Learning Domain Knowledge for Teaching Procedural Tasks

Andrew Scholer, Jeff Rickel, Richard Angros, Jr. and W. Lewis Johnson

Information Sciences Institute & Computer Sciences Department
University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292-6695

ascholer@isi.edu - rickel@isi.edu - angros@isi.edu - johnson@isi.edu

Abstract

Providing domain knowledge needed by intelligent tutoring systems to teach a procedure to students is traditionally a difficult and time consuming task. This paper presents a system for making this process easier by allowing the automated tutor to acquire the knowledge it needs through a combination of programming by demonstration, autonomous experimentation, and direct instruction.

Introduction

Education researchers have long recognized that one-on-one tutoring is a particularly powerful method of instruction (Bloom 1984). Unfortunately, it is a highly expensive one as well. Providing every student with a teacher to watch over him or her and provide situated, interactive instruction is often impractical.

Intelligent tutoring systems (Wenger 1987, Sleeman 1982) attempt to make widespread one-on-one instruction possible by filling in for human instructors and providing some of the same types of interaction. Students can watch a tutoring system demonstrate how to perform a task or they can practice the task under the supervision of the automated tutor. Such systems should ideally allow the same kind of interactivity that a human instructor does. A student should be able to ask the system to suggest or perform a step when they become stuck and should be able to take over for the system when they feel confident with their ability to finish a task being demonstrated. However, providing the knowledge needed by an intelligent tutoring system to provide instruction in such a flexible manner represents a new burden.

For teaching a procedural task, an intelligent tutoring system must have knowledge of the task that is both robust and explicable. At any point, despite possible unexpected actions by the student, the system must be able to both select an appropriate action to take and be able to explain the reason for that selection to the student. The need for the ability to respond to a potentially large set of user actions means that scripting all of the system's actions is highly impractical at best. A better way to equip such a system with the necessary knowledge is to provide it with a model of the task, enabling it to plan how to complete the task and to both dynamically adapt the plan to new situations and to provide explanations of these adaptations.

Traditionally a programmer or knowledge engineer is required to work with a domain expert to formalize the expert's procedural knowledge of a task into code or declarative statements that the intelligent tutor can use. Unfortunately, formalizing this knowledge can be both difficult and time consuming.

Researchers have explored a number of ways to facilitate providing procedural knowledge to an intelligent agent. Systems have been designed that use higher-level instruction languages to make knowledge engineering quicker and more easily understood (Badler, 1998). Alternatively, some systems seek to eliminate traditional programming through instruction (Huffman 1994), example solutions (Wang 1996) or experimentation (Gil 1992). Most of this work focuses on providing agents with the knowledge to perform tasks as opposed to teaching the tasks. While learning how to do and how to teach how to do are similar problems, an agent that is to be an instructor has an extra requirement – it must be able to explain the rationale behind what it learns. In addition, most prior work focuses on learning from one source; each of the systems either learns inductively from actions taken in the environment, or from instruction by a human tutor. Only Pearson and Huffman (Pearson 1995) have tried to implement a system that combines learning by observation with direct instruction.

This paper describes Diligent, a system that integrates programming by demonstration, autonomous experimentation, and direct instruction to learn domain knowledge needed to teach procedural tasks. Diligent has been implemented and integrated into STEVE (Soar Training Expert For Virtual Environments), an intelligent agent that cohabits virtual environments with students to teach them procedural tasks (Rickel 1999, 2000). The next three sections of the paper describe the environment Diligent learns in, the types of knowledge it learns, and its learning methods respectively.

Diligent's Environment

The environment within which knowledge acquisition is assumed to occur consists of the domain simulation, the human instructor, and Diligent.

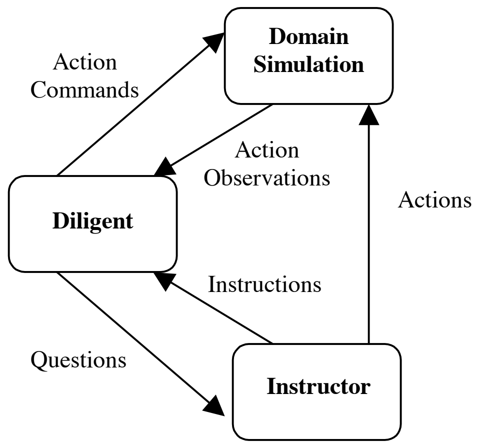

Figure 1: Environment for Learning

We assume that human students learn procedures by working with the intelligent tutoring system in a simulation of a real domain. Students can interact with this simulation by taking actions in a GUI that change the state of the domain. This same domain simulation can be used by the intelligent tutoring system to acquiring the domain knowledge to tutor students.

The domain simulator is responsible for providing an interface (e.g. the same GUI human students use) with which the human instructor can take actions within the domain. The simulator is also responsible for providing the learning system with records of actions (action observations) taken by the human expert or by the intelligent tutoring system. Each action observation consists of a pre-state, the action taken, and the effects of the action. (STEVE actually acquires the information from the simulator in a more complicated interface, but this simplified interface will suffice for this paper.) Finally, the domain simulation is responsible for accepting commands to execute actions in the simulated environment (action commands), so that Diligent can execute the same actions as the instructor.

The human instructor provides input to the learning system through instruction, allowing them to guide its actions and provide information. Instruction may take the form of simple commands, answering questions the learning system poses, and directly specifying elements of knowledge in the ontology.

The learning system itself is responsible for utilizing action observations and instruction to acquire the knowledge in our ontology for teaching procedural tasks. It is assumed that the system can ask questions of the instructor and that it can issue action commands to the domain simulation.

Ontology for Teaching Procedural Tasks

Our task ontology has three main categories of knowledge that the learning system must acquire: procedures, operators and linguistic knowledge.

Procedures

There are two main requirements for our procedure representation. First, the representation must be sufficient to choose the next appropriate action while demonstrating a task or watching a student and, if asked, to explain the role of that action in completing the task. Second, the representation must be sufficient to allow the system to adapt procedures to unexpected student actions.

To meet these requirements, we represent procedures with a procedural net (plan) consisting of a set of steps, a set of end goals, and a set each of ordering constraints and causal links between steps. Each step may either be a primitive action (e.g., press a button) or a composite action (i.e., itself a procedure). The end goals of the task simply describe a state the environment must be in for the task to be considered complete. Ordering constraints impose binary ordering relations between steps in the procedure. Finally, causal links represent the role of steps in a task; each causal link specifies that one step in the task achieves a precondition for another step in the task (or for termination of the task). For example, pulling out a dipstick achieves the goal of exposing the level indicator, which is a precondition for checking the oil level.

Such a representation is not uncommon in the AI planning community and has proven effective in a wide variety of research on task-oriented collaboration and generating procedural instructions (Delin et al. 1994, Mellish and Evans 1989, Young 1997). Used with partial order planning techniques this representation is sufficient to choose the next appropriate action while demonstrating a task or watching a student, and it also enables the system to explain the role of that action in completing the task. (Rickel 1999)

The causal links in this representation are key to both of these abilities. First, they support replanning by allowing the system to determine which parts of the task are still relevant to achieving the task's end goals. Previous tutoring systems based on procedural net representations (e.g., (Burton 82), (Munro 93), and (Rickel 88)), on the other hand, only represent steps and ordering constraints. Without causal links, which represent the role of steps in the task, these systems are incapable of adapting procedures to unexpected circumstances. Second, causal links are the foundation for generating explanations; specifically, by combining knowledge of the causal links in the plan with knowledge of which parts of the plan are still relevant, a rationale for actions or recommendations can be provided. (Rickel 1999)

Operators

Operators are used to represent the effects of actions in the domain by mapping the relation between the pre-state for an action and the effect it has. Many actions will have different outcomes depending on the state of the domain. For example, pressing a power button on a stereo will turn on the machine if it is off and will turn it off if it is already on. Thus the effect model may contain a number of different conditional effects, each one an effect the action has given a particular set of preconditions.

The system is responsible for deducing the set of preconditions that produce each conditional effect from different action observations. While learning, it is likely there will not be enough information to uniquely establish a set of preconditions for a given effect. To represent this uncertainty, the preconditions are stored in a version space (Mitchell 1977) that maintains sets of preconditions describing the most specific and general possible preconditions proven. This makes explicit both the known restrictions on what the preconditions may be as well as the remaining uncertainty at any point during learning.

Linguistic Knowledge

The intelligent tutoring system needs to be able to explain the task to students and the reasons behind decisions the system makes. For this communication to take place, the system needs to be able to describe the procedure and its components (steps, goals, ordering constraints and causal links). Diligent currently learns text fragments for the various elements of its ontology; these can be used with domain independent text templates to support natural language generation in the intelligent tutoring system. In the future, we plan to support the acquisition of more structured linguistic knowledge to support more sophisticated natural language generation techniques.

How Diligent Learns this Knowledge

Diligent, the learning module that we have developed and integrated into STEVE, begins learning about a procedure through a process of *programming by demonstration* (Cypher 1993). The human instructor issues a command for Diligent to observe his actions, and then performs the procedure by taking actions in the virtual environment. During this demonstration, Diligent receives action observation messages from the simulator, processing them to create operators and stores the list of actions as a demonstration.

Diligent records actions as a human instructor executes each step in the task being learned, noting the state of the simulated environment before and after each action. For each action observed, an operator is created to model the effects of the action; each step in the task is recorded as an instance of an operator with a particular effect.

By the time the task is complete, Diligent has learned a series of steps that can be used to perform the task. This is the first portion of the procedure in our ontology. To establish the end goals of the task, the second part of our procedure representation, Diligent uses the cumulative effect of the steps on the environment. The instructor is then allowed to review this list and remove goals that are merely side effects of the procedure.

At this point, Diligent could derive the ordering constraints and causal links from its operator models, but there would most likely be errors. For the most part, during a demonstration Diligent only sees an action taken under one set of circumstances. To refine the operators' models of actions' effects, the system needs to see the actions performed under different circumstances. To produce action observations with different preconditions than seen during the instructor's demonstration, Diligent experiments with the task in the simulated world.

These experiments are conducted by repeating the procedure once for each step in the original demonstration. Each time through, Diligent omits a different step to see how the absence of that step affects subsequent steps. These experiments are a relatively focused way for Diligent to learn on its own with a minimal amount of background knowledge of the task and environment.

Learning is performed by refining the preconditions of operators associated with each action. A modified version space maintains bounds representing the most specific and most general combinations of preconditions possible for that operator. The state of the world before each action the agent takes, and the changes that occur afterwards, are used to create new operators and update old ones. Successful applications of old operators under new conditions can be used to broaden the most specific representation of the operator's preconditions. Conversely, actions that fail to replicate the changes of an old operator may be useful in narrowing down the most general set of preconditions for an operator.

Ideally, these two bounds will meet to define the exact preconditions of an operator. However, even after experiments with a task are complete, it is likely that the general and specific sets of preconditions for each operator will not have met. Many facets of the environment will not change at all during experiments with the demonstration, making it unclear how they relate to the steps in the task being learned. For this reason, Diligent maintains a third set of preconditions, known as the heuristic set. This heuristic set is bounded by the general and specific sets and focuses on preconditions whose states change during the execution of the step or during the execution of a previous step in the task.

This set represents Diligent's assumption that the ordering of steps in a demonstration has significance. The agent assumes that the instructor has a reason for performing each step at a particular point – that effects of earlier actions are likely to be preconditions for later actions. This set gives Diligent a useful piece of knowledge from which to derive ordering constraints and causal links. Thus the heuristic set does not speed the learning of a proven set of preconditions, but improves the quality of a speedily examined set of precondtions. (For a complete discussion of how Diligent uses demonstration and experimentation to learn procedural knowledge see Angros 1997, 2000.)

After the system has finished experimenting, it can use its refined operators to generate a more accurate model of the causal links and ordering constraints between steps in the procedure. The instructor can review the knowledge Diligent has acquired for STEVE by either examining a graph of the procedure representation produced or by allowing STEVE to demonstrate its understanding of the task by trying to teach it back to the instructor. During this review, the instructor can use GUIs to refine the partial order plan produced by Diligent by adding or removing steps, ordering constraints, and causal links. Such modifications may be necessary if Diligent did not execute an action in a wide enough variety of circumstances to fully learn the preconditions of its effects.

Future Work

Although Diligent integrates programming by demonstration, autonomous experimentation, and direct instruction, they are not completely interchangeable. It is expected that portions of the knowledge acquisition will be performed by specific methods. These methods build specific support for the knowledge acquired. Because other methods used to acquire the same knowledge may not build up the same supporting information, it can be difficult to meaningfully integrate the knowledge from different sources. In particular, Diligent has a limited ability to accept instruction and to mix this form of learning with the observations that take place during the demonstration and experimentation. Current work is aimed at making Diligent more flexible, allowing the three methods to be used in any combination at any time.

References

Angros, Jr., Richard; W. Lewis Johnson; Jeff Rickel. 1997. Agents that Learn to Instruct. *AAAI Fall Symposium on Intelligent Tutoring System Authoring Tools*. Menlo Park, CA: AAAI Press.

Angros, Jr., Richard. 2000. Agents That Learn What To Instruct: Increasing the Utility of Demonstrations by Actively Trying to Understand Them. Ph.D. diss., University of Southern California.

Bloom, Benjamin S. 1984. The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher*, 13(6):4-16.

Badler, Norman; Rama Bindganavale; Juliet Bourne; Martha Palmer; Jianping Shi; and William Schuler. 1998. A Parameterized Action Representation for Virtual Human Agents. In *Proceedings of the First Workshop on Embodied Conversational Characters*, 1-8.

Burton, Richard R. 1982. Diagnosing Bugs in a Simple Procedural Skill. *Intelligent Tutoring Systems*. 157-183. Sleeman, D. and J. S. Brown, eds. Academic Press.

Cypher, A. et al., eds. 1993. *Watch What I Do: Programming by Demonstration*. Cambridge, Mass.: The MIT Press.

Delin, Judy; Anthony Hartley; Cecile Paris; Donia Scott; and Keith Vander Linden. 1994. Expressing Procedural Relationships in Multilingual Instructions. *Proceedings of the Seventh International Workshop on Natural Language Generation*.

Gil, Y. 1992. Acquiring Domain Knowledge for Planning by Experimentation. Ph.D. diss., School of Computer Science, Carnegie Mellon Univ.

Huffman, S. B.; and Laird, J. E. 1995. Flexibly instructable agents. *Journal of Artificial Intelligence Research*, 3:271-324.

Mellish, Chris; and Roger Evans. 1989. Natural Language Generation from Plans. *Computational Linguistics* 15 (4).

Mitchell, T. M. 1977. Version Spaces: A Candidate Elimination Approach to Rule Learning. *Procedings of the 5th International Joint Conference on Artificial Intelligence*, Cambridge, MA, 305-310.

Munro, A. ; M. C. Johnson; D. S. Surmon; and J. L. Wogulis. 1993. Attribute-centered simulation authoring for instruction. In *Proceedings of the AI-ED 93 World Conference of Artificial Intelligence in Education*, 82-89. Edinburgh, Scotland.

Pearson, D. J. and S. C. Huffman. 1995. Combining Learning from Instruction with Recovery from Incorrect Knowledge. *ML-95 workshop on agents that learn from other agents*.

Rickel, Jeff. 1988. An Intelligent Tutoring Framework for Task-Oriented Domains. *Procedings of the International Conference on Intelligent Tutoring Systems,* 109-115, Montreal, Canada.

Rickel, Jeff; and W. Lewis Johnson. 1999. Animated Agents for Procedural Training in Virtual Reality: Perception, Cognition, and Motor Control. *Applied Artificial Intelligence* 13: 343—382.

Rickel, Jeff; and W. Lewis Johnson. 2000. Task-Oriented Collaboration with Embodied Agents in Virtual Worlds. *Embodied Conversational Agents*. Boston: MIT Press.

Sleeman, D. and J. S. Brown, eds. 1982. *Intelligent Tutoring Systems*. Academic Press.

Wang , X. 1996. Learning Planning Operators by Observation and Practice. Ph.D. diss., School of Computer Science, Carnegie Mellon Univ.

Wenger, Etienne. 1987. *Artificial Intelligence and Tutoring Systems*. Los Altos, CA: Morgan Kaufmann.

Young, R. Michael. 1997. Generating Descriptions of Complex Activities. Ph.D. thesis, University of Pittsburgh.