Jl. of Artificial Intelligence in Education (1993) 4(1), 95–121

Instructional Design and Intelligent Tutoring: Theory and the Precision of Design

PETER CAPELL[1] AND ROGER B. DANNENBERG[2]
[1]*Software Engineering Institute,* [2]*School of Computer Science*
Carnegie Mellon University, Pittsburgh, PA 15213, USA

Instructional Design aspires to define a sound curriculum by using instructional analysis and concept organization. Along with other criteria, the purpose of instructional design is to ensure integrity among instructional objectives, tasks that students must perform, and the evaluation of their performance. Currently, the methods used in instructional design models have a limited scientific basis. Even with many efforts towards a science of instruction, this goal remains elusive. Computers may provide a positive shift towards systematic and verifiable instructional analysis with the advent of intelligent tutoring systems and the byproducts of their development. One such system, the Piano Tutor, has led to a formal model for curriculum design and analysis and is described in detail.

Instructional Systems Design (ISD) consists of methods, procedures, and theories that are used in developing well-structured curricula. Instructional systems theory has been developing since just after WWII and has been influenced by many schools of thought including systems theory, behavioral and cognitive psychology, and information theory.(Seels, 1989; Richey, 1986) The basic principle of the ISD approach is that all concepts of a curriculum should be defined behaviorally so that what a student is taught is made observable and measurable by performance-based learning criteria.

With the computer revolution, new possibilities began to open up for education. In the late sixties, ambitious new efforts attempted to capitalize on the power of computing. Although the initial attempts were fitful, one

byproduct of that era has been new perspectives on instruction and learning. Computers have added to the exploration of *systematic* educational methods.

Perhaps the biggest advance in these systems is the ability to examine a student's performance on an individual basis to compare it with a model of an ideal performance, and to provide incisive hints and responses in a learner-specific manner. We are now able to address the problems of individual learning capabilities and styles for large numbers of people in all stages of their lives. Computer-based instruction has added extra impetus to the reshaping of Education from its being a field dependent mostly on attitude and belief to being a discipline composed of the systematic study of learning and of the methods of analyzing instructional content.

For example, in order for a computer to mimic the diagnoses that human tutors employ, the knowledge that teachers use must be made explicit in great detail. This byproduct of human-computer interaction may help in moving us beyond school systems that simply cull exceptional students from poorer ones depending on students' capacity to adapt. We are now at the crossroads of being able to provide very specialized instruction to large groups of students.

Basic improvements in the power and cost of computing have yielded a new generation of instructional computing systems. Thanks to these developments and to the creation of advanced instructional systems, we can now analyze student performance that utilizes a wide range of input devices including the standard computer keyboard, a piano or other musical instrument, a mouse, a joystick, or a light pen, and we can provide instructional advice about the many kinds of learning and environments that can be engendered through these devices. We can now teach subjects as diverse as foreign languages, circuit analysis, welding, music, computer programming, algebra, and geometry using instructional computing systems.

This paper takes the perspective that intelligent tutoring systems provide us with an opportunity to examine the nature of instructional design and its future as this future is affected by these systems. The paper is divided into three broad areas:

1. the need for tools of scientific investigation in education,
2. intelligent tutoring as a platform for more tightly controlled educational experimentation, and
3. an example of a formal method for computerized curriculum analysis.

Our ultimate purpose is to stimulate further efforts and discussion and to extend these basic ideas toward sound and testable theories of instructional analysis and design.

THE CONTEXT OF INSTRUCTIONAL DESIGN

A survey by the first author (Capell, 1989) examined the instructional design methods used in intelligent tutoring systems (ITS). The study sought to answer some basic questions about how these systems were designed and about the consistency of their designs with what the literature in instructional design indicates as *sound*. The rationale for the study was that if the design and implementation of an intelligent tutoring system requires exact specification—of the goals of the system, of the system's interaction with students, of how knowledge must be structured in assessing the student, and of how to provide meaningful remediation—then certainly an examination of the design process of these systems would provide discoveries of interest to instructional designers. In the survey, there *were* many interesting aspects to the systems and their designs, most of which tended to ignore any prescribed method.

Two of the systems, Bridge (Bonar,1985) and the Lisp Tutor (Anderson & Reiser, 1985), have been used with students and have been successful as teaching systems—even though both were created without any particular method of instructional design. Part of their success is attributable to at least two basic features: First, any computerized teaching system must use *behavior* as its means of evaluating and remediating student performance (machines tend not to be swayed by their beliefs about students), and second, the systems' developers are bright people who understand successful teaching methods—and the systems are their brain children. It can therefore be easily argued that in essence these systems do follow established instructional design principles even if the application of these principles is mostly unwitting.

Irrespective of these systems' success "without design," the problem before educational technologists is the same with regard to these machines as it is with regard to the classroom; we must have methods that ensure quality of instruction for large numbers of students. With respect to machine instruction, it is much more likely that more and better systems will be created if their designers are able to systematize their procedures.

The holy grail of instructional design is to create *instructional integrity*—the idea that instruction can be planned, its effects measured, and its outcomes predicted. This idea was expressed clearly by Briggs (1977) who said that the purpose underlying all instructional design was to create "congruence among objectives, teaching strategies and performance evaluation." Several factors contribute to instructional integrity:

1. Students should be tested only on material that they have been taught.
2. The objectives of the curriculum should be clearly articulated in behavioral terms, behavior being the best indicator of the state of a student's understanding. This is the fundamental premise of behaviorism in instruction—that we may gauge a student's progress only by virtue of behaviors indicating that the student has *understood* an idea. For example, a math student demonstrates his or her understanding of the concept of *division* by doing some number division problems correctly.
3. The curriculum should describe in unambiguous terms what the student has to do in order to demonstrate an understanding of what has been taught. The designer must carefully consider the audience for the instruction. This can become a matter of intense concern with preinstructional evaluation and therefore with the selection of strategies and material to address students' individual needs. Of course, it is difficult to formalize a process with so many unknowns and unpredictable parts.

Whether one sees these developments as contributing more to Instructional Design's becoming a science or more to its becoming an effective art is irrelevant. It is clear to both educational theorists and practitioners that whatever leverage we gain through an understanding of the methods needed to improve design and instruction comes none too soon in light of the problems in modern education.

INSTRUCTIONAL SYSTEMS DESIGN AND THE PIANO TUTOR

The intelligent tutoring system from which this research was developed is the Piano Tutor (Dannenberg, Sanchez, A. Joseph, Capell, et al., 1990; Dannenberg, Sanchez, A. Joseph, Saul, et al., 1990). The Piano Tutor is a multimedia workstation that is designed to teach first-year piano playing. The system employs realtime score-following technology, expert systems, videodisc, and other media. It is an unusual system for at least two reasons: There has been a concerted effort to use instructional design techniques as the basis of the system, and the domain of instruction is psychomotor. Both of these aspects are uncommon in intelligent tutoring systems.

Lessons are treated as individual modules in the system, and each lesson has associated with it prerequisites, objectives, a presentation, and an evaluation. The prerequisites of a lesson are skills, or discrete pieces of knowledge, that we expect students to master. Skills also form the objective of a lesson. It is important to note that the designers of the system rede-

fined the notion of a *lesson* to mean something slightly different from what we might think. A lesson in the Piano Tutor is a single interaction that attempts to teach very few ideas. Each lesson addresses one or two concepts, presents the student with a task, and then evaluates the student's performance. Finally, the lesson attaches a score to each skill named in the lesson's *objective* slot.

For example, there is a lesson called "Teach 3/4 time" whose purpose is to teach how to play a score with a $^3/_4$ time signature. The skill name, "SK-3/4," belongs to the skill forming this lesson's objective. The lesson also has a list of prerequisite skills such as SK-2/4, SK-NOTE-NAMES, and so on. When the student has completed the task associated with the lesson, the SK-3/4 skill is updated to a value indicating whether or not the student has learned the skill. While the strategy is simple, it is a direct instantiation of instructional design principles.

The Piano Tutor performs various diagnoses of student errors in the context of the lesson and updates the skills accordingly. This action in turn affects the way lessons are selected for the student. A lesson cannot be selected unless the student has mastered the prerequisite skills for that lesson. This is another way in which the basics of instructional design manifest themselves in the Tutor. From this point, having in hand a means of determining the kinds of errors the student is making and the skills affected by those errors, we can influence the ways in which the student receives new instruction so that, in theory, no new instruction will be beyond the student's capabilities.

There is of course a problem. There are many lessons that teach many skills. The system is not constrained to choose the lessons in any fixed order; they simply pop up according to the diagnoses occurring in the context of each lesson and in relation to the skills affected by the diagnoses. This lack of constraint makes it difficult to determine in advance how lessons will interact together, especially when one considers the number of paths that can occur through a total of over 70 lessons. However, computer-based analysis does allow us to model numerous paths through the curriculum to explore the effects of different assumptions about how skills and lessons are organized.

Formal Analysis

One great advantage of using lessons and skills to describe a curriculum is that the representation is amenable to extensive analysis by comput-

er. While the information that can be derived by computer analysis will not tell us whether a curriculum is *good* or *correct*, analysis *can* help, with reference to basic principles of instructional design, to locate points where the curriculum is inconsistent, incomplete, or perhaps incorrectly specified. By locating these deficiencies early in the design process, problems can be corrected before time and money are wasted on realizing a faulty curriculum through textbooks, videodisc, computer software, or other media.

Analysis using formal techniques can also demonstrate global properties of a curriculum that cannot be shown by ordinary testing. For example, analysis might prove that in order for a student to learn a particular skill S, he or she *must* first take lesson L even if L does not directly teach S. With ordinary testing, we might observe that all students who master skill S have previously taken lesson L, but this does not prove that this *must* be the case. In a large and complex curriculum, properties such as this can be both nonobvious and very useful. For example, in this case we know that if a student cannot pass lesson L, he will never learn skill S.

Previous Work

The analysis techniques we have developed are either standard graph algorithms (Aho, 1974) or related to algorithms used by optimizing compilers for code analysis (Wulf, Johnsson, Weinstock, Hobbs, & Geschke, 1975). Thus, the technical aspects of the analysis are relatively standard, but we believe that our application of these techniques to the formal analysis of curriculum designs is novel and unique.

A number of researchers have investigated the general topic of curriculum design and lesson selection for individually tailored instruction (see Halff [1988] for an overview of curriculum issues). Barr, Beard, and Atkinson (1976) describe the Stanford BIP system. The representation used by BIP included lessons and objectives but no prerequisites. Tasks were ordered by skill groups called *techniques* that were taught in strict linear order. In a subsequent paper, Wescourt, Beard, and Gould (1977) describe enhancements resulting in the BIP-II system. Here, prerequisite relations were added between skills but not in order to relate skills to lessons as in the Piano Tutor.

The *skill hierarchy* concept implied in BIP-II is found in other systems as well. Derry, Hawkes, and Ziegler (1988) describe a system for teaching arithmetic word problems that is based on a hierarchy of skills. Tutoring plans are generated and modified as students progress by attaining skills.

McCalla, Peachey, and Ward (1982) use an and/or graph of skills for curriculum planning. McArthur, Stasz, Hotta, Peter, and Burdorf (1978) discuss the generation of appropriate lessons from a skill hierarchy representation.

Wipond and Jones (1988) focus on curriculum *design*. Their system is based on a hierarchical refinement of courses into topics, modules, and submodules of instruction. An expert system monitors the curriculum design process and detects design errors. This is the only reference we have found outside of our own work in which a curriculum design is automatically checked for consistency. The focus of this system seems to be on the refinement process whereas the Piano Tutor's curriculum representation distinguishes between skills and tasks and attempts to verify consistency between the two.

The concept that skills and tasks are distinct is seen in Peachey and McCalla (1986) where operators with prerequisites and objectives, corresponding to *lessons* in the Piano Tutor, are introduced. Peachey and McCalla use these operators to generate course graphs but do not attempt to analyze graphs as in the Piano Tutor.

CURRICULUM ANALYSIS

All of the following analysis procedures assume that a curriculum design consists of *lessons* and *skills*. Each lesson initially has two properties or attributes: *prerequisites*, a set of skills that must be mastered before the lesson can be taken, and *objectives*, a set of skills that are assumed to be mastered after the lesson has been taken. A distinguished lesson named *Ultima* has as prerequisites all skills that the student should have upon completion of the curriculum. The lesson Ultima has no objective skills, and its purpose in our formal system is merely to designate a particular set of skills via the prerequisites set. The analysis procedures compute new and useful properties for lessons and skills. Since some of the analysis procedures build upon the work of earlier ones, it is assumed that the procedures are performed in the order they are presented.

Our purpose in presenting analysis procedures is to show that analysis can be done in a straightforward, deterministic manner. Rather than using a conventional programming language, we have attempted to use a stylized English notation. We feel that this notation will be at least somewhat readable by the nonprogrammer who wants to get a feel for the computational steps that are involved. At the same time, the procedures are complete, and we believe unambiguous to someone skilled in the art of programming.

Thus, they can serve as a basis for developing software (an implementation in the programming language Common Lisp is available from the second author). The reader may wish to skip the algorithms altogether since the processing steps are described in prose as well.

Notation and Terminology

A few words about our notation are in order. Lessons and skills have *attributes*. For example, "taught-by of *s*" refers to the *taught-by* attribute of skill *s*. Attributes are often sets of lessons or skills. A common operation is to insert a new element into an existing set. We denote this operation by a sentence such as "Add L to taught-by of S''. This tells us to take the set of lessons that comprise the taught-by attribute of S and insert the lesson L into the set. After the sentence is performed, L will be an element of the taught-by attribute of S. Finally, comments are introduced by "—" and are written in italics.

In curriculum analysis, we expect that the student is capable of taking, mastering, and passing all lessons, assuming the prerequisites are met. Therefore, when our explanations refer to a student "taking a lesson," it should be assumed that the lesson is mastered and all of the objectives are met. The effect of a student's not mastering lesson objectives can be determined by modifying the curriculum. Either the lesson can be removed from the curriculum or some of the objectives can be removed from the lesson, and then the analysis procedures performed on the modified curriculum.

Computing "Taught-By"

One useful attribute for skills is taught-by. This attribute lists the lessons that have a particular skill in their set of objectives. This attribute makes it easy, for example, to find all skills that can only be taught by one lesson. Furthermore, skills that are not taught by *any* lessons indicate a problem with the curriculum.

The procedure for computing taught-by is

```
—first, initialize the taught-by attributes:
        for each skill S in Skills
            set taught-by of S to the empty set

—now, build up taught-by attributes:
        for each L in Lessons
```

```
let Objs be the objectives of L
for each Obj in Objs
    add L to the set taught-by of Obj
```

This algorithm simply enumerates all objectives, putting the lesson that teaches the objective in the objective's taught-by set (see Figure 1).

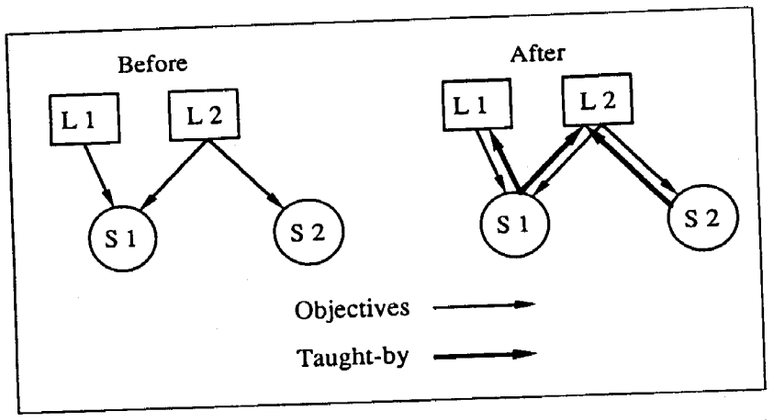

Figure 1. Computing taught-by: The taught-by attribute of skills is the inverse of the objectives attribute of lessons

Computing "Used-By"

The used-by attribute lists, for a given skill, the lessons that have the skill as a prerequisite. A skill that is not used by any lessons is suspect. This attribute also simplifies some of the following analysis procedures.

The procedure for computing used-by is very similar to the previous procedure.

```
—first, initialize the used-by attributes:
        for each skill S in Skills
            set used-by of S to the empty set

—now, build up used-by attributes:
        for each L in Lessons
            let Prereqs be the prerequisites of L
            for each Pre in Prereqs
                add L to the set used-by of Pre
```

Figure 2 illustrates that the used-by attribute is simply the inverse of prerequisites.

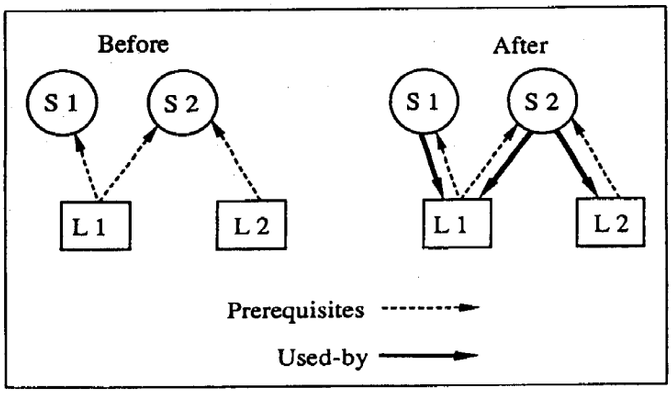

Figure 2. Computing used-by: The used-by attribute of skills is the inverse of the prerequisite attribute of lessons

Computing "Useless" Skills and Lessons

If a skill is not a prerequisite of some lesson, then it is useless in the sense that mastering the skill will not help the student advance in the curriculum. The skill is useless in the context of the curriculum because the skill does not enable the student to take any lessons that could not be taken otherwise. Similarly, a lesson is useless if it teaches only useless skills. Useless skills and lessons are not normally present in a curriculum, so their presence is an indication of a design error.

The procedure for their computation is straightforward:

```
set Useless-Skills to the empty set
set Useless-Lessons to the empty set

for each S in Skills
        if used-by of S is empty
        then add S to Useless-Skills

for each L in Lessons
        let Useful be false—until proven otherwise...
        for each Obj in the objectives of L
```

```
        if Obj is not in Useless-Skills
        then set Useful to true
    if Useful is—still—false
    then add L to Useless-Lessons
```

In Figure 3, S1 is a useless skill because it is not used by any lesson. L1 is a useless lesson because it teaches no useful skills.

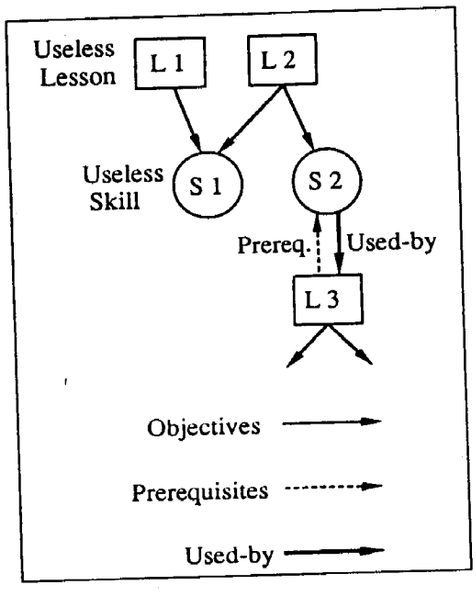

Figure 3. Computing useless skills and lessons: A useless skill such as S1 is not used by any lesson; a useless lesson such as L1 teaches only useless skills

Computing "Patently Unobtainable Skills"

If a skill is not the objective of any lesson, then we say that the skill is *patently unobtainable.* In other words, patently unobtainable skills are not taught by any lesson. The presence of a patently unobtainable skill usually indicates that a lesson has been omitted from the curriculum, or perhaps there is some confusion over the skills that are assumed and the skills that are to be taught.

The procedure to find patently unobtainable skills simply enumerates the skills and tests to see that the skill is taught-by some lesson:

```
set Patently-Unobtainable to the empty set

for each S in Skills
        if taught-by of S is empty
        then add S to Patently-Unobtainable
```

In Figure 4, S1 is not taught by a lesson, so it is patently unobtainable.

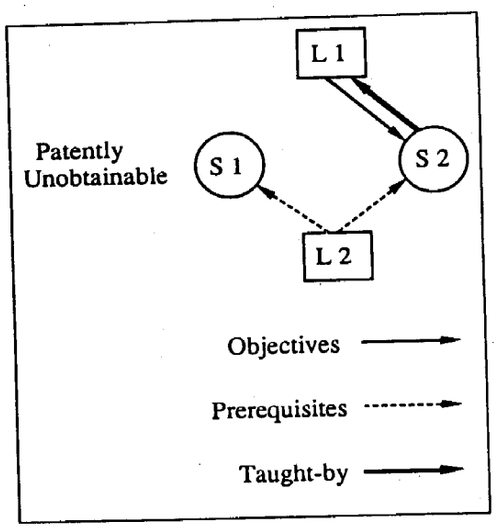

Figure 4. Computing patently unobtainable skills: Patently unobtainable skills such as S1 are not taught by any lesson

Computing Lessons With No Prerequisites

A lesson with no prerequisites can be taken by the student at any time. These are usually introductory lessons or lessons that serve to assess the student's current skill level.

```
set No-Prerequisites to the empty set

for each L in Lessons
```

```
If prerequisites of L is empty
then add L to No-Prerequisites
```

In Figure 5, L1 and L3 have no prerequisites.

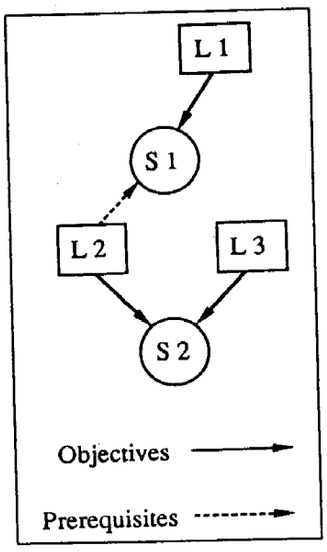

Figure 5. Computing lessons with no prerequisites: L1 and L3 have no prerequisite skills

Computing "Implies"

Depending upon how the curriculum is constructed, the possession of a skill might imply that the student has mastered other skills as well. Studying the implication of skills can give an instructional designer insight into the structure and interrelationships of a curriculum design. For example, if a relatively basic skill implies a relatively advanced skill, then there is probably a mistake somewhere in the curriculum specification.

The implies attribute is computed for both skills and lessons. For a lesson, the implies attribute tells what skills the student will have when the lesson is complete. These will of course include both prerequisites and objectives of the lesson, but there may be other skills that are not specified by

the designer but that are implied by the structure of the curriculum. In the case of lessons, it is useful to look at implied skills that are neither prerequisites nor objectives. If these are not in any way relevant to the lesson, then this analysis indicates a problem with the curriculum.

The procedure to compute the implies attribute is based on the following relations between skills and lessons:

1. A lesson implies the union of its prerequisites and objectives. In other words, a student who completes a lesson will know the prerequisites and objectives. In Figure 6, both L1 and L2 imply S1, S2, and S3 because S1 is a prerequisite of each lesson, and S2 and S3 are objectives of each lesson.
2. If all lessons that teach a skill S imply skill X, then S implies X. In other words, if a student knows skill S, then the student must also know X because any lesson that teaches S implies X. For example, in Figure 6, to acquire S2, the student must have taken either L1 or L2, both of which teach S3. No matter how S2 is acquired, S3 will also be acquired, so we say S2 implies S3. Furthermore, to acquire S2, the student must also have S1, a prerequisite to both lessons that teach S2. Therefore, S2 also implies S1.
3. A lesson implies the union of all skills implied by its prerequisites. Because the student must know each prerequisite skill in order to take a lesson, the student will also know skills implied by prerequisites. For example, in Figure 6, L3 has S3 as a prerequisite, but S3 implies S1 and S2. Therefore, L3 implies S1, S2, and S3.

The algorithm starts by setting the implies attribute of each lesson and skill to the empty set. Next, each attribute is recomputed according to the principles described above. The entire computation is iterated until there is no change in any implies attribute.

This computation eventually must terminate because the set of skills in any implies attribute either stays the same or grows on each iteration (skill sets are monotonically increasing). Because the total number of skills is finite, there is an upper bound to how large any implies attribute can grow. The procedure is as follows:

```
—initialize implies attributes:
for each S in Skills
        —a skill implies itself:
        set implies of S to {S}
for each L in Lessons
        set implies of L to the empty set
```

plies of L
of skills implied by prerequisites:
the empty set
uisites of L
lies to the union of Prereq-Implies
ee of P
s is union of skills implied by prerequisites:
e set-union of Prereq-Implies and

ny change:
the same as implies of L
g to true
or each skill:

nplies of S
lls implied by all lessons that teach S:

-by of S
the set-intersection of implies of L
any change:
t the same as implies of S
lag to true
ange

atently unobtainable skill will imply all rmal logic but somewhat counterintuitive. l information to the instructional designer, avoid this computation altogether until the obtainable skills. Notice also that all skills consequence of formal considerations, but s obvious information when we output the read.

her, then they are considered to be equiva- hangeable in the curriculum design and are ted together. Assuming that the curriculum yms on purpose, the presence of equivalent

```
—iterate until there is no change:
loop
        set Change-Flag to false
        for each L in Lessons
            set Old-Implies to implies of L
            —compute the union of skills implied by prerequisites:
            set Prereq-Implies to the empty set
            for each P in prerequisites of L
                set Prereq-Implies to the union of Prereq-Implies
                and prerequisites of P
            —now Prereq-Implies is union of skills implied by prerequisites:
            set Implies of L to the set-union of Prereq-Implies and
            objectives of L
            —see if there was any change:
            if Old-Implies is not the same as Implies of L
            then set Change-Flag to true
        —now compute implies for each skill:
        for each S in Skills
            set Old-Implies to implies of S
            —find the set of skills implied by all lessons that teach S:
            set Imp-By to Skills
            for each L in taught-by of S
                set Imp-By to the set-intersection of Implies of L
                and Imp-By
            —see if there was any change:
            if Old-Implies is not the same as Implies of S
            then set Change-Flag to true
—repeat the loop if there was a change
until Change-Flag is false
```

As formulated here, any patently unobtainable skill will imply all skills. This is consistent with formal logic but somewhat counterintuitive. In the interest of providing useful information to the instructional designer, the best approach is probably to avoid this computation altogether until the curriculum is free of patently unobtainable skills. Notice also that all skills imply themselves. This is also a consequence of formal considerations, but we find it helpful to suppress this obvious information when we output the implies attributes for humans to read.

Computing "Equivalent" Skills

If two skills imply one another, then they are considered to be equivalent. Equivalent skills are interchangeable in the curriculum design and are redundant whenever they are listed together. Assuming that the curriculum designer has not invented synonyms on purpose, the presence of equivalent

skills indicates that the designer intended to make a distinction, but none was made. Once it is pointed out that two skills are equivalent, the designer can either remove one of them or elaborate the curriculum to make the distinction a real one. For example, a lesson might be added that teaches one skill but not the other. Figure 7 shows a case of two equivalent skills S1 and S2. On the right, the curriculum has been simplified by merging S1 and S2 to form a single skill, S1/S2.

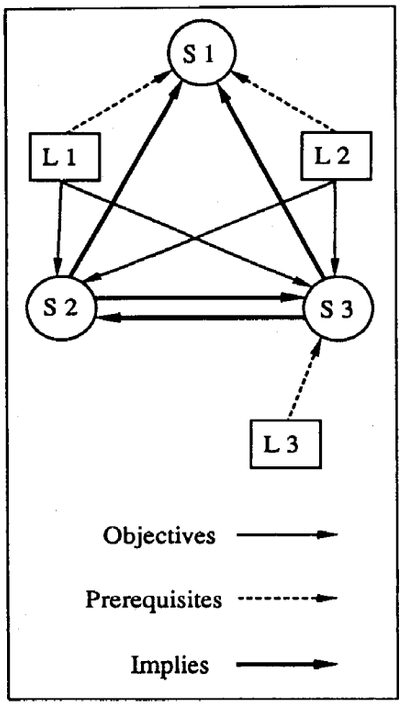

Figure 6. Computing implies: Only the implies attribute of skills is shown here; the fact that each skill implies itself is not shown

The procedure looks at each implied skill to see if there is an equivalence:

```
—initialize the equivalent attributes:
for each S in Skills
```

```
        set equivalent of S to the empty set
for each S in Skills
        for each I in implies of S
            —S implies I, does I imply S?
            if S is in implies of I
            then add I to equivalent of S
```

A skill is always equivalent to itself. As with the implies attribute, it is useful to suppress this obvious information from output intended for the instructional designer.

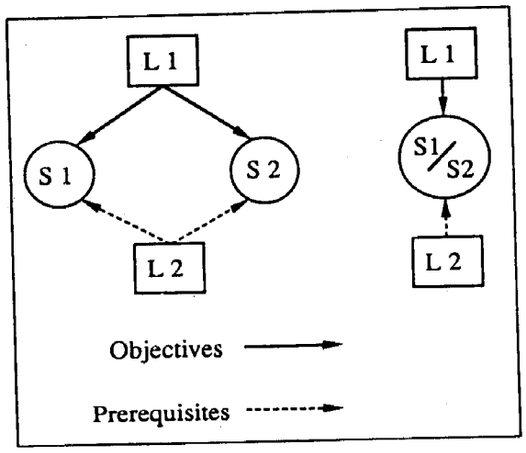

Figure 7. Computing equivalent skills: S1 and S2 imply one another, so they are equivalent; they could be replaced by a combined skill S1/S2

Computing the Shortest Curriculum

Traditional curricula are designed to be linear: It was assumed that students take all lessons in a fixed order. With interactive computer-based teaching systems, it makes sense to offer lessons in any order that is allowed by the prerequisites. This enables students to pursue their own interests and to skip past lessons that offer alternative approaches to the same material.

In a flexible scheme like this, it is impossible for the designer to envision all of the possible paths through the curriculum that a student might take. An interesting question to ask is: Of all possible paths through the

curriculum, which is the shortest one? The shortest path will avoid the greatest number of lessons. The lesson designer can then check to see if the resulting path is acceptable. If not, then perhaps there are skills associated with the untaken lessons that should be introduced into the curriculum and required by the final lesson (Ultima). Or perhaps some prerequisites were omitted from the design, allowing the student to take advanced lessons prematurely.

The procedure that we use computes a short path but not necessarily the shortest path through the curriculum. In the first stage of the procedure, we simulate a student progressing through the curriculum. First, all lessons whose prerequisites are met are labeled as Level 1. We simulate the taking of all Level 1 lessons by marking the objectives of these lessons as *learned*. We also record which lesson taught the skill. We then find all the newly enabled lessons (those whose prerequisites are now met) and mark them as Level 2. We simulate the taking of these lessons and continue in this fashion until there are no more lessons that are enabled but not taken. Figure 8 illustrates a curriculum arranged by levels.

In the second stage of the procedure, we work backwards from Ultima. If Ultima is at Level N, we look for lessons at Level N-1 that were used to teach prerequisites of Ultima. These lessons are added to the path being computed. We then look for lessons at Level N-2 that were used to teach prerequisites of Ultima or any other lesson on the path being computed. This process continues until Level 1 is reached at which point we have identified a short path from Level 1 to Ultima. In Figure 8, this path is Ultima, S4, L3, S1, S2, L1, L2. Notice that the longer path to Ultima via L5 is avoided: The path contains L2 which is not strictly necessary.

```
—first lessons are those with no prerequisites:
set Enabled to No-Prerequisites
for each L in lessons
        —use -1 to indicate that no level has been assigned
        set level of L to -1
for each S in Skills
        set level of S to -1
set Current-Level to 1

—take enabled lessons until none are left:
while Enabled is not empty
        set Update-List to the empty set
        for each L in ~~Lessons~~ Enabled
            if level of L equals -1
            then set level of L to Current-Level
                for each S in objectives of L
                    add the pair [L, S] to Update-List
```

```
            for each pair [L, S] in Update-List
                if level of S equals -1
                then set level of S to Current-Level
                set teacher of S to L
            set Current-Level to Current-Level + 1
            —figure out new set of lessons:
            set Enabled to the empty set
            for each L in Lessons
                —only examine untaken lessons:
                if level of L equals -1
                then set Enab to true
                    —see if every precondition is satisfied:
                    for each P in prerequisites of L
                        if level of P equals -1
                        —the precondition is not met
                        then set Enab to false
                    if Enab
                    then add L to Enabled
—the while-loop continues as long as new lessons are enabled
—see if the simulated student learned Ultima:
if level of Ultima equals -1
then print "Failed to find a path to Ultima"
    stop

—search for a short path from Ultima back to beginning
set Current-Level to (level of Ultima) - 1
set Short-Path to [Ultima]
set Need-To-Learn to prerequisites of Ultima
while Current-Level is greater than zero
            for each S in Need-To-Learn
                —see if needed skill has a teacher at the current level:
                if level of teacher of S equals Current-Level
                and teacher of S is not in Short-Path
                then append teacher of S to Short-Path
            —now compute the skills we need to know:
            set Need-To-Learn to the empty set
            for each L in Short-Path
                set Need-To-Learn to the set-union of Need-To-Learn and
                prerequisites of L
            set Current-Level to Current-Level - 1
—repeat until the while loop finishes
—Short-Path is now a valid path to Ultima
```

Computing Critical Lessons

A *critical lesson* is one that must be taken in order to reach Ultima. If a student cannot master a critical lesson, she cannot master the curriculum because there will be certain essential skills that cannot be taught by any

other lesson. In Figure 9, L1 is a critical lesson because it must be taken to reach Ultima. L2 is not a critical lesson because L3 also teaches S5. The instructional designer might use the set of critical lessons in different ways:

1. Critical lessons might be scrutinized to make sure they are clearly presented and likely to be successful at teaching their objectives.
2. Critical lessons might be tested carefully with human subjects since these lessons are critical to the curriculum.
3. New lessons might be added to provide alternative instructional paths, eliminating critical lessons.

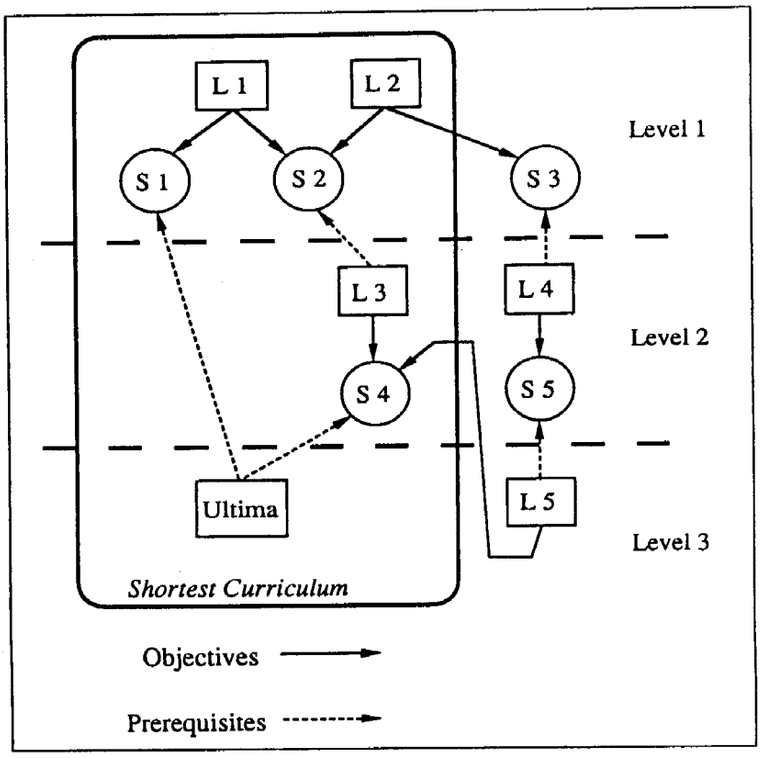

Figure 8. Computing the shortest curriculum: The curriculum is traversed one level at a time until Ultima is reached; then the curriculum is traversed backwards to identify paths to Ultima

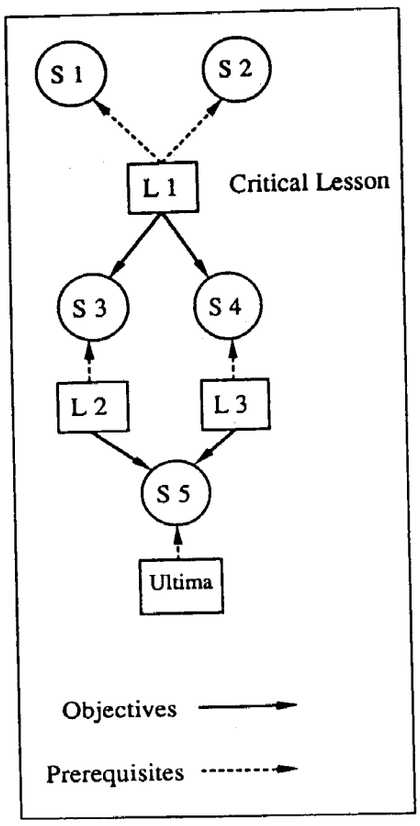

Figure 9. Computing critical lessons: Critical lessons such as L1 must be taken in order to reach the final lesson, Ultima

There are at least two algorithms for computing the set of critical lessons. The first is a "brute force" approach that is easy to understand but is less efficient than the other. In the first approach, we remove each lesson in turn and simulate a student as in the first part of the algorithm in the previous section. If Ultima is not reachable, then the lesson removed must be a critical lesson.

```
set Critical-Lessons to the empty set

for each L in Lessons
        remove L from Lessons
        relabel lessons and skills as in the first half
            of the previous algorithm
        if label of Ultima equals -1
        then add L to Critical-Lessons
        replace L in Lessons
```

In an actual implementation, rather than *removing* a lesson from the curriculum, it may be better to simply mark the lesson by setting an attribute to a special value. The relabeling algorithm would also be modified so as not to simulate taking a lesson if it is so marked.

The second algorithm for computing critical lessons is similar in spirit to the implies computation described earlier. The implied-lessons attribute will tell which lessons must have been passed in order to take a given lesson or master a particular skill. The following relations are true of implied-lessons:

1. A lesson implies itself. This reflects the tautology that a student who takes lesson X has taken lesson X. In Figure 10, L1 implies L1, L2 implies L2, and L3 implies L3.
2. If all lessons that teach a skill S have L as an implied lesson, then L is an implied lesson of S. In other words, if the student knows S then she must have taken L because any lesson that could teach S implies L. In Figure 10, S1 implies L1, indicating that possession of skill S1 implies that L1 was taken.
3. A lesson implies all lessons that are implied by the prerequisites. In other words, if a lesson X is implied by a prerequisite P of L, then L implies lesson X. In Figure 10, both L2 and L3 imply L1 because their prerequisite S1 implies L1.

The procedure is as follows:

```
—initialize implied-lessons attributes
for each S in Skills
        set implied-lessons of S to the empty set
for each L in Lessons
        —a lesson implies itself
        set implied-lessons of L to {L}

—iterate until there is no change
loop
        set Change-Flag to false
        for each S in Skills
```

```
            set Old-Implied to implied-lessons of S
            —find the set of lessons implied by all lessons that teach s
            set Imp-By to Lessons
            for each L in taught-by of S
                  set Imp-By to the set-intersection of implied-lessons of L
                  and Imp-By
            —see if there was any change
            if Old-Implied is not the same as implied-lessons of L
            then set Change-Flag to true
—now compute implied-lessons for each lesson
for each L in Lessons
            set Old-Implied to implies of L
            —compute the union of lessons implied by prerequisites
            set Prereq-Implied to ~~the empty set~~ implies of L
            for each P in prerequisites of L
            set Prereq-Implied to the union of Prereq-Implied
            and implied-lessons of P
            —now Prereq-Implied is union of lessons implied by prerequisites
            set Implied-lessons of L to Prereq-Implied
            —see if there was any change
            if Old-Implied is not the same as implied-lessons of L
            then set Change-Flag to true
—repeat the loop if there was a change
until Change-Flag is false
```

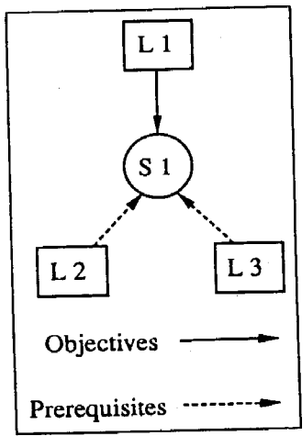

Figure 10. Computing implied-lessons: The implied-lessons attribute tells which lessons must have been passed in order to take a given lesson or to master a particular skill

A byproduct of this procedure is the implied-lessons attribute on each skill and lesson. This attribute tells which lessons *must* have been taken in order to reach a given lesson or skill. The implied-lessons attribute of Ultima is the set of critical lessons for the curriculum.

Computing "Unlearnable" Skills and Lessons

An important property of a curriculum is connectivity or reachability. That is, an instructional designer wants to ensure that each lesson and skill can be reached by taking some sequence of lessons. An unreachable or *unlearnable* lesson or skill indicates a problem in the curriculum.

The procedure to compute unlearnable skills and lessons uses the results of the simulated student from the previous section. Any lesson or skill not visited and marked with a level cannot be reached by any path through the curriculum.

```
set Cannot-Be-Learned to the empty set
set Cannot-Be-Taught to the empty set

for each S in Skills
        if level of S equals -1
        then add S to Cannot-Be-Learned

for each L in Lessons
        if level of L equals -1
        then add L to Cannot-Be-Taught
```

Skill Groups

In many cases, it is useful to give a single name to a group of related skills. For example, if there was a procedure called CHANGE-A-TIRE consisting of many smaller subtasks such as REMOVE-HUBCAP, GET-JACK, JACK-UP-CAR, and so on, we could lump them under a single category called KNOWS-TIRE-CHANGING. This simplifies the domain representation problem and makes formal designs more readable. We call a set of skills a *skill group*.

One way to incorporate skill groups into the lessons and skills model is to expand each skill group into a lesson and a skill. Given a skill group G representing mastery of skills $S_1, S_2, \ldots S_N$, we add to the curriculum a new skill S_G and a new lesson L_G with prerequisites S_1 through S_N and objective

S_G. The new lesson L_G has no content, so as soon as the prerequisites are satisfied, the objective S_G is met, that is, S_G serves as a shorthand for the group S_1, S_2,...S_N. This is exactly the semantics we want for skill groups. Figure 11 illustrates a skill and a lesson that together represent a skill group.

It is also possible to represent skill groups explicitly. Because skill groups have attributes of both lessons and skills, the algorithms become more complex, but there are no fundamental changes. The left half of the figure shows how the curriculum might look without a skill-group to represent S_1 through S_N.

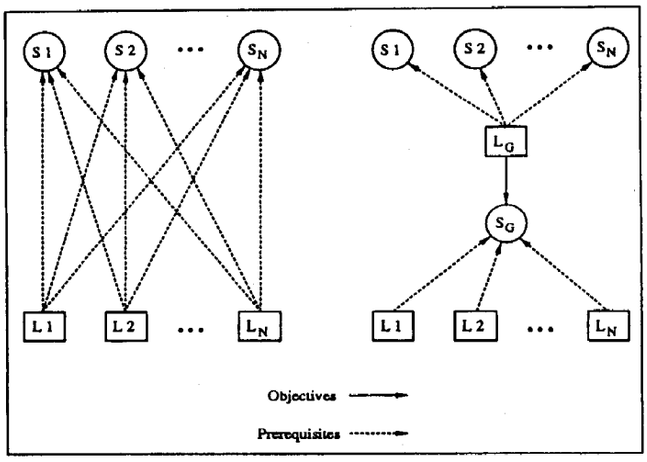

Figure 11. Skill groups represent knowledge of a collection of skills; an empty lesson and a skill can be used to simulate a skill group as shown

CLOSING COMMENTS

From our experience in building the Piano Tutor, we have learned that it is possible to use instructional design as an organizing principle for an intelligent tutor. Specifically, we can use a formal model based on lessons and skills to represent a curriculum and to tailor the curriculum to individual students automatically. We fully expect that improvements can be made to enhance the designer's ability to sort out pertinent relationships among skills and lessons. Additionally, we expect this research will eventually aid in speeding up the process of lesson creation and system design in intelli-

gent tutoring systems. This will also mean further research in this area as well as the possibility of more and better systems available on many subjects at lower cost to educational institutions and the general public. We believe the elements of instructional design supported by our formal curriculum design process will benefit future tutoring systems.

References

Aho, A.V., Hopcroft, J.E., & Ullman, J.D. (1974). *The design and analysis of computer algorithms.* Reading, MA: Addison Wesley.

Anderson, J.R., & Reiser, B.J. (1985, April). The LISP Tutor. *Byte,* pp. 159–175.

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

Bonar, J.G. (1985). *Understanding the bugs of novice programmers.* Unpublished doctoral dissertation, University of Massachusetts, Amherst.

Briggs, Leshe J. (Ed.). (1977). *Instructional design: Principles and applications.* Englewood Cliffs, NJ: Educational Technology Publications.

Capell, P. (1989). A *content analysis approach used in the study of the characteristics of instructional design in three intelligent tutoring systems: The LISP Tutor, Bridge, and Piano Tutor.* Unpublished doctoral dissertation, University of Pittsburgh, Pittsburgh.

Dannenberg, R.B., Sanchez, M., Joseph, A., Capell, P., Joseph, R., & Saul, R.A (1990). Computer-Based Multi-Media Tutor for Beginning Piano Students. *Interface, 19*(2–3), 155–173.

Dannenberg, R.B., Sanchez, M., Joseph, A., Saul, R., Joseph, R., & Capell, P. (1990). An expert system for teaching piano to novices. In S. Arnold & G. Hair (Eds.), *ICMC Glasgow 1990 Proceedings* (pp. 20–23). San Francisco: International Computer Music Association.

Derry, S.J., Hawkes, L.W., & Ziegler, U. (1988). A plan-based opportunistic architecture for intelligent tutoring. In G. Bochmann & M. Jones (Eds.), *Proceedings of the International Conference on Intelligent Tutoring Systems* (pp. 116–123). Montreal: ACM.

Halff, H.M. (1988). Curriculum and instruction in automated tutors. In M.C. Polson & J.J. Richardson (Eds.), *Foundations of intelligent tutoring systems* (pp. 79–108). NJ: Lawrence Erlbaum.

McArthur, D., Stasz, C., Hotta, J., Peter, O., & Burdorf, C. (1978). Skill-oriented sequencing in an intelligent tutor for basic algebra. *Instructional Science, 17,* 281–307.

McCalla, G.I., Peachey, D.R., & Ward, B. (1982). An architecture for the design of large-scale intelligent teaching systems. In N. Cercone & G. McCalla (Eds.), *Proceedings of the 4th National Conference of the Canadi-*

an Society for Computational Studies of Intelligence (pp. 85–91). Saskatoon: CSCSI.

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

Richey, Rita. (1986). *The Theoretical and Conceptual Bases of Instructional Design.* NY: Nichols Publishing.

Seels, B. (1989). The instructional design movement in educational technology. *Educational Technology, 29*(5), 11–15.

Wescourt, K., Beard, M., & Gould, L. (1977). Knowledge-based adaptive curriculum sequencing for CAI: Application of a network representation. In *Proceedings of the National ACM Conference* (pp. 234–240). Seattle: ACM.

Wipond, K., & Jones, M. (1988). Curriculum and knowledge representation in a knowledge-based system for curriculum development. In *Proceedings of the International Conference on Intelligent Tutoring Systems* (pp. 97–102). Montreal: ACM.

Wulf, W., Johnsson, R.K., Weinstock, C.B., Hobbs, S.O., & Geschke, C.M. (1975). *The design of an optimizing compiler.* New York, NY: American Elsevier.

Acknowledgements

We are grateful for support from the Markle Foundation for this work. We would also like to thank Gordon McCalla for his helpful suggestions, encouragement, and pointers to the literature.