ITEAMS - An Intelligent Teaching Environment with Assessment Modules for Self-Study

Dan Dentinger  Matthias Scheutz
University of Notre Dame
Notre Dame, IN 46556
e-mail: {ddenting,mscheutz}@nd.edu

Abstract

We propose a system that provides a framework for an intelligent teaching environment. The strengths of the system are that it is domain independent, it is adaptive to the subject's level of difficulty, and it has an expendable set of metrics that evaluate students' assignments' performance in external systems, (e.g., neural network simulators). The evaluation of students' assignments' adaptability is accomplished through belief networks. We demonstrate the use of the system by applying it to an agent design course in which it evaluates student programs.

1 Introduction

Agent-based programming assignments that require students to specify and implement agent architectures for simulated and robotic agents have become an important aspect of courses in artificial intelligence and even more so in behavior-based robotics (e.g. [Arkin1998], [Murphy2000], [Russell & Norvig1995]). Such programming assignments can be extremely instructive, helping students appreciate the difficulties connected with making agents perform even simple tasks properly. In introductory artificial intelligence courses, they also help students to understand how AI subfields are related. Yet, such assignments require a substantial support system on the part of the instructor to be able to correct students’ approaches and give them feedback that will improve their understanding of agent architectures. Typically, several teaching assistants will have to be involved in testing and grading assignments (in addition to the main instructor) if students are to meet the instructional goals of such courses. We believe that an intelligent teaching environment that can monitor students’ progress in designing agent architectures, adapt to the students’ current level of knowledge and expertise, and to some extent assess the quality of their solutions as well as their knowledge level would be of great benefit to such courses, and likely have utility beyond.

In this paper, we introduce the architecture for our Intelligent Teaching Environment with Assessment Modules for Self-Study, called ITEAMS. After a brief summary of the functional requirements of ITEAMS, we present an overview of the ITEAMS architecture together with a functional specification of its components. Examples from
a sample teaching module developed for ITEAMS are intended to illustrate the use and workings of ITEAMS.¹ We conclude with a short discussion of the current state of implementation and the future plans of development of the ITEAMS system.

2 Motivation

Intelligent tutoring systems (ITSs) can be placed in two groups: systems that are application or field-specific [Zhou & Evens1999] and systems that can be used for multiple fields [Nkambou1999]. Since many tutoring systems are designed for a specific application, special care has been taken to make ITEAMS as general as possible to be applicable in many different subject areas, although it is currently only targeted at courses that use agent-based programming assignments.

Most recent research has focused on how to model the student and (not as much) on how to present the material to the student. Patel [Patel1996b] pointed out that current ITSs do not model the teacher and emphasized that there needs to be an increased focus on the teacher model in ITSs.

(Other researchers, besides Patel), have brought to light the need to pay more attention to the presentation of material and teaching style [Conati2002, Self1990]. Existing systems have attempted to handle multiple styles of teaching, but have typically been restricted to a limited set of styles. ITEAMS, on the other hand, allows instructors to incorporate their own style of teaching into the modules. This draws upon the instructor’s experience as an educator and exposes students to a possibly more familiar learning environment. Consequently, there is no explicit instructional model in our system, although there is support for structuring the lecture materials.

Every learning environment needs some student evaluation schema. Evaluation schemas range from neural networks [Magoulas, Papanikolaou, & Grigoriadou1999] to Bayesian networks [Bunt & Conati2002]. Others have implemented a more straightforward grading approach such as the Marker system [Patel1996a]. ITEAMS uses Bayesian networks to assess the students’ performance and knowledge level.

Assessing the quality of a student’s program that controls an agent is generally a task left to the instructors. Currently, there are no systems that attempt to evaluate the performance of students’ programs or evaluate the programs themselves. Yet there is no reason to believe that, given a performance model of a program, one cannot evaluate another program that performs the same task. ITEAMS uses belief networks to evaluate the performance of students on agent based assignments (see the section on student modeling).

3 The ITEAMS Architecture

ITEAMS consists of two major components: (1) a student model and (2) a model of the domain. The student model contains information about the students that is used in their assessment. The model of the domain contains the material for the module and

¹The teaching module is currently under development and will be used in a behavior-based robotics course taught in Spring 2003 at our institution.
possible instructional paths through the material. ITEAMS also contains several other components: (1) an external interface, (2) a learning module interface and (3) an AI component. The external interface allows ITEAMS to connect to other systems, either simulation environments or, for robotics courses, robots, and allow students to work in that environment for assignments while providing feedback to ITEAMS. The learning module interface allows instructors to prepare and structure lecture material for each learning module. Assessing the students’ performance and programming knowledge is accomplished by causal queries of belief networks.

3.1 External Interface

Since ITEAMS is currently used for courses with agent-based modeling assignments for which students define and implement agent architectures, some form of evaluation of these assignments is necessary. At minimum, our evaluation scheme uses base set of metrics that models how humans judge the visual performance of agents: the metrics, supported by the system so far, which the external system must be able to supply to ITEAMS are: (1) the number of agents, (2) the number of changes in motor commands per agent, (3) the number of collisions per agent, (4) the average velocity per agent, and (5) the distance traveled by each agent in a given time period. The metrics in ITEAMS were chosen to be similar to how a human observer would evaluate an agent. The number of agents increases the complexity of the problem since there are now “moving obstacles” and possible coordination problems. The number of changes in motor commands approximates the way a person observing an agent would perceive the visible movements. The number of collisions, average velocity of each agent and distance traveled by each agent help to approximate the perception of visible moves. These metrics are used to create the belief network for evaluating the student for a programming assignment. The choice of metrics will be discussed shortly. The external interface has been designed for any system that an instructor wants to use in conjunction with ITEAMS. Thus any additional metrics that external systems want to provide are easily added into the belief network.

3.2 Learning Module Interface

The learning module interface gives instructors the freedom to create modules containing lecture materials, questions, and assignments arranged in the instructor’s teaching style. We have defined a simple format for the learning module, which allows the instructor to design the material in terms of goals and sub-goals and associate with these goals questions and assignments of different levels of difficulty.

The instructor needs to define and structure the sequence of lecture material and goals for ITEAMS to be able to present the material to the student in a coherent manner. The arrangement is accomplished by (1) ordering the sections of the lecture material for each goal, (2) arranging the placement of quizzes and assignments, and (3) sequencing the goals. The following definitions show the format for lectures and goal definition:

<begin lecture>
<goal name>
Since questions and assignments are associated with lecture material, a quiz or assignment tag needs to be placed immediately following the associated section tag. Questions are typically in multiple choice format where each answer has some “percentage correct” associated with it, and can be specified by: (1) giving the question text, (2) what material it references, (3) the difficulty, and (4) any answers with their associated correctness. The format of assignments is similar to the format of questions. The instructor specifies programming functions instead of question text. Material is specified by providing a section name, the goal or subgoal with which it is associated, the depth of material being presented and the actual content.

In order to simplify the task of organizing material for the instructor, we have implemented a tool which graphically creates a flow diagram. An example flow diagram of a teaching module is shown in figure 1. Note that goals, dependencies, and assignments/quizzes are graphically distinguished from each other. Goals are depicted as labeled ovals, assignments and quizzes are the hexagons on the links from one goal to another, and links depict different types of dependencies in the goals.

Figure 1: Teaching Module Flow Diagram
3.3 Belief Networks in ITEAMS

There are several decisions ITEAMS must make in the course of an instructional session: (1) what level of material to present, (2) the order in which questions are presented, (3) which supporting code to present in a programming assignment, and (4) what different levels of knowledge the student has obtained. These decisions take place at different times and rely on several different aspects of students’ performances. All of these decisions are made by belief networks. These networks are specified for each teaching module by the number of possible questions, possible programming functions, number of goals included, the material being presented, and several other items. All the inferences performed are causal inferences of the form: \( P(\text{Query} | \text{Evidence}_1 \land \text{Evidence}_2 \land \cdots \land \text{Evidence}_n) \).

Since different students learn best at different levels of abstraction, instructors can define multiple levels at which the material can be presented to students. ITEAMS, in turn, will have to choose which material to present to the student at any given time, as determined by a belief net. There are currently four levels of material supported: overview, intermediate, proficient, and complete. The material for each level does not differ in content or subject matter, but increases in depth as the level increases. The decision as to which level of material to present is determined by current performance of the student for a given goal along with the performance of the student for any related goals.

![Figure 2: Example belief net to determine student knowledge](image)

The decision about which question to present to the student is based on the overall performance of the student. The overall performance of the student takes into account information in the student model, which is discussed in the next section.

“Knowledge”, which refers to students’ knowledge, is an unspecified term in figure 2. What “knowledge” refers to will depend on the course material. The belief network can assess the student’s knowledge at any level. For a behavior-based robotics course, the knowledge can be the student’s programming knowledge, architecture knowledge, background knowledge, etc.

To determine the different knowledge levels of students, a belief net is used that is composed of all the questions, programming functions, and goals the student can be presented. An example is shown in figure 2. The belief nets are constructed dynamically from the questions, goals, and assignments contained in a teaching module. The CPTs are also dynamically created from the “percent correct” value of each question’s
answers. The top three layers shown in figure 2 are only the nodes that determine the knowledge level. As will be discussed in the next section, many different pieces of information affect the conditional probability tables in the network. Some of these include: previous answers given for the current question, number of times the current question has been seen, and number of times a related goal has been reviewed, to name a few. All this information updates the conditional probability tables by either increasing or decreasing the probability that a student understands a question, goal, or programming function.

![Figure 3: Example programming function belief net](image)

Programming assignments are similar to questions in that ITEAMS must decide at which level to present the assignment to the student. If the student must write programs, the system must also decide which functions should be written and which should be provided. Figure 3 shows a belief net that is based on the belief net from figure 2. In order for ITEAMS to decide which functions the student must write, the probability of each node in the bottom layer must be computed. As the depth of the programming increase, so the level of understanding for the student. As the percentage that the student understands the material increases, the number of functions the student must write increases. Which functions students must write is also influenced by the goals in which the students performed best.

![Figure 4: Example belief net to determine knowledge level from a programming assignment](image)

Once students have finished the programming assignment and run their code in the external environment (simulated or real), ITEAMS must interpret the results of the run. This is accomplished by a belief network that is structured similar to the example in figure 4, where for every agent in “Number of Agents” there is an accompanying “observed” belief network that feeds into the knowledge-level node. Any extra information that the external system provides is included in a separate set of belief nodes.
that connect directly to the “Knowledge” node. The CPTs for these belief networks are dynamically created from the level of difficulty associated with the programming functions and the level of difficulty of the assignment. Once the knowledge level is determined from the external environment, ITEAMS must then determine the students’ level of understanding of the programming assignment.

3.4 Student Model Description

Different levels of student information are needed to make decisions within the system. The information ranges from qualitative information on the progress of the students to quantitative data used to calculate their knowledge levels. All of the information recorded is used to update the conditional probability tables of the several belief networks in ITEAMS or is part of the belief network.

3.4.1 Student Activation Record

Whenever a student performs certain actions, they get recorded. A few such actions are: switching goals, and the number of times a student switches answers to a given question. These records assist in evaluating the students’ overall performance as well as the local performance for a given goal. This is done by updating the conditional probability tables (CPTs) of the belief networks.

We believe that keeping track of a student’s switching habits can help in the assessment of knowledge. When a switch occurs from one goal to a related goal, the CPTs for both the questions and programming assignments related to the goals are updated by a positive constant to indicate that there is an increased chance that the student understands the material. The updating does not always occur, since the student could keep switching goals until the system decides the student has a very high understanding of the goals. We plan to conduct experiments to determine if keeping track of switching habits is a good idea and if so, what is the desired rate of change.

We also believe that keeping track of students’ possible answer selections for questions helps in determining their understanding of the material. In order to evaluate the students, this information is used to decrease the probability that they know the material for that question by another constant factor.

3.4.2 Student Solution Records

Knowing the history of a student’s answer to questions has been suggested as a good metric in evaluating the student’s knowledge level [Zhou & Evens1999]. There are two types of answers that the system tracks: question answers and programming solutions. A student’s answer is comprised of three parts: the answer, the quantitative value of the answer (in the case of the question), and the number of attempts the student has made. In the case of the programming assignment, the record consists of the code for a student written function, a qualitative assessment of the performance, and the number of attempts made. Given that students can review the material as often as they want, they could receive the same question more than once. In this case the CPT for that question is updated so that the answers better reflect the possibility of knowledge.
Currently we are experimenting with different values to find the proper amount to shift the probabilities.

The evaluation of a student is based on several items: (1) each question that was answered, (2) each programming function written, (3) the student’s performance in the assignments, and (4) the performance in the goals. The questions are evaluated by computing the probability that the student understands the material based on the question answer, \( p(\text{Understanding}|\text{Current question } Q) \). The programming functions and assignments are evaluated using the belief network shown in figure 4. The goals are evaluated with \( p(\text{Understanding}|\text{Current set of questions } Q \land \text{Current set of Functions } P) \).

### 3.4.3 Goals Assessment

The student model keeps track of the student’s quantitative assessment for each goal and sub goal respectively. It also keeps track of the student’s overall quantitative assessment. This allows the instructor to quickly see a student’s performance without having to re-evaluate each time.

### 3.4.4 Current State

ITEAMS also keeps track of its current state. This state consists of the current active goal, active question if the student stops in mid question, and what section of the goal is active. Keeping this information up to date allows students to exit ITEAMS at any time and then restart at the point where they left it.

### 4 Example of ITEAMS

To illustrate the interaction of students with ITEAMS, figure 5 provides a segment of a teaching module. For the particular goal depicted, a student will have to define an agent architecture that performs obstacle avoidance using the subsumption design methodology ([Brooks1986]). Note that the student could have arrived at the present goal having completed one of two other goals, namely “Simple Behaviors” or “Subsumption Architecture.” Once a student has completed the complex behavior goal, she will have to take a quiz which ITEAMS uses to assess her knowledge level. This information is then used to make informed decisions about how to proceed with the presentation of the material.

Before assigning a particular programming task, ITEAMS determines one of four levels at which the assignment can be presented and completed. The first level is intended for exposition only, since at this level the problem is described and the solution presented immediately after the problem description. At level two, the student can use predefined behaviors (e.g., blocks of code), which need to be arranged in such a way as to give rise to the overall functionality required for the solution. Note that at this level the overall architecture layout is given and blocks simply need to be inserted into empty spaces (“fill in the blank” at the component level) in that layout. At level three, students need to write code that implements blocks, while the blocks themselves are
arranged properly. And finally, at level four, students need to structure the architecture, i.e., provide the layout of the components, and implement them.

In the subsumption architecture example above, the next instructional goal that needs to be satisfied for the programming assignment is “Foraging”. In this particular case, the first level consists of a description of the “foraging” task together with a demo of an agent performing the foraging behavior. At the second level, students can arrange components implementing behaviors pertaining to `avoid_obstacles`, `move_forward`, `move_to_food`, and `detect_food`. At the third, students write the code for the four functions. Finally, at the fourth level students will have to provide a structural layout of a “forage” layer, and have to provide the implementation of the augmented finite state machines implementing the four behaviors.

5 Discussion

ITEAMS is intended to serve two main purposes: (1) it is a tool for instructors that allows them to map out their courses and define assessment criteria for determining the levels of knowledge and competence of their students for a given subject; it will also save them time supervising students while the students are working on assignments; (2) it is also a learning tool for self study for students that monitors and checks the progress of their assignments, and provides feedback based on individual assessment that is targeted at the strengths and weaknesses of each individual student. ITEAMS is currently in the implementation phase and is expected to be completed in time for the trial run in Spring 2003.

At present, ITEAMS is very much targeted at assignments that require the design and implementation of agent architectures for simulated or robotic agents. As such, the system interfaces with specially modified agent control software developed in our lab. Hence, there is considerable room for improvement at the level of the external interface to make ITEAMS work with other systems as well. Also, at the level of the student model, we see much room for improvement. For example, ITEAMS does not take “affective information” of the user into account. Affective information, however,
may prove useful in selecting particular problem sets or in providing help to students (e.g., it may be instructionally more effective to switch to a different problem set or to provide more clues for students, who are already agitated because they cannot solve their assignment). Furthermore, information about the students’ character profile (i.e., a personality model) could help in the self-adaption of the system to particular learning and working styles (e.g., a patient, pragmatically oriented student may prefer to spend more time working out details of the programming code than would a short-on-patience, conceptually oriented student).

References


