Growth and Maturity of Intelligent Tutoring Systems:

A Status Report

Beverly Park Woolf,
Joseph Beck, Christopher Eliot, Mia Stern

Department of Computer Science
University of Massachusetts
Amherst, MA  01003-4610
bev@cs.umass.edu

Running head: Maturity of Intelligent Tutors
1 Motivation and Goals

This research is concerned with improving the ability of software tutors to enhance the quality of individual human-computer interaction and to extend the range of students who can be reached. We address long-term questions, such as how tutors can facilitate learning and how this learning can be measured. Several case studies are presented, along with remaining hurdles on the way toward achieving stronger learning outcomes. Research has led to the development of several effective teaching systems, achieved by modeling learning at the cognitive level and then using artificial intelligence (AI) techniques, such as planning, machine learning, cognitive modeling and dialog management, to adapt curriculum topics and hints based on greater knowledge of the domain and student learning.

However, available cognitive models are not adequate for building all the types of learning systems wanted. Cognitive models have contributed some pedagogical and subject-matter theories, instructional design and enhanced instructional delivery to the field. On the other hand, many issues remain, beginning with understanding learning at a deep level, understanding educational methods and social organization and adequately representing alternative learning styles such as collaborative learning, scaffolding, mentoring and partnering. The lack of AI development tools, e.g., shells and frameworks, similar to those used to build expert systems, also limits development in this field.

Cognitive studies of instruction have shown that learners must remain active and motivated [Fletcher, 1995, 1996; Regian & Shute, 1993; Seidel & Perez, 1994; Shute & Regian, 1993]. Learners must want to learn and be involved, active and challenged to reason about the material presented. Flashy graphics and simulations are not enough; the experience must be authentic and relevant to the learner's work [Woolf & Hall, 1995; Woolf, 1992]. Simple presentations of text, graphics or multimedia often result in systems that encourage passive learning and provide little, if any, adaptability to different learning needs. Students do not learn by pressing buttons and flipping pages, even if the pages contain images, sound or video. Interactive exercises are required that involve students in the material.
AI-based systems overcome such drawbacks through sophisticated feedback, customized curriculum and focused methods for remediating errors. Adaptive systems are based upon explicit representations of tutoring, student knowledge and pedagogy, rules of inference about possible ways to teach content knowledge and dynamic generation of customized paths through the knowledge in response to student behavior. They might reason about stored knowledge, customize curriculum, feedback, help and error detection.

Intelligent tutors have many benefits for education. By adapting the curriculum to each student, learning should be customized to different learners. This allows self-directed learning. In the ideal case, this is similar to one-on-one tutoring and is in sharp contrast to the current conditions of an entire class of 25 students progressing at the same pace. One-on-one tutoring by human tutors has been shown to increase the average student’s performance to around the 98 percentile of students in a standard classroom [Bloom, 1984]. Formal evaluations have shown that intelligent tutors can produce the same improvement that results from one-on-one human tutoring and can effectively reduce by one-third to one-half the time required for learning [Regian, 1997].

AI applications can also make asynchronous learning effective. Current web-based material has proliferated beyond any individual’s ability to evaluate or fully utilize. AI techniques will enable students to learn about selected material at their own rate before it is taught or as a supplement to course activities. Such systems might become routine supporting group collaboration of students-at-a-distance, exploration of hypothetical worlds, and the making and testing of hypotheses. Learning need not be constrained to place and time.

The remainder of this chapter is organized as follows. First, we describe a variety of abilities tutors can demonstrate (generative, student modeling, expert modeling, instructional modeling, mixed initiative and self-improving) and then present four tutors which exhibit some of these abilities. These tutors are described in some detail with special attention to their artificial intelligence components. Next we discuss evaluation of tutoring systems and finally we examine remaining research issues.
2 Abilities of Tutors

Intelligent tutors have certain abilities that set them apart from earlier examples of Computer-Aided Instruction (CAI) [Fletcher, 1988; Lesgold et al., 1990; Regian & Shute, 1993; Fletcher & Rockway, 1986; Seidel & Park, 1995; Seidel et al., 1988; Shute & Psotka, 1995]. Some of these abilities are defined in Table 1, adapted from [Regian, 1997]. No universal agreement exists on which abilities are needed or present in intelligent tutors. However, the most sophisticated tutoring systems include, to some degrees, a large variety of these abilities.

Current published data suggest that, in general, the more effective instructional systems are the more powerful with regard to the above abilities [Fletcher, 1996, 1995]. However, this is probably not universally true. It is likely that the relative importance of these abilities is a function of the nature of the knowledge or skills being taught and the quality of pedagogy applied in the teaching context [Regian & Shute, 1993]. Researchers are now collecting data to quantify the independent contributions of each ability [Regian, 1997]. The goal is to quantify the effectiveness of each ability under varying pedagogical implementations -- for teaching various types of knowledge and skills [Regian & Shute, 1993; Seidel & Perez, 1994; Shute & Regian, 1993].

However, the efficiencies of certain instructional strategies may be dependent upon context, including the skills of the student. For example, students with less prior domain knowledge need more guidance than students with more prior domain knowledge, and that the level of guidance required to support students with less domain knowledge is frustrating and counterproductive for students with more prior knowledge [Regian, 1997]. Student Modeling (Ability 2) tracks student performance, models student knowledge and infers student learning, and Instructional Modeling (Ability 4) allows tutoring systems to appropriately modify the level of guidance provided to each student. In one very effective, highly interactive tutor that taught college statistics, the combination of student modeling and instructional modeling produced an additional 10 percent boost in student outcome performance as compared to traditional computer aided systems [Shute, 1995; Shute & Psotka, 1995]. Research such as
| **Generative.** | Generates appropriate instructional material, including problems, hints and help based on student performance. This ability is to be distinguished from storing multiple canned responses and then selecting one for each student. |
| **Student Modeling.** | Assesses the current state of a student's knowledge and does something instructionally useful based on the assessment. The system identifies presumed student knowledge and makes inferences about his or her grasp of skills. Frequently, a system will use a student model to represent how the student has organized and incorporated new knowledge. The student model might display its changing view of the student's strengths and weaknesses as well as aspects of his or her currently (mis)understood knowledge. |
| **Expert Modeling.** | Models expert performance and does something instructionally useful based on knowledge of the domain. Current systems vary in the type of knowledge they teach, e.g., velocity and acceleration, as well as processes within whole systems, e.g., emergency shut-down procedure in a boiler system [Woof et al., 1986], or a hydraulic process involved in folding a helicopter's blades [Towne et al., 1990]. Some teach formal logic and formal knowledge, e.g., ALGEBRALAND [Foss, 1987] and the Geometry Tutor [Anderson et al., 1985]. Building an expert model requires specification of the relative difficulty of topics, identification of the strategies and tactics used for tailoring instruction to an individual student, and a corpus of analogies, examples and error diagnosis techniques for teaching in the domain. Without the aid of shells, e.g., expert systems shells and authoring systems, which currently do not exist, this task is time consuming. Even with such software tools, each new domain requires identification of curriculum topics prerequisite topics, causal and temporal relations between topics, and the relative difficulty of learning each topic. |
| **Instructional Modeling.** | Changes pedagogical strategies based on the changing state of the student model, prescriptions of an expert model, or both. Tutoring style might also vary across domain types. Explanation, guided discovery learning, coaching, coaxing or critiquing might be preferable depending on the domain. Tutoring style also varies within a tutorial domain. For example, guided discovery learning might be replaced with opportunistic one-on-one tutoring once the student requests a specific activity or shows the need for remediation. How and why human teachers change teaching style is an open research question. |
| **Mixed-initiative.** | Human-computer communication in which the student or system controls the conversation or asks a questions. Such control is now assumed in responsive human instructional situations. The ability to initiate interactions with the student as well as to interpret and respond usefully to student-initiated comments is required. Natural language dialog is sometimes taken as the focus of this ability. Error diagnosis, or system's ability to diagnose mistakes, plausible misconceptions, overgeneralizations and missing information, is often a goal of this ability. A diagnostic tutor compares student behavior with that of an expert before reasoning about how to elicit better learning performance. |
| **Self-Improving.** | The capacity to monitor, evaluate and improve its own teaching performance as a function of experience. Machine learning, or some other technique which changes behavior over time, is required. |

**Table 1: Abilities of intelligent tutoring systems.**
this, which serves to quantify the relative instructional impact of specific abilities and to provide specific pedagogical details, supports a new approach to the implementation of instruction, termed “instructional engineering” [Fletcher, 1995, 1996].

Every intelligent tutor will incorporate one or more of the abilities listed in Table 1. The case studies below discuss the use of these abilities in working tutoring systems.

3 Case Studies

Tutors can provide a variety of interaction modes to address a variety of knowledge types. For example, for problem solving domains, such as physics or mathematics, a tutor might encode “correct” knowledge and then track the student’s actions for explicitly right or wrong answers (see Cardiac and Mathematics Tutors in this section). On the other hand, some inquiry-based activities might provide an environment for exploration and ask a student to generate hypotheses and analyze data. Such tutors (see Engineering Tutors or Adaptive Courses, in this section) will not necessarily check for correct answers. Intelligent technology plays a role in both types of teaching.

Early tutors, that dealt with correctable knowledge, were often omniscient or despotic in nature; incorrect student actions were quickly identified, based on reference to an error knowledge base, and remediated. However, immediate help has several disadvantages: it may compete for short-term memory with newly learned material and students might also become dependent on it [Schooler & Anderson, 1990]. Explicitly correctable knowledge can also be handled in an empathetic manner in which tutors implicitly elicit information about student goals and plans. Such systems require less knowledge about the domain and more information about the student and teaching rules. They coach rather than tutor and reason about several forms of behavior before taking action. In their most sophisticated form they might reason based on knowledge about how people solve problems or make inferences in the domain. Theoretical focus has shifted from exclusive diagnosis and remediation to identifying and supporting students in managing their own cognitive processes.
### Table 2: Cases studies and intelligent abilities

In this section, four tutors are described which use several of the abilities to achieve rich interactivity with the student (see Table 2). These tutors are summarized along with the intelligent abilities that give them their power.
3.1 The Cardiac Tutor

The Cardiac Tutor helped students learn an established medical procedure through directed practice within an intelligent simulation [Eliot & Woolf, 1994]. It incorporated substantial domain knowledge about cardiac resuscitation and was designed for training medical personnel in the skills of advanced cardiac life support (ACLS) [Eliot & Woolf, 1995]. The system evaluated student performance according to procedures established by the American Heart Association [AHA, 1987] and provided extensive feedback to the student.

A formative evaluation based on physician-administered final exams with two classes of fourth-year medical students, suggested that working with the Cardiac Tutor was equivalent to being trained by an emergency room doctor, “running codes” and testing the student’s knowledge of procedures on a mannequin for an equal amount of time [Eliot & Woolf, 1996a; Eliot et al., 1996]. This implies that an emergency room doctor would not be called away from clinical duties to train students, at least for the procedural part of the training.

A primary contribution of the Cardiac Tutor was use of an adaptive simulation to improve student learning. Technologies were integrated, including simulation, tutoring and plan recognition, in novel ways. The algorithm that identified procedural steps, for example, performed plan recognition in a multi-agent and real-time environment using knowledge-based methods for generating common sense plan interpretation in unexpected situations. Representation of expert action, represented as protocols listing a series of procedures, closely resembled the form in which domain experts express their knowledge. Consequently, when new advanced cardiac life support (ACLS) protocols were adopted by the medical community the system was easily modified.

The Cardiac Tutor addressed several issues of reasoning about a user’s knowledge and skills within a real-time system. The training context (i.e., choice of topics) was changed based on a tight interaction between user modeling techniques and simulation management. The tutoring process was enhanced by providing appropriate new patient cases when the user model was accurate, without creating serious
problems when it was not. The impact of tutoring decisions on the student was considered carefully. The student model was constructed using fallable heuristics, so inferences from that model were restricted to ensure that inaccuracies did not create serious problems for the student. Adaptivity built into the Cardiac Tutor was designed to reduce the consequences of incorrect assumptions. These abilities will be more fully explained below.

An Example of the Cardiac Tutor. One problem in treating a patient with a cardiac problem is that the heart sometimes spontaneously changes state and goes into one of several abnormal rhythms or arrhythmias. Proper training for ACLS requires that the medical leader apply specific procedures for each new rhythm. Approximately two years of closely supervised clinical experience in an emergency room, ambulance, or similar medical facility is often required. The cost of this training is high, because medical instructors must constantly supervise medical students to ensure that patient care is not compromised.

Figures 1-3 show screen images from the Cardiac Tutor, a simulation of a patient experiencing a series of arrhythmias. Figure 1 shows that the intravenous line has been installed and the patient is being

![Figure 1. The simulated patient](image)
intubated. The icons on the chest and mouth indicate that compressions are in progress and ventilation is not being used. A pacemaker was installed (shown in Figure 2). The student tried a sequence of drugs, specifically epinephrine and atropine and following the second dose of atropine, the ECG converted to ventricular fibrillation.

![ECG Trace](image)

**Sinus Rhythm**

**Ventricular Fibrillation with pacemaker capture**

**Figure 2. Simulated ECG traces.**

The protocol for ventricular fibrillation requires immediate electrical therapy, repeated at increasing strength up to three times or until conversion to another rhythm is seen. Electrical therapy begins by charging the defibrillator to the desired setting, i.e. 200, 300 or 360 joules. When the unit is ready, the student must press the “Stand Clear” icon to ensure that caregivers are not injured. Once the 'stand clear' command has been processed, the student may apply the countershock by pressing the “defibrillate” icon. Synchronized cardioversion may also be selected or the procedure aborted using the “dump charge” command. All of these simulated actions were monitored by the tutor and evaluated in comparison with medical protocols.

During the retrospective feedback, or post resuscitation conference, every student action was reviewed, and a history a performance shown, listing correct and incorrect actions taken by the student (see Figure 3). Each action in the history and performance review was connected to the original simulation state and knowledge base, so the student could request additional information about his past actions.
Maturity of Intelligent Tutors

Hints

• At 20 seconds you used INTUBATE while the rhythm was ASYS. The ACLS protocol suggests that the correct actions in this state are PACING, START-VENTILATION.
• At 48 seconds you used INTUBATE while the rhythm was ASYS. The ACLS protocol suggests that the correct actions in this state are EPINEPHRINE.
• At 77 seconds you used PRONOUNCE-DEAD while the rhythm was ASYS.

Figure 3. Requesting help in the hints window

Each state transition to a new rhythm was associated with a different probability as shown in Figure 4. The nodes represented states of cardiac arrest or arrhythmias, and the arcs represented probabilities of moving to a new physiological state following a specified treatment event or other significant occurrence during the simulation. The left side of Figure 4 represents the normal traversal of the system from one arrhythmia (VFIB) to alternative possible arrhythmias (VTACH, ASYS and BRADY). The tutor altered the path of traversal to increase the probability that a specific learning opportunity would be available to the student. In other words, if the student needed to practice a Bradycardia rhythm (BRADY), the transverse probability listed on the arc for BRADY would be modified (right side of Figure 4). Biasing the simulation to reach states with good learning opportunities is a novel way to implement goal directed behavior.

Traditional simulations used for training are not truly adaptive to the learning needs of the student. At most, simulations allow users to select among fixed simulation scenarios or to insert isolated events

Figure 4. Underlying representation of arrhythmias
(such as a component failure) [Self, 1988]. On the other hand, the Cardiac Tutor analyzed the simulation model at every choice point to determine if any goal state could be reached. It altered the simulation model dynamically to increase the learning value of the time spent interacting with the student, without eliminating the probabilistic nature inherent in the domain.

**Multi-Agent Systems.** Expert behavior, i.e. a series of correct procedures performed in response to a specific rhythm, was represented as a protocol (Figure 5, third line from the bottom). Domain actions within the protocol recognition algorithm were augmented with planning knowledge to enable the system to make a common sense interpretation of the protocols in situations resulting from student mistakes or unexpected events initiated by the simulation [Eliot & Woolf, 1996b]. The system ensured that every domain recommendation was possible in the current situation and conformed to some interpretation of the student’s actions applied to the protocols.

The system employed planning technology as a knowledge-based critic to determine teaching goal states. The goals in this directed graph corresponded with arrhythmias, different heart rhythms that the student should practice. However, the Cardiac Tutor reasoned about several issues: which arrhythmia to present; interrelations among goals; several goal states in the simulation which might satisfy a single tutoring goal; and the fact that all tutoring goals need to be satisfied eventually. The tutor also reasoned about which tutoring goals might fail, e.g., the student could not reach a rhythm that needs practice.

The tutor moved to a new goal state by searching forward from the current state whenever the simulation reached a branch point. Normally, the simulation would choose a direction randomly based upon the domain specific probabilities of the outcome. Before making this random selection, however, the tutor searched several states forward to determine if a high priority simulation state was easily accessible. When a goal, or high priority state, was accessible, the tutor considered altering the base probabilities of the chosen point. The choice was then made randomly based on “improbability,” see Figure 4.
The Plan Recognition Mechanism. The Cardiac Tutor was based on integrating a simulation, tutoring and plan recognition mechanism. Figure 5 shows a time varying trace of the integrated simulation, planner, plan recognition system, and student-model reflecting the actions of the student, the system and their effects on each other. The bottom line, *Simulation*, represents clinic reality, in this case, the independent succession of rhythms of the heart during a cardiac arrest. Sometimes the heart spontaneously changed state and went into one of several abnormal rhythms or arrhythmias. The student was expected to respond to the state changes correctly, *Expert Model*. However, the student’s response, *User Actions*, was frequently inconsistent with the behavior of the expert model. During such inconsistencies, a *Plan Recognition* phase predicted what an expert would do and compared this with the student’s actions.

The relation among tutor, plan recognition mechanism and simulation was cyclic: the plan recognition module monitored the student interacting with the simulation (User Actions in Figure 5) and produced information that was interpreted by the tutoring module to define goal states. The adaptive simulation responded to the current set of goals so the student spent more time working in problem

![Figure 5. Functionality of the simulation and planning mechanisms.](image-url)
solving situations with high learning value for the individual. As the student learned, the system continued to update its model thereby keeping the curriculum focused on the student’s most important learning needs.

Student learning needs were determined by evaluating past student behavior. Observed deviation from standard medical practice was interpreted as a need to better understand the relevant procedure. Student performance measures were combined with knowledge about the difficulty of each procedure to estimate which topics currently provided the maximum potential for learning, or maximum expected learning value. The maximum expected learning value combined the probability that the student would, in fact, learn, the importance of the topic and the quantity of expected learning. The goal of these combined heuristics was to maximize the student's improvement in performance per unit of time spent in the learning environment. Once learning goals were identified as required, the tutor probabilistically increased the priority of simulation states in which those medical topics had to be applied. We assumed that this would result in increased time on task and hence improved learning.

The student model was built passively by comparing student actions with expert protocols (similar to plans or scripts) representing expert behavior (Figure 5 top line). Medical interventions by the student were mediated by the computer and compared to protocols representing expert behavior. The computer offered automated tutorial help in addition to recording, restoring, critiquing and grading student performance. It customized the simulation to suit different previous levels of achievement and might, for example, require one student to work on a two or three rhythms for an hour, while another student experienced a dozen rhythms and contexts during that same hour. Good or improved performance was noted with positive feedback; incorrect behavior was categorized and commented upon.

The Student as Agent. The plan recognition system modeled an expert’s behavior in the domain for comparison with the student's actions. The student performed actions in the simulation that were evaluated by the tutor to assess consistency with the expert’s actions, encoded as a planning formalism.
The tutoring system reasoned about the student, both as an agent within the simulation and as a student learning a task. The student was responsible for directing simulated agents and most commands were simulated as orders for assistants to perform medical tasks. In other words, the student might initiate electrical therapy by first asking an agent to charge the defibrillator and then pressing the “stand clear” button. The student was represented within the simulation as he directly operated the simulated defibrillator and shouted orders for another agent to perform some action by selecting a menu command. Each order was simulated as realistically as possible, including an appropriate real-time delay as the task was performed.

The domain of cardiac arrest and the protocol of required procedures necessitated complex representations and reasoning mechanisms. Many abilities had to be generalized for multiple instances: multiple agents, (e.g., including doctors, nurses and EMTS), multiple roles (e.g., airways or medications manger), multiple actions (e.g., start iv, intubate, defibrillate) and multiple orders, (e.g., action performed out of order due to parallel activity).

**The Protocol Interpreter.** The system reasoned about the student's ability by comparing student actions with a model of expert behavior. Representation of expert behavior as a series of actions, see Figure 5, third line from the bottom, encoded a leader’s correct procedures in relation to the simulated arrhythmias, see Figure 4 and Figure 5, second line from the bottom. The team leader was expected to perform a limited set of actions and to direct the actions of other members of the resuscitation team. The system varied the number of assistants available in each scenario, thus affecting the possible number of parallel activities. Actions were assigned to a specific agent both for evaluating student performance and while updating the simulation. An action was considered valid if an available agent was ready to perform it. We assumed that simulated agents could remember a sequence of actions and perform them in order.
Figure 6. Protocol recognition.

Figure 6 shows a simplified version of the protocol interpreter. The correct protocol had to be selected initially by analyzing the state of the simulation. In this example, Protocol-3 was selected so Protocol-1 and Protocol-2 remained inactive. If the simulation had been in some other state initially then Protocol-1 or Protocol-2 could have been selected.

Encoding knowledge in this domain required a more sophisticated representation than depicted in Figure 6. Some actions were optional in some situations. For instance, if Act-32 was optional then the current recommendation would be the set including actions: \( R = \{ \text{Act-32, Act-33} \} \). If the student’s actions were always correct, updating the protocol pointer was comparatively straightforward. However, incorrect student actions were allowed to affect the state of the simulation, possibly resulting in movement to a state where currently recommended actions were impossible or meaningless [Broverman, 1991]. The protocol interpreter required additional planning knowledge to detect and correct such problems.

The knowledge base allowed preconditions and post-conditions to be specified for protocol actions. When the protocol interpreter detected an incorrect student action it notified the student, then examined the preconditions of actions in the current set of recommendations and then skipped impossible actions by moving the pointer forward.
Feedback during the simulation was considered carefully. Providing descriptive information during a simulation was found by the students to be quite intrusive. Only a small number of mistakes were considered important enough to force termination of the simulation for immediate feedback. For other mistakes, the tutor would beep and record information for presentation during retrospective feedback. This allowed students to continue working with the simulation despite imperfect performance. The simulation history was saved, so that mistaken actions could be reviewed in context by the student during retrospective feedback, see Figure 3.

**Actions and Synchronization.** Actions of agents were not simple. The role assigned to each action was part of the static knowledge about actions. The assignment of roles to agents was automatically defined at the beginning of each simulation depending upon how many helper agents were available. The number of helpers and their role assignments did not change during a simulation. The simulated team leader was controlled by the student and assigned the single role of leader. The other roles were distributed among the available helper agents as equally as possible.

In sum, the Cardiac Tutor utilized mechanisms for goal selection, plan formation, and plan instantiation within situated contexts. It included an accurate descriptive model of the emergency room environment and general patient status, combined with a causal model of cardiac function and related physiologic systems. The tutor consisted of: a simulation, representing the problem-solving environment; a student model, to guide the learning process; a bias mechanism, making the simulation adaptive; and a plan recognition system, for constructing the student model.

Multiple agent and planning technology enabled the Cardiac Tutor, unlike typical teaching systems, to go beyond simple classification of student actions as correct or incorrect by specifying how an incorrect user action related to the expert action. For example, student actions could be classified as too early or too late. In addition, matching student action against that of an expert enabled the system to recognize an action that was correct, but used an incorrect parameter value. This was labeled partially correct. Dynamic construction of the student model involved monitoring student actions during the
simulation and evaluating these actions in comparison with an expert model encoded as a multi-agent plan.

### 3.2 Mathematics Tutor

The second tutor applied machine learning to the problem of generating new problems, hints and help. AnimalWatch learned how to predict some characteristics of student problem solving such as number of mistakes made or time spent on each problem. It used this model to automatically compute an optimal teaching policy, such as reducing mistakes made or time spent on each problem.

AnimalWatch was a NSF funded arithmetic tutor designed for fourth through sixth grade elementary school students. The top-level goal was to increase girls' confidence in their ability to do mathematics [Beal et al., 1998b]. The hypothesis was that the abilities that make an intelligent tutor powerful, e.g., curriculum sequencing, personalized help and intelligent problem selection, also serve to bolster a student's interest and enjoyment of mathematics. In contrast to most educational software which is designed primarily with the male user in mind, AnimalWatch was tailored to girls’ interests and needs [Arroyo et al., 1999, 2000]. It engaged girls’ interest in math by blending mathematics with environmental biology, the science that is of highest interest to female students. An environmental biology storyline, in which math problems were embedded, unfolded as the narrative progressed. Students selected an endangered species, such as the Right Whale or the Giant Panda, which included an initial story context and joined an environmental team to monitor the behavior of the species. For example, in the case of the Giant Panda, problems involved research at the library about the Panda’s habitat, reading about the birth of a new Panda at the San Diego Zoo, estimating the expenses associated with a trip to China, and analyzing the rate of decline of the Panda population over time, etc.

AnimalWatch helped girls sustain their belief in their ability to learn difficult mathematics concepts [Arroyo et al., 2000; Beal et al., 2000]. After working with AnimalWatch, both boys and girls showed

---

1 This work was supported by grant HRD-9714757 from the National Science Foundation to Drs. Carole Beal, Beverly Woolf and Klaus Schultz at the University of Massachusetts, Amherst.
significant increases in math self-concept, as indicated by responses on a questionnaire originally
developed by Eccles et al. [1993], with girls in some cases matching boys’ responses. In addition,
AnimalWatch improved girls’ attitudes towards their desire to study mathematics [Beal et al., 1998a,
1998b; Hart et al., 1999]. Several evaluation strategies were employed. First, performance data (accuracy
of problem solving; number and type of errors; hints and instruction selected after errors; effectiveness of
hints; latency to solve and problem difficulty level achieved) were automatically collected for subsequent
analysis, and showed that the system was effective for both boys and girls [Beal et al., 2000]. Second, pre
and post test questionnaires to assess math self concept, math value and math liking have showed, based
on standard instruments developed by Eccles et al. [1993], benefits to students’ math attitudes of working
with AnimalWatch. The results of several other evaluation studies indicate that AnimalWatch provided
effective individualized math instruction and had a positive impact on students’ belief in the value of
learning mathematics.

Sophisticated machine learning algorithms reasoned about students’ knowledge and provided
customized examples and hints tailored for each student [Beck & Woolf, 2000; Beck et al., 1999, 2000].
The goal of the machine learning component was to automatically compute an optimal teaching policy,
such as reducing the amount of mistakes made or time spent on each problem by using a “two-phase”
learning algorithm, see Figure 7. The architecture used two learning agents. One learning agent was
responsible for modeling how a student interacted with the tutor (the population student model, or PSM),
and the other was responsible for constructing a teaching policy (the pedagogical agent, or PA). The
student's interaction with the tutor was modeled in the population student model. “Classical” tutors use
cognitive modeling to understand how a novice or expert solves problems in the domain. In
AnimalWatch, the PSM was trained to understand student behavior by observing hundreds of students
using a prototype of the tutor and was therefore capable of predicting an individual student’s reaction to a
variety of teaching actions, such as presentation of specific problem type.
The second learning agent, the pedagogical agent (PA) was responsible for constructing a teaching policy that met a configurable teaching goal. The PA was a reinforcement learning agent and used the PSM to simulate the environment of working with hundreds of students. The reward for the PA was whatever high-level learning goal an instructor specified, such as reducing time spent on problems. Casting the problem in this way permitted instructors to ignore low-level issues, such as specific teaching rules, e.g., which analogy worked best on which problem. The PSM and PA worked together to permit the tutor to learn a good teaching policy directly from observing students using a simplified version of the tutor.

The architecture was evaluated by comparing it to a “classical’’ tutor that taught via heuristics without the aid of machine learning agent. The metrics used to assess student performance included subjective measures, (e.g., mathematics self-confidence and enjoyment) as well as objective measures (e.g., mathematics performance).

Motivation. Use of machine learning was motivated in part to produce more flexible tutors. In other words, to expand the population of students reached by tutors and to expand the quality of responses for an individual student. For example, if a tutor were designed to teach algebra to ninth graders, it might be programmed to assume that all incoming students had a fairly standard set of skills and would acquire
new knowledge about the domain in a fairly standard way. Such assumptions would probably not even be valid for all ninth graders, and it is unlikely they would apply for an eight year old math superstar learning algebra at a young age, or for a thirty year old community college student who has tried to learn algebra in the past and failed. The latter two students are likely to learn very differently. In particular, the community college student probably has some experience with algebra, and may have some misremembered rules. Such misconceptions from prior exposure can be a major stumbling block to reteaching the material [Stern et al., 1996]. It is not feasible to construct a separate tutor for every new population of students. Thus some means of constructing more adaptable tutors are needed.

Another obstacle to the acceptance of tutors in many training situations is the high cost of development and lack of flexibility once deployed. Cost and flexibility are two very similar problems: the actual difficulty is the high cost per student taught. If some means could be found of either decreasing development cost, or broadening the range of uses of a finished tutor, intelligent tutors would be more cost efficient. In addition to problems of cost, institutional acceptance is another factor blocking tutor deployment [Bloom, 1996]. Few organizations are willing to accept teaching software that cannot be modified.

**Research Objectives.** AnimalWatch addressed difficulties in tutor construction and especially the large amount of time required to build tutors. Currently, a large part of the effort of building tutors is spent on encoding a human teaching knowledge [Quafafou et al., 1995].

The machine learning agent in AnimalWatch was able to change the tutor’s *model of how to teach*, rather than simply changing the tutor’s explicit *teaching* in a specific instance. This fundamentally differs from traditional tutor adaptation. A “classical” tutor has encoded rules that enable it to use a student model based on student performance to adjust its teaching. Thus, an individual student may be treated differently from another student, but all such similar students, for the life of the tutor, will be treated with the generation of exactly the same problem, hints or help. Machine learning, on the other hand, enables the tutor itself to change and thus, over time its treatment of this typical student will be different.
Machine learning alters how a tutor reasons and makes inferences about the student. This permits it to reason “outside” of the variables that make up its student model. For example, a tutor could realize that one student who has the multiplication tables memorized can solve multiplication problems with small numbers much more quickly than expected and somewhat more quickly than expected on multiplication problems with large numbers. From this, it could adjust its mechanism for estimating how a student will perform on a problem and indirectly influence its teaching. This form of adaptation is not applicable to an entire population of students, but only to the student currently using the tutor. However, the tutor does not understand why the student is working more quickly. Thus, another student who, for some other reason, produced the same overt behavior would be categorized with the fast multiplier.

By allowing the tutor's modeling to be more flexible, the tutor is usable on a wider range of students. Also it improves the tutor’s adaptivity with a particular student. In the ideal case, machine learning techniques will adjust the system’s modeling techniques on-line as the student interacts with the tutor. This modeling is more challenging than off-line machine learning, which is done before students begin using the tutor (generally with data previously gathered). Students generally do not use a tutor for a long period of time; 20 hours of instruction is current state of the art [Koedinger & MacLaren, 1997] which severely limits the number of training instances available to the machine learning agent.

**Interacting with the Mathematics Tutor.** The expert model in AnimalWatch was arranged as a topic network where nodes represented skills to be taught. The links between nodes frequently represented a prerequisite relationship. For instance, the ability to add is a prerequisite to learning how to multiply. Figure 8 shows a portion of the topic network from AnimalWatch.

Throughout this discussion topics are major components of the curriculum about which a student may be asked a question, while skill refers to any curriculum element (including topics). For example, borrowing (as in subtraction) was not tested separately and so was a skill but not a topic. Whole number subtraction was tested and can be considered both a skill and a topic. When we use the term “topic,” we refer to the type of problem on which the student was currently working.
Subskills are steps within a topic the student performs in order to accomplish a task. For example, adding fractions has the subskills of finding a common denominator, converting the fractions to equivalent form with a new numerator, adding the numerators, simplifying the result, and making the result proper. Note that “add wholes” was both a prerequisite and a subskill for “add fractions.” For a given problem, not all of these were required. Table 3 shows the subskills required for some sample problems.

<table>
<thead>
<tr>
<th>Subskill</th>
<th>Problem 1</th>
<th>Problem 2</th>
<th>Problem 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Problem 1</td>
<td>1/3 + 1/4</td>
<td>2/3 + 5/8</td>
</tr>
<tr>
<td>Find LCM</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Equivalent fractions</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Add numerators</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Make proper</td>
<td>No</td>
<td>No</td>
<td>yes</td>
</tr>
</tbody>
</table>

Table 3: Three sample add-fractions problems

AnimalWatch provided an environment for students to practice mathematics and receive immediate feedback about their work. The tutor generated the topic on which the student worked. Figure 9 shows an example of a problem for addition of whole numbers and Figure 10 is an example of a topic generated from the “pre-fractions” area of the curriculum. Generating the topic on which the student worked was one way the tutor adapted the curriculum to the learning needs of the student. Students moved through
the curriculum only if their performance for each topic was acceptable. Thus problems generated by the tutor was an indication of mathematics proficiency, as described below.

Figure 9: A typical addition problem with a simple text hint
AnimalWatch adjusted the difficulty of each problem presented. For example, different addition of fraction problems can widely vary in their degree of difficulty. This process is illustrated in Table 3. The more subskills required, the harder the problem. Similarly, larger numbers also increased the tutor's rating of the problem's difficulty: it is harder to find the least common multiple of 13 and 21 than it is to find the least common multiple of 2 and 3. AnimalWatch also selected from a variety of hints, some requiring manipulation of small rods on the screen (see Figure 12) and others describing procedural rules. The tutor recorded information about the difficulty of problems presented and abilities of hints, such as the amount of text they contain, whether they were interactive, how much information they provided. Figures 9, 11 and 12 demonstrate hints that provide varying amounts of information. The hint in Figure 9 is brief and text based, while the hint in Figure 12 is interactive. If the student continued to make mistakes, more and more specific hints were provided until he answered the question correctly.

A database of the student's current state of knowledge, or student model, tracked the student's level of ability for each skill being taught. It noted how long the student took to generate a response, both after
the initial problem presentation and (in the event of an incorrect response), the delay in responding after a hint was presented. It also measured the student's level of cognitive development [Arroyo et al., 1999] according to Piaget's theory of intellectual development [Piaget, 1954]. This measure correlated with student's math performance and can was used to further customize the tutor's teaching [Arroyo et al., 2000].
Knowledge about the student and the Tutor. The tutor tracked a variety of information about the student's behavior in the student model. A primary task of this model was to track the student's level of ability for each skill. In addition to “snap shots” of the student's current performance, the tutor also kept track of trend information. A history of the student's proficiency in each topic was maintained; this allowed the tutor to determine if the student was making progress or not. The student model also contained the average amount of time the student required to solve each problem.

The tutor recorded information about problems presented to the student. When a student was shown a problem, the tutor noted the topic of the problem and the operands. The tutor used a set of heuristics to assess the difficulty of the problem based on the topic and operands. For example, the math problem was assigned a numerical difficulty rating based on how many subskills the student must apply and the complexity of applying each such operation [Beck et al., 1997].

The Machine Learning Agent. A layered machine learning architecture (Figure 7) was used to allow each learning component to be considered and optimized independently and separately from the design of the rest of the tutor. One layer of learning (PSM) was concerned with describing student behavior in different contexts. The other layer (PA) determined how to map this description of the student's behavior onto a correct teaching action.

The first objective was to construct a model of how students behaved while using the Tutor. This model predicted how a student would act in a certain situation, given that the tutor had previously taken certain teaching actions. By experimenting with this model of the student, the pedagogical agent determined which actions taken in which context resulted in this (simulated) student achieving the desired objective. This objective was customizable and provided the basis for the Reinforcement Learning (RL) agent’s reward signal. If the RL agent took actions that resulted in the student achieving desirable states, the reward was high. Figure 7 provides an overview of this process.
Thus, the RL agent was provided a model of how students behaved and a desired teaching goal, such as reduce the amount of time an individual student spends solving problems. The agent then output a teaching policy, which took as input a set of abilities that describe the current teaching situation (how the student was doing, prior help given, etc.) and output a recommended teaching action. This teaching policy was used by the tutor to direct its teaching decisions, including which problem to generate and what type of hints to show.

The tutor observed whether the student’s response was correct and how much time it took her to generate this response, a relatively simple set of goals. It did not attempt to create a general simulated learner for all conditions, or even a general learner for this particular domain. Rather, it built a simulation for how a learner responded (at a directly observable, not cognitive process, level) in a specific domain to specific teaching actions. This restriction made the problem tractable.

In order to gather enough information to make predictions, data about the entire population of users were gathered. When AnimalWatch was deployed with hundreds of students, logs from each user were saved and then merged. This allowed a much larger training set than if the Tutor were restricted to data from a single user. This population student model (PSM) made predictions about “an average student with a proficiency of X who has made Y mistakes so far on the problem.”

**Architecture of AnimalWatch.** The learning algorithm for the PSM took as input a state and determined the student's likely behavior. Ideal characteristics of this algorithm were robustness to noise, low computational cost, and being capable of learning from little data. Even though the PSM was first trained off-line, where training data were plentiful, there might also be an on-line learning component that attempts to reason about an individual student while he uses the tutor. In this situation, training data are scarce.

The PA optimized its decisions to maximize the expected future reward it received. Since the Tutor was designed to increase confidence in mathematics, students might be frustrated by spending a lot of time (or making many mistakes) on a problem.
AnimalWatch integrated both the PSM and PA into the tutor. The PA was needed since it directed which teaching decision to use. The PA needed some way of predicting the effects of its teaching actions, and the PSM was responsible for predicting the student's action to a particular action in a given context.

**Evaluation of AnimalWatch.** The performance of the PSM and PA in isolation was evaluated without testing the combined system on actual students. To evaluate the PSM, the data were split into training and a test set. The PSM was constructed with the training set and its predictions were evaluated on the test set (this can be extended to cross-validation techniques). Each prediction of the PSM (e.g. in the current system, likelihood of correct answer and time to generate response) can be verified in this manner.

To measure effectiveness, a variety of performance metrics were used. Pre- and post-test scores used a standard test for mathematics confidence [Eccles et al., 1993] and were used to measure the tutor’s impact on girl’s self-confidence in mathematics. Data concerning student’s qualitative assessment of using AnimalWatch were also gathered. Finally, actual performance data of students using the Tutor were evaluated.

A critical issue was what it meant for this architecture to “work.” It was possible for the PSM and PA to work flawlessly together and produce no improvement in any of the above metrics. The difficulty was that the system tried to optimize an externally defined learning goal.

To determine if the PSM was sufficiently accurate, we compared its predictions to how the students in the training dataset actually performed. Figure 13 shows its accuracy for predicting how long students required to generate a response, the PSM's predictions correlated at 0.629 with actual performance. Training on half the dataset and testing on the other half dropped this correlation to 0.619, which was still very strong for predicting as noisy a variable as time.

It was necessary to validate whether the pedagogical agent had learned anything useful from its training. The PA could not be validated in as simple a manner as the PSM. It was difficult/impossible to split the data into training and test sets, since the PA made active decisions. The actual data may not
follow the path the PA would have chosen. The PA's experience occurred in the context of a simulated student using a simulation of a tutor. To determine if the pedagogical agent would interact correctly with the PSM, we constructed a simulation. This simulation contained all the typical steps in a tutoring session. First, the PA selected a topic and a problem. The PSM was then invoked to see if the student would make a mistake or answer correctly. In event of an incorrect answer, the PA was responsible for finding the hint that appeared best. From the PA's standpoint, this was identical to interacting with an actual student.

Figure 14 shows the improvement of the PA's performance at minimizing the amount of time students spent per problem. The x-axis is the number of trials (in thousands) the agent has spent learning. The y-axis is the exponential-average of the amount of time the simulated student required to solve a problem. Performance started at around 40 seconds per problem, and eventually reduced to around 16 seconds per problem.
To test whether this architecture would work with actual students using an actual tutor, we evaluated the system in a controlled experiment. The first version of AnimalWatch used a fairly weak teaching model with some basic heuristics, and was tested in two local elementary schools. These data were used to train the PSM and PA. The AnimalWatch Tutor was then taken to a different school for testing. The students at this school were broken into two groups. One group, the control (N=39), were taught with a set of teaching heuristics. The second, experimental (N=58), group was taught using just the machine learning (ML) architecture. The ML architecture had the goal of minimizing the amount of time a student spends on a problem.

Students using the ML version of the Tutor averaged 27.7 seconds to solve a problem, while students using the classic version of AnimalWatch averaged 39.7 seconds. This difference was significant at P< 0.001 (2-tailed t-test). Just as important, the difference was meaningful: reducing average times by 30% was a large reduction. Thus, the agent made noticeable progress in its goal of reducing the amount of time students spent per problem. We are not arguing this was necessarily the best pedagogical goal, just what we asked the agent to do.
Further evidence of the ML agents' ability to adapt instruction can be seen in Table 4. Students in the experimental group, i.e. those who used the machine learning version of the tutor designed to minimize the amount of time per problem, solved whole (P<0.001) and fraction problems (P=0.02) significantly faster than students in the control group. Particularly impressive was that experimental group students were faster at solving fraction problems in spite of having a significantly lower Piaget score. Students in the control group, that is students who were taught using a set of heuristics, who got to fraction problems averaged a Piaget score of 7.9. However, students in the experimental group averaged 7.3; this difference was significant at P<0.001. In other words, even students with a weaker cognitive development, as shown by their lower Piaget score, were able to succeed at whole and pre-fraction problems in order to be allowed to move into fraction problems. Thus, in spite of being less restricted about which students saw fraction (i.e. difficult) problems, the experimental group still solved such problems more quickly. For whole number problems, students in the experimental group made 0.28 mistakes per problem, which is significantly fewer mistakes (P<0.001) than the control group (0.44 mistakes per problem). However, for fraction problems both groups made an equivalent number of mistakes (0.39 in the experimental vs. 0.40 in the control).

<table>
<thead>
<tr>
<th></th>
<th>Control</th>
<th>Experimental</th>
</tr>
</thead>
<tbody>
<tr>
<td>Whole</td>
<td>Percentage of problems</td>
<td>73.6%</td>
</tr>
<tr>
<td></td>
<td>Time</td>
<td>43.4 sec</td>
</tr>
<tr>
<td>Prefrac</td>
<td>Percentage of problems</td>
<td>19.3%</td>
</tr>
<tr>
<td></td>
<td>Time</td>
<td>22.7 sec</td>
</tr>
<tr>
<td>Fraction</td>
<td>Percentage of problems</td>
<td>7.2%</td>
</tr>
<tr>
<td></td>
<td>Time</td>
<td>44.5 sec</td>
</tr>
</tbody>
</table>

Table 4: Summary of performance by math topic area

In sum, this research has contributed to the field of intelligent tutoring systems in the automatic construction of a teaching policy. Learning techniques were used to increase a tutor’s flexibility, determine how to teach and allow the teaching policy to change over time for changing student population changes. No pre-built set of teaching rules offers this degree of adaptability. Another contribution was
exploring how to combine data from a single user with data previously gathered from a population of users. For the field of student/user modeling, this was a powerful idea. This application of reinforcement learning allows an agent to view the student as the “environment” and the Tutor as the actor. Finally, the potential to customize teaching goals allows a software tutor to be used in a wider variety of conditions.

3.3 **Engineering Tutors**

The next two tutors provided open design environments to support independent student exploration and implicitly elicit information about student knowledge and goals. These tutors coached rather than taught and required less knowledge about correct answers and more information about the domain. The Engineering Tutors coached about “design for manufacturing” (DFM) and provided students with a realistic understanding of complex processes through interaction with 3-D animations [Woolf et al., 1996, 1997]. Students designed parts and then the tutor simulated fabrication of the dies used in the manufacturing process. This provided immediate feedback about manufacturing costs, 3-D visualizations to supply intuition behind complex geometric problems and environments within which a student could experiment. Individual tutors addressed topics such as injection molding, stamping, forging, die casting, and finite element analysis. Some of these tutors contained an expert system based on visual components of the solution, as explained below.

The **Injection Molding Tutor** enabled students to construct and examine molded polymer part designs. It showed an animation of an injection molding machine along with a simple open/shut mold, see Figure 15. The student then created new designs, using either an “L-bracket” (see Figures 16a and 16b) or a box as the base, as in Figure 15. The student selected abilities to add to the base from a restricted set of abilities, shown in the pallet right side of Figure 16a, including ribs, “thru” holes and

---

2 This work was supported in part by grant from NSF/ARPA, EEC-9410393, The Engineering Academy of Southern New England, involving UMass, UConn and URI, by a grant from NSF/DUE - 9813654, and by the University of Massachusetts, College of Engineering.
bosses. As an example, shown in Figure 16a, a student placed a “thru” hole on the short end of the L-bracket. The tutor provided an animated three dimensional view of the tooling needed to manufacture the part (see Figure 16b), critiqued the design and provided alternatives to save money. The tutor stored an animation of a manufacturing die needed to produce every option a student might take. After the student created a part, the appropriate animated solution was selected from the visualization library and displayed (Figure 16b).

Figure 15. Injection molding tooling required to produce a simple U-shaped part.

Figure 16a. The student created a part by placing a ‘thru’ hole in an L-bracket.
Figure 16b. The tutor showed an animation of the die needed to manufacture the part.

The *Stamping Tutor* helped students understand the relationship between sheet metal part design and the required stamping stations. The tutor identified design issues and showed how they impacted the number of stations required. Design issues included: dissimilar abilities, closely spaced abilities, narrow cutouts, projections and bends. For example, a student might design a new part by dragging a ability (hole, rib, emboss or extruded hole) onto the blank metal strip (Figure 17, bottom left). Using a knowledge-based representation of stamping rules, the tutor dynamically generated a manufacturing solution, including the proper number of moving stamping stations required to build the part designed by the student (Figure 17, top), and included a critique of the design (Figure 17, bottom right). Similarly, when explaining how to bend a metal part, the tutor generated both an animated tooling solution (see Figure 18, bottom) and the plan view (shown on the top). These animated toolings were not stored or canned. The tutor selected graphic objects (film strips) stored in a simple network along with simple rules.
of design and manufacturing. Based on the decision of the expert system, certain stamping stations were
selected to illustrate the manufacturing process and new film loops animating additional stamping stations

![Figure 17. The student designed a stamped part by moving abilities onto a blank metal part (bottom left). The tutor’s solution is generated (top).](image)

were added to the evolving animation. The tutor also suggested design alternatives and critiqued the
student’s design, explaining why certain ability combinations or geometry would result in an inefficient
design. Since the animation and critique were presented in real time, in response to a student design, the
probability was increased that the feedback would address a specific learning opportunity.

Formative evaluation of these Engineering Tutors showed that they were as effective as several
lectures and homework assignments within a traditional classroom setting (see Table 5) [Poli et al., 1999;
Woolf et al., 1997]. The evaluation involved both introductory freshmen and junior engineering majors
who used the tutors as a normal part of the course. The Injection Modeling Tutor was tested with 125
freshman divided into two groups. The first group (Table 5, Software first) used the software first and then took an evaluation test. The other group (Table 5, Lecture first) attended traditional lectures,

![Figure 18. The tooling stations (bottom) and plan view (top) were generated by adding film strips based on expert systems analysis.](image)

worked problems both at home and in class and then took the evaluation test. Both groups eventually received the lecture and were allowed to use the software (Table 5, Both). The scores indicate that the software alone can teach the relevant concepts of injection molding for manufacturing. Students who did not attend lectures and only used the tutors understood the relationship between part geometry and tool complexity and were as knowledgeable about the manufacturing process as those who had the advantage of being exposed to a human manufacturing expert during lectures.

The Injection Molding Tutor was also evaluated with 42 junior engineering majors, 29 of whom (see Table 5, “Previous exposure”) had previously been exposed, via lectures and homework assignments two years earlier, to the processes and concepts of design for manufacturing. The other 13 students

<table>
<thead>
<tr>
<th></th>
<th>Average Score</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Freshmen</strong></td>
<td></td>
</tr>
<tr>
<td>Software first</td>
<td>80</td>
</tr>
<tr>
<td>Lecture first</td>
<td>81</td>
</tr>
<tr>
<td>Both</td>
<td>85</td>
</tr>
<tr>
<td><strong>Juniors</strong></td>
<td></td>
</tr>
<tr>
<td>Previous exposure to injection molding</td>
<td>80</td>
</tr>
</tbody>
</table>
Table 5: Evaluation of learning

<table>
<thead>
<tr>
<th></th>
<th>79</th>
<th>72</th>
</tr>
</thead>
<tbody>
<tr>
<td>No previous exposure to injection molding</td>
<td>79</td>
<td></td>
</tr>
<tr>
<td>Forging</td>
<td>72</td>
<td></td>
</tr>
</tbody>
</table>

(Table 5, “No previous”) included foreign students and transfer students, who had never heard of injection molding and had never been exposed to design for injection molding concepts. After using the Injection Molding Tutor for about 45 minutes, a quantitative evaluation test was administered. The average score achieved by students who two years earlier had been exposed to injection molding was 80 percent. The average score achieved by student with no previous exposure was 79 percent. However, this latter group included two students who received exceptionally low scores and obviously had not clearly read the problem. If these two scores are dropped, the average score of this latter group becomes 85 percent. Again, the software seems to be able to replace several lectures and homework assignments, even among engineering majors.

Junior engineering majors were also evaluated on their understanding of forging concepts for manufacturing after using the Forging Tutor (not described here). These student received no classroom lectures and were not given reading or homework problems dealing with the subject of forging. The overall average of all students was 72 percent. This subject is more complicated than injection molding and so the lower scores were expected. However, the results indicate that the concepts of design for forging, that is knowledge of the combinations of geometry and material selection for manufacturing, can be achieved without the need for standard classroom lectures and assignments.

3.4 Adaptive Courses

The final tutor customized an existing set of video-taped courses (“traditional” MANIC) for an individual student, using an overlay student model that recorded student ability and preferences.
Traditional MANIC (Multimedia Asynchronous Networked Individualized Courseware) delivered audio and HTML versions of lecture slides over the World Wide Web and began to take advantage of the increased interactivity and flexibility provided by the Web as compared to usual lectures. Web-based courses can be considerably more interactive than existing computer-based courses, allowing students to take more control over their learning. For example, in traditional MANIC, students had several options for viewing course material (see Figure 19, left). They controlled the speed, direction, and linearity of both the slides and audio playback. They played the audio from the beginning of the course to the end, or could stop and start the audio and slides, or “randomly” traverse the material using the table of contents provided as a guide. A more detailed description of the traditional MANIC can be found in [Stern et al., 1997a, 1997b].

3 This work is supported by NSF Grant DUE-9813654, by earlier grants from NSF/DARPA and Apple Computer Technology Reinvestment Award (CDA-9408607), NSF awards CDA-9502639 and NCR-9508274, and a University of Massachusetts graduate fellowship.
However, this traditional version of MANIC did not take advantage of all the benefits of web-based delivery. Students still felt they were in a lecture hall and thus a passive role in their learning. MANIC Tutor was designed to give students a more active role in their learning. Although the audio and slides for MANIC Tutor were still taken from existing video-taped courses, the tutor’s courses were not simple direct translations of those courses. By examining which slides were seen and which quizzes taken, the student model in the tutor determined a student’s learning, as explained below, and guided a student through the material, dynamically generating course content and constructing interactive and adaptive quizzes at the appropriate level of difficulty. Generated course content was accomplished by using adaptive hypertext techniques including adaptive navigation support and adaptive content.
MANIC Tutor provided an interface similar to the traditional MANIC, using slides and audio. Students traversed the course using either the “next” and “previous” buttons to linearly proceed through a topic or they used the table of contents to randomly jump to another point in the course. However, since the topic structure was not linear, adaptive navigation support was supplied; when a student reached the end of one topic and guidance was available to choose the next topic. Additionally, MANIC Tutor provided stretchtext, which allowed the student to see more detailed information about the content by clicking on the text to get more information associated with that text.

**Domain Organization.** Lectures, and therefore video-tapes from lectures, are linear from start to finish. However, the on-line version of a course need not be. MANIC Tutor stored topics in a simple semantic network, with links indicating the relationship between a topic and its pre-topics (connected topics preceding the topic in the semantic net). Links represented prerequisite, co-requisite, and related topics, similar to the link types used by Intelligent Guide [Carr & Goldstein, 1977].

Each topic was divided into subtopics, which were themselves topics. These subtopics allowed the tutor to reason at a finer level about the student’s knowledge. When a topic was displayed, a static set of material, consisting of a linear ordering of pieces of text or graphics called content objects was presented to every student. As the topic was presented, it was broken into pages, or “slides,” containing these content objects.

In addition to the topic structure, there were also keywords, or concepts, that were part of the domain. These concepts appeared throughout the content objects which taught a topic. As the tutor presented the topics, and a concept word appeared, it dynamically chose additional content for that concept, for example, extra explanations, graphics and definitions. Each piece of additional content had a level of difficulty assigned by a domain expert, indicating the level of knowledge needed to understand that information.

The tutor also reasoned about where to put page breaks, based on how much additional content was being shown. When displaying a slide, only one screen’s worth of material was presented at a time, to
prevent making the student scroll. However, before the slides were to be displayed to the student, the tutor did not know how much supplementary information it would choose to show (see the adaptive content section for how the tutor makes these decisions). Therefore, as the tutor presented a topic, it dynamically decided where to put page breaks after deciding how much supplementary material to give.

**Motivation for Sequencing Curriculum.** In non-linear domains, users might find themselves “lost in hyperspace” [Brusilovsky et al., 1996]. An intelligent guide can be useful in avoiding this problem. To this end, the MANIC Tutor selected relevant topics on the basis of the current student model [Anderson, 1990; Anderson & Reiser, 1985] and helped sequence the curriculum topics based on a two part process: selecting relevant topics on the basis of the current student model [Anderson, 1990; Anderson & Reiser, 1985]. It was assumed that a student was “ready to learn” a topic only if he performed sufficiently well on its pre-topics. The problem thus decomposed into determining: 1) how well the student had performed on topics in the course, called the “learned” score; and 2) how to combine the ratings for all the pretopics, as well as the rating for the current topic, called the “ready” score.

Traditional intelligent tutoring systems determined how well a topic was learned by examining quizzes and tests. In open environments like the web, additional information was needed because students were free to explore without being required to take tests. For example, items considered to determine how well a topic was learned included: the student's access pattern for viewing the course material, e.g., time spent studying a topic and whether topics were reviewed multiple times.

**Learned score.** Three factors were important in grading a student’s knowledge of a topic: how well the student performed on quizzes, how well the topic was studied, and how much the topic was reviewed. These three pieces of evidence were combined to determine how well a topic was “learned.”

Quizzes were a good source of information about a student's knowledge and were considered important for the student model. Because quizzes provided such concrete evidence of a student’s knowledge, students were required to take quizzes on all topics in the domain. Quizzes were dynamically
constructed from a question database and covered the topics most recently completed, as well as topics that should be reviewed. Each question had a level of difficulty as determined by the course instructor [Anderson & Reiser, 1985; Boyle & Encarnacion, 1994; Brusilovsky & Pesin, 1994; Brusilovsky et al., 1996] indicating the level of mastery a student should have to answer the question correctly. This level of difficulty was used to update the student model after the student answered the questions posed. Clearly, correctly answering a harder question demonstrated a higher ability than correctly answering an easier one. Similarly, failing at a harder question was not as damaging as failing at an easier one.

Students using MANIC spent most of the time reading and listening to course material. Therefore, some judgement was made about the comprehension a student gained through these activities. The problem of determining if a student understood the material became one of judging how sufficiently each content object in the topic was studied. The premise was that students who did not spend enough time studying were not learning the material well and students spending too much time were having difficulty understanding the material. In order to assess the time spent on a content object, the tutor plotted the amount of time spent studying the object using a normal curve, with the mean and standard deviation determined by the course instructor before the course was presented to students. Once the content objects were plotted for a student and the new scores obtained, they were averaged over all content objects in the topic. This became the studied score for the topic.

This measure was flawed in at least two ways. First, it used time, which is an inherently inexact measure in assessing educational effort. If the student left the room for five minutes, the tutor did not detect this. Second, differences in individual learners needed to be considered. Spending a certain amount of time studying material did not imply comprehension. For example, some people read slower than others but comprehend just as well. The tutor needed a way to learn the optimal time for each

---

4 Previous work in Interbook [Brusilovsky & Schwarz, 1997] also used information about pages read to update the student model.
student as he used the course. Future research involves improving the metrics for measuring how well material has been studied and understood.

Additionally, a student often studied a topic multiple times. The reviewed score on a topic recorded how many times the student returned to visit the same topic again. In general, if he reviewed frequently, then perhaps he did not retain information sufficiently and thus did not learn the material. Of course frequent reviewing might reflect individual differences, which were not taken into account.

The three scores on a topic, (i.e., quizzed, studied and reviewed) were combined into a single score to determine how well a topic was learned. A weighted average of the three individual scores was used, giving the most weight to quizzes.

**Selecting the next topic.** Once each topic’s “learned” score was calculated, a “ready” score for other topics was determined. This score determined which topic the student should study next, based on how well the pre-topics were learned. Rules that adjusted the pre-topics’ learned scores took into consideration the link types between a topic and its pretopics in the semantic network. Each link type had a threshold, indicating the minimum score for mastery of the pre-topic. The weights were adjusted based on how close the learned score was to the threshold. Scaling rules were used to give more weight to different kinds of relationships. Once the weights of links were determined, the ranking on the topic was computed by averaging the adjusted link weights of the pre-topics of the topic in question.

When the student linearly progressed through a topic and came to the end of that topic, he had the option of letting MANIC Tutor choose the next topic. Topics which were not sufficiently learned had a higher priority over new topics to study. Thus among topics to be repeated, the one with the lowest “learned” value was chosen as the next topic to study.

If no topics needed to be repeated, the tutor’s goal became guiding the student forward through the curriculum. To fulfill this goal, the “next” topics from the current topic were evaluated to see if they were “ready.” If one or more topics have “ready” values, the one with the highest value was chosen to be taught. If no such next topic existed, then the semantic net was recursively searched backwards from
these next topics, looking a previous topic with the highest “ready” value. This policy ensured that topics that could help the student move on to new topics would be taught next and, thus, momentum through the curriculum preserved.

**Adaptive Content.** The goal of adaptive content in MANIC was to provide a presentation that was not too hard nor too easy, while taking into account a student’s learning style preferences. For example, one student might have preferred pictures to textual explanations, while another preferred definitions at first but examples later on.

A two-pass method was used to determine which supplemental information should be given to each student. The first determined which content objects from the concepts were at the correct level of difficulty, taking into consideration how much a student knew about the concept. The second determined student preference and took into consideration how the student preferred to learn.

The tutor analyzed how well the student knew the concept when deciding which supplemental content objects were at the correct level of difficulty. To do this, the tutor examined how well the concept had been learned. Each concept had a mastery value for each level of difficulty. The tutor simply determined the highest level of difficulty the student had mastered. A level of difficulty is said to be mastered if its mastery value is greater than a threshold, in this case 0.85 (on a 0 to 1 scale). The tutor then chose those content objects that were at the same level of difficulty as the student's highest mastered level of difficulty.

Both quizzes and time spent studying an object were used to update the mastery values for a concept. When a quiz was taken or a content object seen, the mastery value for each level of difficulty of the concept was updated, based on regression equations designed by a domain expert. To create these equations, the expert was given prior values of a level of difficulty mastery value and asked what the next values should be under the various circumstances. For example, if the student's level 0 mastery value was \( x \), and a level 2 object had been seen, what should the student's new level 0 mastery be? For each such
circumstance, the expert provided 5 pre- and post-action pairs. We then used polynomial regression to fit a curve to those points.

A machine learning algorithm was used to decide which additional objects to present to the student. The tutor chose objects that matched the student’s preferences as he demonstrated on previous pages, from among those objects at the correct level of difficulty. Each content object had a set of abilities, including instructional type (e.g., definition, explanation, example), media type (e.g., picture, text), place in concept (e.g., beginning, end), and wanted (e.g., yes, no). The tutor deduced which abilities the student liked to see and which he did not, by analyzing which objects the student had elected to view or hide in previous browsing activities. Those objects comprised the Naïve Bayes Classifier’s example space.

When a concept was to be presented, the tutor examined the content objects at the right level of difficulty and used the Naïve Bayes Classifier to predict if the object would be wanted or not. If the classifier returned a “yes” answer, the object was shown; otherwise, it was not. It should be noted that if an object was shown, the student was given the option to hide the object. Similarly, hidden objects had an option that allowed them to be shown.

**MANIC Tutor Architecture.** The architecture consisted of four main parts: the client, the HTTP server, the port server, and the student model servers (see Figure 20). The client consisted of a web browser (Netscape® Navigator was the preferred browser) and a control applet that allowed the student to traverse the course material. The applet contained buttons such as "next page" and "glossary." Also, the RealAudio® plugin was embedded in the applet.

The HTTP server used Common Gateway Interface (CGI) scripts to interact with the port server and with the student model servers. The port server controlled the creation of the student model servers; one server was created for each student using the system. When a student first logged on to the system, the port server was contacted in order to get a port for this student. This was the port the student used each time he logged in. The port server spawned a student model server on that designated port. These servers ran continuously until a long period of interactivity, at which point they terminated. By running
continuously, the student model server maintained state. This architecture was chosen so that the tutor did not have to rebuild state each time the student performed an action.

![The MANIC Tutor Architecture](image)

**Figure 20: The MANIC Tutor Architecture**

The main communication link in the architecture was between the HTTP server and a student model server. The HTTP server used "cookies" to maintain state with a given client and thus knew which student model server to contact for each interaction. Cookies stored the student's name, the server's IP address and the port on which the student's server was listening.

Once the HTTP server contacted a student model server, it simply waited for the student model server to send back a reply, which was in the form of HTML code. The HTTP server then sent the information received from the student model server directly to the web browser. Thus the HTTP server contained no intelligence at all.
Student model servers were the elements of the system that performed all of the “reasoning” and dynamic construction of course content. Each time the student made an action, his student model server was connected both to log that action and to generate the content as a consequence of the action. The HTML seen by the student was generated dynamically by his student model server.

4 Evaluation of Intelligent Tutoring Systems

Formal evaluation of student performance with intelligent tutors has shown that tutors can achieve performance improvement that approaches the improvement afforded by one-on-one human tutoring compared to classroom teaching [Bloom, 1984]. Several success stories indicate that computer tutors effectively reduce by one-third to one-half the time required for learning. In the past 10 years a series of careful, comprehensive reviews of automated instruction have been conducted [Fletcher & Rockway, 1986; Fletcher, 1988, 1995, 1996; Fletcher & Orlansky, 1986; Fletcher et al., 1990; Johnston & Fletcher, 1995; Lesgold et al., 1990, 1992; Seidel & Park, 1995; Shute, 1995; Shute & Psotka, 1995; Wiggs & Seidel, 1987]. In these studies, traditional computer aided instruction (CAI) effectively raised average student performance from 50 percent to 65 percent, reflecting a 15 percent improvement on the median of the Gaussian curve which typically results when evaluating student performance. Traditional computer instruction commonly achieves an improvement in teaching effectiveness in this range and it is now considered to be routinely obtainable, although it is not universally obtained. However, the addition of AI technology, such as described in this chapter, based on 3 studies (not from this chapter) raised average student performance from 50 percent to 84 percent, reflecting a 34 percent increase in performance due to the tutors [Fletcher, 1995; Regian & Shute, 1992, 1993, 1994; Regian, 1997]. Given the small sample of tutor studies, the tutor results are suggestive rather than authoritative. All things considered, it is safe to say that CAI works quite well in a variety of settings and for lots of different instructional domains. It is
also safe to say that tutors work well, may be considerably more effective than CAI, and could be enormously more effective.

Automated instruction also supports students to achieve a given performance level in a shorter period of time. Thus it is useful in reducing the time required to achieve some desired level of performance [Regian, 1997]. In these studies, students were required to reach predetermined instructional objectives, and allowed to spend more or less time doing so. Traditional computer aided instruction resulted in an average instructional time reduction of 29 percent compared to lectures. These results are commonly reported and routinely obtainable. Tutoring systems, using AI techniques, based on 3 studies, resulted in an average instructional time reduction of 55 percent [Regian, 1997]. Again, given the small sample of tutor studies, the tutor results are suggestive. It is safe to say that traditional computer based instructional systems can reduce the time required to reach instructional objectives, in a variety of settings and for lots of different instructional domains. Tutoring systems can also reduce instructional time, and probably more so than traditional computer-based instructional systems.

In one special case, students working with an Air Force electronics troubleshooting tutor for only 20 hours gained proficiency equivalent to that of trainees with 40 months (almost 4 years) of on-the-job experience [Lesgold et al., 1992]. In another example, students using the LISP tutor at Carnegie Mellon University [Anderson, 1990; Anderson & Reiser, 1985] completed programming exercises in 30 percent less time than those receiving traditional classroom instruction and scored 43 percent higher on the final exam.

Although limited but encouraging results have been shown for tutors, the fact remains that classroom tests often do not provide a measure of success for these systems because the material presented is not the same as that taught in a traditional classroom, and a system's content cannot easily be integrated into a traditional curriculum. For instance, both the original Geometry and Algebra Tutors [Anderson et al., 1985, 1995] automated much of the symbol manipulation, e.g., addition and multiplication, in the domain and provided an environment for students to learn problem solving, whereas the original classroom curriculum focused on symbol manipulation. The systems, in part, redefined the
curriculum content to focus on problem solving, and this made evaluation of learning outcomes difficult. Current versions of the Algebra Tutor, especially PAT (Pump Algebra Tutor) were designed using content guidance from national standards and working closely with classroom teachers. They have achieved a one standard deviation effect in the classroom as well as improvement in the range of 50-100% on the new standards assessment [Koedinger, in this book]. PAT is now the most widely used tutor in K-12 schools and will be in over 300 high school or 1% of the USA high school market in Fall 2000.

Several aspects of evaluation remain major research issues. For example, if a system succeeds, which components should be assigned the credit, and how might the various models be fine-tuned to improve the next generation of systems? Portability between subject matter has been shown in only a few systems [Anderson et al., 1995]; most systems are less effective when rebuilt in another domain, and generalizability has yet to be demonstrated.

5 Research Issues

One crucial issue of tutor research and development is the need for continuous refinement of system behavior based on computer-student performance. That is, results from one iteration are needed to inform and constrain development of the next version if we are to see improvement in the field: working tutors should foster clear refinement and evaluation of AI and cognitive theories and vice-versa. There is nothing so practical as a good theory. However, too little theory currently guides development of new tutors. Additionally the field has many research issues to address, as discussed below.

Cognitive Modeling. Cognitive modeling is now making rich contributions to progress in this field. It is applied to develop pedagogical and subject matter theories, design instruction and present instruction [Regian & Shute, 1992, 1993]. Expert models and student models have achieved the most direct benefit from cognitive modeling thus far, including substantial benefits from modeling subject matter experts.
For instance, Anderson et al. [1990] attribute much of the success of their tutors to the cognitive task analysis of experts in LISP, geometry and algebra.

Instructional modeling, i.e. the actual presentation of instruction, is the area where cognitive modeling has thus far found the least fruitful application, mostly due to a historical accident. Working with classroom teachers and trainers needs to be more common place during research and development of these systems.

**Communication Modeling.** Developing systems that are sensitive to student idiosyncrasies and able to customize their language responses is still very difficult. Achieving flexible mixed dialog between human and machine, whether text- or visually-based, is a current goal in tutoring systems [Graesser et al., 1995; Freedman & Evans, 1996; Freedman et al., 2000]. One early success, e.g.; SOPHIE [Brown et al., 1982], suggested this was achievable, yet advances in reasoning about language as well as statistical and semantic analysis were required before we could achieve natural dialogue. Human-human dialog succeeds despite ambiguity and digressions because both participants model the dialog, subject matter and other speaker and both actively change models and language when working towards success of the dialog. Studies show that even naïve human tutors, including non-experts and fellow students, are successful in part because they use general tutoring strategies [Graesser & Person, 1994]. Several outstanding research projects have shown that portions of the dialog task are feasible. Suthers et al. [1992] showed that responsive explanations can be planned and generated dynamically, and Lester and Porter [1996] produced human-like complex explanations. This suggests that continuing efforts be made to enhance the machine's ability to do its part to model the user and dialog context. Building responsive intelligent interfaces requires building mechanisms to support cooperative dialog and developing a deeper understanding from the viewpoint of the learner.

Choosing and organizing domain knowledge for the communication effort provides the next set of research issues. Control should account for the tutor's ability to switch strategies dynamically according to multiple constraints and to do so in a manner that is sensitive to abilities that human tutors use in tutorial
interactions. Further work is required to research relevant topics, hints and help, especially when multiple perspectives of the topic are available.

**Adequate Models.** Other research issues center on development of adequate models of the student, the pedagogical context and the recognition of how to stimulate the student's own abilities and creativity. Although much is known about student motivation and cognitive development, it is mostly general knowledge about which activities engage particular students and how novice behavior is distinguished from expert behavior [Larkin et al., 1980; Chi et al., 1981]. This field requires information at a finer grain size. Thus we need knowledge about particular student actions in specific contexts. If a student behaved in a certain way, in the last half hour, can we predict his or her behavior (time on a problem or success in completing the problem) on the next problem.

A separate issue concerns how relevant knowledge should be presented once it has been selected. Presentations, whether explanations or examples, should be customized for the learning needs of the user and should enable her to build on existing knowledge. A model of the teacher might include strategies for providing remediation for errors as well as a selection of examples, analogies, and strategies -- including when to remediate, when to provide examples, etc. Teaching strategies of expert teachers are not well understood at the level of granularity needed. A communication model might include dialog and teaching strategies, e.g., which graphic to present, how to phase a hint, which would be couched within principles of good interface design.

**Implementation Issues.** Though some success has been seen in the development of intelligent tutors, one might ask why more systems have not been deployed. The answer is that deep design and implementation issues remain, beginning with the lack of AI development tools, e.g., shells and frameworks, similar to those used to build expert systems. Expert systems shells have sped up development of industrial-strength systems such as systems to evaluate a person’s likelihood of repaying a house mortgage [McDonald et al., 1997], to schedule rail for a large railroad [Murphy et al., 1997], and to configure a passenger aircraft cabin [Kopisch & Gunter, 1992]. Though several AI tutoring shells have
been built [Murray, 1998; Van Marcke, 1998; Major et al., 1997], none has wide usage or has shown the ability to scale up properly. Perhaps authoring tools for tutoring systems will become available as part of the growth of authoring tools for the Internet. Development tools would facilitate large-scale development; a simple tool, such as a simulation tied to an expert system or to a lock-step tutor, might be a practical way for designers to get started on a path of incremental design through feedback from the expert and student. A developer should be able to interact with a variety of tools, in much the same way that a computer artist develops an animation by using one package to model, another to animate, and yet another to add special effects or edit in music and voice. Each tool adds separate functionality and each interoperates smoothly with the others to produce the final product.

Another reason for slow development has been the inability to reduce cognitive task analysis to engineering practice. An excessive amount of time is still required to analyze each task to the depth required for building a tutor. The use of new knowledge representations should result in greater expressive power than that offered by first-generation expert system tools. For instance, qualitative representations might be used to represent domain knowledge [Forbus, K., Citation in this book].

Social and cultural issues provide the final set of barriers to producing generalizable and pervasively deployed tutors. A transition towards student-active discovery or inquiry-based instruction has begun [McNeal & D’Avanzo, in press; Weaver, 1989], and tutors such as described here would play a large role in such classrooms for supporting discovery and open-ended investigations. However, this instructional transition is slow and traditional didactic, lecture-based teaching remains the norm. Although one-on-one tutoring is in fact very old [Plato, 1922; Gordon & Gordon, 1990], its reintroduction through this technology represents a radical change for some educators. This research field can help revive an old tradition of tutoring that is largely inquiry-driven in nature. Since educational change works very slowly, the new technology will require curricular and infrastructure changes as computer-based systems are integrated into learning environments.
6 Discussion

Much success has been achieved, and yet more remains to be done to support the effective use of AI technology in education. Current instructional technology research has succeeded in exploring many domains and some non-traditional pedagogical strategies, such as partnering, mentoring and scaffolding. However, many of the rich and detailed tutoring methods used by talented teachers, such as mentoring and inquiry-based teaching and collaborative learning, still elude researchers, and the next generation of tutors will require development of accessible shells and test-beds to facilitate further experimentation and expansion.

Several predictions can be made about future development of intelligent teaching systems based on near-term goals and long-term opportunities. As the computer price-performance ratio continues to improve, a wide expansion of tutoring systems will continue to be seen in new teaching and training arenas. Initiatives incorporated within these systems, such as qualitative reasoning, machine learning, case-based reasoning, general purpose architectures and multimedia, will facilitate the study of human learning and teaching and will accelerate the power of tutors, their acceptance in the classroom and the willingness of developers to build them.

A paradigm shift has begun in some classrooms in which the educator is no longer the ‘sage on the stage’ and has become the ‘guide on the side’ assisting students in their navigation through remote libraries, museums, databases or institutional archives. The planet has come on-line and yet there is still more information than knowledge available in this global electronic network. Intelligent agents and explainers are needed to retrieve, construct, understand and modify conceptual models for individual learners. Agents will support intelligent retrieval, configure themselves to the learner based on current goals and provide a learning focus. The global information infrastructure enables easy access to information, and artificial intelligence is needed to play a central role in bringing this knowledge to a new level of realism and usefulness for students. The tutors described in this chapter provide merely the foundational technology for what might become powerful and globally sophisticated personal tutors.
References


Fletcher, J.D. (1995). What have we learned about computer based instruction in military training? *NATO Research Study Group, RSG 16: Workshop on Lessons Learned*.


