IEEE Transactions on Systems, Man, and Cybernetics, Vol. 19, No. 1, January/February 1989, pp. 40-57

Intelligent Computer-Aided Instruction: A Survey Organized Around System Components

Jeff W. Rickel

Abstract

This paper presents a survey of the issues and previous research in Intelligent Computer-Aided Instruction (ICAI). Unlike previous surveys of ICAI, which have been organized around the seminal ICAI systems, this one concentrates on the contribution of each effort to our understanding of the various components of ICAI systems. Along the way, comparisons are made between ICAI and CAI, although a CAI straw man is used due to the focus of the paper. Various learning scenarios are discussed, including computer coaches, gaming environments, mixed initiative dialog, Socratic tutors, articulate experts, interactive simulation, and discovery learning. Various forms of knowledge representation are discussed along with relevant issues and examples. Several techniques for student modeling and diagnosis are presented, as are their respective advantages and disadvantages. Pedagogical knowledge, its role in ICAI, and several examples are highlighted. The evolution of discourse management techniques for ICAI is outlined. Techniques for the automatic generation of problems from a general base of domain knowledge are presented. Finally, issues in the design of user interfaces for ICAI systems are briefly discussed. The purpose of this paper is to place previous ICAI research in a form which is appropriate for use by future designers of ICAI systems.

I Introduction

Although computer-aided instruction (CAI)[1] has been around for some time now, there has been much discontent with the current state of educational software. Traditional "frame-oriented CAI" requires teachers to fully specify presentation text, all questions and their associated answers, and a strict flow of control through the course, allowing at best different branches to be taken based on the student's pre-enumerated possible responses. This style of teaching has often been referred to as "electronic page turning."

A step in the right direction was made with generative CAI. In these systems, techniques such as textual templates, problem-generation grammars, and random numbers were used to generate problems for the student. Still, the systems lacked the ability to adapt to the student or relate his incorrect answers to the correct ones, and the problems were usually limited to simple drill-and-practice questions [40].

[1] Because this paper concentrates on ICAI, a CAI straw man is used for comparison. In fairness to CAI researchers, it should be noted that the problems with CAI were well known, and many practitioners developed ingenious solutions for them. Nevertheless, our characterization of CAI is adequate for the purposes of this paper.

Educators and computer scientists alike saw the need for "intelligence" in these systems. Dubbed ICAI for "intelligent" CAI, these systems would have a rich representation of their domain which would allow them to use their knowledge in ways unspecified by the course designer. Also, by modeling the student himself, these systems could individualize their instruction, tailoring their presentation to the level of the student's knowledge and his rate of learning.

Although there has been much research in ICAI, few if any commercial systems exist due to the current cost of building such systems and the lack of experienced people in the area. As noted by Psotka et al. [43], "When one considers the ICAI systems that have been built to date, one is impressed with the fact that each is somewhat like the work of a craftsman, a combination of technical skill and artistic creativity." ICAI is still very much a research topic, with today's systems suffering the same basic limitations as more general AI systems (i.e., lack of commonsense reasoning). Yet ICAI is a natural test bed for AI techniques since it requires reasoning from a rich knowledge representation, modeling the user, and communication of information structures. While today's ICAI systems are carefully crafted, one of the primary motivations for further research is the potential for educational systems which generate their pedagogical abilities automatically given only a base of domain knowledge. As AI and ICAI research continue to advance, computer-based tutors should become both more effective and more economical.

The remainder of this paper will survey the important contributions of past ICAI research efforts. Most previous surveys have been organized around the seminal systems. While this gives an important historical perspective, it leaves to the ICAI system designer the task of drawing together the contributions of each effort to our understanding of the various components of a typical ICAI system. Having been through this process, the author will present in this paper this latter perspective so that the designers of future ICAI systems can readily leverage the lessons learned by their predecessors. The interested reader is referred to Fletcher [19], Feigenbaum and Barr [18], and Park et al. [40] for good surveys of past ICAI systems, and to Park et al. [40] for an extensive comparison of CAI and ICAI.

II Learning Scenarios

A learning scenario is the situation in which the student's learning is to take place. The most important criterion with which to view a learning scenario is the balance of control between student and tutor. Along this dimension, learning scenarios range from the simple "rule-example-practice" paradigm used in traditional frame-oriented CAI programs to controllable simulation to mixed-initiative dialog to discovery learning [20]. This dimension forms a continuous spectrum from frame-based CAI, in which the student can't explore things of interest, to open-ended discovery environments, such as LOGO [39], in which the student may get stuck or waste a lot of time. The knowledge required by the tutoring system is minimal on either ends of this spectrum, but increases towards the middle [7].

Evidence suggests that learned information is retained longer if the student is an active participant in the learning process (rather than a passive "absorber") and if the presentation involves several of the student's senses. One study reports that people retain about 25 percent of what they hear, 45 percent of what they see and hear, and 70 percent of what they see, hear, and do [17]. Experts in diverse fields suggest that true understanding of knowledge gained from formal training comes when the student later combines that knowledge with actual experience and application [38, 53]. Indeed, the backlash against traditional CAI has focused on the passive role of the student and has prompted educators and instructional designers to seek more effective uses of computers for education. This explains the more ambitious computer learning scenarios that will be described in this section.

One particularly popular form of instructional philosophy is that of the computer coach [22]. Brown and Moskowitz sum up the coaching scenario as follows: "At your request, the intelligent coach 'looks over your shoulder' while you attempt to carry out tasks, offering timely but unobtrusive advice" [8]. One advantage of computer coaches is that training becomes more cost-effective since the coach is available at the convenience of the student and thus off-site learning is minimized. The coach is available not just for formal training, but also to help out during normal work activities, saving the user time that might otherwise "be spent thumbing through a manual or waiting for help from a colleague" [8]. In this sense, we can see that tutorial ICAI and job aiding need not be mutually exclusive [38, 49]. Coaching is most effective in tasks where the student's performance tends to reach plateaus [25]; the exposition of key knowledge often opens up a whole new area for exploration by the student.

Gaming environments can combine the characteristics of both coaching and informal discovery learning [10]. In such a scenario, a game is designed to teach more general skills. The student learns these skills by applying them in the context of the game and may even discover them through their positive influence on his game position. A coach can point out when the student is consistently employing a suboptimal strategy and can demonstrate the use of the desired skills in winning the game. This learning scenario has the advantages of discovery (the student often discovers the skills himself), application (the student learns the skills by using them), coaching (the computer coach can lift the student off plateaus and guide him when he seems misdirected), and, most importantly, the student's attention is heightened by his interest in the game itself.

The prototypical example of a game-based learning scenario is the WEST system of Burton and Brown [10]. WEST is a computer tutoring system which coaches students through the game "How the West was Won." The game requires students to combine numbers, using arithmetic operators and parentheses, into expressions which evaluate to the number of board positions they would like to move. Proficiency in the game thus requires mathematical as well as tactical skills, and it is the job of the WEST coach to determine whether or not the student is employing the right skills and when he needs guidance. Their paper [10] is an excellent treatise on the issues involved in computer coaching, especially in gaming environments.

Many educators have lamented the one-sided nature of CAI. They claim that the student should be allowed to ask questions of the tutor and in general affect the directions of the tutoring session. In 1970, Jaime Carbonell launched the field of ICAI by using AI techniques to build a *mixed-initiative* tutoring system. His seminal tutor, SCHOLAR [13], tutors students in geography by carrying on a natural language dialog with the student in which the student can respond to computer questions or initiate a line of questioning which SCHOLAR must answer. Carbonell emphasizes the benefits of mixed-initiative tutorial dialog, pointing out that it results in a more reactive tutoring system and that allowance for student initiative is crucial to effective education [13].

Closely related to mixed-initiative dialog is the use of the *Socratic* teaching method [42]. The Greek philosopher Socrates believed that education could not be attained through passive exercises such as reading or listening, but instead came from actual problem solving. His technique involved posing problems to the student, each one carefully crafted to require the student to use new knowledge, to point out gaps in the student's knowledge, or to entrap the student into discovering his own misconceptions. Stevens and Collins have explored the use of a Socratic method in their WHY tutor [55, 58], which tutors students on the causal relationships and factors affecting rainfall.

Gaming environments and mixed-initiative dialog scenarios differ in the demands they put on the tutoring system. In a dialog system, the tutor must be able to articulate its knowledge in response to queries, whereas a game-based tutor need only recognize that a student's answer/action is not optimal and provide guidance. On the other hand, a computer coach must infer student shortcomings from the context of the game, whereas mixed-initiative dialog systems allow the computer tutor to test hypotheses by asking new questions [10]. Woolf and McDonald [63] draw an important distinction between "context-dependent" tutoring systems, in which the tutoring system adapts its strategy based on the state of the student, and "retrieval-oriented" systems like WEST that place their emphasis on retrieving the correct answer, and suggest that the former provide more reactive tutoring. Both learning scenarios offer unique advantages and disadvantages, however, and each introduces its own architectural requirements.

Another popular instructional paradigm is that of the *articulate expert* exemplified by the famous SOPHIE systems [6]. SOPHIE[2] is designed to teach students about troubleshooting electronic circuits by allowing them to view the tutor solving problems before they are given their own problems. The student is allowed to insert faults in an electronic circuit, and SOPHIE then troubleshoots the circuit, explaining along the way not only the justifications for its decisions, measurements, and conclusions, but also its high-level troubleshooting tactics. The student, after watching several expert performances, is then given some problems to solve while SOPHIE watches and occasionally provides guidance and advice. In particular, SOPHIE won't let students make measurements unless that particular measurement can actually distinguish between several candidate faults.

Many researchers have pointed out the importance of rationalizing each action when teaching procedural skills. Beyond learning *when* actions are appropriate, a student must know *why* those actions are appropriate. A tutoring system must therefore present procedural skills at varying levels of abstraction and should explain actions in terms of their causal effects in the domain. For instance, in the Fault Isolation Procedure (FIP) tutor [38, 37], the current goal hierarchy is maintained. Tracing up this goal hierarchy answers the question "Why am I doing this action?". This goal-oriented perspective, along with qualitative, causal knowledge

[2] There are actually three distinct SOPHIE systems, known as SOPHIE I, II, and III. I speak of them in general under the name SOPHIE.

of how a system works, enables students to fully understand why and when certain actions are appropriate [38, 53].

Another way to give students a feel for the causal relationships in a domain is to provide them a simulation of that domain with which they can explore the consequences of their actions without fear of real-life consequences. Also, by thrusting the student into the ultimate performance environment, and coaching him into the right actions, the student will be more likely to carry his training into real-life situations.

STEAMER [29] is the archetypal example of an *interactive, inspectable simulation.* Although STEAMER grew into more generic tools, its original purpose was to provide simulation-based instruction on the workings of a basic steam propulsion system. It is interactive in that students can manipulate controls and see their effect. It is inspectable in that STEAMER includes an extensive graphics interface which allows students to view the workings of the steam system in a much more animated form than would otherwise be possible. Its authors coined the expression "dynamic graphical explanations" to describe STEAMER's style of instruction, noting that "relationships that are difficult to describe unambiguously in words are often easily depicted graphically" [29].

In a more recent attempt at simulation-based training, Beverly Woolf built a tutoring system for the Kraft recovery boiler whose goal "is to challenge an operator to solve boiler problems and to maintain his incentive to play with the tutor" [65]. In the Recovery Boiler Tutor (RBT), students are presented with a graphical window into the process and given problems to correct. Based on their actions, RBT offers coaching and advice. Although RBT is a real-time simulation, updating its process variables every one or two seconds, operators who have used it seem to like the ability to stop the process to ask questions or explore boiler characteristics [65]. Figure 1 shows an actual RBT tutorial dialog[3] extracted from [65].

The authors of STEAMER raise an important point about simulation-based training. It is not enough to simply present the simulation as a black box with which the student can play. The training tool must provide *conceptual fidelity* [29]; that is, the information must be presented to the student in a way that helps him build the correct cognitive constructs for thinking about the domain. These cognitive constructs need not accurately model the internal behavior of the domain, but should instead reflect the way experts think about it. For instance, if electronics experts think about electricity as discrete electrons flowing through the circuit, the student should be presented with this graphical model rather than the more accurate quantum mechanical model or just the voltage and current readings at various points. Give the student a useful mental model of the domain that is at the appropriate level of abstraction.

As a final note on simulation-based training, it should be noted that simulations can be time consuming to construct and are challenging from an educational standpoint because it becomes harder to detect student misconceptions (compared to mixed-initiative dialog, for instance). Most successful simulation-based instructional programs have combined a simulator with more traditional ICAI methods to produce a more versatile tutoring system.

[3]Student input in RBT is done through menus and tutor output is canned text extracted from emergency-specific text files.

[Note: A partial blackout is in progress and it will lead to deterioration of the combustion process if no action is taken. Meters indicate that CO, SO_2, and O_2 readings are abnormally high.]
Tutor: What action are you going to take?
Student: **Check smelt spout cooling water.** [Student's action does not result in a change in the boiler.]
Tutor: It looks as though the liquor isn't burning . . .
Student: **Help. What is going on?** [Student requests help.]
Tutor: A partial blackout is occurring and this causes TRS, CO, SO_2, and O_2 to increase. Steam flow and steam temperature are dropping. I think part of the bed is not burning.
Student: **Rod the primary air ports.** [A small improvement in the boiler readings can be observed.]
Tutor: Your action is correct; however, at this time rodding alone is not effective . . .
Student: **Raise the liquor temperature.** [Student's action does not reverse the emergency and a new alarm buzzer sounds.]
Tutor: Rodding the air port apparently was not enough to undo the blackout.
Student: **Increase the oil fuel supply.** [Student's action reverses the emergency.]
Tutor: That solved the problem alright. Good thinking.

Figure 1: RBT tutorial dialog.

Discovery learning is at the opposite end of the spectrum from frame-based CAI. This learning scenario can take many forms, but the basic philosophy is to present the student with some sort of microworld in which to explore and discover general rules and concepts. The idea is that the student, being in full control, can let his own curiosity and interest drive the session, and that those principles that he discovers on his own will be more likely to stay with him. Obviously, this sort of learning also prepares the student for future scientific inquiry and real-life problems since it forces him to develop his own problem-solving skills and learn to think creatively. We will have little more to say about this learning scenario since it is very different in character from the other basic approaches and in fact deserves a survey of its own.

Having explored the various forms of ICAI learning scenarios, it is now relevant to ask what knowledge is needed to support their respective requirements and what form this knowledge should take. This is the topic of the next section.

III Domain Knowledge Representation

The biggest problem with traditional CAI systems is that they have a poor knowledge of their domain. This knowledge typically consists of a canned presentation of the information to the student, canned problems with which to test his knowledge, canned answers, and prespecified branches in the presentation based on a pre-enumerated set of possible responses. It is hardly surprising when these systems are criticized for their lack of adaptation to the student. Their canned text prevents them from adapting the presentation to the needs of the student. Their prespecified branches prevent them from handling unanticipated answers. They have no facilities with which to ferret out the student's true misconceptions. They cannot construct problems which specifically address the student's needs. And their prespecified answers leave them with no criteria for judging student responses other than as right or wrong. A human teacher with the same level of knowledge of his subject could not do much better. We expect more domain competence on the part of our human teachers; we should strive for no less in computer tutors.

In a nutshell, the goal of ICAI is to plug the necessary knowledge into computer-based tutors to correct the above limitations. However, the knowledge needed to correct all the above shortcomings is tremendous, and no tutoring system to date has encompassed all the requisite knowledge even for the most limited domain; this indeed would exceed the current state of the art in AI. On the other hand, all the above shortcomings have been solved at some level in one system or another. The challenge for each new ICAI system is to provide a domain knowledge representation rich enough to support the desired level of understanding and therefore flexibility in teaching.

If there is one lesson that thirty years of AI research has taught us, it is that knowledge, rather than general thought mechanisms, is the key to intelligent behavior. Furthermore, the form in which we store this knowledge is crucial to our abilities to use it. Finally, no general form seems suitable for all knowledge. Instead, different types of reasoning and types of knowledge require different representations for efficient and effective use. The bottom line, then, is that intelligent behavior (and thus tutorial capabilities) requires extensive knowledge *structured in such a way as to facilitate the desired types of reasoning and cognitive capabilities.* Our challenge in ICAI is to determine the types of knowledge required, and suitable representations for that knowledge, to support teaching particular subjects. To quote several of the most influential ICAI researchers, "By far the most important issue we have been forced to confront is the overwhelmingly central role of detailed, domain-specific knowledge in governing almost every aspect of the tutorial process" [58].

The structure and interrelationships of knowledge needed for tutoring are much greater than for typical problem-solving expert systems. This important lesson was clearly demonstrated by research at Stanford University in which they tried to use the famous MYCIN expert system [9] as a tutor for medical students. The resulting system, GUIDON [14], was shown to be inadequate in many ways, all resulting from MYCIN's uniform rule-based knowledge representation. Woolf and McDonald summarize: "GUIDON helped demonstrate the need for a cognitive base to the expert module and defined the importance of placing logical and relational abstractions, both of which are important in learning and tutoring, into the expert knowledge base .... The suggestion has been made that the original rules of the MYCIN system represent 'compiled' knowledge devoid of the low-level detail and relations necessary for learning and tutoring" [63]. Bill Clancey, the prime architect of GUIDON, concurs: "Teaching and explanation, we came to recognize, place different demands on an expert than simply solving problems. A teacher can provide analogies, multiple views, and levels of explanation which are unknown to MYCIN. In building MYCIN, we did not make explicit how an expert organizes his knowledge, how he remembers it, and strategies he uses for approaching problems. We need to be able to articulate how the rules fit together, how they are constructed" [16]. A tutoring system, then, needs not only knowledge of its domain, but also the perspective on this knowledge that allows it to convey it to the student.

- Factors: "Do you think the amount of moisture in the air affects the amount of rainfall?"
- Results of Processes: "Condensation is the process by which moisture in the air becomes liquid water again."
- Functional Relations: "What happens to the temperature of the air as it rises?"

Figure 2: Knowledge types in WHY.

The complexity of an ICAI system far exceeds the more typical frame-based CAI techniques. Building a knowledge-based CAI system requires investigation of such questions as "What is an effective taxonomy of question types, from both a semantic and syntactic point of view?", "What classification of errors should be utilized, if it is to apply regardless of the specific subject matter?", and "What different efficient techniques can possibly be defined and used for diagnostic and remedial purposes?" [13]. Although the expert module or domain knowledge of the system does not necessarily have to have the ability to solve problems it poses to the student, it must at least recognize an incorrect answer [63]. Goldstein and Papert [23] suggest the use of a "glass-box" expert that can both solve domain problems and explain its reasoning to the student. Burton and Brown [10], on the other hand, state that a "black-box" expert which solves domain problems in the most efficient (albeit possibly obscure) way may be more efficient than a more complex glass-box, articulate expert. They suggest the combination of a black-box expert, which solves domain problems and compares its answers with the student's to detect errors, with a possibly incomplete glass-box expert, which cannot solve all domain problems, but can solve enough to be able to explain to the student how his solution differs from the ideal. Many such tradeoffs exist, and it would be foolish to think that one best answer applies to all domains.

The choice of knowledge representation in a tutoring system depends on both the type of knowledge to be stored and its intended tutorial use. WHY [55, 58], for instance, is a Socratic tutoring system which teaches the causal relationships and factors affecting rainfall. Stevens et al. [58] list the sorts of knowledge that are represented in WHY along with sentences which illustrate each type; these are shown in Figure 2. The intended use of this knowledge is to test the student's understanding of the major causal factors involved in rainfall. When the student has a misconception with a particular causal factor, they would like to shift their discussion down one level to discuss the more intricate causal relationships at work in that factor. Therefore, their knowledge dictates a representation which emphasizes causality, while their intended teaching strategy necessitates different levels of abstraction from high-level factors to lower-order ones.

The representation which they chose was the *script* [51]; their use of scripts is described in detail in [58]. Their version of the script representation has nodes, which represent processes and events, and links between these nodes which express such relations as X *enables* Y or X *causes* Y (where X and Y are processes or events such as evaporation or condensation). Thus, each script depicts a partially-ordered sequence of processes and events linked by temporal or causal connections. Furthermore, each node can have a hierarchically-embedded subscript which describes the causal relationships at work within it. Finally, these scripts have *roles* (e.g. air-mass or body-of-water) that will be bound to particular geographic or meteorological entities when the script is applied to a particular case (e.g. Ireland). The scripts therefore capture the causalities involved in rainfall.

Their representation not only expresses the causalities of rainfall, but also provides the necessary organization to support their desired teaching strategy. They test the student over the relationships in the top-level script. Missing or erroneous steps cause the tutor to shift down to the appropriate subscript in order to determine why the student doesn't understand the process which that subscript describes. If a student masters steps at one level in the script, the tutor has the choice of moving down to the subscript level for a more detailed examination or proposing a specific case, such as a particular geographic location, and asking the student to apply his general knowledge to this specific case. WHY provides a perfect example of how the domain knowledge and teaching strategies together drive the choice of knowledge representation.

SCHOLAR, on the other hand, demonstrates the needs of a mixed-initiative, fact-oriented tutoring system. SCHOLAR stores its knowledge in a *semantic network* [44], which is essentially a highly-interconnected network of nodes and links representing objects and their properties. Carbonell sees the semantic network representation being used in the following ways: "answering questions not specifically anticipated, constructing questions on given topics, and generally carrying on a mixed-initiative contextual dialog with a human in a rather free and comfortable subset of English" [13]. Obviously, this requires a highly-structured data base in which concepts and facts are connected along many dimensions and in which linguistic information can be embedded. The semantic network knowledge representation fills this need.

Rather than store any specific pieces of text, questions, predicted answers, errors, or branching information, SCHOLAR's contribution was that it generated all this from its semantic network of geographic knowledge [13]. Each unit of the semantic net is organized as an object with properties. Each property consists of the property name (attribute), a set of tags, and the property value. Values themselves can either be properties or pointers into other units. The tags tell the executive program how to interpret the properties and values and thus lend "semantics" to the structure. Furthermore, procedures may be embedded into the network which can use the semantic network to make inferences; an example is a procedure that infers climate from local conditions like latitude and altitude [13]. By organizing the information in such an interconnected way, SCHOLAR avoids storing information redundantly in the data base, and it can index into the information in a variety of ways, thus supporting a very flexible method of query and reasoning.

Jonathan Wexler [61] independently developed a system very similar to SCHOLAR. In Wexler's domain-independent system, knowledge is encoded in an information network and is used to generate questions, statements, and answers. Net objects are grouped into classes and each object includes a value/character string and unlabelled links to other objects. More complex relationships can be built by creating new classes, such as family (which could be used to link mother, father, and children) [61]. Questions, statements, and answers are constructed from the network using prespecified *skeleton patterns*, which are textual templates annotated with information net paths and constraints. Thus, Wexler's information net is similar to SCHOLAR's semantic net, and is used for similar purposes, albeit through different mechanisms.

Reasoning in a semantic net involves a lot of pointer following. In Wexler's system [61], the net is searched (using backtracking) for nodes which satisfy the question skeletons. The search is begun at class nodes and proceeds to the various instances. Carbonell [13] notes that Rockart, who uses a semantic network to represent accounting information [48], "makes an interesting use of the semantic representation to find relations via intersections between two concepts in the data base." SCHOLAR uses a similar intersection search to answer questions like "Is Buenos Aires in Brazil?" SCHOLAR can determine what two things have in common by searching both their SUPERP (superpart) or SUPERC (superconcept) links for an intersection [18]. This allows SCHOLAR to relate a student's answer to the correct one and thus realize that if a city with certain characteristics was asked for, the student is not as far off if his answer is an incorrect city as if he answered with some country [13]. There are many other techniques for information retrieval in a semantic net, but the point is that, by endowing the executive program with knowledge of the semantics of the various links, most of the facilities needed for effective tutoring are reduced to graph search procedures.

Besides representing geographic knowledge, SCHOLAR's semantic net represents other world knowledge and even *meta-knowledge* (knowledge about knowledge). Representing commonsense knowledge such as reasonable temperatures enables the system to recognize ridiculous answers, such as 200 degrees in response to "What is the average temperature in Minnesota?" [13]. In order to reason about the extent of its knowledge, properties are tagged as to whether SCHOLAR has complete or incomplete knowledge of them. For instance, I have complete knowledge of my family members, but incomplete knowledge of all AI researchers. Since Carbonell places great emphasis on determining the contextual relevancy of properties, all properties are tagged with their relevancy to their attached object. For example, the author of a book may be more important than its Library of Congress number. Carbonell notes that it would be nice if the relavancy of one node to another was simply the distance betweeen them in a graph-theoretic sense, but it turned out that solution was not refined enough [13]. Although not included in SCHOLAR, Goldstein [25] suggests that an estimate of cognitive complexity for different tasks should be maintained so the presentation and necessary remediation for that topic can reflect the expected difficulty the student will have; the tutor should know if it is teaching a hard task or an easy one. The goal to which each of these commonsense and meta-knowledge annotations aspires is to provide the tutoring system with the same commonsense world knowledge as its students, but obviously such a rich knowledge base is beyond the state of the art in AI.

The semantic network has a rich history of use in intelligent tutoring and continues in popularity today. Brown et al. [5] discuss the use of a semantic net to store factual knowledge and an augmented finite automata structure for describing causal process relationships. Recently, Woolf [64] used KL-ONE, which is a semantic net representation tool, annotated with pedagogical information about the relative importance of domain topics, for her Ph.D. work on tutorial dialogs. The attraction of semantic networks is that they have been used in many branches of artificial intelligence, including natural language understanding, question answering, problem solving, and learning, and many general-purpose techniques have been developed for their use in knowledge-based reasoning.

Simulation-based training has quite a different set of representational requirements. Obviously, one component of such a system is a simulation, so the challenge is the integration of this simulation with the rest of the tutoring system. The SOPHIE systems teach electronic troubleshooting and hence include a circuit simulator. Brown et al. [6] report that the simulator used in SOPHIE II requires careful use since violation of the assumptions on which it depends could yield unexpected and inaccurate results. To deal with this problem, SOPHIE II uses pieces of code termed *procedural experts* which know how to run the simulator, how to use the results, and what the limitations are [6].

STEAMER [29], the interactive, inspectable, simulation-based tutor described earlier, is connected to a detailed mathematical simulation of a steam propulsion system. The STEAMER graphical editor allows graphics objects to be tied to the underlying simulation. They allow icons to reflect the value or state of various simulation variables and also permit the user to change variables by clicking on these icons. Hollan et al. [29] discuss the difficult problems of this so-called "tapping." In an effort to achieve the *conceptual fidelity* mentioned in Section II, STEAMER also represents operating procedures in a form that allows explanation at different hierarchical levels, and it associates operating procedures with their underlying engineering principles. They report the development of "a frame-based representation system[4] that supports multiple perspectives and permits an integration of the vast amount of structural, functional, topological, and graphical information contained within STEAMER" [29].

SOPHIE III [6], the successor to SOPHIE II referenced above, takes a different approach to simulation than its predecessor. While SOPHIE II used a black-box circuit simulator with procedural experts as the glue between the simulator and tutoring system, SOPHIE III includes specially-designed circuit simulation techniques which are more articulate. Based on propagation of constraints, their circuit reasoning component can not only predict circuit behavior, but it can explain *why* it predicted this behavior. Connections between circuit wires (i.e., a transistor, resistor, or even physical connection) are modeled as constraints on those wires (which are the terminals of the constraint box). By propagating values from known measurements (i.e., current or voltage) through the many constraints in the circuit, SOPHIE III can predict measurements at other points. Furthermore, these inferences can be recorded as they are made so that SOPHIE can justify the predictions to the student [6]. Such an articulate simulation has pedagogical benefits which far exceed a simple black-box simulator.

Besides its use of scripts, WHY [55, 58] also makes use of constraint-based reasoning, in this case to describe functional relationships at work in processes. For instance, a process such as evaporation can be modeled by a constraint prototype with actors (the instance variables or terminals which are being constrained). The constraint can be instantiated and connected to real actors, such as a body of water and a land mass. "Factors" used by the constraints are attributes of the actors, such as temperature and size. Problems can be generated which provide all necessary inputs but one. The constraint can infer the remaining one and compare this with the student's answer. The constraint knows how to describe itself in the abstract and

[4] *Frames* [36] are really very similar to semantic nets, although the two have different conceptualizations. While a semantic net is a graph of nodes and links, a frame encodes the same information in structures with slots. However, since these slots are usually links to other frames (nodes), semantic nets and frames have evolved into largely equivalent data structures, with the difference lying mainly in the mental picture they evoke.

can give examples using its instantiated actors. This provides an articulate inference procedure for causal, factor-oriented concepts.

The paper of Stevens et al. [58], an excellent treatise on the knowledge representation issues in tutoring and one that has greatly influenced this author's understanding, emphasizes the need for "multiple representational viewpoints." Using their WHY system as an example, they point out that scripts serve well to guide the sequencing of topics, but are particularly weak in diagnosing student misconceptions. To overcome this problem, they added a second viewpoint, the above-described *functional* perspective, which exposes the functional relationships between rainfall parameters (e.g., evaporation is a function of a body of water and an air mass, and increases proportionally with increases in the water temperature, the air temperature, and their relative proximity). While scripts are "ordered and sequential," the functional perspective is "non-linear and interactive" [58]. This functional perspective embodies the viewpoint that was lacking with scripts; while the scripts convey the causal sequencing of events, the functional perspective describes the "interaction microstructure" [58]. They note that the availability of multiple viewpoints allows a tutoring system to view student errors as a reflection of some perspective missing in the student's knowledge. With such a multiple-perspective view of knowledge representation, we see the enterprise of knowledge-based CAI in a new light, in terms of the partitioning of knowledge between different viewpoints and the determination of when and how to invoke a particular viewpoint.

Condition/action rules, also known as production rules, have enjoyed a widespread popularity in AI, and this has spread into ICAI. Such rules, which model behavior as "recognize a condition, initiate an action," have long been considered as a possible model of human intelligence and thus serve as the domain knowledge for many different tutoring systems. One example is Goldstein's WUMPUS tutor [25]. This tutor serves as a coach for students playing "Hunt the Wumpus," a game requiring mathematical and logical skills whose goal is to find and kill the deadly Wumpus in a dark cave filled with many perils. The requisite skills are modeled by condition/action rules which choose moves based on probabilistic and logical evidence. When the student makes suboptimal moves, he is postulated to be missing the appropriate rule. More recently, Anderson and Reiser used rules in their respective Geometry [1] and LISP [46] tutors. The attractiveness of rules is their modular nature, since each rule is, ideally, independent of all others and thus rules can be added and deleted with only an incremental effect on system performance. The criticism of rule-based approaches has been that their uniformity masks the underlying abstractions and cognitive models of the domain necessary for teaching [16].

While most ICAI systems revolve around their knowledge representation, some attempts have been made to combine heterogeneous knowledge in a more modular architecture. Bonar et al. [4] describe an object-oriented tutoring architecture in which domain knowledge classes, called "Bites," are organized into a hierarchical inheritance lattice to take advantage of shared structure. Generic classes of Bites include Abstraction Hierarchy Bites, which represent an ordering of concepts in the curriculum; Definition Bites, which are relatively standalone; I/O Bites, which represent black-box (functional) behaviors; and Discovery Bites, which include skills such as scientific inquiry which allow the student to explore an environment with minimal coaching [4]. Abstraction Hierarchy Bites are especially critical in student diagnosis since only they have connection to other related Bites and so can test for understanding of concepts that span several Bites. In contrast with other efforts that separate "explainer," "diagnoser," "tutor," "user modeler," etc. components, an object-oriented architecture provides these facilities as "mix-ins" which can be inherited by domain knowledge classes. There can be several different diagnoser mix-ins, representing different strategies, and different knowledge Bites can include different diagnoser classes. In this way, multiple forms of knowledge can coexist in the same knowledge base, each with its own preferred techniques for presentation, student diagnosis, and problem solving.

This section has emphasized the importance of domain knowledge and its representation in tutoring systems, but the important point is that this knowledge representation is driven by both the domain knowledge to be modeled and, equally important, by the intended use of this knowledge in the tutor. Since the differentiator in an intelligent tutoring system is the flexibility with which it handles student misconceptions, the choice of knowledge representation is driven largely by the needs of the student diagnosis module. This module is the topic of the next two sections.

IV Student Modeling

The student model is simply a model, maintained and used by the computer tutor, of the student's knowledge and capabilities. Its purpose is to evaluate and account for the student's actions and responses. Human tutors do an excellent job of judging the student's answers in the context of his assumed level of understanding and past learning behavior, thus effectively adapting their instruction to the student's competence and abilities. Yet traditional CAI has done a poor job in this area.

The most common technique for student modeling in CAI has been a tally of the student's correct and incorrect answers. Adaptation to the student was limited to branching to different points in the instruction based on the student's answer, his tally exceeding some threshold, or his tally as a percentage exceeding some threshold. Thus, these programs did not keep track of the student's current knowledge except at some coarse level, and what little they did maintain about the student was insufficient for anything beyond grading the student's performance over the session.

Ideally, a tutor should know what the student knows, his level of competence, his past learning behavior (i.e., slow, fast), the presentation methods to which he responds best, and even his areas of expertise and interest beyond the current subject matter. Armed with this information, the tutor can choose a suitable level and method of presentation, it can relate new topics to things with which he is familiar, and it can evaluate his responses in terms of the areas he knows well and those in which he is more likely to have misunderstandings. Although ICAI still has a long way to go in this area, it has made much progress over earlier CAI attempts.

Ira Goldstein of MIT introduced the now widely-used method of *Overlay Modeling*, in which the student's knowledge is viewed in terms of the tutor's domain knowledge; that is, the student model is embedded (conceptually anyway) in the tutor's knowledge [24]. This permits an easy comparison between that which the student knows and that which he should know. Of course, annotating the tutor's knowledge base with student model information implies that there is a knowledge base to annotate, which explains the simpler models used in CAI.

In later papers, Goldstein showed that it may not be sufficient to model the student's knowledge in terms of the coach's mature, general knowledge. He suggested the explicit representation of intermediate forms of knowledge such as specialized rules that the student may discover before he recognizes the general form. Goldstein uses a *genetic graph* exposing relationships between intermediate and mature pieces of knowledge such as analogy links, specialization links, etc. [25, 26].

Different approaches have been taken to the construction of the student model. In Wexler's system [61], the student model is constructed as a semantic net, with nodes and links added as they are taught. Carbonell [13] states that building a student model from scratch is too inaccurate and that the system should instead start with what it thinks the student knows (perhaps an ideal student) and tweak this model as is necessary. This opinion is reflected in SCHOLAR, which, in contrast to Wexler's approach, starts with the expert knowledge base as the student model and annotates deviations that are subsequently discovered [13]. Burton and Brown's WEST tutor [10] typifies a *skill modeler*, which is an overlay modeler in which expert knowledge is represented as skills, and the student is modeled by the set of skills he has mastered (used successfully). They also introduce the notion of *differential modeling* [10], which is similar to Carbonell's method of modeling explicitly the differences between expert and student knowledge. As mentioned earlier, Goldstein's WUMPUS tutor [25] includes intermediate forms of knowledge linked with the expert knowledge in a genetic graph [26] and therefore can model the student not only on his mastery of the expert knowledge but also on his progress toward this mastery. All these methods are variants of the basic overlay modeling paradigm.

Other domains, however, have been shown to defy simple overlay modeling techniques. The research described by Burton [11] clearly indicates that the bizarre nature of children's arithmetic bugs suggests that no subset of the expert's knowledge could explain the incorrect procedures used by novices. Instead, Burton's *Buggy model* employs both correct and "buggy" rules which the student may be following [11]. Understanding a student's error then becomes a task of finding a suitable combination of these correct and buggy rules which together would produce the same incorrect answer as was produced by the student. This approach is different from Goldstein's work on genetic graphs in that buggy rules may be in no special way related to correct rules, but instead may be empirically-determined typical errors.

While ICAI student modeling techniques are designed to overcome the limitations with CAI tallies of right and wrong answers, such tallies are often still embedded within richer student models. Burton and Brown [10] have pointed out the difficulties with keeping such tallies, pointing out that even after a student has mastered an area, this model would continue to reflect his past weaknesses. Pieces of the model should really decay with time, yet the computational costs involved would then be prohibitive. In their WEST tutor, they dealt with this by allowing successful immediate behavior (the last three moves) to clear out past problems.

Student models more complicated than those presented in this section have been developed and tested, although they are beyond the scope of this paper. The interested reader is referred for instance to Sleeman [54]. It should be pointed out that the student model is potentially very open-ended, and its design should therefore be driven by the domain knowledge representation [38, 41] and, more importantly, by the needs of the student diagnosis module, which will be discussed in the next section. In order to diagnose student misconceptions and explain student errors, it is necessary to understand his current state of knowledge and past history. The design of the student model is driven foremost by the level of abstaction needed to make sense of his errors.

V Student Diagnosis

As Schank and Slade have noted, the opportunity to fail is an important part of learning and one which the computer can readily provide without the fear of peer or teacher judgement [52]. But the computer must capitalize on each student failure as an opportunity to correct a misconception. If it merely tells the student that he is wrong, it has done no teaching at all, but has rather pushed the problem off on the student, who must somehow determine why he was wrong. As Brown and Moskowitz note, "... good teachers don't just mark off points. Rather, they try to determine the student's misunderstandings as the best basis for correcting those misunderstandings" [8].

The techniques for student diagnosis are intertwined with student modeling methods; the reader will notice the close connection between topics in section IV and this section. The way to view this connection is that the types of domain knowledge and the desired level of student diagnosis drive the domain knowledge representation, which in turn largely determines the actual student diagnosis techniques, which then determines the student modeling needs and methods.

Student diagnosis is similar to other diagnostic tasks such as equipment or disease diagnosis. Carbonell views student errors as symptoms of diseases (misconceptions) and notes that our diagnostic reasoning should operate on an open set of alternatives [13]. Koffman and Blount [31] similarly suggest that an intelligent tutoring system should make and test hypotheses concerning the source of student errors, considering his past problems. We in fact have a rich base on which to make such hypotheses, including the student model, typical mistakes with the topic we are teaching, the constraints of the current problem, and, hopefully, additional knowledge about the student's answer. The rest of this section will outline several different techniques that have been used.

The experience of Stevens et al. with WHY led them to state perhaps the most important maxim of student diagnosis: "*The types of misconceptions in a student's knowledge that a system can diagnose are heavily dependent on the knowledge represented in the system*" [58]. For instance, student shortcomings in discovery learning environments can only be detected through comparison with a computer-based expert, because you need some way of determining when particular knowledge is applicable [10]. In their WEST tutor, Burton and Brown use an "Issue Recognizer" to detect student shortcomings and an "Issue Evaluator" which uses the student model to decide if the student is really lacking the skill or is using an equally valid strategy. This is based on a differential model of the student's behavior (as compared with the expert's behavior). Such a differential model is effective only when there is a relatively "... unique and well-defined closed set of correct answers ..." [13].

An alternative to differential modeling is direct interpretation of the student's answer in terms of semantic memory, considering the compatibility and relevancy of the answer in relation to the question [13]. Wexler's system [61], for instance, which stores its domain knowledge in a semantic net, evaluates the student's response by searching the net for nodes which satisfy the conditions of the question. If no such node matches the student's answer, it then tries to fit the answer into the question skeleton by successively removing the constraints on the answer. If the resulting skeleton matches anything in the network, it relates this to the student. For instance, if the question was "What is the capital of Texas?" and the student replied "Madison," I would imagine Wexler's system replying "Madison is the capital of Wisconsin." The issue here is giving extra information about the student's incorrect answer and making sure that this information is relevant.

Student diagnosis is not as easy as the above example would suggest, however. Sometimes the student has the correct answer but expresses it in a fundamentally different way [63]. Similarly, in gaming environments, the student may appear to be missing certain skills when instead he is employing a totally different strategy from the expert [10]. Finally, Stevens et al. [58] note that sometimes bugs interact, so we must always consider the possibility that an error resulted from a combination of student misconceptions.

One of the few techniques which handles multiple bugs is Burton's Buggy model [11]. In this model, the system has rules corresponding to both correct domain skills and typical buggy techniques. Applying this model to subtraction, Burton built *procedural networks* [50] of the techniques (both correct and incorrect) needed to perform subtraction. These procedural networks are partially-ordered sequences of arithmetic operations, noting not only when certain skills such as borrowing are needed but also when particular buggy techniques are likely to be used (such as erroneous forms of borrowing). The student's answer is evaluated by searching for a path through this network of skills that yields the student's answer. If the path contains only correct skills, we know which skills he has successfully used. If the path contains buggy skills, we know exactly which misconceptions he has. The problem with this approach is that the number of possible paths through the network grows exponentially with the number of correct and buggy techniques, so that even for a simple domain such as subtraction we must resort to some heuristic search. Also, this method requires an explicit enumeration of possible bugs, which certainly runs counter to Carbonell's opinions.

Similar in spirit to the Buggy model, Genesereth suggests evaluation of student actions through *plan recognition* [21]. Plan recognition is the inverse of planning; while in planning we search for a sequence of operators which will accomplish our goals, building such a plan through top-down refinement of those goals, in plan recognition we search for the goals which might underly the apparent plan corresponding to the student's actions. This is similar to language parsing; domain actions serve as terminals, goals and subgoals serve as non-terminals, and we want to find a parse of the student's actions which will tell us the goals he is following.

Genesereth's system uses a combination of top-down and bottom-up parsing and considers both correct and buggy actions. The difference between plan recognition and the Buggy model is that the former considers more levels of abstraction.

While not everyone subscribes to the Buggy method, most see the need for some sort of error taxonomy. As Stevens et al. [58] so aptly point out, "A good teacher must possess knowledge of the types of misconceptions that arise in the domain being taught". Carbonell [13] suggests a taxonomy of errors, which would allow the tutor to determine the significance of various errors, choose appropriate remediation strategies, and notice patterns of related errors in the student's answers/actions. Misconceptions that can arise in a particular domain should be represented, and their commonalities should be abstracted into more general misconceptions ranging from basic misconceptions that people experience, such as misusing a metaphor, down to the ways in which these misconceptions manifest themselves in particular domain concepts [58].

There have been various approaches to such an error taxonomy, each being somewhat tied to a particular type of knowledge. Many people have distinguished between problems local to the immediate topic, problems involving relationships between various topics or pieces of knowledge, and problems that indicate downright confusion on the part of the student. Carbonell [13] suggests the following generic error types (which, not surprisingly, have strong ties to semantic net representations): (1) missing information, (2) misfiled fact, (3) wrong entry, (4) lack of a concept, (5) wrong superordinate (superpart or superconcept), (6) overgeneralization, (7) failure to draw some superordinate inference, (8) failure to draw some negative inference (i.e., that some information contradicts the rest). Stevens et al. [58] identified sixteen common bugs that accounted for 58% of the errors of students in the area of rainfall (72% ignoring omissions). They discuss the more abstract error types which underly these sixteen bugs, including application of an incorrect metaphor and missing generalizations. The idea of a taxonomy, then, is a useful one; at some level of abstraction, human errors are domain independent, but teachers must also recognize instances of these abstract errors that occur in the domain to be taught.

While most systems treat diagnosis as a separate module, the object-oriented approach of Bonar et al. [4] allows different classes of knowledge to use different diagnosis techniques. Each knowledge class inherits diagnostic capabilities from a particular Diagnoser class. Using these inherited capabilities, a knowledge Bite sends messages to each component Bite asking if that component Bite has knowledge relevant to the student's response, the current tutoring goals, and the current mode. If some component does, the Diagnoser routines then check to see if the student has misused the knowledge contained in this component Bite. Misuse is defined by that component Bite's own Diagnoser class. The Diagnoser then updates the student model, which of course is local to each knowledge Bite, and invokes the Task Selector to choose the next action [4].

It is important not to become so wrapped up in diagnosing the student's misconceptions that we lose sight of our goal. Carbonell [13] reminds us that student diagnosis is a means rather than an end and that the real objective is for the student to overcome his misunderstandings, not for the teacher to magically infer them. If we can narrow down the set of possible misconceptions, it is usually easy to determine through further interaction where the student is confused. Often we can use the Socratic method of posing carefully selected problems to the student in order to allow him to see his own misconceptions. Finally, a little meaningful feedback in the style of Wexler [61] discussed above can go a long way towards helping the student find his error.

VI Pedagogical Knowledge

A good teacher is more than just knowledgeable. The communication of a body of knowledge to a person doesn't simply follow a protocol of information transfer as between two computers. It is guided by strategies and techniques which are selected and combined dynamically in reaction to the student. Therefore, teaching itself can be considered a knowledge-based skill which we can model computationally. In the words of Brown and Moskowitz [8], "... we must formalize the intuitive knowledge that guides good teachers".

Instructional strategies deal with choosing an effective presentation method, determining the balance of student and tutor control, governing the amount and timing of feedback, and choosing evaluation criteria with which to judge student competence [20]. Perhaps the two most important pedagogical decisions are when to interrupt and what to say [10]. There is a distinction between determination of what to say (the content) and the actual choice of words and input/output behavior of the machine. The scope of the tutoring knowledge ends with choosing the content, just as it ends with the choice of a presentation method or an evaluation strategy [63]. Thus the pedagogical knowledge guides the overall goals and strategies of the tutor, but the other modules carry out these intentions.

Although, as outlined above, the pedagogical knowledge is tasked with many responsibilities, the most important is the determination of a strategy to deal with student errors. Since students are seldom consistent, and since diagnosis of one's own misconceptions is often more effective than too much guidance, a tutor cannot just provide remediation whenever the student errs. Woolf and McDonald [63] suggest that the pedagogical component be applied "after the expert and student modules have been accessed and some assessment [has been] made about the level of the student's knowledge." At this point, the tutor must select between ignoring the error, pointing it out, correcting it, or somehow guiding the student towards recognizing and correcting it himself. Obviously this choice must be made based on instructional heuristics.

Most intelligent tutoring systems have represented pedagogical knowledge in the form of production rules. These rules trigger on such things as student errors, recognized situations in the tutorial session, the history of the tutorial session, significant student actions, the nature of the knowledge being taught, and comparisons between student and tutor knowledge. Probably no two systems have adopted the same tutorial philosophy or set of rules, thus pointing out another recognized advantage of using production rules for pedagogical knowledge: the rules can easily be added, deleted, or modified independent of one another (ideally), thus allowing the course author to experiment with various strategies.

In a seminal paper with respect to the development of a theory of teaching, Stevens and Collins [57] discuss their analysis of a number of human tutorial dialogs and present a detailed theory in the form of condition/action rules of the goals, strategies, and priorities that guide an

- "If the student is about to lose interrupt and tutor him only with moves that will keep him from losing" [10].
- "If (1) a student proposes a rule or makes a prediction based on one or more factors that are insufficient, or (2) is entrapped by a rule (ENS 1 or ENS 9) based on one or more factors that are insufficient,
  then (3) pick a case that has the values specified on the insufficient factors, but not the value specified on the dependent variable" [57].
- If "the present topic is complete, but the tutor has little confidence in its assessment of the student's knowledge," then generate "... an expository shift from detailed examination of a single topic, to a shallow examination of a number of topics" [64].
- "IF: The recent context of the dialog mentioned either a 'deeper subgoal' or a factor relevant to the current goal THEN: Define the focus rule to be the d-rule that mentions this focus topic" [14].

Figure 3: Some example pedagogical rules.

effective teacher. In a less detailed way, Burton and Brown [10] list twelve general guidelines that tutoring systems should follow. To give a general feel for the content of various systems' instructional strategies, Figure 3 lists some example rules.

As a further perspective on instructional strategies, we can survey some of the conclusions reached through other research efforts. Often these conclusions depend heavily on the particular learning scenario used; for instance, instructional strategies for discovery learning differ markedly from mixed-initiative tutoring on geography. However, it is interesting to note the similarities across domains and learning scenarios.

Many researchers have investigated the tradeoffs between correcting a student explicitly, trying to entrap him into discovering the error himself, and simply letting him make mistakes and view the consequences. Burton and Brown [10] discuss this point at length, pointing out that "every time the Coach tells the student something, it is robbing him of the opportunity to discover it for himself." Their paper points out the importance of allowing students to observe the results of incorrect decisions. The coach's role, in their view, is to intervene when the student is likely to miss the evidence of his error or misconception. Woolf [65] contrasts her philosophy of letting students explore possible wrong paths with Anderson et al. [1] and Reiser et al. [46], whose Geometry and LISP tutors immediately correct student errors. This immediate feedback is justified by its authors because errors in geometry and LISP are so ambiguous and delayed that much fruitless effort could be wasted on erroneous solution paths. Thus, although immediate feedback is not always desired, it may be chosen for domain or technical reasons.

Besides deciding when to interrupt, pedagogical rules must decide what to say (the content). For instance, tutor responses in the Recovery Boiler Tutor (RBT) [65] fall into three categories: redirect the student, synthesize data (point out relationships), and confirm correct actions. In RBT, instructional condition/action rules based on specific emergencies and student responses are "designed to verify that the student has asked the right questions and has made the correct inferences about the saliency of his data" [65]. Pedagogical rules in WHY fire on the type of knowledge being taught and the type of misconception the student is having and suggest a strategy for exposing this misconception to the student [57]. Rules in Meno-tutor [63, 64] take the tutor off some default dialog path and suggest a different tutorial strategy or tactic. Woolf's thesis work [64] in particular is an attempt at a domain-independent set of pedagogical rules to guide tutorial discourse; this work will be described further in section VII.

Finally, the course author should be able to specify high-level strategies for ordering topics. For instance, a teacher should be able to tell the tutor to teach about a phylum, then its classes, then the class species, and finally the characteristics of each species [61]. Other teachers may prefer a different ordering, and so the tutoring system should give each teacher access to these ordering heuristics.

As has been stated, the general purpose of pedagogical rules is to indicate discourse directions, strategies, and tactics. In the next section, the more global problem of discourse management, of which pedagogical rules are a component, will be considered.

VII Discourse Management

As Carbonell [13] so elegantly put it, "Man-computer interaction is basically a communication between two information structures, including their computational abilities". Without a formal protocol, this communication must be driven by the goals and needs of both parties. Each side must not only communicate its knowledge, but must reactively structure its dialog to meet the needs and goals of the other. Effective communication lies not in the words, but rather in the determination of what the tutor and student need to communicate [63].

Stevens and Collins [56] argue further that a good human tutor doesn't follow a prespecified syllabus, but instead lets the responses and misconceptions of the student drive the dialog. The tutor should "opportunistically adapt material to the needs of the dialog" [18]. While traditional CAI efforts notoriously hardcoded the syllabus (allowing prespecified branches based on predicted student responses), this problem of discourse management and topic selection has been addressed at some level by most ICAI efforts. Flexibility in the tutorial discourse is a hallmark of ICAI.

In SCHOLAR, the executive program that uses the semantic network to generate teaching material and questions is almost independent of the subject matter in the semantic net [13]. Carbonell [13] discusses the issue of "contextual continuity" and topic coverage, noting that traditional "ad-hoc frame-oriented" (AFO) systems have a large degree of anticipation (they know what they are going to do next), while SCHOLAR has almost none (it is very reactive). He describes SCHOLAR's use of a simple push-down list (stack) of contexts, which it builds and maintains dynamically, to maintain contextual continuity, but admits that more anticipation may be desirable and could be implemented by a straightforward use of deterministic strategies for material presentation and question generation, although he prefers the more reactive system.

As ICAI research developed, more sophisticated discourse control schemes were utilized. Koffman and Blount [31] briefly describe a control scheme based around a student model and a concept tree of prerequisites. The system, based on the student model, determines whether the student prefers to advance quickly to new material or build a solid foundation on current material. Based on the student model, the system "decides at which plateau of the concept tree the student should be working" [31]. The concepts on this plateau are evaluated with respect to recency of use, the student's progression, his current knowledge, the impact of the concept on his current knowledge, and relevance to other concepts. The concept which scores the highest by this criteria is selected, a problem is generated based on his experience level, and the solution is evaluated.

A similar idea was developed further by researchers at Stanford University and resulted in the notion of a *curriculum information network*, used in the Stanford BIP system [3], in which skills to be taught are related to tasks that exercise those skills. The curriculum net is organized as a large set of problems (in their case, BASIC programming problems) indexed in terms of the generic skills they require. Thus the curriculum net also serves as a student model; when the student successfully solves a problem, they know which skills he has used and can therefore choose a new problem with an appropriate set of requisite skills. In this way, although the problems are canned, they are selected and sequenced reactively based on student successes and difficulties. In a second-generation BIP system, known as BIP-II, the curriculum net was generalized into a semantic network of skills, each interconnected to other skills via relational links which denote prerequisites, relative difficulty, and analogies, as well as taxonomic links such as component and kind [60]. They also developed a sophisticated task selection procedure which uses the various links to guide the sequencing and selection of problems for the student. Note the similarities between this new curriculum net and Goldstein's genetic graph.

In [55], Stevens and Collins present a theory of the goal structure of a Socratic tutor based on an analysis of human tutorial dialogs. This tutorial control structure has four parts: (1) an agenda of goals and subgoals, (2) priority rules for adding goals and subgoals to the agenda, (3) strategies for selecting cases which satisfy the goals, and (4) the student model. They argue that human tutors have a few initial goals for the tutorial session and that further pedagogical goals and subgoals are added to the agenda dynamically in reaction to the student. Although they acknowledge the possibility of inserting goals at arbitrary places in the agenda, they note that most tutorial discourse can be seen as a pushing and popping of goals and subgoals onto a push-down stack. Their priority rules determine the order in which to add new goals to the agenda when several are uncovered simultaneously, and their strategies are used to select examples with which to teach certain goals. Their work is significant because of the experimental data (human tutorial dialogs) on which it is based.

The control structure used in the object-oriented architecture of Bonar et al. [4] uses a similar scheme. They have a stack on which TutoringMode objects are stored. Each TutoringMode object includes a satisfaction criteria and a floundering threshold. The satisfaction criteria determines when the TutoringMode object can be popped, while the floundering threshold determines when a remedial object should be pushed onto the stack. They currently have six modes, including exploration, experimentation, elaboration, didactic, demonstration, and coaching [4].

TDUS [47], an acronym for Task-oriented Dialog Understanding System, is a natural language understanding system built at SRI International as part of research into the knowledge structures and reasoning mechanisms needed to guide a human apprentice through a task. Their thesis was that knowledge of the human's current progress through the task would allow them to disambiguate and therefore understand his questions and comments. Their basic task representation is the procedural network [50], which models the task as a partial order of actions organized into a hierarchy of goals. They also represent the objects which are associated with actions, such as tools or the target of the action. With this representation, they not only have a way of guiding the advisory discourse (through the task), but they also have a way of recording the apprentice's progress through the task, inferring actions not explicitly mentioned, and understanding his natural language utterances in terms of the current context.

The most recent work in tutorial discourse management was performed by Beverly Woolf as her Ph.D. thesis [64]. She uses a *discourse management network* (DMN) to guide the tutorial session, plan pedagogical strategies and tactics, and determine natural language utterances. The discourse management network has three successive levels: the tutoring approach, the strategy, and the tactics. These are organized into states of an *augmented transition network* (ATN) [62], with additional meta-rules which can affect the transitions [63].

In a nutshell, an ATN is a finite state machine in which nodes and arcs can store information in global registers for later use, and arcs are dynamically activated or deactivated based on these register values and other input. ATN's were popularized by work in natural language understanding in which the transition network is used to parse natural language sentences and semantic information can be incrementally added to the registers during the parse to further constrain later transitions. ATN's are also hierarchical in that a particular node may itself consist of another ATN which must be traversed before continuing the higher-level ATN. Besides natural language parsing, ATN's have been used in many other contexts including natural language generation [34] and plan recognition in a programming tutor [33].

Woolf's DMN is isomorphic to an ATN but is conceptually a top-down refinement of tutorial goals through the strategies and tactics which implement them. At the top level, states represent the basic tutoring approach such as "introduce" or "tutor." These states lead to refinement states that specify the strategy, such as "explore competency" or "repair misconception." Strategy states are further refined through tactical states such as "teach specific knowledge", "evaluate input", or "suggest new topic." The current tutoring mode is a state in this DMN and determines the next action of the tutor. Transitions through the states follow a default path, but pedagogical rules fire off the student model, current mode, and student responses to lift the tutor off its default path and onto some other state in the network, thus creating a discourse transition [63]. These transitions could be between states at the same level, such as two tactical states, or they could lift the tutor from its current level up to a higher level, effecting a more radical shift in the tutorial directions. Woolf points out the absence of any need for a goal agenda, as used by Stevens and Collins [55], and suggests that her approach more closely mirrors the behavior of human tutors [64]. She also claims that her DMN is relatively domain independent, although she acknowledges that it may need some further refinement. Figure 4 shows her discourse

Figure 4: Woolf's discourse management network.

management network.

Work in discourse management for tutorial purposes has thus followed several distinct transitions. In a backlash against the very deterministic branching structure of early CAI systems, Carbonell's SCHOLAR [13] included a very loose organization of tutorial goals and basically just reacted to the student. Work by Stevens and Collins [55] showed that human tutors maintain an agenda of goals and that the challenge is to add and remove goals from this agenda in a last-in-first-out (LIFO) manner. Other work, exemplified by BIP [3], showed the utility of a richly structured knowledge of the course curriculum which can be used to dynamically sequence the domain topics. Finally, Woolf's work [64] returns us to a prespecified network of tutorial states, but now the network is a domain-independent model of discourse knowledge rather than the earlier-used sequences of domain topics. This trend indicates continuing realization that more and more knowledge is necessary for tutoring, ranging from the pedagogical strategies of teachers, through their rich knowledge of the course topics and their interrelationships, right down to commonsense knowledge about discourse.

VIII Problem Generation

Early CAI efforts included canned problems with canned answers. There are several difficulties with this approach. First, it places a large burden on the course author to develop a broad set of problems on which the tutoring system can draw. Second, it is very difficult to match canned problems to student needs and abilities and impossible to customize the problems to fit the situation. Finally, canned answers don't allow meaningful feedback on student errors unless the error can be anticipated, and justifications for the answer and explanations must also be canned. Thus, canned problems and answers reduce the flexibility of the system to exactly those interactions that were anticipated and prespecified by the course author.

Early attempts to correct these problems were called *generative* CAI due to the programs' abilities to generate new problems. Although the ideas developed in this period have been of further use, the techniques were usually limited to simple drill and practice in arithmetic and language vocabulary and were rarely adapted to the student's learning needs [40]. While early attempts at problem generation substituted values for variables in canned question templates, later efforts generated problems from deeper structures, yielding a richer mix of problems which were potentially better suited to the student's needs [32]. As work progressed, people noticed the applicability of AI techniques and the suitability of rich knowledge representations as a base from which to both generate and solve problems.

Problem generation can be viewed as a tree-structured decision process in which each level represents another decision on what to include in the problem, each branch represents one alternative, and the branches can be augmented with probabilities based on the student's competence and prior experience [31]. Koffman and Blount [31] go into some detail on such a generative scheme, pointing out the need to avoid regenerating problems the student has solved. They discuss adaptation to the student as a process of determining at which plateau of the concept tree the student should be working and the effect of this decision on problem generation. They also make the interesting observation that the tutoring system should solve subproblems for the student that he has already solved before, thus allowing him to concentrate on those aspects of the problem from which he can derive the greatest benefit.

Ramani and Newell [45] provide a more thorough treatment of problem generation. They suggest that one must first identify and analyze the various sets of problems in the domain and then develop suitable structures from which to generate these sets. Like Koffman, they view the basic control structure of problem generation as a tree-structured decision process and, like Koffman, they show how a context-free grammar can be used to implement this technique. In essence, each choice shifts control to the part of the grammar that includes alternatives for lower-level choices. The limitations of this approach, they note, are that the representation is not rich enough to generate broad but meaningful classes of problems and that even simple holding variables which record choices in order to constrain later choices are not possible. Too much information must be specified in the grammar itself, which is not the right level of abstraction [45].

They sum up the problems with generative grammars by explaining that problem generation needs semantic information, while grammars bundle syntax and semantics all into syntactic structures. They then shift their analysis of the problem to view it as a design procedure, in which decisions must be made, with some decisions independent and some dependent on other decisions. While this view still has a basic tree-like decision process, it can operate on a richer knowledge base such as a semantic net, it allows some decisions to refer to earlier decisions, and it generalizes the criteria for decisions to include the rich knowledge base and the student model. While generative grammars are driven by top-down refinement of choices, this new view on problem generation is more constraint based, a paradigm that was later to have profound influence on planning and design research in AI.

The basic techniques of Ramani and Newell's [45] problem generation theory are relatively simple. The underlying data structure is a semantic net which encodes the types of manipulable objects (i.e., numbers, symbols, sets, and sequences) and relevant attributes of these objects. For each class of problems, there is a generative procedure whose job is to fill in the particulars of the problem (i.e., the slots). Slots can be filled through various means; a slot could be a random or probabilistic selection from a list of alternatives or class of objects, it could be a function of other slot values (to implement relationship constraints among slots), it could be a function of external data such as the student model, or there could be hierarchical relationships between candidate values which allow increasing difficulty with time. By suitably structuring the semantic net of candidate objects and classes, much of the above functionality becomes simple retrieval; for instance, we could define many classes of objects organized into a taxonomy, and this semantic net of objects can include annotations of previously-used values as well as embedded functions which reference the student model or other objects. Text generation routines can be associated with problem classes and can "know" how to word a problem given the slot values. Thus, a richly-structured knowledge base again gives us the desired flexibility (if the reader is noticing that this is a recurrent theme of this paper, he is correct).

Of course, if new problems can be generated, they must also be solved, since answers cannot be canned if the problems aren't. This is where the domain knowledge comes into play and serves as a sort of expert system operating on the representation of the problem. Basically, some executive module must take the problem data structure as input and reason from the domain knowledge to produce the corresponding answer. The techniques mentioned in Section III are all relevant.

As one last observation linking problem generation, expert problem solving, and student diagnosis, consider that problems can be viewed as a set of constraints on their solution. If the generated problems in an ICAI system are expressed directly in terms of such constraints, we can easily evaluate student answers by checking that all constraints are satisfied. In this manner, we need not even have the ability to solve the problems ourselves, because we not only have an easy way of validating the student's response, but we can also give him meaningful feedback on wrong answers by telling him which constraints he failed to satisfy.

IX User Interface

Although most of an intelligent tutoring system can be considered as user interface, it is useful to distinguish those aspects that are explicitly concerned with communication with the student. This includes the actual presentation of text and graphics as well as acceptance of user input. Because this topic is so broad, I won't even attempt to provide a survey, but instead will present a few representative examples.

As was stated earlier, one study reports that people retain about twenty-five percent of what they hear, forty-five percent of what they see and hear, and seventy percent of what they see, hear, and do [17]. This is a strong argument for user interfaces that include not only text but also visually exciting graphics and the ability for the user to interact with the graphical environment as though he were a part of it. The authors of the STEAMER paper [29] feel that graphical depiction of causal relationships and topology are very important and that some concepts can be elucidated through graphical animation which defy textual explanation. They also give the student the ability to control the steam propulsion plant by touching icons on the screen. RBT [65] similarly uses graphical animation of its process, a recovery boiler, including visual clues such as a darkened smelt bed, acoustic clues such as ringing alarm buzzers, and textual help and explanations. The user of RBT responds to process alarms by selecting actions from menus and watching their real-time effect on the screen. Although the process simulation runs in real time, students are allowed to stop the simulation in order to ask questions, a feature that many users mentioned that they like [65]. The more a student feels connected to his learning environment, the richer his experience will be, and interactive graphics help with this connection.

Since communication with the student is primarily through English, the system must have the abilities to both understand the student's response and generate text of its own. Many different approaches have been taken for text generation in tutoring systems, most of which avoid true natural language mechanisms. In Woolf's RBT, tutor output is created by extracting text from emergency-specific text files which are loaded when the emergency is invoked [65]. In WEST, textual explanations are attached to each issue (skill); each issue is responsible for presenting a few lines of text explaining itself [10]. WEST therefore distributes prestored text to the actual knowledge structures and combines the text dynamically to fit the situation. In Woolf's Meno-tutor, a description of the content of the textual output is constructed and sent to a surface language generator, although she dummied up the surface language generator for her thesis work [64]. In SOPHIE, each concept has an associated grammar rule that provides alternate ways of expressing the concept in terms of other concepts [18]. Thus, none of these systems generates English from deep linguistic knowledge.

One of the few tutoring systems to incorporate rich natural language generation mechanisms is SCHOLAR. Carbonell separated question generation into two distinct levels: semantic and syntactic. Semantic information comes from the semantic net and determines what is being asked. For syntax, Carbonell identified classes of questions, such as what, which, where, who, and how many questions, fill-in questions, multiple-choice questions, and true-false questions. Each class contains relevant syntactical information. For statements, Carbonell's strategy was to make use of short sentences with simple syntax and a limited repertoire of verbs, mostly variant forms of the verb "to be." All knowledge in the semantic net contained linguistic annotations such as its part of speech. Thus, no text is canned in SCHOLAR, and all sentences and questions are processed completely from the semantic internal representation into English [13].

Another approach to text generation, intended for use in explanation and tutoring, was recently developed by Perry Miller [34]. His PROSENET is an ATN-based approach to text generation. Each arc in the ATN has three components. The action component can be push, pop, jump, a prose fragment to output, a function call which may generate text, an option, a sequence of arcs, or an endsequence, which terminates a sequence. The other two components are a destination state and an action routine which is invoked whenever this arc is traversed. The system places the information to be conveyed into global registers, and the appropriate PROSENET network is activated. To maintain coherency in the text, sentence-level, paragraph-level, and global constraints may be introduced as the text is generated [34].

Natural language parsing is another difficult area; even a survey of this area is beyond the scope of this paper. Few systems, however, have allowed true natural language input. SCHOLAR included rich natural language facilities that allowed it to understand most student questions and answers. SOPHIE used a technique called *semantic grammars*, in which the basic idea is to look for understandable fragments in the input rather than parse the entire sentence [6]. Most other systems have bypassed the natural language problem by using graphical or menu-based input.

It is worthy to note that experts in the field of intelligent tutoring have labeled natural language the "Achilles' heel" of potential tutoring systems [2, 28], and warn against its use because of its potential to consume project resources. However, it is often difficult to provide adequately-flexible interactions with the student without some level of natural language.

The bottom line on user interface in intelligent tutoring systems is that input should be robust, allowing flexibility on the part of the student, and output should be rich, appealing to as many of the student's senses as possible. Future interfaces will likely be superior to today's due to technological advances. The use of varied media (i.e., mixing speech, graphics, text, sound, video disk, etc.) will someday be the norm.

X Summary

As we have seen, the design of an intelligent tutoring system is complex, with many issues to consider and still more that remain topics for research. It all begins with the choice of a good learning scenario which is appropriate to the domain knowledge. This learning scenario, along with the nature of the domain knowledge and the types of misconceptions we must detect, determines the necessary functionality and hence structure of the knowledge representation. The knowledge representation has a profound influence on the explanatory capabilities of the tutor as well as its competence in student diagnosis. In order to customize instruction to the student's abilities and determine the cause of his errors, we must maintain a student model. Finally, both pedagogical knowledge and knowledge of discourse patterns are needed, and we must wrap it all up in a sensory-stimulating user interface to achieve maximal tutorial efficacy.

This survey has provided a springboard for those interested in understanding or designing ICAI systems. Much research has been done and many important lessons have been learned, but much remains. While this paper has provided a tutorial survey, it also acts as an index into the literature for more detailed studies of particular techniques. ICAI has produced some exciting results, but the real excitement lies in the future.

XI Acknowledgement

I would like to thank the following people for their help: Rick Herrod and Don Candy for making this research possible; Ellen Rickel and Mike Smith for their proofreading; William Pervin, Fillia Makedon, and Ivor Page for reading an earlier version and suggesting that I publish it; Dan Russell and Allan Collins for improving the accuracy of my references; the anonymous referees for their excellent comments; and my wonderful wife Rickie for being such a wonderful wife.

References

[1] J.R. Anderson, C.F. Boyle, and G. Yost, "The Geometry Tutor," *Proc. IJCAI-85*, Los Angeles, CA, pp. 1-7, August 1985.

[2] J.R. Anderson, "The Expert Module," *AFHRL Research Planning Forum for Intelligent Tutoring Systems*, San Antonio, TX, September 1986.

[3] A. Barr, M. Beard, and R.C. Atkinson, "The Computer as a Tutorial Laboratory: The Stanford BIP Project," *International Journal of Man-Machine Studies*, Vol. 8, pp. 567-596, 1976.

[4] J. Bonar, R. Cunningham, and J. Schultz, "An Object-Oriented Architecture for Intelligent Tutoring Systems," *Proc. of OOPSLA-86*, Portland, OR, September 1986, pp. 269-276.

[5] J.S. Brown, R.R. Burton, and F. Zydbel, "A Model-Driven Question-Answering System for Mixed-Initiative Computer-Assisted Instruction," *IEEE Trans. Systems, Man, and Cybernetics*, Vol. 3, No. 3, pp. 248-257, 1973.

[6] J.S. Brown, R.R. Burton, and J. deKleer, "Pedagogical, Natural Language and Knowledge Engineering Techniques in SOPHIE I, II and III," in *Intelligent Tutoring Systems* (eds. Sleeman and Brown). Cambridge, MA: Academic Press, 1982, pp. 227-282.

[7] J.S. Brown, R.R. Burton, and W.J. Clancey, "Applications of AI to Training and Education," AAAI-84 Tutorial No. 2, Austin, TX, 1984.

[8] J.S. Brown and Gary Moskowitz, "AI: Windows of Opportunity in Office Automation," Xerox Corporation, July 1985.

[9] B.G. Buchanan and E.H. Shortliffe, Eds., *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Reading, MA: Addison-Wesley, 1984.

[10] R.R. Burton and J.S. Brown, "An Investigation of Computer Coaching for Informal Learning Activities," in *Intelligent Tutoring Systems* (eds. Sleeman and Brown). Cambridge, MA: Academic Press, 1982, pp. 79-98.

[11] R.R. Burton, "Diagnosing Bugs in a Simple Procedural Skill," in *Intelligent Tutoring Systems* (eds. Sleeman and Brown). Cambridge, MA: Academic Press, 1982, pp. 157-183.

[12] R.R. Burton, "The Environment Module of Intelligent Tutoring Systems," *AFHRL Research Planning Forum for Intelligent Tutoring Systems*, San Antonio, TX, September 1986.

[13] J.R. Carbonell, "AI in CAI: An Artificial Intelligence Approach to Computer-Assisted Instruction," *IEEE Trans. Man-Machine Systems*, Vol. 11, No. 4, pp. 190-202, December 1970.

[14] W.J. Clancey, "Tutoring," in *Rule-Based Expert Systems* (eds. Buchanan and Shortliffe). Reading, MA: Addison-Wesley, 1984, Part 8.

[15] W.J. Clancey, "From Guidon to Neomycin and Heracles in Twenty Short Lessons," *AI Magazine*, Vol. VII, No. 3, pp. 40-60, Conference 1986.

[16] W.J. Clancey, "Methodology for Building an Intelligent Tutoring System," in *Artificial Intelligence and Instruction* (ed. Kearsley). Reading, MA: Addison-Wesley, 1987.

[17] M. Edwards, "The Mercedes Benz of Interactive Video," *Hardcopy*, Vol. 14, No. 5, pp. 74-80, May 1985.

[18] E.A. Feigenbaum and A. Barr, Eds., *The Handbook of Artificial Intelligence*, Vol. 2. Los Altos, CA: William Kaufmann Inc., 1982, Chapter 9.

[19] J.D. Fletcher, "Intelligent Instructional Systems in Training," in *Applications in Artificial Intelligence* (ed. Andriole). Princeton, NJ: Petrocelli, 1984.

[20] R.S. Freedman and J.P. Rosenking, "Designing Computer-Based Training Systems: OBIE-1:KNOBE," *IEEE Expert*, pp. 31-38, Summer 1986.

[21] M.R. Genesereth, "The Role of Plans in Intelligent Teaching Systems," in *Intelligent Tutoring Systems* (eds. Sleeman and Brown). Cambridge, MA: Academic Press, 1982, pp. 137-155.

[22] I.P. Goldstein, "The Computer as Coach: An Athletic Paradigm for Intellectual Education," AI Memo 389, MIT, Cambridge, MA, 1976.

[23] I.P. Goldstein and S. Papert, "Artificial Intelligence, Language, and the Study of Knowledge," *Cognitive Science*, Vol. 1, No. 1, pp. 84-123, 1977. Also AI Memo 337, MIT, Cambridge, MA, March 1976.

[24] I.P. Goldstein, "Overlays: A Theory of Modelling for Computer Aided Instruction," AI Memo 406, MIT, Cambridge, MA, February 1977.

[25] I.P. Goldstein, "Developing a Computational Representation for Problem Solving Skills," AI Memo 495, MIT, Cambridge, MA, October 1978.

[26] I.P. Goldstein, "The Genetic Graph: A Representation for the Evolution of Procedural Knowledge," in *Intelligent Tutoring Systems* (eds. Sleeman and Brown). Cambridge, MA: Academic Press, 1982, pp. 51-77.

[27] R. Guindon and P. Sladky, "Analysis of User-Expert Dialogues: Task Networks, Subdialogue Boundary Markers and Antecedent Distribution," MCC Technical Report Number HI-084-85, Microelectronics and Computer Technology Corporation, Austin, TX, December 1985.

[28] H.M. Halff, "Curriculum and Instruction in Automated Tutors," *AFHRL Research Planning Forum for Intelligent Tutoring Systems*, San Antonio, TX, September 1986.

[29] J.D. Hollan, E.L. Hutchins, and L. Weitzman, "STEAMER: An Interactive Inspectable Simulation-Based Training System," *AI Magazine*, Vol. 5, No. 2, pp. 15-27, Summer 1984.

[30] W.B. Johnson, "Pragmatic Considerations in Development and Implementation of Intelligent Tutoring Systems," *AFHRL Research Planning Forum for Intelligent Tutoring Systems*, San Antonio, TX, September 1986.

[31] E.B. Koffman and S.E. Blount, "Artificial Intelligence and Automatic Programming in CAI," *Artificial Intelligence*, Vol. 6, pp. 215-234, 1975.

[32] E.B. Koffman and J.M. Perry, "A Model for Generative CAI and Concept Selection," *International Journal of Man-Machine Studies*, Vol. 8, pp. 397-410.

[33] M.L. Miller and I.P. Goldstein, "PAZATN: A Linguistic Approach to Automatic Analysis of Elementary Programming Protocols," AI Memo 388, MIT, Cambridge, MA, December 1976.

[34] P.L. Miller, *A Critiquing Approach to Expert Computer Advise: ATTENDING.* Boston, MA: Pitman Advanced Publishing Program, 1984.

[35] J.R. Miller, "Human-Computer Interaction and Intelligent Tutoring Systems," *AFHRL Research Planning Forum for Intelligent Tutoring Systems*, San Antonio, TX, September 1986. Also MCC Technical Report HI-284-86, Microelectronics and Computer Technology Corporation, Austin, TX, September 1986.

[36] M. Minsky, "A Framework for Representing Knowledge," in *The Psychology of Computer Vision* (ed. Patrick Winston). New York: McGraw-Hill, 1975. Also in *Readings in Knowledge Representation* (eds. Brachman and Levesque). Los Altos, CA: Morgan Kaufmann, 1985.

[37] T. Moran, D.M. Russell, D.S. Jordan, F. Halasz, J. Orr, and J. Shrager, "ARI Project Phase 1 Report: Problems in Training and Procedure-Oriented Job Performance Aids," Xerox PARC Internal Report, Intelligent Systems Lab, Palo Alto, CA, October 1984.

[38] L.H. Nawrocki, "Artificial Intelligence Applications to Maintenance Training," in *Artificial Intelligence and Instruction* (ed. Kearsley). Reading, MA: Addison-Wesley, 1987.

[39] Seymour Papert, *Mindstorms*. New York: Basic Books, 1980.

[40] O.C. Park, R.S. Perez, and R.J. Seidel, "Intelligent CAI: Old Wine in New Bottles, or a New Vintage?," in *Artificial Intelligence and Instruction* (ed. Kearsley). Reading, MA: Addison-Wesley, 1987.

[41] P.L. Pirolli and J.G. Greeno, "The Problem Space of Instructional Design," in *Intelligent Tutoring Systems: Lessons Learned* (eds. Psotka, Massey, and Mutter). Hillsdale, NJ: Lawrence Erlbaum Associates, 1988.

[42] Plato, *Laches, Protagoras, Meno, and Euthydemus* (trans. Lamb). Cambridge, MA: Harvard University Press, 1924.

[43] J. Psotka, D.B. Pliske, and W.D. Gray, "Intelligent Design Environments and Assistant Systems," *Proc. Expert Systems in Government*, October 1985.

[44] M.R. Quillian, "Semantic Memory," Report AFCRL-66-189, Bolt Beranek and Newman Inc., Cambridge, MA, 1966.

[45] S. Ramani and A. Newell, "On the Generation of Problems," SDL 348, Carnegie Mellon University, Pittsburgh, PA, November 1973.

[46] B.J. Reiser, J.R. Anderson, and R.G. Farrell, "Dynamic Student Modelling in an Intelligent Tutor for LISP Programming," *Proc. IJCAI-85*, Los Angeles, CA, pp. 8-14, August 1985.

[47] A.E. Robinson, D.E. Appelt, B.J. Grosz, G.G. Hendrix, and J.J. Robinson, "Interpreting Natural-Language Utterances in Dialogs About Tasks," SRI Memo 210, SRI International, Menlo Park, CA, March 1980.

[48] J.F. Rockart, M.S. Morton, and Z.S. Zannetos, "Associative Learning Project – Phase-1 System," Working Paper, Sloan School of Management, MIT, Cambridge, MA, January 1970.

[49] D.M. Russell, T.P. Moran, and D.S. Jordan, "The Instructional Design Environment," in *Intelligent Tutoring Systems: Lessons Learned* (eds. Psotka, Massey, and Mutter). Hillsdale, NJ: Lawrence Erlbaum Associates, 1988.

[50] E. Sacerdoti, *A Structure for Plans and Behavior*. New York: Elsevier North-Holland, 1977. Also SRI Memo 109, SRI International, August 1975.

[51] R.C. Schank and R. Abelson, *Scripts, Plans, Goals, and Understanding*. Hillsdale, NJ: Lawrence Erlbaum Associates, 1977.

[52] R.C. Schank and S. Slade, "Education and Computers: An AI Perspective," Report 431, Yale University, October 1985.

[53] J. Shrager, D. Jordan, T. Moran, G. Kizcales, and D.M. Russell, "Pragmatic Issues in Qualitative Modelling: Lessons Learned from Modelling Xerography," *Communications of the ACM*, Vol. 30, No. 12, December 1987.

[54] D.H. Sleeman, "Assessing Aspects of Competence in Basic Algebra," in *Intelligent Tutoring Systems* (eds. Sleeman and Brown). Cambridge, MA: Academic Press, 1982, pp. 185-199.

[55] A. Stevens and A. Collins, "The Goal Structure of a Socratic Tutor," BBN Report 3518, Bolt Beranek and Newman Inc., Cambridge, MA, 1977.

[56] A. Stevens and A. Collins, "Diagnosing Student's Misconceptions in Causal Models," BBN Report 3786, Bolt Beranek and Newman Inc., Cambridge, MA, 1978.

[57] A. Collins and A.L. Stevens, "Goals and Strategies of Inquiry Teachers," in *Advances in Instructional Psychology II* (ed. Glaser). Hillsdale, NJ: Lawrence Erlbaum Associates, 1982, pp. 65-119.

[58] A. Stevens, A. Collins, and S. Goldin, "Misconceptions in Students' Understanding," in *Intelligent Tutoring Systems* (eds. Sleeman and Brown). Cambridge, MA: Academic Press, 1982, pp. 13-24.

[59] K. VanLehn, "Student Modelling in Intelligent Teaching Systems," *AFHRL Research Planning Forum for Intelligent Tutoring Systems*, San Antonio, TX, September 1986.

[60] K.T. Wescourt, M. Beard, and L. Gould, "Knowledge-Based Adaptive Curriculum Sequencing for CAI: Application of a Network Representation," *Proc. of ACM-77*, Seattle, WA, pp. 234-240, 1977.

[61] J.D. Wexler, "Information Networks in Generative Computer-Assisted Instruction," *IEEE Trans. Man-Machine Systems*, Vol. 11, No. 4, pp. 181-190, December 1970.

[62] W.A. Woods, "An Experimental Parsing System for Transition Network Grammars," in *Natural Language Processing* (ed. Rustin). New York: Algorithmics Press, 1972, pp. 113-154.

[63] B. Woolf and D.D. McDonald, "Building a Computer Tutor: Design Issues," *IEEE Computer*, pp. 61-73, September 1984.

[64] B. Woolf, "Context-Dependent Planning in a Machine Tutor," Ph.D. dissertation, Dept. of Computer and Information Science, University of Massachusetts, Amherst, MA, 1984.

[65] B. Woolf, D. Blegan, J.H. Jansen, and A. Verloop, "Teaching a Complex Industrial Process," *Proc. AAAI-86*, Philadelphia, PA, pp. 722-728, August 1986.