INTELLIGENT TUTORING SYSTEMS

The advent of increasingly powerful and inexpensive computer hardware late in this century has enabled the development (and, in some cases, deployment) of advanced, computer-based instructional software tools based on the principles of artificial intelligence. Unlike the early computer-assisted instruction (CAI) systems that preceded them, most of which were offshoots of Skinnerian ideas about programmed instruction (1,2) and contained canned knowledge of experts in a given domain, these newer, intelligent systems aimed to embody the domain expertise itself (3). That is, rather than simply deliver instruction, they aimed to *generate* instruction (4), tailoring it to individual students' needs (5). Indeed, many such systems have aimed to use the computer as a kind of "cognitive microscope" (6), revealing the processes occurring in the mind of the human learner. Although such systems are more costly to produce than CAI systems, in time, money, and effort (7), their benefits often outweigh "the costs of the I in the ITS" (8).

Such tools have appeared over the last two decades under various labels, including: intelligent computer-assisted instruction (ICAI) systems; intelligent tutoring systems (ITSs); microworlds or discovery worlds; coached apprenticeship systems; reactive learning environments; and, more broadly, intelligent learning environments (ILEs). Different theories of learning or instruction underlie the various systems, from production-system models of individual instruction (7) to theories of cognitive apprenticeship and situated cognition, often involving groups (9). Among the most salient differences between the various pedagogical approaches are the type, amount, timing, and structure of the feedback from the system's intelligent agent(s) to the user(s).

In general, the availability of artificial intelligence tools has permitted a major shift in the nature of computer-based training and education. Prior to the appearance of intelligent tools, computer-based instruction consisted mostly of either pure didactic exposition or simplistic scoring of student answers to verbally posed questions. In recent years, simulations driven by numerical algorithms and using fancy graphics have also appeared (e.g., variations on the SimCity game series). Intelligent (i.e., knowledge-based) systems now support several improvements in computer-based learning:

- The computer can more deeply evaluate a student's performance.
- The computer can solve a problem itself and compare its solution to the student's, offering advice, critique, and modeling of better ways to perform.
- The computer can assess a student's strengths and weaknesses and select learning opportunities (e.g., problems to solve) that best fit the student's current knowledge state.
- The computer can explain why an expert might attack a problem differently.

Because intelligent computer-based training and education is in a rather early stage of evolution, there are many cases of prototype and demonstration systems but few cases of fielded and practical systems. Of those, few have undergone substantial evaluations. This article surveys a subset of intelligent tools that have undergone some form of evaluation. The list of systems reviewed, while certainly not exhaustive, is representative of the existing tools for which evaluation results were readily available. Overall, these systems provide a good sense of the intelligent learning tools developed to date, and the evaluations we summarize help establish the state of the art in intelligent learning system technology.

Some of the questions we consider in this article include the following: To the extent that individual systems have been deemed helpful for teaching the target domain or skills, in what ways do they help? What aspects of the system's various components help or hinder the user's learning? To what extent does feedback from the system help? How well does the system foster individual learning versus collaboration among multiple users, when appropriate?

Given the range of disciplines and tasks for which ILEs have been built, it is difficult to compare approaches to ILE design and delivery of system feedback without accounting for the types of skills they support. Clancey (10) describes how problem solving operators and inference procedures differ across various domains. McKendree (11) suggests that more complex or ambiguous tasks may require a greater degree of informative feedback than more constrained ones, for which more directive feedback often suffices. Others in the field have described the process of learning from an ILE as a four-way interaction of learner style, desired knowledge outcome, type of instructional environment, and subject matter (12). Choice of subject matter has also been shown to influence the relative effects of human tutoring (13), and may also affect the results of intelligent system evaluations (6). Thus, our review categorizes systems by subject matter domain, wherever possible.

SOME BACKGROUND

Tutoring

Many of the intelligent learning environments developed to date, most notably the ITSs, focus on the benefits of one-on-one tutoring. The reasons for this are simple: studies of human tutoring have shown achievement advantages of up to two standard deviations over traditional classroom instruction (14). While effect sizes vary with the subject matter [e.g., higher for mathematics and lower for reading (13)], human tutoring is superior not only to traditional instruction but also to other classroom improvements upon it such as mastery learning (14).

In an effort to reap the same benefits, early ITS developers sought to build systems that mimicked, equaled, or even surpassed human tutors (5,7,15). For example, the CIRCSIM-Tutor system, a combined ITS and simulation of the human circulatory system, has been modeled after the patterns of hinting behaviors exhibited by humans tutoring medical students about cardiovascular physiology (16). However, obtaining sufficient information to draw meaningful comparisons is often difficult because the processes underlying expert human teaching or tutoring are not well understood (17,18). For example, the extent to which expert human tutors do extensive student diagnosis has been widely debated (5,16,19,20). Partly because of this, and partly because of the limited bandwidth in human-computer interaction (21), student diagnoses by successful ILEs also vary in their degree of detail. Nevertheless, the clear benefits of human tutorial guidance have led many system developers to try to emulate characteristics of human tutors.

System Components

The majority of ILEs we surveyed are complex systems, made up of a number of different components or modules. Most ITSs are comprised of four components: a domain knowledge or expert module, a student model, a tutoring or pedagogical module, and a user interface (22). ILEs may contain other components as well, including a simulation environment (e.g., 23,24); a learning component, for systems that employ machine-learning techniques to improve themselves (2,18); and a control component, to coordinate all the other components (18,23). Although often regarded as discrete units, the various system components are usually interrelated in function and often in features (25). We describe briefly each of the four most common ILE components, those of an ITS.

The expert module of an ITS contains the target knowledge of the domain that the system is designed to teach. This knowledge may be declarative (concepts, system models, etc.) or procedural, or both (20). Ideally, this knowledge has been verified by human domain experts, who either examine the knowledge base or interact with prototypes of the learning system (18). In most cases the expert module consists of a working model capable of solving target problems in the domain (3). Such expert models may differ in the degree to which their problem solving processes correspond to those of human problem solvers (3,26). For instructional purposes, a model that can explain what it is doing is much more useful than a model with optimal, but unexplainable behavior. So, for example, an expert model consisting of a probability network with no conceptual underpinnings would not be very helpful for instruction, since humans cannot readily assimilate complex systems of probabilities.

Student modeling components, which diagnose students' emerging competence, appear in several forms. An *overlay* model represents a student's knowledge as a subset of the expert model's knowledge. Overlays can reveal those pieces of expert knowledge that are either missing or misapplied in a student's mind, but they cannot capture student knowledge that is qualitatively different from an expert's (10,21). So, for example, an overlay model for mechanics could represent that a student does not know that objects maintain their velocities absent outside forces, but it would not be able to capture that a student uses the term acceleration in ways that subsume parts of acceleration and parts of velocity.

Another type of model matches student errors to a *bug library* of common misconceptions in the domain. Such models can detect not only the correct knowledge students do not possess but also much of the incorrect knowledge they do possess (10,21). Some bug libraries are comprised of prespecified bugs, while others generate models of student bugs from a set of underlying principles (3,10,21). Some other models try to simulate the student's learning processes, relying on the difficult assumption that human and computer learning processes can be equated (2). However, Reusser (20) notes that "machine tutoring based on cognitive simulation of the student is still not possible across a full range of tasks and in open-ended domains."

Student models diagnose in different ways, including *knowledge tracing* [keeping track of which skills or pieces of knowledge a student has mastered (27)], *model tracing* [matching a student's solution steps to those of an expert problem solving model (5)], *plan recognition* (inferring a student's plan based on subgoals accomplished so far), and *decision trees* (21). Generally, student diagnosis is more difficult in student-controlled systems than in systems that maintain some degree of control over the interaction (26); but even in the latter systems, student models should diagnose conservatively.

While expert knowledge is fully in tune with real-world systems and hence constrained by them, students sometimes begin with very divergent and idiosyncratic beliefs and backgrounds. Consequently, it is easy to misdiagnose a student's misunderstandings, making a conservative approach to student diagnosis appropriate (6). Developers must consider the dangers of misdiagnosis (28), as well as the validity and reliability of computer-generated diagnoses (18). A variety of measures are available for evaluating student models' diagnostic success (29). It is also possible for a model simply to query the student to resolve ambiguities in diagnosis (10).

The pedagogical module structures the interaction between the system and the user, deciding what task material to present and what kind of feedback to provide, if any (20). Its behavior depends upon the domain knowledge and student modeling components (18). Pedagogical styles can differ along such nonorthogonal dimensions as prescriptive versus discovery learning, tutoring versus coaching, and student-directed versus system-directed (19,20). Some pedagogical approaches are directive (7), some are noninterventionist (19), and some are in between, such as the cognitive apprenticeship teaching methods of scaffolding and fading (9,28). Thus, feedback from a system can play any number of roles, from corrective to regulative to informative (3,30), and it may differ in relative amount and timing (e.g., immediate vs. delayed vs. on demand (1,5,28,31)). The pedagogical module is best evaluated against standards of its underlying instructional theory or of expert human teachers or tutors in the respective domain (5,18).

Although often de-emphasized relative to the other modules, the user interface component of an ILE is critical to its success or failure (3,6,10,18). In addition to being the means by which the user and the system communicate, the interface can also function as an external memory for the user, reducing his or her cognitive load (6). Common screen interfaces may contain text, hypertext, graphics, or even animation. While graphical interfaces are generally more engaging than text-based ones (26), in some domains they have been found to be disadvantageous (32). Some interfaces are capable of adapting to individual users, although resultant changes in screen displays may cause confusion (6). In some systems, the interface may be rich enough to also play a pedagogical role, in which case the separate effects of the interface and the system's pedagogy may be difficult to ascertain (5,6).

Evaluation

Why Evaluate?. While the specifications of a declarative knowledge or CAI system can be validated via careful inspection by experts in relevant field(s), this type of validation is insufficient for complex systems such as ILEs (18). Generally there is no way to guarantee that such a complex system does what it purports to do unless its effects can be demonstrated in the behavior of actual users in the target population (33). One radical view of evaluation in the educational technology field is "to move towards a situation where we have reliable, precise theories from which systems may be formally derived which will produce the learning benefits predicted by the theories" (34), thereby eliminating the need for evaluation. However, we have not yet reached that utopian position.

There are other, more specific reasons for evaluating an ILE. A system under development should be tested in the field to assess "its actual impact on a broad array of teacher and student behaviors" (35). Developers must assess not only the effectiveness of a system but also the likelihood that it will be fully accepted into the work or school culture of the target audience (36).

Formative versus Summative. The ultimate validation of a learning tool involves formal, controlled experiments that follow a scientific methodology (6,37). However, the literature shows that there are many fewer controlled evaluations than systems (3,15,38). In most cases, a summative type of evaluation is reserved for completed systems. Some have argued that such evaluations are inappropriate because formal summative techniques for systems as complex as ILEs do not yet exist (18,23). While global effects can readily be measured, we have insufficient tools for validating specific mechanisms within a system or for verifying exactly what is learned from a system. Thus, many developers prefer to conduct formative, internal evaluations. Although much more informal and less rigorous than summative evaluations, formative evaluations can produce results of much greater detail (18).

Such detail is especially important during system development. Many developers use formative evaluation studies for rapid prototyping and incremental improvement of system parts. In early stages of system development, a "Wizard of Oz" (6) approach is used, in which humans simulate missing system components (18). Techniques used in formative evaluations include additive design comparisons, to assess the impact of various components on the overall effectiveness of the ILE; and lag sequential analysis, to measure the impact of system feedback on the user (29). Even pilot evaluation of completed systems is usually formative in nature (38), often progressing from laboratory sessions to field testing (18). Whereas the purpose of a summative evaluation is to validate a system's advantages, a formative evaluation should emphasize weaknesses and other negative aspects, so that they can be corrected (6).

Assessing Educational Impact. An ILE evaluation may address a variety of issues. Developers and researchers may ask how much users gain by interacting with the system, while teachers, administrators, and other outside parties may ask whether users learn the same things as students of more traditional instruction (37). Corporate trainers may focus on transfer, the extent to which the system not only trains workers for a specific job but also prepares them to quickly learn other jobs that are closely (near transfer) or more distantly (far transfer) related. In addition to transfer, an evaluation may measure attitude change about computers (39).

Mark and Greer (18) list several criteria for evaluating an ILE's effects on achievement and affect. Achievement measures include transfer, retention, time to mastery, and dropout rates. Affective measures include student motivation (including the intrinsic motivation of computer use), self-esteem measures, and time on task. While most achievement measures are objective, many affective ones are subjective and usually gathered by questionnaire.

SYSTEMS CATEGORIZED BY DOMAIN/SKILL

In the sections that follow, we present a number of examples of intelligent learning tools for various subject matters, commenting on any evaluation data that are available.

Programming

Many of the ILEs we reviewed were designed to teach computer programming. This was an attractive domain for early intelligent tutoring system efforts. The subject matter was familiar to developers (2), the procedures to be taught were well defined, and programming novices were readily available. Most of these systems teach only the basic, introductory elements of a programming language, with some covering the content of a one-semester course.

The LISP (LIST Processing) Tutor. One of the most well-known and thoroughly evaluated ILEs is the CMU (Carnegie Mellon University) LISP Tutor, also known in various incarnations as GREATERP (4), LISPITS (27), and the GRAPES LISP Tutor (31). The system is an ITS that has been used to teach introductory LISP programming, both in laboratory studies and in one-semester college courses. Its design principles are based on Anderson's ACT* theory of cognitive skill acquisition (7,11,27,40). It contains a problem-solving expert component, a bug catalog, and a tutoring module for assessing student knowledge, assigning appropriate problems, and providing feedback (4). Problem-solving rules are represented as productions (IF-THEN statements) in GRAPES (Goal Restricted Production System) (41), and the system uses model tracing to diagnose student solution plans (7) as well as knowledge tracing to select appropriate problems for students to solve (27). In essence, the student tries to write programs prescribed by the tutor, and the tutor intervenes with advice whenever the student's activity deviates from what the expert model would do. The interface includes a structured editor using LISP templates, so that the student does not have to concentrate on checking syntax. Whenever the student makes an erroneous step, the system intervenes with immediate feedback.

Two early evaluations of the LISP Tutor (7) clearly demonstrated its educational effectiveness. One study compared groups of students learning LISP from a human tutor, from the ITS, or on their own. Although post-test scores were equivalent across groups, the human-tutored and ITS-tutored groups took significantly less time to cover the material. The second study found that ITS-tutored students took less time and scored better on a final exam than control students working on their own (7,27). Students in the studies liked the tutor and rated it as superior to traditional programming courses. Results showed that while a human tutor was still best, the ITS was a close second, far ahead of classroom instruction (4). The performance and mastery time data were consistent with the 1.0 effect size found in evaluations of some other extensive ITSs (42).

Corbett and Anderson (27) found that students who had received more explanatory feedback from the ITS made fewer errors per goal than those who had received less explanatory feedback, but did no better on post-tests. They also found that students had equal post-test performance but longer solution times when they controlled the timing of feedback presentation than when the system did, suggesting that students either were more careful about making errors or were spending more time detecting and correcting them (27). Students rarely requested immediate feedback from the ITS; most of them wanted feedback only when they were finished coding a problem (43,44).

GIL. Another LISP tutor, GIL [Graphical Instruction in LISP (45)], was built using many of the same principles as the CMU tutor. One difference is a graphical interface. This allows students to learn programming concepts without having to deal with syntax concurrently, and it imposes a lower cognitive load than a text-based system. The basic idea is to describe a program as a graph of processes that connect inputs to outputs. With this format, students can construct program graphs both forward from a problem's input(s) and backward from its output, thus eliminating the top-down, left-to-right constraint found in the LISP Tutor. Another improvement is that GIL's rule base can be used not only to solve LISP problems but also to explain to the student why a particular step is appropriate in a given situation.

In a study comparing four versions of GIL with different degrees of feedback (46), undergraduate LISP novices wrote program graphs with GIL and then completed a post-test based on elements of similar problems. Students who had received greater degrees of feedback tended to commit fewer errors, to immediately fix them more often, and to request system help less often than those in the minimal and delayed feedback conditions. They also scored significantly higher on the post-test. Thus, unlike in Corbett and Anderson's (27) feedback study, GIL students who received the most informative feedback both solved the LISP problems better *and* apparently learned the material better (46).

A visually explicit interface alone can serve pedagogical functions, even in a system with little or no tutoring. Another study (cited in Ref. 5) compared students using the standard version of GIL to those using an exploratory version without model tracing. Although the exploratory students took twice as long as the standard GIL students to complete the training problems, they scored as well as them on post-tests.

ADAPT. Another programming system, ADAPT (ADA Packages Tool), was designed to teach a second language to programmers already experienced in Pascal or C (47). Since syntax from these prior languages shows positive transfer to ADA but solution planning shows negative transfer, the focus with ADAPT is on planning rather than syntax. ADAPT's user interface includes plan menus, from which plan components are chosen. Some of the plans are buggy; immediate feedback is delivered when one of these is chosen. ADAPT is more flexible than the LISP Tutor, generally allowing planning of steps in any order and enforcing top-down, left-to-right order only at the coding level. In a formative evaluation study (47), six undergraduates who knew both Pascal and C solved problems using ADAPT. As in the coding-order manipulation studies with the LISP Tutor (27,44), students did not exercise control over the order of step planning; for the most part, they developed plans in the sequence in which they appeared in the interface. Positive transfer of prior syntax to ADA was found, as expected; and on the few ADA syntax errors that students did commit, the system's error-location feedback was usually sufficient for them to be corrected immediately (47).

EGO. Ego is an ITS that teaches Gries' methodology for developing programs and proofs in parallel (48). Although the system teaches program writing, it helps focus students on the overall methodology by helping them with algebra, logic, and code syntax. The system also contains a context-sensitive advisor, available on demand. Ego allows students to make and correct their own errors, although the system can intervene to prevent excessive drift down a bad solution path. The system maintains an overlay student model and a goal library, and each goal module incorporates both tutorial knowledge and a bug catalog specific to that goal. Ego thus can utilize different teaching strategies as appropriate in the context of particular goals. Unlike many other programming tutors, its interface allows a student to undo a bad path back to the origin of the error.

PROUST. PROUST is an ITS for beginning Pascal students that identifies and provides feedback on program bugs. Unlike the other programming tutors we reviewed, PROUST accepts only complete, syntactically correct programs. The system is noninteractive, and does not tutor on writing correct code per se, but rather compares a student's submitted code to its library of plans, identifying bugs (3,15). While noninteractiveness may seem like a limitation of the system, recall that students in one of Corbett and Anderson's (44) studies requested feedback only when they had *finished* coding their programs. This form of tutoring is consistent with the revised ACT-R theory's claim that students can learn from complete problem solving products, not just from correcting erroneous steps en route to complete solutions (43). In addition, structured editors are available to facilitate the writing of syntactically correct program code; therefore, PROUST's input constraints are not extraordinarily difficult to satisfy. In general, tutoring based upon completed problems will likely work so long as the student is afforded enough help to assure problem completion.

Summary. The programming ILEs we reviewed differ along several dimensions. Some teach or enforce correct syntax, while others focus more on solution planning than coding syntax. Some impose a strict order on program development, while others are more flexible and allow users to choose the order in which they work (even if they usually opt not to exercise their choice). Some utilize model tracing to provide immediate feedback, keeping students on-path during solution coding; one provides delayed feedback, allowing students to commit and correct errors; and one delivers feedback only on completed programs. Of those ILEs that provide immediate feedback, most provide information on at least the location of coding errors, if not more explanatory feedback. Generally, in each of the studies comparing degrees of feedback, more was found to be better than less.

Geometry/Diagrammatic Reasoning

After programming, the ITS community began to focus on mathematics instruction and then on a number of other areas. Geometry received considerable attention, partly because it relies more heavily on reasoning with diagrams and partly because it was found that the proof process could be understood more readily when presented graphically as the task of finding a path from premises to conclusions. Conveniently, geometry proofs start with a conclusion to prove, and involve the relatively simple operators of forward and backward inference (10).

The Geometry Tutor. The Geometry Tutor (also known as GPTutor or just GPT), also inspired by the ACT* theory, teaches students how to construct geometric proofs (7,11,35,49). Much like GIL's program-graph interface, the Geometry Tutor's interface uses graph-like displays (proof trees) to represent both directions of inference and to reduce students' cognitive loads (40). The tutor intervenes immediately if students attempt illegal inferences, but will allow them to proceed with any legal inference, even if it is not on a solution path included in its expert model (7).

In a formal evaluation study (49), GPTutor students averaged a letter grade higher on exams than the control class. In another study (11), using a version modified to treat legal but nonoptimal inferences as illegal moves, high-school students who received only minimal feedback on their moves made significantly more errors per proof on a post-test than those receiving any combination of goal and condition-violation feedback, and also tended to immediately fix fewer of their errors. These results are consistent with those found in the aforementioned feedback study with the GIL programming tutor (46).

ANGLE. ANGLE (A New Geometry Learning Environment) is a more recent tutor based on principles similar to the GPTutor, but further emphasizing the diagrammatic reasoning inherent in expert geometry planning (33). Diagrams also facilitate *novice* problem solving by making subgoals explicit and reducing cognitive load, in domains including geometry (11), propositional calculus (6), programming (45), argumentation (50,51), and certain procedural task simulations (39).

A formative study of the ITS in geometry classes showed that ANGLE students tended to solve more proofs on a post-test than control group students, although the results varied by teacher. However, in another study, ANGLE students made more execution errors than GPT students on a post-test. This may reflect ANGLE's higher emphasis on tutoring proof planning over execution, or possibly ANGLE's more flexible interface, which does not enforce a planning approach (52).

Summary. Both of the geometry ILEs share many features with the LISP Tutor and GIL, including diagnosis by model tracing and the graphical nature of the latter's interface. The tutors differed from each other in the extent of their emphasis on planning, as with the programming ILEs. As in the programming domain, minimal feedback on errors was less beneficial than more informative feedback, with goal-related feedback being the most valuable.

Algebra

PAT. PAT (Practical Algebra Tutor) is another ITS based on the ACT theories of skill acquisition (53). It was designed specifically to support a new algebra curriculum, produced by the Pittsburgh Urban Mathematics Project (PUMP), emphasizing real-world problems. PAT users work on word problems using tables, graphs, and symbolic equations, while the system does model tracing using correct and buggy rules. The system also does knowledge tracing, displaying cumulative skills acquired by the user in a "Skillometer" window. In an admittedly confounded formative evaluation of PAT (53), algebra classes used PAT plus the new curriculum, in comparison to control classes using the traditional curriculum without the tutor. The PAT students scored 100% better than controls on tests of the new curriculum, but also 15% better on standardized tests of the traditional curriculum. The tutor has been integrated into many of the ninth-grade algebra classes that implement the new curriculum and is being adopted by districts elsewhere in the country.

RAND Algebra Tutor. Another algebra tutor, developed by the RAND Corporation (37), has a less sophisticated student model, based on the number of problems tried and solved. Although it can sort problems based on student ability (via a form of knowledge tracing), and provide hints and answers to student questions, it cannot tailor its feedback to individual student characteristics. Field evaluations of multiple versions of the tutor with different pedagogical goals for teaching high-level skills have met with mixed success (37).

Pixie. Different instructional strategies were also compared in the Pixie algebra tutor (cited in Ref. 2). Developers exposed students to two pedagogical strategies, Model Based Remediation (akin to human or ITS tutoring) and Reteaching (akin to CAI). Although each strategy led to superior performance over control students, who received error notification only, performance with both strategies was equivalent. Although the developers concluded that ITS effects were similar to those obtained with traditional CAI approaches, their instructional manipulation was on too small a sample and for too short a duration for meaningful comparisons to be drawn (42). When the amount of content conveyed and the criterion for evaluation are both minimal, there is no reason to expect intelligent systems to excel over older approaches.

Summary. The ACT-based algebra ILE (which used both model and knowledge tracing) demonstrated tangible benefits for its users. Studies of the other two systems, however, had equivocal results. Possible reasons for this include division of focus between different pedagogical goals and strategies, less effective diagnosis (via knowledge tracing alone in one case), and problems with experimental manipulations and with cultural and other field factors.

Other Mathematical Skills

GIDE. GIDE (54) is a goal-based diagnostic system for problem solving in algebra and statistics. It assumes that problem-solving errors are systematic, and that they must be considered in the context of a student's solution plan in order to permit "intention-based" diagnosis (54), similar to PROUST. GIDE uses buggy plans and rules as necessary for diagnosis, and attributes missing or skipped steps in student solutions to inferred prerequisite knowledge or other such conceptual dependencies. Evaluations were conducted of two implementations of the system, one for statistics and one for algebra word problems (54). GIDE-Stat was able to recognize almost all of the goals, including implicit ones, in problem solutions from students in an introductory statistics course. It also identified most of the missing goals in students' errors, but the implicit inference engine was *too* powerful, often attributing known or acquired concepts to students where an expert instructor would not. GIDE-Algebra's evaluation was more extensive, involving thousands of problem solutions. The system made interpretable diagnoses for all of the solutions, and its diagnoses for a random subset of the solutions corresponded highly with those of human raters.

POSIT. POSIT (Process-Oriented Subtraction Interface for Tutoring) teaches subtraction by presenting declarative knowledge from a bug library to correct student errors. The statements it presents are based on the system's error diagnoses. In a formative evaluation, POSIT correctly diagnosed most student errors, and misdiagnosed fewer errors than it failed to diagnose at all. In a summative evaluation, time to mastery was shortest with a human tutor and longest with classroom instruction, with the ILE group in between. The effect size for POSIT relative to classroom instruction was 0.74 (29).

Summary. The ILEs in this section cover the domains of subtraction and diagnosis in algebra and statistics. Each system's diagnoses were extraordinarily successful. One reason for this could be the relative simplicity and formality of both the problem solving operators and the inference procedures in these domains, in comparison to programming and to natural domains such as physics and medical diagnosis (10).

Electrical and Economic Law Induction

MHO. The MHO system teaches basic principles of electricity, using an overlay student model based on mastery of a collection of curriculum issues or "bites," each of which was a simple concept or a combination of concepts taught earlier (3). It does knowledge tracing to select issues to address in problems it presents to students, based on prerequisite knowledge mastered thus far (3). A summative evaluation study (39) compared two versions of the tutor, one that provided rule-application feedback (i.e., it told students directly which principles to apply) and one with rule-induction feedback (i.e., students had to induce principles based on the relevant variables). More exploratory students did better if they received rule-induction feedback, and less exploratory students did better with the rule-application version.

Voltaville. Voltaville is a microworld for learning about electric circuits via self-directed experimentation in a computer-based circuit laboratory (55). There is no direct instruction by the system. Rather, students try to discover as many electric laws as possible, which they submit to the system for feedback. Voltaville returns feedback both on the hypothesized laws and on the sufficiency of the evidence gathered in support of them. In an evaluation study, undergraduate students' electrical knowledge improved from pre-test to post-test, and they discovered most of the laws included in the microworld. Students who showed the most improvement with the system were better at algebra and on learning indicators such as data management, devising correct hypotheses, controlling variables, and interpreting evidence in their simulated experiments (55).

Smithtown. Smithtown (12) is a similar microworld, using essentially the same interface as Voltaville, for law induction in the domain of microeconomics. It also has no fixed curriculum, and seeks only to coach scientific inquiry skills via a form of knowledge tracing (3). While interacting with the system, users can alternate between simply observing the effects of manipulated variables (exploratory mode) and devising and testing hypotheses about them (experiment mode). Smithtown's coach intervenes when a student exhibits buggy behaviors while in experiment mode, but remains silent while the student is in exploratory mode. The coach's intervention threshold can be modified to present anything from immediate feedback to complete silence.

In one experiment, Shute and Glaser (12) compared undergraduates in an introductory economics course using Smithtown, receiving only classroom instruction, or receiving no economics instruction at all (control). Both treatment groups outperformed the control group; and although Smithtown did not teach economics directly, students who used it did as well as classroom students on post-tests, after having spent less than half of the latter group's time on task. The better Smithtown students were differentiated by learning and performance indicators similar to those in the Voltaville study (55). Shute and Glaser (12) ran a second experiment on a larger sample (over 500), of military recruits. Better learning correlated with more hypothesis-driven learning indicators; results from less-able learners showed them to be limited to data-driven indicators.

Summary. The ILEs reviewed in this section were much less directive than many of the aforementioned systems, precluding model tracing in the purest sense (3). However, these systems were able to get a lot of mileage out of knowledge tracing; evaluations of each system showed at least some learning benefits for its users. The extent of benefits varied with user ability or practice. Given the less powerful forms of student modeling employed in exploratory environments, this is not surprising.

Operative Skill

GT-VITA. GT-VITA [Georgia Tech Visual and Inspectable Tutor and Assistant (23)] is a tutoring architecture for training NASA satellite ground controllers. It teaches operative skill, or "how to use declarative and procedural knowledge to manage complex systems in real time" (23), using a cognitive apprenticeship approach. The system includes an operator function model (OFM) to teach and evaluate operative skill, and an associated expert system (OFMspert) for presenting context-sensitive advice. The system also has a pedagogy module to provide immediate feedback (early in an interaction), coaching at critical checkpoints in a task (later in an interaction), and other "lesson objects" (23). The system also combines overlay and buggy student models for diagnosis. In a field evaluation (23), the tutoring architecture was used in the context of a payload-operations control center by novice satellite ground controllers. Students had difficulty at first with some of the operational skill demands during real-time satellite pass simulations; however, they did well on all declarative and most procedural prerequisites, and eventually on the essential operational skill measures. Students rated the system highly, and based on the system's effectiveness, NASA has adopted a newer version of the architecture to be used in required ground control training (23).

ALM. The ALM (Advanced Learning for Mobile subscriber Equipment) tutor (29) is a system for training operative skill for Army communications equipment. It uses an overlay student model and provides advice when students commit procedural errors. ALM's advice templates cover anywhere from five to seven procedural steps, which may impose an excessive cognitive load. As a potential remedy, developers created MALM, a modified version of the tutor, in which each advice template covers only one procedural step. In an informal evaluation, initial error rates were comparable with both tutors, as were error and correction rates immediately following advice. However, MALM led to better performance that persisted, whereas the benefits of ALM's advice lasted through only a brief interval (29). Thus, MALM's simplified feedback worked better than ALM's more complicated feedback, suggesting that the cognitive load imposed by the latter was nontrivial.

Summary. These systems differ from those reviewed previously in that they add real-time complexities to already complex tasks. Thus, cognitive load is even more a concern than with the other domains discussed above. ALM's developers were able to achieve improved performance by simply reducing the amount of feedback presented by the system at any given time. However, because GT-VITA employed different types of feedback at different times, as well as different forms of student modeling, it is difficult to determine which of its complex features are most effective or which could be best improved.

Miscellaneous Systems

Andes. Andes is a tutoring system for teaching university physics (56). Students use the system to solve physics problems and study example problem solutions. Andes uses quantitative as well as qualitative physics problems, which are less likely to perpetuate students' misconceptions than traditional quantitative problems because their focus is not on algebraic manipulation [56; see also (6)]. Based on results of a pilot study in which the majority of students' requests for help were made when they were lost (56), Andes' student model has been extended to do diagnosis by plan recognition. Its coaching component has been billed as "the first computer tutor aiming to improve learning by guiding self-explanation" (57), the extent of which is gauged by using a "poor man's eyetracker" (56) to measure students' reading times for various elements of example problems. Andes provides immediate error-flagging feedback on a student's problem-solving steps, except when such feedback could lead to error correction via simple guessing; in such cases, Andes instead presents questions focusing on the student's reasoning.

VCR Tutor. Mark and Greer (8) developed a device tutor to teach VCR programming. They created four versions of their VCR Tutor with different pedagogical approaches, to examine the role of knowledgeable feedback on a task as predominantly procedural as programming a VCR. Tutorial interactions ranged from simply forcing the user through a predetermined programming procedure, to giving error notification feedback, to giving informative feedback drawn from a conceptual model of the task. Only the most informative version employed student modeling and error diagnosis, using a bug catalog. Students who had used the most informative version had fewer steps, errors, and error types, and did marginally better on all other post-test measures than students who had used any of the other three versions. Thus, although each of the four versions was sufficient for teaching VCR programming, knowledgeable feedback led to performance advantages at no additional cost in training time. This is consistent with McKendree's (11) finding that "tasks that are quite constrained may not require maximally informative feedback, but even these tasks may be learned at least as effectively given the more informative feedback."

CATO. CATO (50) is an ILE designed to teach beginning law students to argue with cases. Despite the limited feedback it generates, the system fosters argumentation skill via an interface that reifies argument structure and helps to manage the complexity that a solely text-based system would present. It provides a set of core argument moves that students can use, and a hierarchy of factors that represent similarities and differences between legal cases. A controlled evaluation study (50) using first-year law students found no differences between groups on either a pre-test or a post-test of basic argument skills, but the control group did better on a more advanced, memo-writing assignment. The evaluators concluded that CATO was able to improve basic argument skills as much as the traditional instructor, but because its method of teaching was not holistic like the instructor's, it was unable to prepare students for more integrative tasks such as memo writing (50).

Turbinia-Vyasa. Turbinia-Vyasa is an instructional system for training operators to troubleshoot failures in marine steam power plants (24). It includes an intelligent tutor and a domain simulator with high degrees of dynamic, structural, and temporal fidelity to power plants on naval vessels. As with many engineering applications, such plants are quite complex and contain many interrelated subsystems, making component failures difficult to troubleshoot. To help minimize the operator's cognitive load during training, as well as the computational requirements for representing such a complex system, the simulator employs qualitative approximations of system states rather than numerical values. The tutor includes highly organized system and troubleshooting knowledge, including limited case-based diagnostic knowledge linking symptoms to components. A student model keeps track of students' failure hypotheses, and the tutor also uses a record of students' actions to infer their misconceptions about the plant system. The tutor responds immediately to student queries and also provides feedback when it infers a student misconception, either immediately or at the end of the training session depending on context. At session's end students can also review correct problem solutions with explanations.

An experiment compared groups of Naval ROTC cadets trained with the simulator and active (system-initiated), passive (student-initiated), or no tutoring. Both tutored groups learned to formulate and test failure hypotheses well, while the untutored group mainly used guessing. Some students became overly dependent on the active tutor's feedback, using it to evaluate their hypotheses instead of theorizing and seeking evidence themselves. In another experiment, groups of cadets solved troubleshooting problems with some combination of diagnostic cost and time limits, to better reflect fidelity of interaction in real-world diagnostic tasks. While the unlimited group was most successful, cadets subjected to *both* limits were second best. The imposed limits induced them to abandon bottom-up, experimenter strategies in favor of more efficient, top-down, theorist strategies (24).

Sherlock. Sherlock is an extensively evaluated ILE for training a technical job in avionics troubleshooting. Specifically, it is a "computer-coached practice environment" (17) that combines intelligent coaching facilities with a realistic work-environment simulation, emphasizing the latter over the former and over student modeling precision (58). Sherlock II, the most recent incarnation of the ILE, utilizes hierarchical, fuzzy student modeling variables that approximate an overlay onto both its expert and curriculum models (59). Sherlock I provided both conceptual and procedural hints on demand, on an ascending scale of explicitness (58). Hint levels were matched to the current student model (59), and were faded as the student became more proficient (17), in order to keep students "in the position of almost knowing what to do but having to stretch their knowledge just a little in order to keep going" (36). Sherlock II added facilities to support reflective follow-up after problem solving, including goal-related presentations such as intelligent replays of problem solving steps, critiques of those steps, and information about what an expert might have done (60). These capabilities were added to help compensate for the learning opportunities that are precluded by the high cognitive effort expended during problem solving (36,61), as well as to coach situations in which students were able to solve the problems but did so in a nonoptimal way (62).

Sherlock follows a cognitive apprenticeship approach of holistic instruction, rather than pacing students through a series of separate lessons (17,58). Because traditional on-the-job training often spans many years, Sherlock accelerates the skill-acquisition process (62). A benchmark study showed that trainees with 20–25 hours of experience with Sherlock I performed at a level equivalent to that of technicians with four years of on-the-job experience, with 90% retention of performance gains after six months (36,59). Thus, even relatively brief interactions with Sherlock produced troubleshooting skills that were durable (36).

In one controlled evaluation (17), using groups of airmen matched for ability on pre-tests, Sherlock students solved significantly more problems, used more expertlike problem solving steps, and executed fewer bad steps than a control group. Use of Sherlock I led to more expert troubleshooting solutions in fewer steps, for both low-ability and high-ability students (17). An evaluation of Sherlock II was conducted with Air Force master and apprentice technicians (62). Again, experimental and control groups were matched for ability, based on verbal troubleshooting tests. The Sherlock students scored higher on post-tests both of standard avionics troubleshooting tasks and of tasks involving another, fictitious troubleshooting system, thus showing transfer to novel troubleshooting tasks. Control students performed many more nonoptimal solution steps, such as swapping of electronic components, in both troubleshooting environments than tutored students and master technicians. Sherlock I effect sizes for some post-test measures were greater than 1.0. Effect sizes as great as 2.0 were obtained in the evaluations of Sherlock II (60).

Clearly, both generations of Sherlock have been successful. However, because Sherlock incorporates multiple elements and instructional strategies, it is difficult to attribute its success to any one of them (17,36). Its developers have proposed ways of partialing out its effectiveness, such as applying elements of Sherlock to other domains or gauging Sherlock's effectiveness with certain elements or pedagogical strategies removed. However, they concede that "it may be inevitable that successful training requires confounding of approaches" (17).

Collaborative Systems

Beginning with the advent of serious interest in computer-supported collaborative work (CSCW) in the late 1980s, a recent trend in the design of ILEs is to support collaboration

(19). Collaborative systems offer many potential benefits over single-user systems. One benefit is cost-effectiveness; more students may be able to use fewer computers at the same time (25,39). Students working together in groups may be able to diagnose and teach each other, relieving the computer of such burdens as diagnosis and natural language parsing (39) as well as exposing all students in a group to alternative hypotheses and multiple perspectives (63,64).

Some collaborative systems involve "pseudo-social" interactions between a human learner and computerized agents (65). Developers may envision the human and the computer as a single system (65) and evaluate the various elements of their software in terms of how they foster activity in the entire human-computer system (63).

HERON. HERON (20), named for a Greek mathematician, is a graphics-based tool for helping students in grades three through nine solve mathematics story problems. It supports the use of graphical solution-trees for problem representation and planning, in which students can link problem concept-nodes using arithmetic operators to map out a solution plan, working forward or backward. Erroneous input will cause HERON to intervene, based not on student diagnosis but on solution-tree content. A field evaluation (20) compared pairs of fifth-graders solving story problems with HERON to pairs working without HERON. The system improved story problem comprehension and solutions on a post-test, promoted useful dialogue among the pairs, and was liked by both students and teachers.

GDSS. Alavi (66) conducted a study using a group decision support system (GDSS) with classes of MBA students. The software included tools for brainstorming; categorizing and ranking ideas; and scoring, rating, and voting on alternatives. Effectiveness of collaborative learning was measured by students' self-perceptions of learning and evaluative ratings of their classroom experiences. Results showed a significant effect of GDSS use over the traditional group; it positively affected students' perceived learning and skill development, interest in the subject matter, and appraisal of the group learning exercises and overall classroom experience. GDSS students had significantly higher final exam scores than control students. It was unclear which tools or features of the GDSS had the greatest impact on group learning, and neither experimenter bias nor novelty effects could be ruled out (66).

CLARE. CLARE (Collaborative Learning and Research Environment) is a system that supports collaborative knowledge construction from published research papers (64). The elements comprising CLARE are a knowledge representation language, a process model for collaborative learning, and a hypertext-based interface that integrates them. The representation language includes node primitives to denote epistemological concepts (e.g., *claim, theory,* and *question*) as well as relationships between them. Students are led, in two phases, "from an external, isolated and individual position inward toward an internal, integrated and collaborative perspective" (64).

Most students agreed that the two-phase collaborative process model helped to promote the formation of individual views. Students also found CLARE to be useful for collaboration, for understanding research papers in a novel way, and for understanding other perspectives. Although students were not pleased with CLARE's interface, most students said that the representation language helped to expose different points of view, and that the primitives were among CLARE's most useful features. However, many students used the primitives incorrectly (e.g., stating evidence as claims, listing problems as disagreements). The evaluators note that even incorrect use of the node primitives can be useful for collaborative learning because, as indicated in their study, it stimulates discussion among students about the roles of the different concepts in understanding the research papers (64).

Belvedere. Formative evaluation studies of Belvedere (51) have shown similar patterns. Belvedere is a networked graphical environment designed to foster scientific argumentation skills in middle-schoolers. Students use node and link primitives similar to CLARE's to construct graphical argument representations of scientific problems. Problems can come from any source, but Belvedere's developers have created specialized databases about several scientific debates, which are accessible via a World Wide Web browser. Belvedere's interface was designed to resemble that of familiar drawing programs, so that students could learn to create argument diagrams with only minimal training. With both diagram sharing and chat facilities, Belvedere enables students to discuss and reflect upon their argumentation processes and products. A computerized coach is available on demand to provide guidance in developing argument diagrams (67).

Several formative evaluation studies of Belvedere were conducted with middle- and high-school students (51). Students used Belvedere's node and link primitives in ways that were inconsistent both with their intended usage and with their own and other students' usage, much like students in the CLARE evaluation. Although Belvedere's developers agreed with CLARE's that such unintended usage actually served to stimulate collaborative discussions (63), such usage can cause problems for the automated diagnostic coach.

Although the coach was still under development during the aforementioned formative evaluation studies, it has since received some empirical validation (67). In addition to syntactic node patterns, the coach is now able to respond to consistency relations between any nodes in a diagram that were copied from one of Belvedere's semantically annotated knowledge bases. Developers applied the coach to a subset of one of the knowledge bases used in the formative evaluations and found that, in most cases, its consistency judgments agreed with their own. The authors then semantically annotated that entire knowledge base, and then had the coach evaluate a diagram produced by students from one of the earlier studies. The coach agreed with all but one of the students' links; and on that particular link, the developers agreed with the coach. Thus, with only minimal knowledge engineering via semantic annotations to an existing knowledge base, developers were able to extend the capabilities of the coach to include consistency checking. Such capabilities could be useful for coaching collaboration by pointing out inconsistent relations between or within students' diagrams (67).

Adapting Existing Systems. Note that none of the previously discussed collaborative systems involves a student-modeling component. Partly this is due to the difficulties inherent in trying to maintain separate models for each student in a collaborative setting, especially when only one computerized agent is involved (26). Another reason is the viewpoint of many developers that "collaboration should be concerned with what is happening on the screen" (65), as opposed to knowledge hidden inside the learners' heads. However, diagnosis-based tutoring or coaching, as employed in many of the systems we reviewed, appears to provide substantial benefits for single users. It seems a shame to have to sacrifice such system intelligence in order to support collaboration among groups of students.

One system that has attempted to bridge this gap is Sherlock, whose developers argue that "affording students significant opportunities for collaborative learning is not going to be any harder than developing high quality computer-based systems for solo learning" (61). They somewhat circumvent the student-modeling problem by defining different roles for Sherlock's coach in a collaborative setting (9). In one scenario, Sherlock can work with a student as a problem-solving collaborator; in another, students work in groups as a single student, with Sherlock as coach.

The developers also envision situations in which Sherlock can help ease a student's transition from self-critique to peer-critique. In one, Sherlock acts as a peer during review of the student's problem solving trace, constructing explanations jointly with the student. In another, a group of students constructs such explanations, with Sherlock available on demand for assistance. In these scenarios, Sherlock is concerned not with modeling individual student knowledge but with fostering peer interaction and critique involving single or joint problem solving activity; however, it can still apply its diagnostic expertise to promote such activities (17,25,61).

DISCUSSION

In focusing on systems for which evaluation results were accessible, this article does not completely reflect the diversity of domains for which ILEs have been, and continue to be, developed. The majority of systems we reviewed support well-defined tasks and domains, many of which contain largely procedural components. This is due partially to the course of history in the field of ILE development, which began almost exclusively with such constrained domains (2) and only more recently began to steer toward more ill-defined, ambiguous tasks. Nevertheless, across the domains and systems we did review, a number of similarities and differences emerged.

With few exceptions, users liked the various systems and were motivated to use them, regardless of domain. This was borne out not only by attitude measures but also by observations of their use of the systems. Although only a subset of the systems have been judged favorably enough to be adopted by their target audiences, at least most of them have hurdled the initial barrier of capturing user interest.

Among the other similarities between systems were the beneficial aspects of their user interfaces. Across many domains, interfaces functioned as external memories for their users. This was true for both single-user and collaborative systems. Many interfaces also served some pedagogical functions, usually by design but sometimes by accident (6). Interfaces helped to reify forward and backward problem solving or inferencing not only in geometry but also in programming (GIL) and mathematics (HERON). Evaluation studies revealed unanticipated shortcomings in some interfaces, one of which (ADAPT's) precluded observation of the effects of other system components. Such findings further underscore the importance of formative evaluation to system development.

Approaches to student modeling were also similar across domains. No clear within-domain preferences for particular student modeling techniques emerged from our review. For example, it was not the case that ILEs for one domain used bug libraries exclusively while those for another domain used only overlay models. Some ILEs for programming and operative skills (Ego and GT-VITA, respectively) even used both types of model. Similarly, variants of both model tracing and knowledge tracing were used for diagnosis in programming and mathematics domains. Only in the discovery-world environments were knowledge tracing variants used exclusively, because model tracing was not feasible with such unguided instruction (3). In general, the relative effects of systems' pedagogical modules tended to vary with the specifications of their underlying instructional approaches. Among the primary characteristics that differentiate these approaches is the type of system intervention they advocate.

Immediate Feedback

A major issue in the design of interactive learning environments is the role of system-generated immediate feedback. Although used to varying extents in the operative skill tutors and in limited ways in the economics microworld, immediate feedback approaches dominated many of the tutors for programming, geometry, and mathematics, including all of the ACT* tutors. Indeed, the ACT* commitment to providing immediate feedback in its tutors is one of the theory's most controversial features (27). One reason for preferring it is to ensure that feedback is delivered in the context in which it is needed, that of the student's current goal and working memory states (27). Another reason to provide corrective feedback immediately is to prevent students from floundering while trying to recover from lengthy incorrect solution paths (11,27,40). Although the revised ACT-R theory and its newer tutorial instantiations permit off-path problem solving (43), they still focus students toward correct solution paths, and immediate feedback still plays a major role in the interaction.

However, research has shown immediate feedback to be disadvantageous in certain situations and with particular tasks (1). In one experiment using a modified LISP Tutor, students who received immediate feedback solved training problems faster than students who received delayed feedback, but when solving test problems took more time and made more errors than delayed-feedback students (31). In addition, delayed-feedback students seemed to be better at planning problem solutions than immediate-feedback students. The experimenters argued that the absence of immediate feedback in the delayed condition allowed students to redeploy their cognitive resources toward developing secondary skills such as error detection and correction. A study comparing versions of the GIL tutor (5) provides further evidence of this: Students who did not receive GIL's immediate model-tracing feedback scored better on a transfer test of program debugging skills than those who did. Thus, the value of immediate feedback seems to vary with not only the task but also the desired learning outcomes of the intervention.

Cognitive Load

Cognitive capacity must be considered when assessing the actual or potential benefits of system-generated feedback. Because information processing limitations arise more often in the student than in the computer, the cognitive load issue pervades nearly all domains and design rationales to at least some degree. Tutorial feedback is but one of many things that must compete for a user's cognitive resources while he or she is interacting with an ILE. With more complex tasks, feedback may be best left for post-problem reflection, when cognitive resources are no longer being taxed by immediate problem-solving demands (36). Perhaps, with further research, McKendree's (11) claims (see previous section) regarding the relationship between task complexity and feedback content could be extended to account for feedback timing as well.

Feedback Content

In addition to McKendree, several other researchers have conducted studies, using ILEs included in our review, which manipulated feedback content. In many of these studies, performance of users given varying amounts of feedback was compared. In some cases, minimal feedback (e.g., error notification) was sufficient to allow users to eventually solve problems. However, in nearly all cases, the maximal amount of feedback resulted in the greatest learning outcome, whether measured by test scores, time to mastery, or both. In some cases, the highest level of feedback involved goal-related information (e.g., GPTutor, GIL). Given the cognitive capacity limitations delineated earlier, this is not surprising. One can easily lose track of goal-related information while engaged in complex problem solving. Sherlock tries to compensate for this by presenting goal-related information during a reflection period after the problem has been solved.

Developers have described methods for varying the nature of system feedback content based on problem characteristics (Andes) or on the chronological point in the interaction during which one type may be more appropriate than another (GT-VITA). Researchers have discussed the prospects of fading out feedback content as users become better able to proceed without it (e.g., Andes, Sherlock). Other researchers have investigated the effects of removing tutorial feedback entirely (e.g., comparing versions of an ILE with and without feedback). A study using GIL (see previous section) showed that its model tracing feedback did have an effect beyond the pedagogical effects of its interface.

Concluding Remarks

Aside from the general similarities and differences already discussed, it is difficult to identify any domain-specific effects of, or any clear preferences between, the various approaches to providing feedback. Again, this may be due partially to the preponderance of well-defined, primarily procedural domains represented in the literature on evaluated ILEs. However, it could also be the case that a simple classification of approaches based solely on the domains or skills they involve is insufficient. Indeed, although such a classification is a useful starting point, our review suggests that perhaps other cross-domain factors, such as task complexity and priorities of learning outcomes, must be considered as well. We are inclined to agree with Shute and Glaser's (12) characterization of learning from an ILE as an interaction of many factors, including domain subject matter, targeted learning outcomes, and the type of instructional strategies employed by the system.

BIBLIOGRAPHY

1. J. A. Kulik and C.-L. C. Kulik, Timing of feedback and verbal learning, *Rev. Educ. Res.,* **58** (1): 79–97, 1988.
2. R. J. Siedel and O. C. Park, An historical perspective and a model for evaluation of intelligent tutoring systems, *J. Educ. Comput. Res.,* **10** (2): 103–128, 1994.
3. E. Wenger, *Artificial Intelligence and Tutoring Systems: Computational and Cognitive Approaches to the Communication of Knowledge,* Los Altos, CA: Morgan Kaufmann, 1987.
4. J. R. Anderson and B. J. Reiser, The LISP tutor, *Byte,* **10**: 159–175, 1985.
5. D. C. Merrill et al., Effective tutoring techniques: A comparison of human tutors and intelligent tutoring systems, *J. Learn. Sci.,* **2** (3): 277–306, 1992.
6. M. Twidale, Redressing the balance: The advantages of informal evaluation techniques for intelligent learning environments, *J. Artif. Intell. Educ.,* **4** (2/3): 155–178, 1993.
7. J. R. Anderson, C. F. Boyle, and B. J. Reiser, Intelligent tutoring systems, *Science,* **228**: 456–462, 1985.
8. M. A. Mark and J. E. Greer, The VCR Tutor: Evaluating instructional effectiveness, *Proc. 13th Annu. Conf. Cogni. Sci. Soc.,* 1991, pp. 564–569.
9. J. S. Brown, A. Collins, and P. Duguid, Situated cognition and the culture of learning, *Educ. Researcher,* **18** (1): 32–41, 1989.
10. W. J. Clancey, Qualitative student models, *Annu. Rev. Comput. Sci.,* **1**: 381–450, 1986.
11. J. McKendree, Effective feedback content for tutoring complex skills, *Hum. Comput. Interact.,* **5** (4): 381–413, 1990.
12. V. Shute and R. Glaser, A large scale evaluation of an intelligent discovery world: Smithtown, *Interact. Learn. Environ.,* **1**: 51–77, 1990.
13. P. A. Cohen, J. A. Kulik, and C.-L. C. Kulik, Educational outcomes of tutoring: A meta-analysis of findings, *Amer. Educ. Res. J.,* **19**: 237–248, 1982.
14. B. S. Bloom, The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring, *Educ. Researcher,* **13** (6): 4–16, 1984.
15. D. C. Littman and E. Soloway, Evaluating ITSs: The cognitive science perspective, in M. C. Polson and J. J. Richardson (eds.), *Foundations of Intelligent Tutoring Systems,* Hillsdale, NJ: Erlbaum, 1988, pp. 209–242.
16. G. D. Hume et al., The use of hints as a tutorial tactic, *Proc. 15th Annu. Conf. Cogni. Sci. Soc.,* 1993, pp. 563–568.
17. S. P. Lajoie and A. Lesgold, Apprenticeship training in the workplace: Computer coached practice environment as a new form of apprenticeship, *Mach.-Mediated Learn.,* **3**: 7–28, 1989.
18. M. A. Mark and J. E. Greer, Evaluation methodologies for intelligent tutoring systems, *J. Artif. Intell. Educ.,* **4** (2/3): 129–153, 1993.
19. E. De Corte, Changing views of computer-supported learning environments for the acquisition of knowledge and thinking skills, in S. Vosniadou et al. (eds.), *International Perspectives on the Design of Technology-Supported Learning Environments,* Mahwah, NJ: Erlbaum, 1996, pp. 129–145.
20. K. Reusser, From cognitive modeling to the design of pedagogical tools, in S. Vosniadou et al. (eds.), *International Perspectives on*

the Design of Technology-Supported Learning Environments, Mahwah, NJ: Erlbaum, 1996, pp. 81–103.

21. K. VanLehn, Student modeling, in M. C. Polson and J. J. Richardson (eds.), *Foundations of Intelligent Tutoring Systems,* Hillsdale, NJ: Erlbaum, 1988, pp. 55–78.
22. M. C. Polson and J. J. Richardson (eds.), *Foundations of Intelligent Tutoring Systems,* Hillsdale, NJ: Erlbaum, 1988.
23. R. W. Chu, C. M. Mitchell, and P. M. Jones, Using the operator function model and OFMspert as the basis for an intelligent tutoring system: Towards a tutor/aid paradigm for operators of supervisory control systems, *IEEE Trans. Syst. Man Cybern.,* **25**: 1054–1075, 1995.
24. T. Govindaraj et al., Training for diagnostic problem solving in complex engineered systems: Modeling, simulation, intelligent tutors, in W. B. Rouse (ed.), *Human / Technology Interaction in Complex Systems,* vol. 8, Greenwich, CT: JAI Press, 1995, pp. 1–66.
25. S. Katz and A. Lesgold, The role of the tutor in computer-based collaborative learning situations, in S. P. Lajoie and S. J. Derry (eds.), *Computers as Cognitive Tools,* Hillsdale, NJ: Erlbaum, 1993, pp. 289–317.
26. R. R. Burton and J. S. Brown, An investigation of computer coaching for informal learning activities, in D. Sleeman and J. S. Brown (eds.), *Intelligent Tutoring Systems,* New York: Academic Press, 1982, pp. 79–98.
27. A. T. Corbett and J. R. Anderson, LISP Intelligent Tutoring System: Research in skill acquisition, in J. H. Larkin and R. W. Chabay (eds.), *Computer-Assisted Instruction and Intelligent Tutoring Systems: Shared Goals and Complementary Approaches,* Hillsdale, NJ: Erlbaum, 1992, pp. 73–109.
28. A. Collins, Design issues for learning environments, in S. Vosniadou et al. (eds.), *International Perspectives on the Design of Technology-Supported Learning Environments,* Mahwah, NJ: Erlbaum, 1996, pp. 347–361.
29. P. J. Legree, P. D. Gillis, and M. A. Orey, The quantitative evaluation of intelligent tutoring system applications: Product and process criteria, *J. Artif. Intell. Educ.,* **4** (2/3): 209–226, 1993.
30. P. M. Fischer and H. Mandl, Improvement of the acquisition of knowledge by informing feedback, in H. Mandl and A. Lesgold (eds.), *Learning Issues for Intelligent Tutoring Systems,* New York: Springer, 1988, pp. 187–241.
31. L. J. Schooler and J. R. Anderson, The disruptive potential of immediate feedback, *Proc. 12th Annu. Conf. Cogni. Sci. Soc.,* 1990, pp. 702–708.
32. M. Oliver, T. O'Shea, and P. Fung, An empirical study into which style of representation works best for learners of modal logic, in B. du Boulay and R. Mizoguchi (eds.), *Proc. AI-ED 97 World Conf. Artifi. Intell. Educ.,* Tokyo: IOS Press, 1997, pp. 371–379.
33. K. R. Koedinger and J. R. Anderson, Effective use of intelligent software in high school math classrooms, in S. Brna, S. Ohlsson, and H. Pain (eds.), *AI-ED 93: Proc. 6th World Conf. Artifi. Intell. Educ.,* Charlottesville, VA: AACE, 1993, pp. 241–248.
34. J. Self, *Computational mathetics: Towards a science of learning systems design* [online], 1995. Available www: http://www.cbl.leeds.ac.uk/~jas/cm.html
35. J. W. Schofield, D. Evans-Rhodes, and B. R. Huber, Artificial intelligence in the classroom: The impact of a computer-based tutor on teachers and students, *Social Sci. Comput. Rev.,* **8** (1): 24–41, 1990.
36. A. Lesgold, Assessment of intelligent training technology, in E. L. Baker and H. F. O'Neil, Jr. (eds.), *Technology Assessment in Education and Training,* Hillsdale, NJ: Erlbaum, 1994, pp. 97–116.
37. C. Stasz et al., An intelligent tutor for basic algebra: Perspectives on evaluation, *Instructional Views of Intelligent Computer-Assisted Instruction: Data and Issues,* Symp. Annu. Meet. Amer. Educ. Res. Association, San Francisco, March 1989.
38. V. J. Shute and J. W. Regian, Principles for evaluating intelligent tutoring systems, *J. Artif. Intell. Educ.,* **4** (2/3): 245–271, 1993.
39. J. W. Regian and V. J. Shute, Evaluating intelligent tutoring systems, in E. L. Baker and H. F. O'Neil Jr. (eds.), *Technology Assessment in Education and Training,* Hillsdale, NJ: Erlbaum, 1994, pp. 79–96.
40. J. R. Anderson et al., Cognitive principles in the design of computer tutors, *Proc. 6th Annu. Conf. Cogni. Sci. Soc.,* 1984, pp. 2–9.
41. J. R. Anderson, R. Farrell, and R. Sauers, Learning to program in LISP, *Cognit. Sci.,* **8**: 87–129, 1984.
42. P. J. Legree and P. D. Gillis, Product effectiveness evaluation criteria for intelligent tutoring systems, *J. Comput. Based Instruct.,* **18** (2): 57–62, 1991.
43. J. R. Anderson et al., Cognitive tutors: Lessons learned, *J. Learn. Sci.,* **4** (2): 167–207, 1995.
44. A. T. Corbett and J. R. Anderson, The effect of feedback control on learning to program with the Lisp tutor, *Proc. 12th Annu. Conf. Cogni. Sci. Soc.,* 1990, pp. 796–803.
45. B. J. Reiser et al., A graphical programming language interface for an intelligent LISP tutor, *Proc. CHI'88, Conf. Hum. Factors Comput. Syst.,* 1988, pp. 39–44.
46. J. W. Connelly, *An empirical investigation of the effective degrees of feedback content in GIL, an intelligent tutor for programming,* unpublished manuscript, 1989.
47. V. Fix and S. Wiedenbeck, An intelligent tool to aid students in learning second and subsequent programming languages, *Comput. Educ.,* **27** (2): 71–83, 1996.
48. F. Ng, G. Butler, and J. Kay, An intelligent tutoring system for the Dijkstra Gries methodology, *IEEE Trans. Softw. Eng.,* **21**: 415–428, 1995.
49. R. Wertheimer, The geometry proof tutor: An "intelligent" computer-based tutor in the classroom, *Math. Teacher,* **84** (4): 308–317, 1990.
50. V. Aleven and K. D. Ashley, Teaching case-based argumentation through a model and examples: Empirical evaluation of an intelligent learning environment, in B. du Boulay and R. Mizoguchi (eds.), *Proc. AI-ED 97 World Conf. Artifi. Intell. Educ.,* Tokyo: IOS Press, 1997, pp. 87–94.
51. D. Suthers et al., Belvedere: Engaging students in critical discussion of science and public policy issues, in J. Greer (ed.), *AI-ED 95: Proc. 7th World Conf. Artifi. Intell. Educ.,* Washington, DC: Association Advancement Comput. Educ., 1995, pp. 266–273.
52. K. R. Koedinger and J. R. Anderson, Reifying implicit planning in geometry: Guidelines for model-based intelligent tutoring system design, in S. P. Lajoie and S. J. Derry (eds.), *Computers as Cognitive Tools,* Hillsdale, NJ: Erlbaum, 1993, pp. 15–45.
53. K. R. Koedinger et al., Intelligent tutoring goes to school in the big city, in J. Greer (ed.), *AI-ED 95: Proc. 7th World Conf. Artifi. Intell. Educ.,* Washington, DC: Association Advancement Comput. Educ., 1995, pp. 421–428.
54. M. M. Sebrechts, From testing to training: Evaluating automated diagnosis in statistics and algebra, in C. Frasson, G. Gauthier, and G. I. McCalla (eds.), *Intelligent Tutoring Systems,* Berlin: Springer-Verlag, 1992, pp. 559–566.
55. L. Schauble et al., Causal models and experimentation strategies in scientific reasoning, *J. Learn. Sci.,* **1** (2): 201–238, 1991.
56. K. VanLehn, Conceptual and meta learning during coached problem solving, in C. Frasson, G. Gauthier, and A. Lesgold (eds.), *ITS96: Proc. 3rd Int. Conf. Intell. Tutoring Systems,* New York: Springer, 1996, pp. 29–47.
57. C. Conati, J. H. Larkin, and K. VanLehn, A computer framework to support self-explanation, in B. du Boulay and R. Mizoguchi

(eds.), *Proc. AI-ED 97 World Conf. Artifi. Intell. Educ.,* Tokyo: IOS Press, 1997, pp. 279–286.

58. A. Lesgold et al., A coached practice environment for an electronics troubleshooting job, in J. Larkin and R. Chabay (eds.), *Computer-Assisted Instruction and Intelligent Tutoring Systems: Shared Issues and Complementary Approaches,* Hillsdale, NJ: Erlbaum, 1992, pp. 201–238.
59. A. Lesgold et al., Possibilities for assesment using computer-based apprenticeship environments, in J. W. Regian and V. J. Shute (eds.), *Cognitive Approaches to Automated Instruction,* Hillsdale, NJ: Erlbaum, 1992, pp. 49–80.
60. A. Lesgold, Ideas about feedback and their implications for intelligent coached apprenticeship, *Mach.-Mediated Learn.,* **4**: 67–80, 1994.
61. A. Lesgold et al., Extensions of intelligent tutoring paradigms to support collaborative learning, in S. Dijkstra, H. P. M. Krammer, and J. J. G. van Merriënboer (eds.), *Instructional Models in Computer-Based Learning Environments,* Berlin: Springer-Verlag, 1992, pp. 291–311.
62. S. P. Gott, A. Lesgold, and R. S. Kane, Tutoring for transfer of technical competence, in S. Dijkstra et al. (eds.), *Instructional Design: Vol II: Solving Instructional Design Problems,* Mahwah, NJ: Erlbaum, 1997, pp. 221–250.
63. D. Suthers and A. Weiner, Groupware for developing critical discussion skills, in J. L. Schnase and E. L. Cunnius (eds.), *Proc. CSCL '95: 1st Int. Conf. Comput. Support Collaborative Learning,* Mahwah, NJ: Erlbaum, 1995, pp. 341–348.
64. D. Wan and P. M. Johnson, Experiences with CLARE: A computer-supported collaborative learning environment, *Int. J. Hum.-Comput. Stud.,* **41** (6): 851–879, 1994.
65. P. Dillenbourg, Distributing cognition over humans and machines, in S. Vosniadou et al. (eds.), *International Perspectives on the Design of Technology-Supported Learning Environments,* Mahwah, NJ: Erlbaum, 1996, pp. 165–183.
66. M. Alavi, Computer mediated collaborative learning: An empirical evaluation, *MIS Quart.,* **18** (2): 159–174, 1994.
67. M. Paolucci, D. Suthers, and A. Weiner, Automated advice-giving strategies for scientific inquiry, in C. Frasson, G. Gauthier, and A. Lesgold (eds.), *ITS96: Proc. 3rd Int. Conf. Intell. Tutoring Syst.,* New York: Springer, 1996, pp. 372–381.

JOHN CONNELLY
ALAN LESGOLD
University of Pittsburgh

INTELLIGENT VEHICLE HIGHWAY SYSTEMS (IVHS). See INTELLIGENT TRANSPORTATION SYSTEMS.

INTERACTIVE TELEVISION. See SET-TOP BOXES.