pieces, which VanLehn discusses.

Research in computer-based instruction

In contrast to research in cognitive psychology, computer-based instruction programs give a more precise idea of task sequencing in a one-on-one tutorial setting. Unfortunately the task sequencing embedded in the early computer-aided instruction (CAI) programs usually consisted of a simple algorithm for branching among a few fixed alternative questions (e.g., Chambers and Sprecher, 1980). Such rigid plans do not provide a model of how a tutor can adapt the generation of tasks to suit the particular needs of each student. Other early programs, for example TICCIT (Mitre Corporation, 1976), leave most decisions about task selection to the student; they essentially engage in no reasoning about task sequences. Some generative CAI programs do attempt to dynamically compute tasks for the student (Palmer and Oldehoeft, 1975). However, as O'Shea and Self (1983) point out, these programs represent very little of the knowledge required to reason intelligently about task sequencing. For example, they do not possess explicit models of task difficulty, like those encoded in our skill network.

More recent research in intelligent tutoring systems offers more promise in supplying a cognitive theory of task sequencing, but, to date, few have focused this issue. Ohlsson (1986) reviews a wide variety of intelligent tutors, and notes that few use their expertise to influence the global structure of lessons. Most effort has been aimed at the issues of student modeling (knowledge of student) (e.g., Anderson, Boyle and Yost, 1985; Sleeman and Smith, 1981), and domain expertise (e.g. Clancey, 1979), instead of teaching knowledge.

Programs such as WEST (Burton and Brown, 1982) and GUIDON (Clancey, 1979, 1983) formalize teaching tactics but offer little in the way of more global or strategic rules for overall control of learning. BIP (Barr, Beard and Atkinson, 1976; Wescourt, Beard and Gould, 1977) is still one of the few attempts to generate intelligent sequences of tasks in a tutoring system. Like the current proposal, BIP is skill-oriented, embedding a curriculum of concepts that need to be learned. However, BIP cannot generate tasks, but rather relies on a library of preconstructed questions. Goldstein's WUMPUS tutors (Goldstein, 1982) uses a genetic graph representation of skills that is a descendent of those used in BIP. It encodes not only generalization, specialization, analogy, and prerequisite relationships between skills, but also correction or deviation relationships that help capture the evolution of skills from formative to mature states. Goldstein (1982) discusses several possible uses of genetic graphs in tutoring, including their potential in suggesting topics and tasks for the student. However, Goldstein mainly focuses on the role of genetic graphs in supplying multiple explanations, representing the syllabus, and providing a basis for student modeling. In this paper we provide a more detailed computational model of how skill networks, or graphs, can be used to reason about tasks for the student.

Peachy and McCalla (1986) attempt to use planning ideas from robotics as a basis for dynamically creating and revising lesson plans, but their work is only at the formative phase. To our knowledge they have not yet implemented a significant task sequencing structure. MENO-TUTOR (Woolf and McDonald, 1984) is a program that attempts to model the discourse strategies of human tutors. It embeds several levels of tutoring knowledge, including tutoring tactics, strategies, and more general pedagogical states. MENO-TUTOR is one of the few ITS to include a significant pedagogical component. Nevertheless, its tutorial knowledge is not used for task sequencing so much as to provide sensitive feedback to a student within a single task.

Generally, most intelligent tutors are clever at the level of individual tasks and individual responses. They can supply sophisticated feedback about the student's performance and possible misconceptions. However, they are not particularly intelligent at the lesson level. They do not use what they have learned about a student on previous tasks to influence strategic decisions about subsequent tasks for the student. Our research on task sequencing is an attempt to rectify this situation.

Implementing task sequencing in an intelligent computer tutor for basic algebra

In the following section we explain how we are implementing the general model of task sequencing in our intelligent tutor for basic algebra. As background to the implementation discussion, we first briefly overview the tutor environment. The tutor is described in greater detail in McArthur, Stasz and Hotta (1987).

Overview of the algebra tutor

Our intelligent algebra tutor currently runs on Sun Microsystems workstations. It has been tested at The RAND Corporation, using local high school students, and we have located six workstations at Santa Monica High School, where the tutor will be more extensively tested and developed over the next several years. There are several versions of the tutor, each presenting a slightly different interface to students, defining different roles for the students, and concentrating on different learning goals. Here we focus on a version in which students solve equations, much as in homework practice.

The student sees the tutor as a collection of windows and menus, as shown in Figure 2. The menus on the left allow the tutor and student to converse about reasoning and problem solving. To the right of the menus, on the bottom, is the "WorkSpace," where the student creates each new line in his or her solution. New lines or reasoning steps can be created by either selecting commands from menus (as in Figure 2), typing in algebraic expressions, or writing them on an electronic tablet. To the right of the Workspace is the "CommentSpace," where the tutor sends textual feedback to the student.

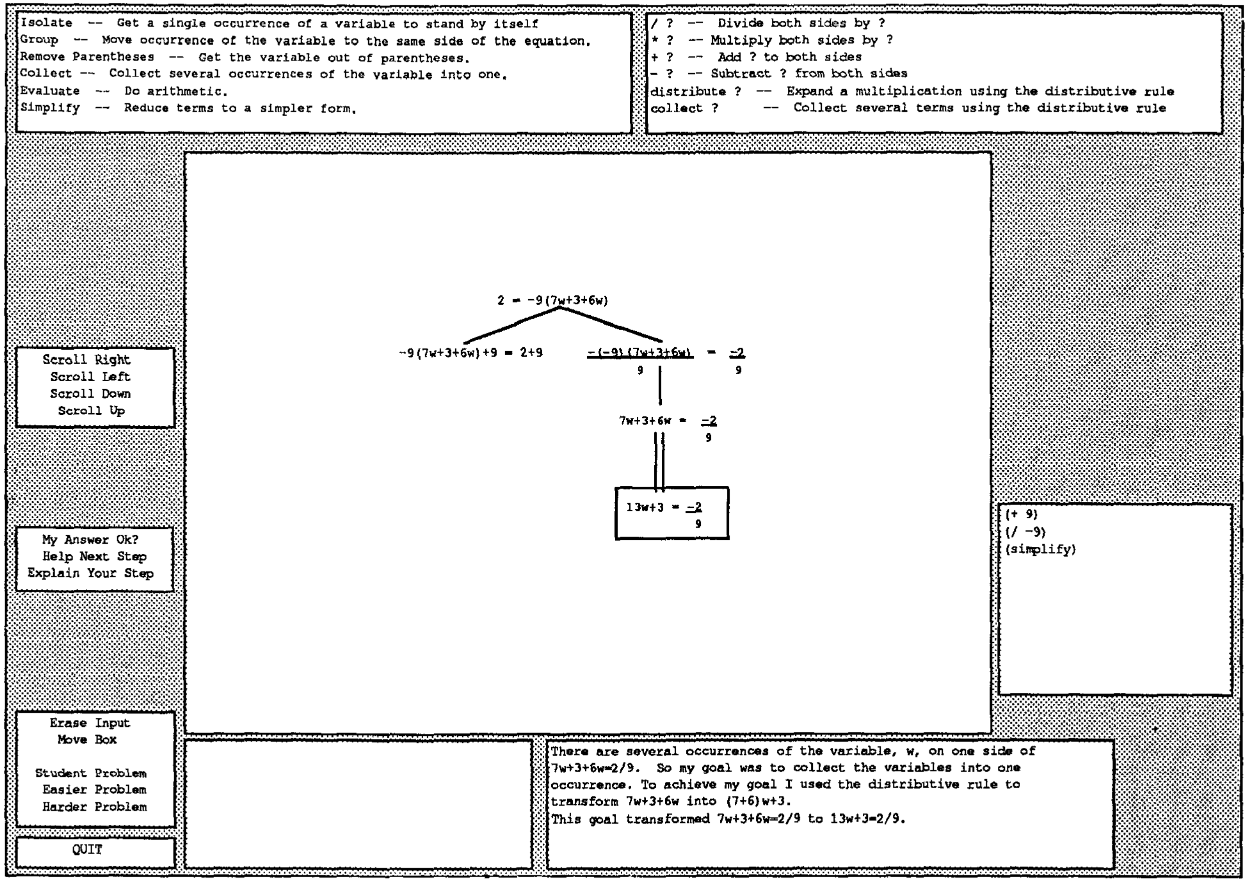
The large window in the upper right is the "DisplaySpace," where the student's reasoning is recorded and queried. Problem solving is represented here as a *reasoning tree*. Many of the menu items to the left are used to manipulate the "nodes" in this tree. For example, "Explain Your Step" permits the student to point at parts of the reasoning tree done by the tutor and obtain justifications for the tutor's reasoning. Similarly, "Help Next Step" allows the student to obtain several levels of hints from the tutor (see Figure 2).

Like AlgebraLand (Collins and Brown, 1986), the tutor displays the student's work as a solution tree, thus "reifying" the student's reasoning process by showing connections between steps. Each branch in the tree represents an alternate solution, or line of attack on the problem.[2] Hence a tree representation allows easy comparison of different solutions, both the student's and tutor's. Moreover, menu items like "Move Box" permit the history to be exploited effectively. The student selects this item when he or she wants to return to a previous solution path. The tutor then allows the new current expression (i.e., the one that it boxed) to be changed by pointing to any other expression in the tree. That expression then becomes the current one.

In general, the skills of the tutor in providing hints and explanations comprise its *intra-task* tutoring expertise. These tactics enable the tutor to coach a student through the local complexities of solving a single task. By contrast, the task sequencing component of the tutor embeds *inter-task* skills; tutorial reasoning on a more global or strategic level.

Current implementation of task sequencing

We have currently implemented only a subset of the general model of task sequencing. It includes operating versions of student performance measures, the skill network, pedagogical rules, lesson updating (which modifies the current lesson skills), and task generation (which creates the next task) (see shaded components of Figure 1). In spite of its limitations, we believe the current task sequencing implementation represents a significant advance in formalizing the expertise required to make strategy-level tutoring decisions.



```
Isolate  --  Get a single occurrence of a variable to stand by itself
Group  --  Move occurrence of the variable to the same side of the equation.
Remove Parentheses  --  Get the variable out of parentheses.
Collect --  Collect several occurrences of the variable into one.
Evaluate  --  Do arithmetic.
Simplify  --  Reduce terms to a simpler form.
```

```
/ ?  --  Divide both sides by ?
* ?  -- Multiply both sides by ?
+ ?  --  Add ? to both sides
- ?  -- Subtract ? from both sides
distribute ?  --  Expand a multiplication using the distributive rule
collect ?     --  Collect several terms using the distributive rule
```

```
There are several occurrences of the variable, w, on one side of
7w+3+6w=2/9.  So my goal was to collect the variables into one
occurrence. To achieve my goal I used the distributive rule to
transform 7w+3+6w into (7+6)w+3.
This goal transformed 7w+3+6w=2/9 to 13w+3=2/9.
```

Figure 2. View of the algebra tutor interface

At the heart of our current implementation of task sequencing is a set of rules of the form "*if <conditions>, then <actions>*". The *<conditions>* include various tests and predicates on student performance, while the *<actions>* are recommendations about how the tutor should adjust the current lesson skills that are being using to generate tasks for the student. In the next section we describe the measures of student performance that enter into the *<conditions>* of the rules. The following sections discuss the *<actions>* in more detail and show how tasks are actually generated.

Implementation of student performance measures

As a student answers an algebra problem we record several kinds of information. Individual problem records are accumulated into student histories. Student histories are retained in files and may extend across many lessons. The records consist of relatively superficial pieces of information about the student's performance, in the sense that they contain no abstract inferences, only "facts". Thus, student modeling, or diagnosis of the misconceptions underlying student performance, plays no part in the current task sequencing structure. Since student modeling is bypassed, these measures act directly as input to the pedagogical rules that decide which skills to focus on at any given time.

The intelligent tutor records many different kinds of information for each question the student answers:

- (A) The question itself.
- (B) The skill set that generated the question (see the section on skill sets).
- (C) Whether the student answered the question acceptably or not.
- (D) A list of invalid steps the student generated in answering the question.
- (E) A list of the inappropriate steps the student generated in answering the question. An inappropriate step is one that is logically valid, but which does not get the student closer to a solution.
- (F) Number of times the student asked the tutor to do a step in the solution of a problem before the student obtained a correct answer by himself or herself.
- (G) Number of times the student asked the tutor to go back to some previous step in a solution.
- (H) Number of times the student asked the tutor to elaborate a step that the tutor has done in a solution. Elaborations give the student more detail about the tutor's reasoning, often indicating that the student has not understood the tutor's actions.
- (I) Number of times the student asked the tutor to explain a step that the tutor executed in solving a problem.

- (J) Number of times the student asked the tutor to give a hint about what to do for the next reasoning step.
- (K) Number of times the student asked the tutor to indicate whether or not a step done by the student was correct.
- (L) Post-solution queries. All the above requests for information are counted only before the student has obtained the correct final answer. The same requests after the right answer has been obtained have a different intent than "pre solution" queries and are accumulated here.
- (M) Total time required to solve the question, and the time required for each step or action within a solution.

These raw measures are not used directly in the rules governing task sequencing. Instead, a score is computed for each question, using these factors, and the score enters into the conditions of the rules. The score for a question is computed using the following formula:

if C = "no", score = 0

otherwise, score = max(0, 100 – (20*D+10*E+10*F+5*J))

If the student answers the question incorrectly, he or she scores 0. Otherwise, the student may get a maximum of 100 for a question, with points taken off for various kinds of mistakes made and assistance received. Each invalid step costs 20 points; each inappropriate step costs 10 points; each request asking the tutor to do a problem solving step for the student costs 10 points; and each request for a hint costs 5 points. No other requests enter into the score for a question. Scores for a question are not shown to a student, but are used only for internal computations.

The weightings in the formula were determined empirically. In our classroom experience, they resulted in assessments of student performance that enabled the tutor to generate sequences of tasks that appeared natural to students and observers. The value of the formula is that it includes a variety of factors that are not usually available to CAI programs that assess student performance (e.g., number of inappropriate steps, number of requests for explanation). Most of these factors are available from the algebra expert system embedded in our tutor. In spite of these advantages, we view the formula as a short-term solution that is subject to change. Little importance should be associated with the specific weights assigned to the various factors, or even to the decision to exclude some factors while including others. For example, the number of explanations the student requests in arriving at an answer might be a reasonable indicator of his strength on a particular skill. Hence, in the future, we might wish to include that factor in our computation. Such changes will be made as we see how the formula performs with students in a classroom setting. A more ambitious change we hope to consider shortly is to eliminate use of quantitative measures of performance, in favor of more qualitative judgements.

Implementation of the skill network

The measures of student performance discussed in the previous section are referenced in the conditions, or antecedents of rules, that the tutor uses to reason about task sequencing. The actions, or consequents of the rules, reference the skills that should be the topic of subsequent tasks, given the current conditions. Skill sets are lists of basic algebra skills, some of which are outlined in Table 1.

The skills in Table 1 are a sample of a relatively large number we have compiled to cover the whole range of equations in basic algebra. We emphasize that they constitute a reasonable analysis of the subskills required for this simple domain, but not the only possible analysis. We used our intuitions, and observations of students in classroom settings, to decide which properties of equations are important enough to establish a distinct skill. For example, we decided that $x-4=5$ differs from $x+4=5$ with respect to the skills required to solve it, because moving a positive term to the other side seems a cognitively distinct skill from moving a negative term. On the other hand, we decided that $x-4=5$ is not different than $x-3=5$ because we expect that skills for moving negative terms generalize trivially to different integers.

Table 1. Skill sets (lists of basic algebra skills)

Skill	Example	Description
isolate+pos	x+2=5	Isolating a variable by moving a positive number added to it to other side.
isolate+neg	x–2=5	Isolating a variable by moving a negative number added to it to the other side.
isolate+sc	x+0=5	Special case of simple isolation with 0
isolate*pos	2x=4	Isolating a variable by moving a positive number multiplied to it to the other side.
isolate*neg	–2x=4	Isolating a variable by moving a negative number multiplied to it to the other side.
isolate*pos-sc	x=4	Special case where coefficient to be moved is an implicit 1.
isolate*neg-sc	–x=4	Special case where coefficient to be moved is an implicit –1.

Neither of these decisions is necessarily correct. Their plausibility rests on psychological assumptions about learning and problem solving. For instance, if students easily generalize from moving a positive number to moving a negative number, then $x-4=5$ should not be treated differently than $x+4=5$ and one subskill for moving added terms, not two, should be posited. However, we have observed that students generalize very slowly. For a long time, they see moving a positive term as different than moving a negative term. Eventually, these skills may generalize into one procedure, possibly using composition and generalization techniques, such as those described by Anderson (1982). However, in our analysis of component subskills of algebra we are concerned with representing the skills of a student who is still learning how to solve equations, not with those of an ideal problem solver. Thus, our subskills often distinguish cases which might be logically subsumed by a more general skill.

While students do learn individual skills, it is also important for them to practice sets of skills together. For example, after students have mastered questions that exercise `isolate+` (e.g., $x-5=9$), and `isolate*` (e.g., $2x=8$), they should work problems that tap both skills (e.g., $2x-5=8$). We have defined *skill sets* to encode such compounds (including degenerate compounds with only one member). In the current task sequencing structure, skill sets, not individual skills are the focus of tutoring.

We have compiled a database of skill sets representing all the sets we might wish to tutor in basic algebra. Our database of skill sets does not include all logically possible skill compounds, since only a few skills are reasonable to teach together. For example, a task that elicits the compound `[isolate+, eval+]` (e.g., $x+3 = 5-4$) is probably useful; however the compound `[isolate+, eval+, eval+, eval+, eval+]` (e.g., $x+3+4= 4+5+7$) is probably not. Thus the skill set database encodes an important pedagogical decision about which skill compounds are good goals for teaching, and which are not.

The skill set database is not a random collection of skill sets. Instead, it is a network, where skill sets are connected to other sets by various relationships. This is shown in Figure 3 which represents a portion of the skill set network of linear equations. Each node represents a skill set (in parentheses), and gives an example question that would elicit the set. Connections between nodes denote different kinds of relationships between skill sets. "S" indicates that one set is a specialisation of another; "G" indicates that one set generalises another; and "C" means that a skill set is a compound of several others. Arcs labeled "P" indicate that one set is a prerequisite for another.

Skills in braces "{}" represent skills that necessarily combine with others in the set. Other non-singleton sets are deliberate combinations. In both cases, a pedagogical decision has been made that the combination is worth teaching explicitly.

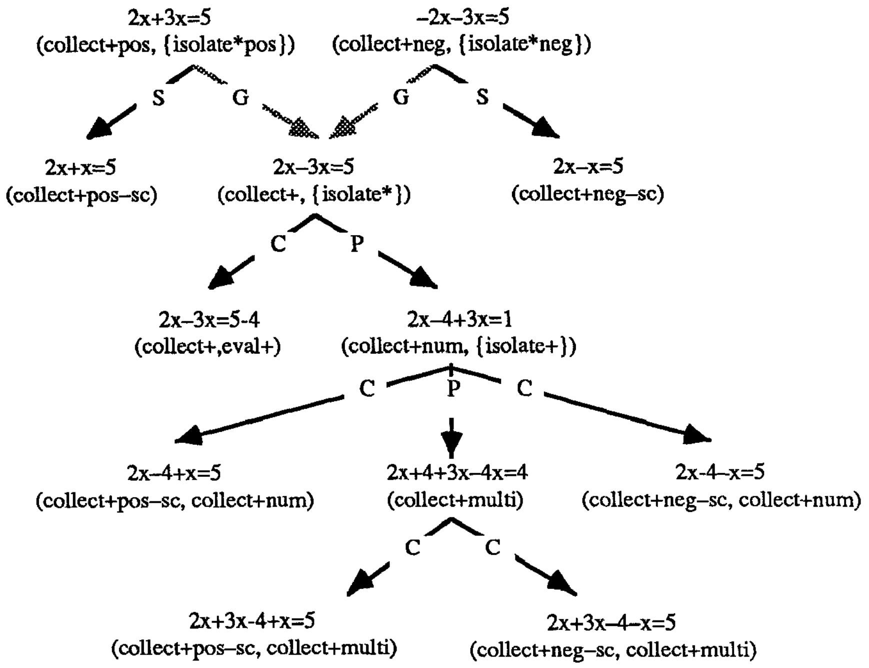

2x+3x=5
(collect+pos, {isolate*pos})
-2x-3x=5
(collect+neg, {isolate*neg})
S
G
G
S
2x+x=5
(collect+pos–sc)
2x–3x=5
(collect+, {isolate*})
2x–x=5
(collect+neg–sc)
C
P
2x–3x=5-4
(collect+,eval+)
2x–4+3x=1
(collect+num, {isolate+})
C
P
C
2x–4+x=5
(collect+pos–sc, collect+num)
2x+4+3x–4x=4
(collect+multi)
2x-4–x=5
(collect+neg–sc, collect+num)
C
C
2x+3x-4+x=5
(collect+pos–sc, collect+multi)
2x+3x–4–x=5
(collect+neg–sc, collect+multi)

Figure 3. Skill set network (collect component)

The collect group of skills reduces multiple occurrences of a variable to a single instance. Several skills for complex types of collection are included. `collect+num` collects two occurrences of a variable, when a number is interpolated. `collect+multi` collects more than two occurences of a variable, with interpolated numbers. This group has the isolate group as a prerequisite.

As Figure 3 indicates, the relationships between skill sets fall into several different categories.

- *Specialization* (labeled "S" in Figure 3). One skill (or set) is a specialization of a given skill if it applies to tasks that are special cases of the tasks to which the given skill applies. For example, collecting terms with an implicit coefficient (e.g., $2x+x=10$) is a special case of collecting terms with integral coefficients (e.g., $2x+3x=10$). Thus, the skill `collect+pos-sc` is a specialization of `collect+pos`. It is not necessary that students learn special-case skills, since there is a general skill that takes care of all regular and special cases. However, empirically, special cases are treated quite differently when students are learning. Students typically are taught, and learn, special case skills after learning regular-case skills.

- *Generalization* (labeled "G" in Figure 3). One skill (or set) generalizes a given set of skills if it applies to the union of the set of tasks to which the skills in the given set apply. For example `collect+` applies to exactly the union of all tasks to which `collect+pos` and `collect+neg` apply.
- *Compound* (labeled "C" in Figure 3). One skill set is a compound of two (or more) other skill sets if the skills in the compound are a union of the skills in the components. Note the subtle difference between generalization and compound. One skill generalizes another if the *tasks* to which it applies are the union of the problems to which the less general skills apply. Either (`collect+neg`) or (`collect+pos`) (but *not* both) will apply to a problem that is appropriate for (`collect+`). On the other hand, compounds represent unions of *skills*. For example (`collect+, eval+`) is a compound of the skill sets (`collect+`) and (`eval+`). An appropriate problem for this compound skill (e.g., $2x-3x=5-4$) will elicit *both* component skills.
- *Prerequisite* (labeled by "P" in Figure 3). The notion of a prerequisite is not as well-defined as specialization, generalization or compound. Roughly, one skill (set) is prerequisite to a given skill if the problems for the given skill are harder than that for the skill or set. For example, a problem like $2x+3+5x=6$ is more complex than $2x+5x=6$; consequently we say the skill collect+ is a prerequisite for collect+num. The prerequisite relationship is distinct from both generalization and specialization. Unlike generalization, the problems to which a prerequisite skill applies are not a subset of the more complex skill; a prerequisite skill applies to a different and simpler set of tasks. Nor is a prerequisite skill identical with a special-case skill. Prerequisite skills are easier to learn than more complex ones; special case skills are harder to learn than regular case ones.

Implementation of pedagogical rules and lesson updating

The pedagogical rules for the current task sequencing structure can each be thought of as functions that map current skill state information and student performance information onto a new skill set state. Two skill set constructs are referenced: (i) the most recent skill set used to generate a question (Sk), and (ii) the lesson stack (LS). The lesson stack stores skill sets, and permits the tutor to suspend one tutorial goal (i.e., a skill set), interpolate another goal (i.e., focus on a new skill set), then resume the original goal.

The following are rough English translations of the rules used in the current task sequencing structure.

(1) If Sk is null and the student has done no previous questions
    then Sk to be a skill set with no prerequisites.

(2) If Sk is null and the student has done some previous questions
then set Sk to be the skill set associated with the last question type.

(3) If the student's score on the last 3 questions totals 0
and they are all from the same skill set (Sk)
then push Sk on the lesson-stack and set Sk to be a simpler skill set.

(4) If the student's score on the last 6 questions totals < 350
and they are all from the same skill set (Sk)
then push Sk on the lesson-stack and set Sk to be a simpler skill set.

(5) If the student's score on the last 4 questions totals 370
and they are all from the same skill set (Sk) and the LS is not empty
then pop the LS and set it as Sk.

(6) If the student's score on the last 4 questions totals 370
and they are all from the same skill set (Sk) and the LS is empty
then select a more complex skill set.

The rules are no doubt too simple to yield intelligent task sequencing in all cases. However, they provide a reasonable starting point and are easily modified. Gathering data on their performance in the classroom will permit us to experiment with various rule sets and improve them in a principled way.

Rule (1) enables the tutor to "bootstrap." If it knows nothing about the student, it figures the student is a beginner and decides to tutor skills that assume no previous knowledge. Rule (2) is similar. It says that if you know nothing other than the last question done by the student, use the skills it exemplifies to generate a new question. Rules (3) and (4) tell the tutor when the student's performance is poor enough to merit going to simpler skills. Note that the current skill set is not abandoned. Rather, it is remembered on the lesson stack and, when the student has shown mastery of the simple skills, the tutor will return to the remembered set. Rule (5) permits skill sets to be restored from the lesson stack, while rule (6) selects new skill sets when the student has mastered the current ones.

The above English specification of the task sequencing rules leaves considerable ambiguity. In particular, notions like "a simpler skill" and "a more complex skill" are vague. The skill network structure permits us to explicate these terms more precisely. Simpler skills are those a tutor might want to revert to, should the given set prove too difficult for a student. Given our skill network, skill-set 1 can be simpler than skill-set 2 in four distinct ways: (i) skill-set 1 might be a prerequisite for skill-set2, (ii) skill-set 2 might be a compound skill that includes skill-set 1, (iii) skill-set 2 might be a specialization of skill-set 1, or (iv) skill-set 2 might be a generalization of skill-set 1. Similarly, more complex skill sets than a given set, skill-set 1, are simply all those skill sets that have skill-set 1 as a prerequisite, or which generalize, compound, or specialize skill-set 1. Generally, relationships among skill sets imply important pedagogical constraints, as did our decision to include only a few of the logically possible skill combinations in the skill set database. The relationships tell the tutor which skill sets are natural tutorial goals when a given skill set has been learned satisfactorily or poorly.

Because the different types of relationships among skill sets define four ways in which a skill set can be simpler or more complex than another, the skill set network, by itself, does not uniquely determine which simpler or more complex skill we should select when we must change the current skill set. Many strategies are possible and we should look to expert human tutors for answers to a variety of implied questions. For example: is it best to introduce special cases of a skill immediately after the regular skill has been taught? Or should the teacher quickly introduce problems that exercise the new skill compounded with other important skills? Similar questions arise when the tutor has to find a simpler skill set for the student.

In the absence of any justified choice, when trying to find a simpler skill set, the tutor first tries to find skill sets for which the current skill set is a compound. In other words, it prefers to back off to simpler skill sets that are components. If it cannot find a component skill set, it will pick a set that is simpler in any of the other three senses. In selecting a more complex skill set, it chooses from specializations, generalizations, compounds or prerequisite successors at random.

We regard the current implementation of simpler and more complex skill sets to be interim solutions, of no great importance by themselves. More important than any specific such implementation is the fact that we have defined a language in which many *different* implementations can be defined. Below, under "Future directions", we discuss in more detail how our approach to task sequencing can be viewed as a language in which several interesting problems about task sequencing can be posed, and a technology for implementing and testing answers.

Implementation of task generation

In the current task sequencing structure, task (i.e., question) generation is the final step in choosing a new task for the student. Prior to task generation, all measures of student performance for the previous question have been computed, and the above pedagogical rules have been executed to modify the current skill set and lesson stack. To select an appropriate question now becomes a matter of generating a question that, if solved correctly by the student, will elicit the skills in the current skill set.

Task generation in the algebra tutor has been relatively simple to achieve. We have defined a library of question types (qtypes). Each qtype has associated with it an abstract pattern and a list of skills. Below is an example of a qtype, represented in standard infix notation:

pattern: {{?integer * ?variable + ?integer1} = ?integer2 * ?variable}

skills: (attract+pos collect+ isolate*)

The pattern can be thought of as a template that instructs the tutor in putting together a specific question. All terms beginning with a "?" are pattern variables that the tutor will replace with specific items when creating a new problem. Pattern variables like "?variable" tell it to create a variable name by selecting at random from its known list of variable names (e.g., u, v, w, x, y). Multiple appearances of the same pattern variable must be replaced by the same term. Pattern variables of the form "?integer" mean the replacing term should be an integer, while "?+integer" means the replacing term should be a positive integer.

Adhering to these rules, the tutor might use the pattern above to produce the following problem:

{{2y+3} = 9y}

The brace constructs ("{}") add further generality to the ability of patterns to generate a diverse set of questions. The braces tell the tutor that the terms appearing inside can be in any order. Thus, when generating a specific question, the tutor randomizes the order of these terms. Assuming the given substitutions for the template shown above, the actual questions the tutor generates could be any of:

2y+3 = 9y

3+2y = 9y

9y = 2y+3

9y = 3+2y

In general, a pattern template defines a large class of questions that can be generated, each embodying a few important properties, but otherwise random in its appearance. The common properties of questions generated by a template insure that a select set of skills will be exercised in solving its problems. This set of skills is remembered as the skills list for each qtype. For example, the skills list for the pattern described above says that each question will exercise skills for attracting occurrences of a variable closer together, collecting like terms, and isolating the variable. For instance:

2y+3 = 9y	[given]
3 = –2y+9y	[attract+pos]
3 = 7y	[collect+]
3/7 = y	[isolate*]

Using qtypes with such patterns and skill lists it is relatively simple to generate a question from a given target skill set. The tutor simply takes the target skill set, matches it against the skills list of the qtypes in our library, and selects the best match. There may be several qtypes whose skills list contain the target skill set. In such cases, we pick the qtype whose skills list has the fewest additional, unmatched skills. Intuitively, we are selecting the question type that will exercise our chosen skills and the fewest additional ones.

Future directions

The implementation described above is a first attempt to provide a task sequencing structure for our intelligent tutor for basic algebra. It generates reasonable behavior, but is limited in several respects. Here we summarize some main shortcomings in the selected components of the task sequencing structure and suggest an agenda of future research topics.

The skill network

Our representation of mathematical skills is currently limited to one type. Students learning algebra ultimately need to understand the basic axioms of mathematics, local heuristics for how to use the axioms in problem solving, and higher-level meta-cognitive skills. At present, our intelligent algebra tutor supports the learning of such skills (e.g., we can present the student with "buggy" solutions and ask him or her to fix them). However, the task sequencing facility cannot yet *reason* about how to combine the learning of higher- and lower-level skills. We have currently limited ourselves to representing, reasoning about, and tutoring of just the local heuristics. Whether our view of task sequencing can extend to these important skills is a question we have yet to answer. Our approach is to begin to decompose the metacognitive skills involved in algebra, much as we decomposed the subject-matter skills, above. This task appears more challenging than the subject-matter decomposition because the structure of metacognitive skills is relatively poorly understood. Nevertheless, we are encouraged by our observations, discussed in the earlier section "The role of task sequencing in learning", that skills in a wide range of subjects appear amenable to a decomposition into related components.

Within the realm of local heuristic skills, several improvements can be envisioned. Our skill set network encodes which logically possible skill compounds should be candidates for teaching. Ultimately, we would like to remove such pedagogical expertise from these data structures and situate it in rules that can generate reasonable skill combinations. In effect, then, the skill network could disappear because it could be constructed as needed. This can be accomplished only if it is possible to articulate general heuristics that experts use in deciding which combinations might be pedagogically useful. If this ability is domain specific, then the skill set network will probably remain unchanged.

Student modeling

As we noted earlier, the current implementation lacks any sophisticated capability for computing student diagnostic inferences or for maintaining such inferences in a student model. To rectify this shortcoming we can draw on our work and that of other researchers in this area (e.g., Anderson, Boyle and Yost, 1985; Sleeman and Smith, 1981). However, we anticipate that traditional work on student modeling may be of limited value since the role we wish to ascribe to diagnosis is different than its traditional role. Typically, student modeling is used to compute local feedback for a student. Our choice to use it primarily for task sequencing may force us to investigate new kinds of diagnostic inferencing capabilities.

Although inferring students' misconceptions from their overt behavior is generally a difficult problem, situating it in the context of task sequencing may simplify the process by providing "top-down" constraints (Wenger, 1987) on the inferential process. For example, since our tutor now knows all the skills that a given question should elicit, when looking for ways to interpret a student's mistake, it can begin by looking to see if the student's performance would be predicted by any "buggy rules" (Brown and Burton, 1978) or "mal-rules" (Sleeman and Smith, 1981) that are variants of the skills that should have been used. In addition, the skills we have already explicitly represented are an ideal place to store the buggy skill variants; hence accessing plausible interpretations of students' visible errors should be an efficient process. Goldstein (1982) makes similar comments concerning the potential role of his genetic graphs in student modeling.

Task generation

Our current question generation module does not comprise a complete task generation facility in several respects. First, we are limited to the generation of equations for the students to solve. The different kinds of tasks that a good human tutor may use to help students are not yet considered in the current tutor. In the future, we may develop a more sophisticated model that uses a distinct set of pedagogical rules to decide what kind of task to employ and to determine the coaching policy for the task, once a skill set is determined.

Although we are considering this modification, it still may be too simple to account for the sophisticated skills of human tutors. Perhaps a tutor can *first* decide that he or she is going to present a certain type of task, and only then decide which skills to focus on. For example, the tutor could decide that in the next task the tutor and student will share problem solving in some way, and then decide that the task should exemplify some very complex skills, since the tutor will be helping the student. If such cases arise, we will have to revise our theory, since it says that computing the current skill set is independent of, and precedes, task generation.

Second, question generation, while flexible, may be inadequately controlled. Currently, the generator expects a list of skills as input, and computes a question that should elicit those skills. All properties of the question not determined by the given skill list are established at random by the generator. In particular, values for variables and constants cannot be controlled. However, such control may be desirable. To take one instance, we have observed that human tutors generate simpler questions that are maximally similar to complex ones that the student cannot answer. For example, if the student fails to answer $2x+4 = 8$, the tutor might back off to $2x = 8$ (McArthur and Stasz, 1987). The tutor is not only generating a question with a simpler skill set, but also constrains the variables and constants in the new question to be identical to those in the previous one.

Similarly, question generation must be expanded in the future to insure that questions will not elicit skills outside the current skill set that the student has not yet mastered. Currently, it just picks a question type that best matches the skill set. While this seems to work in practice, it means that the qtypes and matching hide an important pedagogical decision. When student model information is available, we will make the decision about which skills to exclude an explicit, reasoned process.

Pedagogical rules

The pedagogical rules we use to determine changes to the skill set are naive. A minor problem involves the use of numbers in the rule antecedents (e.g., "If the student's score on the last 3 questions totals 0"). It is probable that good human tutors rely less on such quantitative measures of student progress and on more qualitative assessments. A more important shortcoming is that the rules neither implicitly or explicitly encode a significant theory of how tasks should be sequenced. A rough paraphrase of the sequencing notions embedded in our current rules might be: (1) If the student is doing well, select skill sets of increasing difficulty; (2) if the student is doing poorly, interpolate tasks with simpler skills until he does better; otherwise (3) continue working the current skill set. While this view is probably not wrong, an interesting theory of task sequencing should contain much more.

Collins, Brown, and Newman (1987) and Schoenfeld (1983, 1985) suggest several additional ideas we may consider implementing. First, tasks should not only be of increasing complexity, but of increasing diversity. In other words, we should give tasks where the current skill set *does not* apply, after drilling students on tasks where it does apply, to help students understand the range of applicability of the skill. This idea could be implemented in our current framework by creating rules that first would focus on a single skill set, then alternate that skill set with "similar" or recently completed sets. Second, Collins *et al.* suggest that global skills should be taught before local skills. For example, Lave (cited in Collins, Brown and Newman, 1987) indicates that apprentice tailors first learn to create a whole suit out of precut parts, before learning how to fashion the parts themselves. This ordering enables the learner to establish an overall "cognitive map" for the local skills he or she is to learn. Implementing this idea in our current framework of pedagogical rules is challenging because, in our skills network, skills are all at the same level. A skill like `eval++` is no more or less global than `collect+multi`.

Although the present rules for task sequencing lack an interesting theoretical basis, they have an important value. In a sense, we are not providing a theory of how tasks should be sequenced as much as proposing a framework for representing different ideas about sequencing. Different rule sets can be easily constructed, encoding a variety of different theoretical approaches. Since little research has been done to describe the important features of intelligent sequencing, our framework can provide a useful way of empirically testing out alternate ideas about sequencing.

Acknowledgements

This research is being funded by the National Science Foundation (Applications of Advanced Technologies Program). Views or conclusions expressed herein do not necessarily represent the policies or opinions of the sponsor.

We would like to acknowledge the insightful comments of Andrea diSessa and two anonymous reviewers.

Notes

1. For purposes of this paper, we assume the information in knowledge sources is fixed. This is not strictly true, since tutors can acquire new teaching knowledge. However, such acquisition is independent of the issues dealt with in this paper.

2. In the sections describing the implementation of task sequencing in our algebra tutor, the terms "task", "problem" and "question" will be used synonymously, since the only kinds of tasks the tutor can now generate are algebra problems to be solved by the student.

3. Requests for reprints and other correspondence concerning this article should be addressed to the first author.

References

Anderson, J. R. (1982). Acquisition of cognitive skill. *Psychological Review*, *89*, 369–406.

Anderson, J., Boyle, D. F. and Yost G. (1985). The geometry tutor. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*.

Barr, A. M., Beard, M. and Atkinson, R. C. (1976). The computer as a tutorial laboratory: the Stanford BIP project. *International Journal of Man-Machines Studies*, *8*, 567–596.

Bell, A. W., Costello, J. and Kuchemann, D. E. (1983). *A review of research in mathematical education (Part A)*. Berks., U.K.: NFER-Nelson.

Block, J. and Burns R. (1976). Mastery learning. *Review of Research in Education, 4*, 3–49.

Brown, J. S. and Burton R. R. (1978). Diagnostic models for procedural bugs in basic mathematical skills. *Cognitive Science, 2*, 155–192.

Brown, J. S., Collins A. A. and Harris, G. (1978). Artificial intelligence and learning strategies. In H. O'Neil (Ed.), *Learning strategies*. New York: Academic Press.

Burton, R. R. and Brown, J. S. (1982). An investigation of computer coaching. In D. H. Sleeman and J. S. Brown (Eds.), *Intelligent tutoring systems* (pp. 79–98). New York: Academic Press.

Chambers, J. A. and Sprecher J. W. (1980). Computer-assisted instruction: current trends and critical issues. *Communications of the ACM, 23*, 332–342.

Clancey, W. J. (1979). Tutoring rules for guiding a case method dialogue. *International Journal of Man-Machine Studies, 11*, 25–49.

Clancey, W. J. (1983). "GUIDON". *Journal of Computer-Based Instruction, 10*, (1 and 2), 8–15.

Collins, A. and Brown J. S. (1986). The computer as a tool for learning through reflection. In H. Mandl and A. Lesgold (Eds.), *Learning issues for intelligent tutoring systems*. New York: Springer-Verlag.

Collins, A., Brown, J. S. and Newman S. E. (1987). Cognitive apprenticeship: teaching the craft of reading, writing, and mathematics. In L. B. Resnick (Ed.), *Cognition and instruction: issues and agendas*. Hillsdale, N. J.: Lawrence Erlbaum Associates.

Collins, A. and Stevens, A. (1981a). Goals and strategies of effective teachers. In R. Glaser (Ed.), *Advances in instructional psychology* (Vol 2). Hillsdale, N. J.: Lawrence Erlbaum Associates.

Collins, A. and Stevens, A. (1981b). A cognitive theory of interactive teaching. In C. M. Reigeluth (Ed.), *Instructional design theories and models: an overview*. New York: Academic Press.

Gagné, R. M. and Briggs, L. J. (1974). *Principles of instructional design*. New York: Holt, Rinehart and Winston.

Goldstein, I. (1982). The genetic graph: a representation for the evolution of procedural knowledge. In D. H. Sleeman and J. S. Brown (Eds.), *Intelligent tutoring systems*. New York: Academic Press.

Lawler, R. and Yazdani, M. (Eds.) (1987). *Artificial intelligence and education: learning environments and intelligent tutoring systems*. Norwood, N. J.: Ablex.

Matz, M. (1982). Towards a process model for high school algebra errors. In D. H. Sleeman and J. S. Brown (Eds.), *Intelligent tutoring systems*. New York: Academic Press.

McArthur, D., Stasz, C. and Hotta, J. (1987). Learning problem-solving skills in algebra. *The Journal of Educational Technology Systems, 15*(3), 303–324.

McArthur, D. and Stasz, C. (1987). Tutoring techniques in algebra. Paper presented at the American Education Research Association national conference, Washington DC, April.

Mitre Corporation (1976). An overview of the TICCIT program. *Report M76–44*, Washington: Mitre Corporation.

Newell, A. and Simon, H. A. (1972). *Human problem solving*. Engelwood Cliffs, N. J.: Prentice-Hall.

Ohlsson, S. (1986). Some principles of intelligent tutoring. *Instructional Science, 14*, 293–326.

O'Shea, T. and Self, J. (1983). *Learning and teaching with computers*. New York: Prentice-Hall.

Palmer, B. G. and Oldehoeft, A. E. (1975). The design of an instructional system based on problem-generators. *International Journal of Man-Machine Studies, 7*, 249–271.

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

Schoenfeld, A. H. (1983). Problem solving in the mathematics curriculum: a report, recommendations and an annotated bibliography. *The Mathematical Association of America Notes*, No. 1.

Schank, R. and Abelson, R. (1977) *Scripts, plans, goals, and understanding*. Hillsdale, N. J.: Lawrence Erlbaum Associates.

Schoenfeld, A. H. (1985). *Mathematical problem solving*. New York: Academic Press.

Shute, V. and Glaser, R. (1986). An intelligent tutoring system for exploring principles of economics. *Technical Report:* Learning Research and Development Center, University of Pittsburgh, Pennsylvania.

Sleeman, D. H. and Smith, M. J. (1981). Modeling student's problem solving. *Artificial Intelligence, 16*, 171–188.

Smith, R. (1986). The alternate reality kit: an animated environment for creating simulations. *Proceedings of the 1986 IEEE Computer Society Workshop on Visual Languages*, 99–106.

Stallings, J. A. and Stipek, D. (1986). Research on early childhood and elementary school teaching programs. In M. Wittrock (Ed.), *Handbook of research on teaching* (3rd edition). New York: MacMillan.

VanLehn, K. (1983). *Felicity conditions for human skill acquisition: validating an AI-based theory.* Doctoral dissertation, MIT, Cambridge, MA.

VanLehn, K. (1987). Learning one subprocedure per lesson. *Artificial Intelligence, 31*, 1–40.

Wenger, E. (1987). *Artificial intelligence and tutoring systems*. Los Altos, CA: Morgan and Kaufmann.

Wescourt, K., Beard, M. and Gould, L. (1977). Knowledge-based adaptive curriculum sequencing for CAI: application of a network representation. *Proceedings of ACM, 77*, 234–240.

Woolf, B. and McDonald, D. (1984). Building a computer-tutor: design issues. *IEEE Computers*, September, 61–73.