A Survey of Artificial Intelligence in Computer-assisted Instruction

Avron Barr
Institute for Mathematical Studies in the Social Sciences
Stanford University

A review of current research in the use of AI techniques in computer-assisted instruction: student-machine dialogue in English, incisive analysis of student's solutions and errors, and the modelling of curriculum structure and student cognitive processes. Implications for CAI, author languages and the authoring process in general are discussed.

Computer technology made an early debut in the education arena, with promises of making a private tutor with great wisdom and infinite patience available to all students. However, the computer was originally used as a text presentation device which could grade student's responses to questions, primarily because that was all it was capable of doing in the early sixties. The software advances that
would make the machine a viable teacher were slow in coming, and their development has closely followed the growth of the field called Artificial Intelligence, which has had a similar struggle to match high expectations of performance over its 20 year history. This paper reviews the development of "tutorial" CAI and describes some of the most interesting examples yet created of powerful machine tutors.
I. The Development of the Use of AI Techniques in CAI

There are three general types of application that computers have found in education, so far. The "ad lib" or "environmental approach" typified by Papert's LOGO laboratory (1970) allows students more or less free-style use of the machine; they are involved in programming and learning takes place as a side effect. The second category is the use of games and simulation as an instructional tool, where once again the student is involved in an activity, playing a game, for which learning is an expected side effect. The final computer application in education is computer-assisted instruction (CAI), which makes an explicit attempt to instigate and control learning (Howe, 1973). It is this last use of computer technology that this paper deals with.

The first instructional programs took many forms, but all followed essentially the same pedagogical philosophy. The student was usually given some instructional text (sometimes this was done off-line) asked a question requiring a brief answer, and then told whether his answer was right or wrong. The student's response was sometimes used to determine his path through the curriculum, or the sequence of problems he was given (see Atkinson & Wilson, 1969); either the program branched to remedial material if the student made an error, or in more sophisticated software, the student's wrong responses were anticipated by the courseware author, who could specify branches to material on the
basis of why he thought the student would have made the error he did. Branching on the basis of the response was the first step toward individualization of instruction.

This style of CAI has been dubbed ad-hoc, frame-oriented (AFO) CAI by Carbonell (1970) to stress its dependence on author specified units of information. The branching strategies of some AFO programs have become quite involved, incorporating the best learning theory that mathematical psychology has produced (Atkinson, 1972; Fletcher 1975b; Kimball, 1973) and many of these systems have been used successfully and are offered commercially. Most AFO courses, however, are not appropriate uses of computer technology:

In most CAI systems of the AFO type, the computer does little more than what a programmed textbook can do, and one may wonder why the machine is used at all....When teaching sequences are extremely simple, perhaps trivial, one should consider doing away with the computer, and using other devices or techniques more related to the task. (Carbonell, 1970, p. 201)

In this pioneering paper, Carbonell goes on to define a second type of CAI, information structure oriented (ISO) CAI which he distinguishes from AFO CAI in that ISO programs:

- have a representation of the subject matter they teach
- carry on a human-like dialogue with the student

and

- use unanticipated answers to deduce the student's misunderstanding.

This style of CAI is often called "generative" CAI (Wexler, 1970) since it is typified by programs that generate problems using a large data
base representing the subject they teach. * However, generative CAI has been used to characterize many different types of programs. The simplest, like the arithmetic strands program described in Section III, simply used a random number generator to fill in author-specified problem skeletons. The kind of courseware that Carbonell was describing in his original paper was to be more than just a problem generator, rather a computer-tutor that had the inductive powers of its human counterparts and could offer what Brown (1975) calls a "reactive learning environment", one where the student's interests and misunderstandings drive the tutorial dialogue.

Kimball (1973) states that the characteristics of a good tutor which should be modelled by a CAI program include:

- transmitting problem solving heuristics
- choosing appropriate examples
- dealing with arbitrary student examples
- handling a wide range of student backgrounds

and

- learning student heuristics if they are superior.

This goal has been expressed by other researchers who have tried to write CAI programs that extend the medium beyond the limits of frame-selection:

Often it is not sufficient to tell a student he is wrong and indicate the correct solution method. An intelligent CAI system should be able to make

* See Koffman and Blount (1975) for a review of some early generative CAI programs and an example of the possibilities and limitations of this style of courseware.
hypotheses based on a student’s error history as to where the real source of his difficulty lies. (Koffman & Blount, 1975).

Its realization, however, has involved increasingly complicated computer programs that display truly intelligent behavior. This has led CAI researchers into the area of Computer Science called Artificial Intelligence, whose practitioners attempt "to construct computer programs which exhibit behavior that we call intelligent behavior when we observe it in human beings." (Feigenbaum & Feldman, 1963.) These CAI programs are quite different from even the most complicated frame-oriented, branching program.

Traditional approaches to this problem using decision theory and stochastic models have reached a dead end due to their oversimplified representation of learning. It appears within reach of AI methodology to develop CAI systems that act more like human teachers. (Laubsch, 1975)

Brown has dubbed these hybrid systems intelligent CAI (ICAI). Peele and Riseman (1974) feel that the development of intelligent CAI programs is inevitable and characterize four phases of the use of Artificial Intelligence in courseware:

- benevolent mentor: the computer acts as a flexible, sensitive tutor
- cognizant tool: the computer amplifies the student’s problem solving power
- problem solving partner: meaningful dialogue about the problem
- learner: student learns by teaching the computer
Inevitable or not, the development of intelligent CAI programs has been slow and difficult. The next sections list some of the developments in AI that form the foundations of ICAI systems, the unique difficulties discovered in applying AI techniques in instructional programs, and the specific parts of the CAI programs that have been augmented in this way. The third section of the paper describes many interesting examples of ICAI programs.
II. Applying AI in Intelligent CAI

Artificial intelligence work in natural language understanding, the representation of knowledge, and methods of inference as well as specific AI applications areas like algebraic simplification, calculus and theorem proving has been applied by various researchers toward making CAI programs more intelligent and more effective. Smith (1976) has noted that in many cases these researchers borrowed AI techniques, but had to modify them, often making the inference routines less powerful, but forcing them to follow human reasoning patterns, so as to better explain their methods to the student. Thus Smith concludes that "CAI is not just an area for application of AI techniques but also contains its own point of view towards computer intelligence." Brown (1975) has also noted the caveat: "ICAI systems can't be AI systems warmed over... The problem is that these techniques have barely scratched the surface." In other words, being able to do something is not the same as being able to teach someone else how to do it.

The AI techniques that have been used in ICAI systems include the representation of knowledge, natural language understanding, and specific application area techniques. The applications are interrelated, not only in their goal, namely intelligent tutor-like behavior, but also in the techniques employed. For instance, modelling a student's understanding of a subject is closely related conceptually to figuring out a representation for the subject itself or for the
AI techniques for the representation of knowledge have been used in three different aspects of CAI programs. Knowledge of the subject matter was originally envisioned as a huge static database incorporating all of the facts to be taught. This idea was implicit in the early drill-and-practice programs, and was made explicit in generative CAI. Recent systems (discussed in the next section) have used procedural subject matter knowledge, to show the student how to do things (take measurements, make deductions, etc.). Once again, the fact that these subject matter experts must not only solve problems in their domain but also explain how to do so requires that they follow human-oriented heuristics, a requirement that puts additional constraints on the original AI techniques.

Laubsch (1975) has discussed this more general idea of subject matter knowledge (see Section III) and adds: “A teacher needs some knowledge of his student’s knowledge and goals aside from knowledge about the subject matter.” The student model is the second type of knowledge the program needs, and includes some estimate of what the student knows as well as other information that might be used to "tune" the system to his needs. Fletcher (1975b) has made an exhaustive survey of the use of mathematical models of learning in CAI and concludes:

* See Self (1974) for an excellent discussion of the differences and interrelations of the types of knowledge needed in an intelligent CAI program.
The central problem in computer-assisted instruction is the translation of instructional practice, which is fairly vague, to computer programs, which are quite precise. Models of the learner may be essential in translating instruction to effective procedures. (p. 118)

AI techniques for modelling the student have included simple pattern recognition applied to the student's response history (Koffman & Blount, 1975), flags in the subject matter semantic net representing areas the student has mastered (Brown, 1974; Carbonell, 1970; Wexler, 1970) and a rather sophisticated idea of modelling the student by a program that makes the same errors the student makes, and then tries to debug the program (Brown, 1975; Ruth, 1974; Self, 1974).

To debug the program, or the student, requires yet a third type of knowledge, which would deal with questions like "When is it appropriate to offer a hint?" or "How far should the student be allowed to go down the wrong track?"

These are just some of the problems which stem from the basic fact that teaching is a skill which requires knowledge additional to the knowledge comprising mastery of the subject domain. (Brown, 1975)

This additional knowledge, required beyond a representation of the subject domain and the student, is knowledge about how to teach.

It should be mentioned that there are no standard techniques, from AI or anywhere else, for representing knowledge of these sorts.

In fact,

* Self's paper is once again recommended as an excellent critique of AI-like student models in CAI. It also presents a very interesting idea for a student model, involving evaluation of procedures on two distinct PLANNER-like data bases, one representing the subject matter, and the other the student's current understanding of it.
researchers developing these methods for CAI applications are sometimes in a uniquely advantageous position to understand how we understand:

The precise thinking and the generality required by ISO CAI with its information-processing formulation will translate, we hope, into our better understanding of the processes of teaching, learning, and personal verbal communication. (Carbonell, 1970, p. 196)

This argument, or belief, has been echoed by many researchers in ICAI (Goldstein, 1975; Norman, Gentner, & Stevens, 1976; Ruth, 1974; Smith, 1976) and is a local rendition of a fundamental belief of the AI community: that by trying to imitate intelligence we will come to understand it.

The notion of "cognitive science" might be incorporated here.
III. Examples of AI applications

The review presented here is not an exhaustive list of intelligent CAI programs. In particular, it is biased toward the author's experience at Stanford. Nevertheless, it will serve as an introduction to the different kinds of AI applications, and to the historical background of current issues.

Suppes: Math Strands

One of the earliest examples of a well thought out CAI course was the arithmetic drill and practice curriculum developed at Stanford (Suppes, Jerman, & Brian, 1968; Suppes & Morningstar, 1972; Suppes, Searle, & Lorton, in press). The gradual development of this classic piece of courseware, some version of which is now available on most commercial CAI systems (Computer Curriculum Corporation, Honeywell, Hewlett-Packard, to name a few) illustrates the essence of the process of making CAI intelligent. The original system, built for a prototype IBM 1500 system (Suppes & Morningstar, 1972), simply presented arithmetic problems on the screen and checked answers. Later work, by Lorton and Searle (personal communication), made the course "generative"; the problems were generated, rather than retrieved from a pre-specified curriculum, using a random number generator to fill in author-supplied problem skeletons. This effort, which required careful study of how arithmetic was learned and taught in order to specify the
nature and interrelations of the problem skeletons *, involved a major reorganization of the curriculum and even led to a theory of learning in this area (Suppes & Morningstar, 1972).

Certainly drill and practice in arithmetic was an appropriate application of computer technology to teaching. The program could conduct the standard drill with as much individualization as the teacher, with more privacy, and perhaps more attention to the implications of each student's errors than even a human tutor could give. However, the program was not effective in correcting the student's misunderstandings (or "teaching"). The drill and practice paradigm does not involve teaching of this sort.** So what has this easy, already operational drill and practice course to do with intelligent CAI: complex CAI applications of work in artificial intelligence that might not be completed for decades?

[The arithmetic Strands program is intelligent CAI]. Its performance is not only adequate, it is (within the pedagogical framework of "present - evaluate - correct") all we can reasonably expect of an arithmetic teacher in the form of a teletype. (It is possible, but not certain, that future audio-visual interfaces may make the program more effective.) In other words, its limitations do not stem from the shortcomings of the computer. The reason for this is

* called "equivalence classes" ** See Burton and Brown (1976) for a discussion of a very interesting "intelligent" arithmetic tutor project based on the game "How the West Was Won."
that the computer is inherently an expert at arithmetic, and can be
made an expert at arithmetic drill by relatively easy programs. In a
sense AI is machine intelligence beyond arithmetic. In examining later
attempts at applying Artificial Intelligence techniques to instruction,
we will do well to keep this not so artificial example in mind as a
yardstick. 

Carbonell and Collins: SCHOLAR

Carbonell's program to teach South American geography was an
explicit attempt to employ current AI techniques in CAI courseware, and
remains a landmark in the field. The original idea was to use a
semantic-net to represent all of the relevant facts about South
American geography and their interrelations. Thus, all of the cities
in Chile, and their populations, were stored in the data base. The
instructional program included inference procedures that could deduce,
for example, which is the largest city. The SCHOLAR program can
understand and generate English sentences and conduct a dialogue with
the student, both answering the student's questions and initiating
questions of its own.

Semantic networks were a very important development in AI and
cognitive psychology: a way of representing knowledge that was formal,
heirarchical and thought to be similar to the representation in our
brains (Quillian, 1968). Although current thinking on representing
knowledge has drifted away from the static semantic net,* the SCHOLAR program remains a prototype for Intelligent CAI, and the SCHOLAR group at BBN is continuing work on teaching procedural subjects with a SCHOLAR-like system (Grignetti, Gould, & Hausmann, 1975) and on discovering tutorial strategies that humans use and incorporating them as heuristics in the instructional program (Collins, Warnock, & Passafiume, 1974).

Suppes: Axiomatic Mathematics

One of the early AI successes was the development of theorem-proving programs that could verify deductions in the propositional calculus. (For an example, see Newell, Shaw, & Simon, 1963.) Applications of mechanical theorem proving to CAI appeared immediately (Easley, Gelder, & Golden, 1964) and by the late sixties a second generation, full scale course in elementary mathematical logic was in daily use at Stanford. (Goldberg, 1973; Goldberg & Suppes, 1972).

The original theorem provers had to be modified to be used

* Writing SCHOLAR for a more computational or algorithmic topic would require "a semantic network much richer in procedures than that of more descriptive subjects" (Carbonell, 1970, p. 191). The representation of procedural knowledge, knowledge of how to do things, is different from what a semantic net of this sort was designed to do. Recent concentration on what is called the "procedural representation of knowledge," i.e., representing knowledge by properly indexed programs that "know" how to do things (Bobrow & Collins, 1975; Hewitt, 1972; Winograd, 1972) has shown that static as well as procedural knowledge can be conveniently represented this way. Current work (Anderson, 1976; Norman & Rumelhart, 1975) on semantic net representations incorporates a procedural component into the net.

15
effectively for teaching (Goldberg, 1973). A program that can verify the logical validity of a thousand-line proof cannot necessarily determine what is wrong with a ten-line almost-proof or give meaningful comments to a befuddled student who has "gone wrong" somewhere. Current work at IMSSS has concentrated on turning this early work into really effective computer-based teaching. Using more powerful heuristic theorem provers coupled with English language ability, current programs in logic, set theory and more advanced and applied topics in axiomatic mathematics discuss in a relatively easy and informal grammar the nature of the student's error and assist him when he gets stuck mid-proof. The currently operational courses in Logic and Set theory will embody the current state of the art in its use of synthesized audio, natural language processing, theorem proving and proof checking (Sanders, Benbassat, & Smith, 1976; Smith & Blaine, 1976; Smith, Graves, Blaine, & Marinov, 1975).

I think that it is important to note the developing nature of CAI in the area of proving theorems, one of the first problems to be cracked in AI research. There is a big difference between theorem provers like Newell's Logic Theorist (Newell et al., 1963) and the prover used at the heart of the Stanford courses. The two programs were written to solve two different problems: one to show that a computer could be programmed to perform as an "intelligent" logician, and the other to turn the machine into an intelligent logic teacher. Two different intelligent behaviors, two different AI problems.
Pioneering AI work had to precede the latter application, but it is a difficult AI problem in its own right. Clearly, it is the same difficult problem.

Kimball: Methods of Integration

Another early AI success, mechanized symbolic integration and algebraic simplification programs (Slagel, 1963, for example), directly inspired an effort to produce a CAI tutor for the methods of integration (Kimball, 1973). Kimball was particularly interested in stimulating creative problem solving with the computer. As a result his thesis research involved not only the application of symbolic integrators to discovering and correcting student errors, but also significant efforts to model the student's state of knowledge in an effort to optimize his learning experience.

The finite number of skills involved in the subject being taught, as well as the general power of the symbolic integration routines available made the course on methods of integration a great choice for explorations of what could be done with an intelligent CAI system. The approach taken by Kimball was to describe the student's state of knowledge as a state vector, one state for each integration technique (trigonometric substitution, integration by parts, etc.). Transition matrices were then used to describe the learning process, with particular emphasis on the points where the student's lack of knowledge caused him to fail in his attempt to solve an integral. The instructional program, incorporating an expert integrator, was able to
diagnose his difficulty at those points and give reasonably appropriate assistance.

Although simple Markov models may prove to be inadequate for more complex and involved subjects (Barr & Atkinson, 1975; Fletcher, 1975b), Kimball’s calculus tutor provides a fine example of CAI made intelligent enough to offer an environment that can no longer be considered merely an extension of the classroom/book curriculum; it is something new. The amount of work by both AI and by CAI practitioners required to build equally powerful instructional laboratories in subjects that are not yet solved AI problems is considerable, but the resultant courseware seems like what we want the computer’s impact on education to look like.

Atkinson: Individualization of Instruction

The interesting characteristic of Atkinson’s elementary reading program, an effort to automate drill in word recognition and phonics for young children, was an early consideration of individualization from a very general perspective. Although the models of the student’s state of knowledge were crude, there was an explicit recognition of the importance of modelling the learner for effective CAI (Atkinson, 1972; Atkinson, Fletcher, Lindsay, Campbell, & Barr, 1973). The work of this group at Stanford has explored the possibilities for optimization of instructional allocation and individualization (Barr, Beard, & Atkinson, 1975; Paulson, 1973). But early decision-theoretic models of
learning are inadequate for complex tasks, and no theory of what a student model should be has yet evolved (Atkinson, 1972; Fletcher, 1975b; Self, 1974).

For example, recent work in the Complex Instructional Strategies group at Stanford has concentrated on individualizing instruction in complex subjects that are not yet as completely mechanized as arithmetic. An innovative laboratory for teaching computer programming, the BASIC Instructional Program (BIP), was developed as a testbed for experiments on optimal task-selection procedures (Barr, Beard, & Atkinson, 1976). The method developed involves representation of the author's implicit idea of the interdependence of the problems in a pre-written curriculum via a Curriculum Information Network interrelating the skills required to be developed. Current work on describing typical student "bugs" in terms of these fundamental skills will allow still greater tutoring power, by using "failure points" to update a rather detailed student model. [All of this would of course not be necessary if the program could analyze students' programs on its own and figure out what's wrong with them (compared to its own generated solutions).] In some subject areas this goal has been approached, but as we discuss below, computer programming is not yet well enough understood.

* The "failure point" concept is from the Markov learning model used by Kimball. It is an example of the belief in the "bug" as a powerful tool for individualizing learning, as espoused by Papert (1970), Sussman (1973), Weyer & Cannara (1975), and Winston (1973), among others.
CAI in Programming

Programming the computer to teach programming is a natural and elegant idea that occurs from time to time to people in the Computers in Education field:

Whatever the state of computer-aided instruction with respect to other subjects, the computer is the ideal instrument for teaching its own use. (Fenichel, Weizenbaum, & Yochelson, 1970)

Since his own specialization in CAI has been CAI in computer programming, the author feels compelled to add that the extensive work done here by many diverse researchers indicates clearly how far we have to go before our instrument can intelligently teach very much at all.

The earliest attempts at a CAI course in programming that I know of were the SIMPER system (Lorton & Slimick, 1969) at Stanford and the TEACH system (Fenichel et al., 1970) at MIT. SIMPER was a programming laboratory, a simulated three-register machine with a simple instruction set (alterable for pedagogic reasons) that was to be programmed at an interactive console. TEACH was actually an operating system for students which offered lesson material as well as access to an interpreter for a language called UNCL which, like SIMPER, was especially designed for instruction.

[It is important to note that neither system paid very much attention to what the student did while he was programming.] They were primarily designed as special operating systems for interactive student programming, with, in the case of TEACH, a frame-oriented curriculum.
driver added on. Two large scale production CAI systems for programming were built along roughly the same lines a few years ago: the AID system at Stanford (Friend, 1973) had a large curriculum with a lesson-branching design that was used with limited success to investigate optimization strategies for frame-oriented courseware. The ACSES program on PLATO (Nievergelt et al., 1973) was also a large curriculum effort (it taught several languages) in the predominantly frame structure of PLATO. An interesting aspect of the ACSES system was an attempt to help the student through the problem-solving phase by letting him follow an AND-OR graph representation of the solution stored in the curriculum (Danielson & Nievergelt, 1975).

The first real AI type effort in computer programming CAI was Koffman's generative course in machine language programming. (Koffman & Blount, 1975). This system uses a set of programming primitives to generate programming tasks (by combining the right primitives) which it can both present to the student (in English) and solve with a program (since it can solve all of the primitive tasks). Decisions about what primitives are to be used in each problem are made on the basis of the student's previous successes: a simple student model. Unfortunately, in order to tutor the student when he makes an error, the system requires that he follow the primitive steps given in the problem statement in a lock-step fashion. Koffman argues that this is acceptable in the very beginning course on programming, because the student is given a systematic way of going about writing a program. He
agrees, however, that it is not the ideal. If we could look at the student's completed program, determine if it is correct, and if not, figure out what is wrong with it, "it would then be preferable to allow the student to design his own program to the extent he is able." (Koffman & Blount, 1975).

That is a big "if". Current work in program verification (Floyd, 1967; Manna & Waldinger, 1971) although it may be applied successfully to interactive verifying environments for professional programmers (Floyd, 1971; King, 1967), is unlikely to be useful in CAI for two reasons: First, these verifiers require the programmer to specify the intent of his code in a language, like predicate calculus, that is different from the programming language but just as powerful, and therefore likely to be just as hard to learn and use correctly. Second, although the verifiers are capable of determining whether a given code segment adheres to the specifications given, they are not very good at determining why a faulty code segment is wrong (Ruth, 1974). This kind of knowledge is essential for increasing the effectiveness of the machine tutoring environment.

Some more recent work in program understanding has followed a different approach which may be more promising for CAI. Perhaps the most exciting work in this area is the recent studies of program debugging at the MIT AI Project. Sussman's HACKER program (1973) writes procedures to perform actions in a hypothetical environment of physical objects manipulated by a robot. It then "debugs" its own
procedures by trying them out, identifying why they don't quite work, fixing them, and generalizing the patch to as large a class of similar "bugs" as possible. Goldstein's MYCROFT system (1975) is designed along similar lines with the intention of using a program that knows about programming and debugging for instruction. This system takes programs in a limited subset of TURTLE-LOGO along with a line-by-line, user-supplied specification of its performance, and locates certain types of bugs.

A final MIT AI project is Ruth's system (1974) for examining programs in CAI. This system determines the nature and causes of bugs in programs written by students in a restricted ALGOL-like language to solve a specific class of sorting problems. Using a grammar that generates all solutions to the problems using all approaches, the system determines which approach the student has used. It then determines where he has gone wrong by modifying the grammar to include known bug types (inspired, at least, by Sussman's work) and attempting to reproduce the student's error. If it is successful the system has a genuine understanding of the student's error and the context in which it was made, which it can then use to instruct the student in appropriate debugging techniques.

The important point here is that we do not have yet a very good understanding of what programming is: We do not have a sufficiently explicit knowledge about programming to enable a machine to be an expert programmer. And without that knowledge we cannot begin to build
an intelligent CAI tutor. Having taught the subject conventionally for many years doesn't seem to help. The implicit understanding of the material that a book author or lecturer has is not sufficient to teach a machine to teach. And I suspect that programming is not a unique subject area in this respect. AI successes in "real" subjects (Buchanan & Lederberg, 1971; Shortliffe, 1974) have involved tremendous efforts to get human experts to make their knowledge and strategies explicit. The beginnings of a database of this sort in computer programming is the "knowledge about bugs" incorporated in the programs of Sussman, Goldstein, and Ruth discussed above. These researchers derived their databases primarily by introspection, however, and much systematic work on the codification of programming knowledge remains to be done (Green et al., 1974). That body of knowledge could be a powerful pedagogical tool in itself, before it was ever incorporated in an intelligent CAI course.

J. S. Brown: SOPHIE

Perhaps the greatest success in tutorial CAI in technical subjects is the SOPHIE system developed by Brown and his associates (Brown, Burton, & Bell, 1974). SOPHIE is a tutorial laboratory for electronic troubleshooting that provides the student with facilities and guidance for testing and repairing faults in a simulated electronic device. The system monitors the measurements the student makes in trying to diagnose an unknown circuit fault (the student's problem-
solving path) and can comment when he makes redundant measurements or when his diagnosis does not follow from the measurements that he has made. SOPHIE's ability to provide "intelligent" tutorial interaction rests heavily on its specific knowledge of the single power supply circuit it simulates and its use of "physical" measurements of the circuit in the representation of the solution path; these features allow the student's problem-solving behavior to be analyzed with respect to the semantics of the problem domain. While SOPHIE illustrates some of the intelligent behaviors that can be achieved by a CAI system that interprets complex solutions, it does not attempt to model the student's understanding in a global way in order to adapt instruction to his needs. (In fact, SOPHIE has no curriculum; instead, it generates new problems by randomly introducing faults into the circuit. SOPHIE does not attempt to provide an individualized sequence of problems.)

\[\text{This seems to me to be more a reflection of philosophical tenets and AI interests than a serious developmental gap. Sophie (I believe) includes considerable amounts of explicitly taught about behaviors of electronic components, etc.}\]
The development of intelligent CAI systems is just beginning and seems far removed from the world of practical CAI running in schools today. There are several reasons for this, and if ICAI systems are eventually successful, there are some predictable effects on CAI authoring and systems:

1) The kind of course design and programming required is far beyond the subject matter expert. Author teams will have to include computer professionals.

2) Data structures of this nature are beyond the capabilities of all current CAI languages. This kind of work requires the versatility of full-fledged AI languages (See Laubsch (1975) and Lorton & Slimick (1969)).

3) Some capabilities, like building and retrieving from information networks, parsing English input, and making logical inferences, can be built into CAI systems as they are now built into AI systems. In fact for reasons of efficiency they must be.

Will these giant programs be worth the effort and expense required to develop them? Can we build programs that transmit the excitement that a good teacher brings to the subject?

There is, of course, little evidence that the intuition that more knowledgeable computer tutors do provide better instruction is a sound one. Until such programs are implemented there cannot be. (Self, 1974)