Intelligent CBI Systems for Diagnostic Testing and Instruction

Joseph M. Scandura, University of Pennsylvania
Alice B. Scandura, Intelligent Micro Systems, Inc.

The research discussed herein:

1. reviews the structural learning theory (SLT) and shows how it provides an explicit basis for engineering intelligent CAI systems;
2. details a design for an intelligent "RuleTutor" based on these conceptualizations with emphasis on diagnosis and remediation;
3. describes a concrete implementation of the RuleTutor in the context of whole number arithmetic; and finally,
4. determines the strengths and limitations of this implementation and suggest ways in which the RuleTutor design can be enhanced.

To state the problem concisely: can a conceptual approach which uses available knowledge of cognitive processes provide an explicit basis for designing current and future intelligent CAI systems?

To utilize structural learning principles in designing instruction, the essential first step is to identify: (1) the educational goals — what the learner is to be able to do after instruction and (2) prototypic cognitive processes or rules — what the learner must learn if he is to successfully perform tasks associated with the educational goals.

Consider column subtraction problems as a simple prototype. In this case assume the educational goal is to find the difference. The prototypic rule or cognitive process in this case refers essentially to what the learner must master in order to subtract numbers.

The flow diagram in Figure 1 depicts the procedural portion of a column subtraction rule. In this rule, it is implicitly assumed that each operation acts only on digits, rows and columns — consequently, the previously referred to need to assume certain minimal encoding/decoding abilities. The decisions of this procedure (e.g., is top digit greater than bottom digit?) constitute additional assumptions concerning minimal cognitive ability. According to structural learning principles, only to the extent that these assumptions are met will this subtraction rule provide a useful and operationally precise basis for designing efficient and effective instructional strategies.
The teaching-learning process involves a generalization of the above in which there are two participants, one an "idealized" teacher and the other, a learner, as diagrammed in Figure 3.

![Figure 3. Interactions between an idealized teacher and a learner in Structural Learning Theory](image)

Notice, in particular, that in this model the idealized teacher knows (i.e., has direct access to) all of the rules (identified via S/A) and can recognize and/or generate arbitrary problems in the Problem Domain. In addition, this idealized teacher has built into it all of the theoretically optimal machinery for diagnosing learner difficulties and for providing optimally efficient remediation.
General Design Principles for a RuleTutor

Any CAI tutorial system based on the Structural Learning Theory must as a minimum include: the content specific rules (procedures) to be taught and a general purpose RuleTutor (driver program) which takes the content specific rules and uses them within a general interactive, instructional test/management system. The theoretical rationale of structural analysis (by which the content specific rules/procedures are defined and explained) is illustrated in Scandura (1982, 1984a, 1984b). In this chapter, a design based on these conceptualizations for an intelligent “RuleTutor” will be described.

An effective and practicable instructional system should have at least five distinct instructional capabilities as well as appropriate student performance data management capabilities. The instructional capabilities should include:

1. Diagnostic pretest administration:
2. Prescriptive (adaptable) instructional sequencing based on pretest results.
3. Capability for incorporating additional instructional objectives such as “meaning,” short cuts, verbalization of procedures, etc.
4. Systematic Mastery Testing of simpler paths to confirm learning of basic procedures necessary for later instruction.
5. Efficient posttest administration to check overall retention.

The management system should, at a minimum, record pretest, instructional and posttest results for individual students. Additional management capabilities might include: ability to override pretest prescriptions, assignment of an individual to any place in the sequence, adaptation of criteria for pretest, lesson and posttest progression, analysis of an individual’s capabilities in areas such as “meaning,” verbalization, “timed” practice, etc., meaning, “metacognition” (or verbal awareness of what one knows) and short-cuts commonly achieved by experts.

Since the RuleTutor is a driver program, it needs some form of software for generating problems (tasks) and procedures for solving these tasks. The RuleTutor can utilize these capabilities in deciding which problems to present during testing and which instruction to provide during training. The basic system is quite general and could in principle be used with arbitrary cognitive procedural tasks.

Let us describe in more detail how this RuleTutor functions. Given the content-specific information, the diagnostic testing portion of the system efficiently determines a student’s entering level, as described above. More specifically, it stores a “checklist” for the current student (in the student records disk file) of the known and not-yet-known paths. This checklist is read and updated by the instructional portion of the system as it teaches the student in turn each of the not-yet-known paths.

The instruction on each path has a number of components or instructional levels. For example, the RuleTutor:

1. teaches the meaning of the process,
2. teaches the relationship between this meaning and the process itself,
3. teaches the process itself, providing help where necessary,
4. helps the student to verbalize the cognitive processes he or she has learned by having the student name the processes used or observed, and
5. helps the student to automate the process (once the rule has been learned), thereby increasing his degree of skill.

Within each of the above instructional components or levels, the RuleTutor system also can vary the difficulty of the material and can adapt to the student by increasing problem difficulty (or type) at a rate depending on the student’s prior learning efficiency. For students who are currently learning very efficiently the difficulty level will increase relatively rapidly, while the difficulty level will increase relatively slowly for students who are currently not learning as efficiently as they might. Furthermore, learners may skip some of the instructional levels mentioned above if warranted by their learning efficiency on previous paths in the domain.

The instructional portion of the RuleTutor system stores detailed information about a student’s performance in the student records disk file. This information can be accessed via the management portion of the system, and the parameters representing, for example, the rate at which problem difficulty is increased for a given student or the paths on which that student should be given instruction can be explicitly altered by an instructor.

In the following sections, we describe the RuleTutor design more explicitly. Anticipating the arithmetic implementation, many of the ideas are illustrated in this context.

Diagnostic Pretests

Each Pretest simultaneously evaluates the student in two ways:

1. It determines which types of problems the student can and cannot solve.
2. Independently, it determines whether the student has adequately mastered the prerequisites (e.g., basic facts).

The problem types used in any particular implementation will be determined via structural analysis, which yields types which are homogeneous in the sense that all problems of given type require the same cognitive processes for solution.

For purposes of pretesting, these problem types are arranged in a partial ordering. For example, consider the following lattice in Figure 4.
In this lattice the seven nodes refer to problem types. Notice that these nodes are arranged at five different levels. Type 7 is more difficult than Types 5 and 6, which are prerequisite to it. Conversely, Types 5 and 6 must be mastered before Type 7. On the other hand, the relative difficulties of Types 3 and 4, or for that matter Types 3 and 6 or 5 and 6, cannot be determined from the lattice.

In general, diagnostic pretesting begins at or near the middle level (here Level III). In our example, the student first would be tested at a relatively simple problem of Type 3 (i.e., on a problem involving the smallest numbers of digits for that type). If the student fails, the system would infer that the student would be unable to solve more complex problems (i.e., of Types 5 or 7) which are above Type 3 in the hierarchy; testing then would progress to Type 4 problems. If the student succeeds, the next problem presented would be the most complex of Type 3 (that can be displayed on the screen). Success here would cause the system to infer mastery not only of Type 3 problems but also of all prerequisites of Type 3 problems, specifically those of Types 2 and 1. Then, testing would progress to Type 4.

Note that the pretest system may automatically infer capabilities (or lack thereof) on more than one type at a time. It is this capability which makes testing so efficient.

The above simple, then complex, problem pattern is used with all problem types that have not at any given point been determined to be mastered or not mastered.

Suppose, for example, that a student has mastered Type 3 and hence also Types 1 and 2, but that he or she fails Type 4 (implying also a lack of mastery on Types 5, 6, and 7). In this case as few as three problems are sufficient to determine mastery on all problem types and, hence, that the student should begin instruction at Type 4 before moving on to Types 5, 6, and 7.

If testing had indicated mastery of both Types 3 and 4, then testing would have continued at Level IV (Types 5 and 6). In general, testing will move to higher or lower levels in the hierarchy as determined by the student’s performance — just until the system has been able to infer which types the student has and has not mastered.

During pretests (as well as lessons and posttests) the type of problem along with the number of rows (e.g., rows in addition), the number of digits in each row and the time allowed for each response, can be determined directly from the screen. In the upper right hand corner of the screen, “T” for Type is set equal to the problem type number; “D” for Difficulty Level is set equal to the number of numbers (e.g., rows in an addition problem), followed by the number of digits in each number; and “S” for Speed is set equal to the allowed number of seconds for student responses. (On Lessons, “I” for Instruction-Level indicates the type of instruction.)

At the same time the system is assessing problem types, it also is determining how adequately the prerequisites (e.g., basic facts) have been mastered.

As soon as pretesting has been completed, the system displays for the student the types of problems, including prerequisites, on which instruction is needed.

**Lessons**

Lessons are organized according to Problem Types, Instruction Levels and Difficulty Levels (of problems).

Whole number subtraction problems, for example, may be partitioned into the seven problem types illustrated in the lattice of Figure 4:

1. Single Digit Facts
2. Facts From 10 to 19
3. No Regrouping
4. All Regrouping
5. Mixed Regrouping
6. Regrouping Across 0
7. Mixed Regrouping Across 0

There are up to nine Instruction Levels (I-Levels) for each Problem Type:

1. MEANING
2. MEANING/RULE RELATIONSHIP
3. DIRECTIONS/RULE
4. RULE/REMEDIAL
5. DESCRIBING RULE PROCESSES
6. SHORTCUT INSTRUCTION
7. SHORTCUT (MENTAL) CALCULATION
8. TIMED PRACTICE
In general (e.g., unless the instructor decides otherwise or a given problem type does not require that I-Level), instruction begins, first, with the MEANING (I = 1) of the process (or RULE) as applied to the given problem type.

Note that the term RULE refers to the basic cognitive skill under consideration — for example, the column subtraction algorithm. MEANING refers to a process corresponding to the RULE which relates more directly to its meaning — for example, in the case of subtraction the process is related to numbers in expanded notation (e.g., 346 = 300 + 40 + 6) which in turn is equivalent to the use of Dienes' blocks, sticks grouped by 1's, 10's, 100's, etc.

Instruction at the meaning level is extremely detailed. To minimize the amount of keyboard entry that is required of the learner, inessential parts of the required responses are generated by the computer automatically. For example, in expanded notation problems, the 0's are generated automatically as soon as the learner types the corresponding digit (e.g., the "3" in "300"). Various corrective feedback occurs at any incorrect input during the lessons.

1. Meaning
   Show all work.
   \[
   \begin{align*}
   200 + 20 + 8 & = 228 \\
   - 90 - 9 & = 199
   \end{align*}
   \]
   Now you do it, Name!

At the second I-Level, the system visually shows the relationship between the meaningful process and the targeted RULE. Throughout, the student must interact with the system.

2. Meaning Rule
   Show all work.
   \[
   \begin{align*}
   300 + 10 + 0 & = 310 \\
   - 200 - 40 - 2 & = 242
   \end{align*}
   \]

Third, the student is taken through the rule step by step. A description of the step is displayed on the screen and the student is required to carry it out.

Fourth, the student must go through the process on his or her own for the first time. If errors are made the system automatically reviews the meaning of the process up to the point where the error was made.

5. Describing Rule
   Describe the step shown.
   \[
   \begin{align*}
   T & = 4, I = 5 \\
   D & = 3, 2
   \end{align*}
   \]

At the sixth I-Level, the student is taught to perform the rule mentally without going through the steps overtly. The student is shown the detailed steps on one display of the problem, up to just before each digit in the answer; the student must input the correct digit in the second display.

6. Shortcut instruction
   Compute in your head.
   \[
   \begin{align*}
   T & = 4, I = 6 \\
   D & = 4, 4
   \end{align*}
   \]
   Rename in your head, then subtract.
Seventh, mental practice (i.e., drill and practice) is given and eighth, where the problem type is deemed especially important, further practice is provided under graded time pressures.

7. Shortcut (mental) calculations
Compute in your head.  
T = 4, 1 = 7
D = 4, 4
5150
- 2963

8. Timed practice
Compute in your head.  
T = 4, 1 = 8
Try to beat the beep.  
D = 4, 4
S = 5
4631
- 2695

Students are signed off the system automatically as soon as they complete all (normally seven or eight) levels of instruction on a given problem type (except for prerequisites where only I-Level 8 is used). (This is to encourage students to start the corresponding Mastery Test when fresh.) The next time such a student signs on, he or she automatically will be assigned to a mastery test.

In general, Difficulty Level refers to the number of numbers (e.g., being added), the number of digits per number, and the times allowed for response. There is a range of possible difficulty levels and response times for each problem type/instruction level combination. In a few cases, screen size prohibits the presentation of certain problem/type levels. When this happens, the system makes the needed adjustments automatically.

Lesson instruction is individualized at three basically different levels. First, the pretest automatically determines the type of problems on which the learner should begin.

Second, the learner progresses through the material at his or her own rate. Mastery is required at each difficulty level, instruction level and problem type before advancing to the next level. Conversely, poor performance results in the learner's moving back levels. If a student fails a Mastery Test, then he or she is required to start over at Instruction Level 7, except for Prerequisite Types or Facts where all practice is timed (i.e., at Instruction Level 8). (All steps on a given problem must be correct during instruction in order for that problem to count toward advancement. Two or more errors on a problem count toward moving back; one error counts for neither but just results in another problem being presented.)

Third, and perhaps most subtly, the system automatically adjusts “Step-Size” either increasing it or decreasing it, according to how efficiently the learner has learned previous material.

Two types of step-size are used, one involving difficulty level and the other instructional level. The smallest Difficulty-Level Step-Size (0) requires the learner to go through and master all difficulty levels associated with the given problem type/instruction (L-Level) combination. If a learner masters a given L-Level efficiently (say by missing fewer than 10% of the questions asked during instruction), then he moves to the intermediate step-size (1). Here, the learner must first master the minimum difficulty level and then the maximum before progressing. The learner skips all intermediate levels. If a learner does poorly (say by missing more than 25% of the instructional questions), then, as you might expect, the step-size will be decreased — unless the learner is already at the smallest step size.

At the highest Difficulty-Level Step-Size (2), the learner only receives instruction at the highest difficulty level for each problem type/instruction level combination. Instruction in this case can progress quite rapidly.

There are also three levels of Instruction-Level Step-Size. The smallest Instruction-Level Step-Size (0) requires the learner to go through and master all instruction levels for each problem type. The intermediate size (1) allows the learner to skip Instruction Levels 1 (Meaning) and 3 (Directions/Rule) on the assumption that the missed material is readily inferred from Instruction Levels 2 (Meaning/Rule Relationship) and 4 (Rule/Remedial), respectively. When at the largest step-size (2), instruction begins with levels 4 and 5 (Descriptive).

Level 6 (Shortcut Instruction) also is skipped based on the assumption that fast learners will be able to make the necessary adjustments independently. As with difficulty levels, transitions from one Instruction-Level-Step-Size to another are determined by learning efficiency — this time based on performance on previous problem types.

Optional criteria for determining both difficulty and instruction level step sizes may be assigned by the instructor in Option 8 of the Management System. Option 5 further allows the teacher to override the other two individualization features and to design students to whatever problem types and instruction levels might be desired.

Tutorial Arithmetic System

The RuleTutor design described above has been fully implemented and field tested on the Apple II microcomputer (using DOS 3.3, AppleSoft BASIC and 6502 Assembler) by Standard Training Systems as part of its Micro Tutor II line of software.

Paralleling the preceding analysis and design principles, the RuleTutor has two modular main sections, supplemented by three supervisory or utility programs.

1. The first of the two main sections is the content-specific code and data. As its name implies, there is a version of this for each
content area. It includes: code for generating random problems of each problem type, code for displaying problem components in windows (called “slots”) on the screen (and a file of data called a “template” which specifies the positional relationships between the slots), code for generating each solution step in the solution algorithm to be learned, a file of descriptions/verbal instructions for each step in the algorithm, and a lattice hierarchy of problem types along with an array of allowable instruction levels for each problem type. The content specific code was generated manually, although work is currently being done on a system that would facilitate the authoring of such procedural material. All of the problem generators used until now involve random problem generation (subject, of course, to the constraints necessary to guarantee that a problem belongs to the desired problem type), but it would also be possible to use predetermined problems of essentially any sort — as might be desired, for example, in verbal or language instruction.

2. The other main section is the common code, which is used with all content areas. This is essentially the driver program. It accesses the content specific code as necessary for diagnostic testing and instruction. During diagnostic testing, for example, it determines from the lattice the appropriate initial and subsequent problem types to be presented (in accordance with the theory described above). Then, it calls on the problem generator to generate a problem of the desired type and on formatting/display routines to display it on the screen. After that it gets student inputs and compares them with solution components generated by applying the content specific solution algorithm to the problem. Next, it draws inferences concerning other problem types in the lattice as described previously (i.e., types subordinate to a passed type are marked as passed; types superordinate to a failed type are marked as failed). Finally, the process is repeated until the pass/fail status of all problem types has been determined.

During instruction, the common code builds on student prescriptions determined during pretesting. Given the prescribed problem type, the common code determines the next allowable instruction (I) level for that type and calls on the program generator and display routines. Then it executes the steps of the solution algorithm one by one, providing for each step the type of instruction called for by the current I-Level, while maintaining various kinds of information concerning student responses.

3. The common and content-specific code are embedded in or called from general supervisory management code. The management system does such things as present title or introductory screens, obtain the student’s name and access or update the student records. It also maintains teacher-modifiable system characteristics, such as for mastery, and it controls access to the RuleTutor and the records transfer and data access systems mentioned below.

4. The records transfer program is a utility made necessary by the fact that pretests, instruction and post-tests are on separate disks. The system simply transfers the records of students from pretest to instruction (or from instruction to posttest) disks.

5. The access program is a fairly long program in its own right which provides the teacher user with access to and a means of modifying all of the student and system records stored on disk (by the management system). This code could not be integrated with the other common code because of memory limitations.

Scope and Sequence

In whole number column addition, there are nine problem types and 2-9 instruction/mastery test levels with 1-5 difficulty levels for each, for a total of up to 405 different instructional combinations. Space is provided for maintaining the records of 30 students on each Learning/Mastery Test disk.

In whole number subtraction, there are seven problem types and 2-9 instruction/mastery test levels with 1-5 difficulty levels for a total of up to 315 different instruction levels. Space is provided for maintaining the records of 30 students on each Learning/Mastery Test Disk.

In whole number multiplication, there are nine problem types and 2-9 instruction/mastery test levels, with 1-5 difficulty levels for a total of up to 405 different instructional combinations. Space is provided for maintaining the records of 30 students on each Learning/Mastery Test Disk.

In whole number division, there are seven problem types and 2-9 instruction/mastery test levels with 1-5 difficulty levels for a total of up to 315 different instruction levels. Space is provided for maintaining the records of 30 students on each Learning/Mastery Disk.

Limitations and Future Directions

The MicroTutor II Arithmetic implementation of the Intelligent RuleTutor has been used by a large number of elementary school students with considerable success. Although no statistical study of its effectiveness has been carried out, most teachers have observed that students enjoy working with the system and learn effectively from it.

In spite of its practical utility, however, certain theoretical problems are being dealt with in future research. First of all, the Micro Tutor II Arithmetic was designed to provide diagnostic testing and instruction on individual rules. Consequently, even in principle, Micro Tutor II Tutorial Arithmetic is limited to algorithmic content.

For another thing, MicroTutor II Arithmetic is implemented in AppleSoft (a version of BASIC developed for the Apple II computer) and 6502 machine language. Consequently, it is not easily transport
able. Moreover, limited memory and use of the BASIC language made it difficult to adhere strictly to the modularity (e.g., the distinction between common and content-specific code) we have strived for.

Finally, implementation did not reflect the underlying theory as accurately as possible. For example, meaning, metacognition and automation were treated in an ad hoc fashion. Accurate implementation would have called for introduction of higher-order rules (rules which operate on rules). This was not done for both practical (i.e., resource limitations) and technical reasons (e.g., the limited capacity of the Apple II computer). In future research on the RuleTutor concept, it would be desirable to improve the design, thereby yielding a new RuleTutor more powerful and generalizable than the existing one. Among other things, future designs should make provision for the following.

First, rule diagnosis and rule instruction in the current RuleTutor design are totally independent activities. Thus, all diagnostic testing is completed (in a sequential and highly efficient manner) before any instruction is provided. In fact, however, testing and teaching are highly interrelated both in practice and in principle. Thus, partial information from testing may provide a sufficient basis for (some) instruction. Conversely, instruction on a portion of a rule may influence test performance on other items and, hence, reduce the amount of instruction that otherwise might be prescribed.

Second, the current RuleTutor involves instruction on individual rules (i.e., cognitive procedures). Consequently, the current RuleTutor design could not readily be extended to deal with sets of lower- and higher-order rules, even in principle. Among other things, doing so would require implementation of the universal "goal switching" control mechanism proposed by Scandura (e.g., 1971, 1973, 1981).

Third, although it cannot deal with higher-order rules, the current RuleTutor was designed so as to achieve some of the practical benefits they would have provided. However, the RuleTutor's ability to deal with such things as rule meaning and verbal awareness was bought at the price of some loss of extensibility.

Fourth, even though modularity and structured programming were at the forefront of our original RuleTutor development effort, our use of the Basic language (because of its broad availability on microcomputers) and memory limitations of the Apple II computer resulted in some unavoidable compromises along these lines.

In summary, the Arithmetic RuleTutor case study has shown that the structural learning theory provides a strong foundation for the design and implementation of highly practical and useful intelligent CAI systems. And, as the above discussion suggests, the theory can be used in a deeper way as the basis for even more flexible and efficient CAI systems. In the past 40 years we have seen an extraordinary rise in the amount of computational power that can be purchased with a given amount of money, and we also have witnessed considerable progress in the understanding of how to determine what people do and do not know, as well as how to provide remedial help. The combination of these two trends is barely beginning to have a large-scale impact on education, and we are eager to continue our involvement in this revolution.

References


