Learning Domain Knowledge for Teaching Procedural Skills

Richard Angros, Jr., W. Lewis Johnson, Jeff Rickel, and Andrew Scholer
USC Information Sciences Institute and Computer Science Department
4676 Admiralty Way, Suite 1001, Marina del Rey, CA, 90292
rhangros@raytheon.com, johnson@isi.edu, rickel@isi.edu

ABSTRACT

This paper describes a method for acquiring procedural knowledge for use by pedagogical agents in interactive simulation-based learning environments. Such agents need to be able to adapt their behavior to the changing conditions of the simulated world, and respond appropriately in mixed-initiative interactions with learners. This requires a good understanding of the goals and causal dependencies in the procedures being taught. Our method, inspired by human tutorial dialog, combines direct specification, demonstration, and experimentation. The human instructor demonstrates the skill being taught, while the agent observes the effects of the procedure on the simulated world. The agent then autonomously experiments with the procedure, making modifications to it, in order to understand the role of each step in the procedure. At various points the instructor can provide clarifications, and modify the developing procedural description as needed. This method is realized in a system called Diligent, which acquires procedural knowledge for the STEVE animated pedagogical agent.

Categories and Subject Descriptors

I.2 [**Computing Methodologies**]: Artificial Intelligence

General Terms

Algorithms, Design, Experimentation, Human Factors

Keywords

Interface agents, knowledge acquisition, machine learning, pedagogical agents, programming by demonstration

1. INTRODUCTION

Computer tutors are a valuable addition to simulation-based training. By practicing in a simulation of a real work environment, students learn to perform procedural tasks (e.g., operating and maintaining complicated equipment) in a wide range of situations, without the expense and hazards of mistakes in the real world. Mentors play a valuable role in such training by demonstrating procedures, answering questions, and providing feedback on student performance. While human mentors are a scarce resource, computer tutors that can provide such guidance can be available whenever the student desires.

It is useful to design computer tutors in simulation-based training environments to act as autonomous agents, able to pursue goals in dynamic environments. A tutor should exhibit the same flexibility as learners, who will need to perform procedures flexibly and adaptively under changing conditions, instead of blindly following them by rote. Moreover, the tutor needs to be able to adapt its instruction based upon the actions and requests of the learner, and be able to explain what to do and why in the current situation.

This paper addresses the problem of how to enable agents to acquire the knowledge necessary to act as procedural skill tutors. The agent needs to acquire more than the sequence of steps in the procedure: the agent needs to know what conditions are necessary to execute each step, what ordering constraints exist between steps, what effects the steps entail, and how those effects may constrain the performance of subsequent steps. Specifying all this information can be a difficult and error-prone process, particularly for instructors and instructional designers who may be unfamiliar with knowledge representation techniques.

To cope with this knowledge acquisition bottleneck, researchers have explored three common approaches: direct specification, programming by demonstration, and autonomous experimentation. The first approach, direct specification, requires a domain expert or knowledge engineer to formalize the knowledge as code or declarative statements in the system's representation language. This approach is especially useful for adding particular elements of knowledge, but it is difficult and time consuming to formalize large amounts of knowledge. The second approach, programming by demonstration, allows the system to inductively learn domain knowledge by watching a domain expert perform tasks. This approach is simpler for the domain expert, but the system may require many demonstrations to accurately understand the causal relations among task steps. In the third approach, autonomous experimentation, the system actively experiments in the simulated world in order to learn the preconditions and effects of actions. This approach presents the least burden on the domain expert. However, without an initial domain theory or some guidance on relevant goals and actions, the system may explore many dead ends and useless portions of the domain, and hence the acquisition of the relevant knowledge may be impractically slow.

This paper describes Diligent [1], a system that learns procedural knowledge for simulation-based tutoring systems by exploiting the complementary strengths of these three approaches. Diligent has been fully implemented and integrated into STEVE [21, 22], an animated agent that cohabits virtual environments with students

AAMAS'02, July 15-19, 2002, Bologna, Italy.

to teach them procedural tasks. The next three sections of the paper formulate the problem that Diligent solves, describe its learning methods, and provide empirical evaluation of its approach, respectively, and the final section provides conclusions and discusses areas for future work.

2. THE KNOWLEDGE ACQUISITION PROBLEM

2.1 Domain Knowledge

To support intelligent tutors for procedural tasks, Diligent acquires three types of domain knowledge: (1) task models, which describe domain procedures, (2) operators, which model the preconditions and effects of domain actions, and (3) linguistic knowledge, which allows a tutor to talk about the domain.

The representation for task models must satisfy two requirements. First, it must allow the tutor to determine the next appropriate action while demonstrating a task or watching a student, even if this requires adapting standard procedures to unexpected student actions. Second, it must allow the tutor to explain the role of any suggested action in completing the task.

To meet these requirements, Diligent uses a relatively standard hierarchical plan representation to formalize tasks [23]. A task model includes a set of steps, a set of end goals, a set of ordering constraints, and a set of causal links. Each step may either be a primitive action (e.g., press a button) or a composite action (i.e., itself a task). The end goals of the task describe a state the environment must be in for the task to be considered complete (e.g., "valve1 open" or "engine on"). 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 particular 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.

This representation was chosen for its compatibility with relevant prior work. It has proven effective in a wide variety of research on task-oriented collaboration and generating procedural instructions [5, 11, 34]. For simulation-based tutoring, Rickel and Johnson [21] have shown how partial-order planning techniques can use this task representation to construct and revise plans for completing a task in the face of unexpected student actions, as well as dynamically generate explanations for the role of each plan step in completing the task. The causal links in the representation are especially important. They support plan revision by allowing the tutor to determine which task steps are still relevant to achieving the task's end goals. They also support explanation generation; the tutor can combine its knowledge of the causal links in the task model with its knowledge of which parts of the task are still relevant in order to explain the role of an action in completing the task.

To complement the task models it acquires, Diligent also acquires a set of operators that model the preconditions and effects of the domain actions in the task models. Task models only specify those preconditions and effects of task steps that are relevant to completing the task. In contrast, operators specify all the possible effects of a domain action, and the required preconditions for each effect. Like most work on planning, our work focuses on actions with discrete (rather than continuous) effects.

Finally, to allow the tutor to talk to students, Diligent acquires linguistic knowledge. Specifically, Diligent currently learns text fragments for the various elements of its ontology (e.g., tasks, steps, and goals); these can be used with domain-independent text templates to support natural language generation during tutoring. In the future, we plan to support the acquisition of more structured linguistic knowledge to support more sophisticated natural language generation techniques. In the remainder of this paper, we will ignore linguistic knowledge for simplicity.

Some prior systems have focused on methods for acquiring sufficient knowledge for an agent to perform tasks, as opposed to teaching tasks. While learning how to perform tasks and how to teach tasks are similar problems, an agent that must teach has an extra requirement: it must be able to explain the rationale behind its task decisions. Of these prior systems, some only learn reactive rules that lack sufficient knowledge for generating explanations [3, 8, 18]. Other systems learn representations that would be difficult to explain [27, 2, 16]. Thus, our target domain knowledge distinguishes our work from these prior systems. Of these efforts the work of Nicolescu and Mataric on teaching robots by demonstration [16] is closest in objectives, but it assumes that the procedures are built out of behaviors that have already been specified for the robot.

2.2 Sources of Knowledge

In simulation-based training, students learn procedural tasks by practicing them in a simulation of their work environment. The simulator has a user interface that allows students to perform domain actions and see their effects. Diligent exploits the same simulator to acquire its domain knowledge.

Diligent makes few assumptions about the simulator. First, it assumes that the human instructor can perform domain actions, so that the instructor can demonstrate tasks for Diligent. Second, it assumes that Diligent can perform the same actions itself by sending commands to the simulator; this enables it to experiment with domain tasks. Third, it assumes that the simulator will send Diligent an *action observation* whenever Diligent or the human instructor performs an action. The action observation should include the simulation state in which the action was performed, the action that was taken, and its effects.[1] For compatibility with a wide range of simulators, Diligent assumes that the simulation state is represented as a set of state variables and their values; an effect is a new value for a state variable. Finally, Diligent requires the ability to save a state of the simulator and restore it later, so that it can repeat experiments from a known state.

Unlike some other systems [19], we do not assume that Diligent can make arbitrary changes to the simulation state. While the ability to make arbitrary changes greatly simplifies learning, allowing the agent to see the effect of each change on an action's effects, this ability is impractical for most simulators. Many simulators enforce a wide variety of constraints among state variables. While they are able to maintain these constraints for legal domain actions, they often cannot maintain them in the face of arbitrary changes to individual state variables. Even if they could, propagation of these constraints would violate the agent's desire to make individual changes.

It may seem as if a computer tutor could simply use the domain logic in the simulator, rather than acquiring it indirectly by learning from the simulator. However, this is impractical as a general solution. A wide variety of simulators exist, each with its own (often custom) representation. Simulation standardization efforts such as HLA provide ways to access simulation state data, but do not offer standard representations for simulation logic. Rather than build

[1]Our current implementation uses STEVE, which acquires action observations from simulators using a more sophisticated and efficient interface [21], but this simplified interface will suffice for this paper.

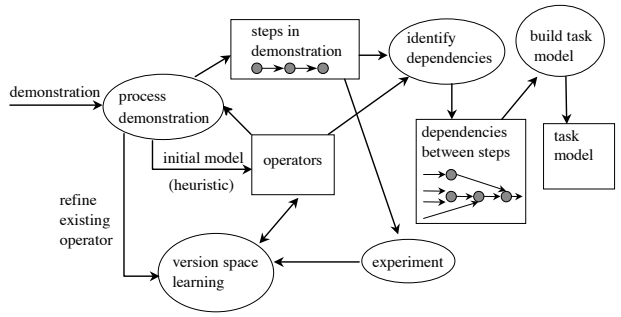

Figure 1: An overview of Diligent's approach. Arrows indicate data flow.

tutoring methods that can exploit these different representations, or build methods that can automatically convert from these representations into our target representation, our goal is to build general learning methods that can acquire our target domain knowledge from any simulator that supports the API described above.

In addition to the simulator, Diligent has a second source of knowledge: the human instructor. Our research focuses on human instructors that have domain expertise but not necessarily any abilities to program or formalize knowledge. Thus, teaching Diligent should be, as much as possible, like teaching another person. The instructor teaches Diligent by demonstrating tasks in the simulator and by using a GUI to issue commands, directly provide elements of domain knowledge, answer Diligent's questions, and review Diligent's knowledge.

3. INTEGRATING DEMONSTRATION, EXPERIMENTATION, AND DIRECT SPECIFICATION

Diligent builds on ideas from a variety of prior systems, but these prior systems have placed a greater burden on instructors. Some ask a large number of questions [31, 29, 10, 24], some require a large number of demonstrations [32, 33, 17], and some require an initial domain theory [6, 7, 20, 13, 25, 14, 19]. Diligent minimizes the need for demonstrations, questions, and an initial domain theory through a novel combination of programming by demonstration, autonomous experimentation, and direct specification.

3.1 Demonstrations

Diligent begins learning about a procedure through a process of programming by demonstration [4]. The human instructor issues a command for Diligent to observe his actions, and then performs the procedure by taking actions in the simulator. During this demonstration, Diligent receives action observation messages from the simulator, processing them to create operators, and stores the sequence 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, as described in Section 3.2, an operator is created to model the effects of the action; each step in the demonstration is recorded as an instance of an operator with a set of particular effects.

By the time the demonstration is complete, Diligent has learned a sequence of steps that can be used to perform the task. This provides Diligent with the set of steps required for its target task model, as well as one possible ordering of those steps; at this point, as far as Diligent knows, other orderings may also work.

To establish the end goals of the task, the second part of our task representation, Diligent hypothesizes that likely goals are the final values of state variables that changed value during the demonstration. The instructor is then allowed to review this list and remove goals that are merely side effects of the task.

At this point, Diligent could derive the ordering constraints and causal links from its operator models, but there would most likely be errors. During a demonstration, Diligent typically only sees an action taken under one set of circumstances. This is insufficient to identify the precise preconditions of operators, which are needed to identify the ordering constraints and causal links among steps. To refine how operators model the effects of actions, 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.

3.2 Experimentation

Diligent uses the initial demonstration to generate experiments. The experiments are conducted by repeating the task 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 focused way for Diligent to learn on its own with a minimal amount of background knowledge of the task and environment.

Diligent learns the preconditions of operators associated with each action using a version space algorithm [12]. A version space maintains bounds representing the most specific (s-set) and most general (g-set) combinations of preconditions possible for each effect of the operator. The state of the world before each action, and the changes that occur afterwards, are used to create new operators and update old ones. To reduce version space memory requirements, Diligent uses the INBF algorithm [28] in which the s-set and g-set are each represented by a single conjunctive condition. Initially, the s-set is the state before the action and only matches that state, while the g-set is empty and can match any state. Successful applications of old operators under new conditions can be used to broaden the s-set by removing preconditions. Conversely, actions that fail to replicate the effects of an old operator may be useful in narrowing the g-set by adding additional preconditions. The combination of the instructor's original demonstration and Diligent's own experiments provide a range of examples of each operator, and the version space representation represents the conclusions Diligent has drawn about these operators and any remaining uncertainty due to insufficient examples.

3.3 Direct Specification

After the system has finished experimenting, it can use the original demonstration, the end goals of the task, and its refined operators to generate a representation of the learned task. Recall that the record of the demonstration contains the state before and after each step. Since Diligent associates an operator with each step in the demonstration, it can use its model of that operator to identify the preconditions that were required in the state before the step in order to produce the effects that were observed after it was executed. (Section 3.4 explains how Diligent handles operator models whose g-set and s-set have not converged.) This allows Diligent to augment the demonstration to include the preconditions of each step as well as its effects. Given the initial state, Diligent can then analytically derive how the initial state and earlier steps establish both end goal conditions and preconditions for later steps. This allows Diligent to identify the causal links and ordering constraints of the task.

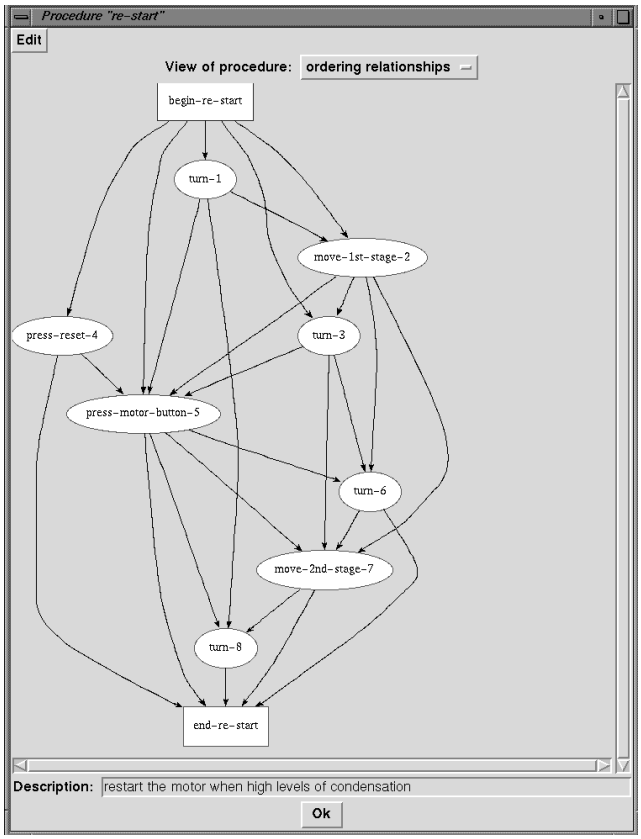

Figure 2: Diligent's graph of a task model for review by the instructor.

The instructor can review the knowledge Diligent has learned by either examining a graph of the task model (e.g., Figure 2) 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 Diligent's task model by adding or removing steps, ordering constraints, and causal links, as well as modifying operator effects preconditions. 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.

3.4 Heuristics for Refining Operators

The g-set and s-set in Diligent's operator models represent conservative, provably correct bounds on the true preconditions (given the assumption of conjunctive preconditions). When they converge, Diligent knows the true preconditions. Before they converge, Diligent has a representation of its uncertainty.

In order to derive a task model and display it to the instructor before the operator models have converged, Diligent must make a reasonable guess at the true preconditions given the current bounds. The system addresses this problem by using heuristics to create and maintain a working set of likely preconditions (h-set). The heuristics are designed to add a small number of preconditions to the g-set that hopefully cover all or most of the actual preconditions.

The likely preconditions (h-set) are currently identified using two heuristics:

Earlier steps establish preconditions for later steps. The steps in a demonstration are related, and the instructor probably has reasons for demonstrating steps in a given order. A likely reason is that the state changes of earlier steps establish the preconditions of later steps. In this case, Diligent uses as preconditions the values of state variables that were changed by earlier steps. The system ignores possible preconditions that were true before the procedure started because only state variables that change value can differentiate between orders of steps that achieve the goal state and those that do not.

Focus on state variables that change value. If a step changes a particular state variable as one of its effects, the previous value was probably a precondition.

The heuristics are only used to initialize the h-set of each operator. As Diligent sees additional examples of an operator, the h-set is refined with the inductive version space techniques described above, so the h-set is guaranteed to lie between the g-set and s-set bounds.

3.5 Example

To illustrate these concepts, we will discuss authoring a very simple task. Assume that Diligent has no prior knowledge of how the demonstration's steps will affect the simulation.

1. The human instructor tells Diligent that he wants to author a new procedure, and he provides a name ("shut-valves") to identify the procedure.

2. The instructor demonstrates the procedure by using the mouse to manipulate objects in the simulation's graphical window. This particular demonstration consists of selecting (i.e., clicking on) three valves sequentially: valve1, valve2, and valve3. When the instructor selects a valve, the valve closes.

3. After the demonstration, the procedure's goals must be specified. Diligent hypothesizes that likely goals are the final values of state variables that changed value during the demonstration: (valve1 shut) (valve2 shut) (valve3 shut). In this case, the instructor verifies that these are the correct goals.

4. Now the instructor tells Diligent to experiment with the procedure. While Diligent could generate a task model without experimenting, the task model would contain errors because Diligent does not yet understand the preconditions of each step (e.g., preconditions for shutting valve1). The experiments attempt to understand if and how each step depends on earlier steps. In this example, Diligent will repeat the procedure twice; the first time it skips step one (closing valve1) to see the effect on steps two and three, and the second time it skips step two (closing valve2) to see the effect on step three.

   To better understand why these experiments are useful, consider the operator that represents shutting valve3. When creating an operator during the demonstration, Diligent hypothesizes that state changes from earlier steps are likely preconditions (h-set). This is shown in Figure 3. The version space g-set is empty because no preconditions have been shown to be necessary, while the version space s-set contains many irrelevant conditions. In contrast, the h-set is more focused on how the demonstration manipulates the simulation's state.

   The demonstration's three steps are independent and could correctly be performed in any order. However, because Diligent uses likely preconditions (h-set) to generate the task model, Diligent initially believes that valve1 needs to be shut before valve2, and valve2 needs to be shut before valve3. Diligent's experiments correct this problem, and, after the

Name: toggle-valve3
Preconditions:
g-set:
empty
h-set:
(valve1 shut)(valve2 shut) (valve3 open)
s-set:
Simulation state before shutting valve3
State changes:
(valve3 shut)

Figure 3: The initial operator for shutting valve3

Name: toggle-valve3
Preconditions:
g-set:
empty
h-set:
(valve3 open)
s-set:
Simulation state before shutting valve3, except for the state of valve1 and valve2
State changes:
(valve3 shut)

Figure 4: The final operator for shutting valve3

experiments finish, Diligent can build a correct task model. Consider the operator for valve3 after the experiments (Figure 4). Even though there have only been three training examples (performance of the action in the original demonstration and the two experiments), the refined operator's likely preconditions (h-set) are correct. However, the version space's g-set is still empty and useless, and the version space's s-rep still contains many unnecessary conditions, but it has been refined so that it no longer contains conditions for valve1 and valve2 because they have been shown to be irrelevant when shutting valve3.

3.6 Extensions

To allow Diligent to model more complex procedures and to support better interaction with the instructor, several extensions to the above mechanisms have been implemented. One extension is support for hierarchical task representations, where a subtask is treated as a single step in a larger task. This allows the reuse of existing tasks and improves scalability because large, complicated tasks can be broken into simpler subtasks. An instructor can specify a subtask during a demonstration either by defining a new task as a step in the current procedure or by indicating that an existing task should be performed at that point.

To treat a subtask as a single step, the internal details of the subtask are ignored and only its preconditions and effects are considered by the parent task. To achieve a subtask's desired effects, Diligent uses a version of STEVE's partial-order planning algorithm [21] for deciding how to perform a subtask. Diligent uses the likely preconditions (h-set) of the subtask's steps to determine ordering constraints, causal links and which steps to perform. However, Diligent will execute all subtask steps that achieve necessary effects, even if preconditions do not appear to be met, because Diligent's knowledge of the subtask preconditions might be incomplete or incorrect.

Besides hierarchical tasks, Diligent also supports information gathering or sensing actions. A *sensing action* gathers information about the simulation's state without changing it (e.g., looking at a light). Because sensing actions do not change the state and might be performed at any time, there needs to be a mechanism to insure that they are performed in the proper context (e.g., look at the light when the motor is off). For sensing action preconditions, Diligent uses the values of the state variables that have already changed value in the current task. To record that a sensing action has been performed in the proper context, Diligent creates internal "mental" state variables, which are not present in the simulation's state. To require that sensing actions are performed, the mental effects are added to the task's goals.

The above discussion only mentioned a single demonstration of each task. However, Diligent allows instructors to provide multiple demonstrations of tasks. This allows instructors to iteratively refine task models, correct mistakes, or elaborate on subtasks. However, multiple demonstrations raise issues of which initial state to use for each step when deriving causal links and ordering constraints. Diligent's algorithms for processing multiple demonstrations, described in [1], provide a foundation and are sufficient for the relatively short procedures that we have looked at. Incorporating algorithms for more complicated domains is an area for future work.

4. EVALUATION

Diligent is fully implemented and integrated with a simulation environment developed independently by our colleagues, and it has been tested on a variety of tasks for operating the gas turbine engines that propel naval ships. The simulator was implemented by the USC Behavioral Technology Laboratories using their RIDES [15] simulation authoring software. The graphics for the simulation environment were developed by Lockheed Martin using their Vista Viewer software [30]. Diligent, RIDES, and Vista run as separate processes that communicate by passing messages via a central message dispatcher [9]; our current implementation uses Sun's ToolTalk as the message dispatcher.

We evaluated Diligent to determine whether it simplifies the job of authoring procedural knowledge. Since Diligent employs two techniques to assist instructors, namely demonstrations and experiments, we evaluated their separate contributions in facilitating authoring. Our hypothesis was that both demonstrations and experiments would reduce the effort required by the instructor and result in more accurate task models. To test the hypothesis, fifteen computer science graduate students were divided into three groups that authored procedural knowledge differently: group G1 (four subjects) used demonstrations, experiments, and direct specification; group G2 (six subjects) used only demonstrations and direct specification; and group G3 (five subjects) used only direct specification.

Two dependent measures were used to assess ease of authoring and quality of the resulting knowledge base. *Edits* is the number of deliberate changes made to the knowledge base (e.g., demonstrating a step or changing a precondition). *Errors* is the number of mistakes in the task model, either missing or spurious steps, ordering constraints and causal links. These errors are typically caused by missing or spurious preconditions and goal conditions. The sum of the edits and the errors metrics, called total required effort, estimates the total work required to create a correct task model.

For each subject, the evaluation covered two consecutive days. The subjects were trained for approximately 2 hours the first day and had a 30 minute review on the second day. After training, the subjects authored two machine maintenance procedures (proc1 and

	Edits		Errors		Total Effort	
	Mean	std	mean	std	Mean	Std
G1:proc1	10	2	11	1	20	4
G2:proc1	35	13	35	14	70	1
G3:proc1	38	8	53	1	90	9
G1:proc2	9	2	15	6	24	6
G2:proc2	17	6	16	1	32	7
G3:proc2	26	4	18	10	44	13

Table 1: Comparison of the means and standard deviations for the three groups (G1,G2,G3) for both procedures (proc1 and proc2).

proc2) of differing complexity, proc1 the more complex of the two.[2] Both procedures were authored with an initially empty knowledge base. While a subject was authoring, the system collected data about his activities. Afterwards, the resulting task models were manually compared against the desired task models. When authoring each procedure, subjects were given a time limit, which many subjects reached. Reaching the time limit was likely to reduce the edits and increase the number of errors.

The authoring was an iterative, multistage process. Subjects were given functional descriptions of the procedures that gave them enough information to reconstruct the formal details (e.g., steps, ordering constraints, and causal links) without specifying them directly. After studying the functional descriptions to understand the procedures, they used Diligent to specify the steps through either demonstrations (G1 and G2) or direct specification (G3). After demonstrating, G1 subjects could refine the initial task model by asking Diligent to experiment with the demonstration. All subjects could then edit the task models and test them with the STEVE tutor. The process was iterative because subjects could add more steps, experiment again, perform additional editing, and retest the task models.

The results are shown in Tables 1 and 2. Because of the limited number of subjects, the statistics are less important than the trends they suggest. First, consider the effect on edits. For proc2, the groups were significantly different (ANOVA) and group G2 (only demonstrations) is significantly better than G3 (only direct specification). Next, consider errors. For proc1, the groups have a significant difference (ANOVA) for both errors of omission (.001 probability)[3] and total errors. For errors of omission on proc1, group G2 is significantly better than G3. Finally, consider total effort. Both procedures have significant ANOVA for total effort, and for proc1, group G1 (experiments) is significantly better than G2 (only demonstrations). Overall, though, the effects with the complex procedure, proc1, are much greater than they were for proc2, suggesting that Diligent's assistance is most beneficial with complex procedures that have significant opportunities for error.

The results suggest that demonstration and experimentation both played a role in improving the quality of the task models and reducing the work required to create them. Task models acquired through demonstrations typically contained spurious elements that the instructors then needed to delete; however it was easier for them to recognize spurious elements than to notice when elements were missing, as group G3 was required to do. Experimentation eliminated many of the spurious elements, further reducing the effort involved. These effects were most prominent in the more complex procedure containing many elements.

[2]The procedures involve restarting a motor after it was shut down by either high levels of condensation or high air pressure, respectively. See [1] for full details.

[3]Errors of omission are not directly shown in tables 1 and 2, which show an error metric that includes both errors of omission and commission.

	Proc1			Proc2		
	Edits	**Errors**	**Total effort**	**Edits**	**Errors**	**Total effort**
ANOVA	.08	.03	.003	.001	.8	.02
G1,G2	.1	.1	.008	.08	.99	.4
G2,G3	.96	.2	.08	.03	.8	.2
G1,G3	.1	.03	.003	.001	.8	.02

Table 2: Comparison of the variance between the three groups (G1,G2,G3). ANOVA indicates the probability that the variance is random. G1,G2 and G2,G3 indicate the probability that the difference between the two groups is random (Scheffé's simultaneous confidence intervals). G1,G3 is shown for comparison but was not part of the experimental design.

5. CONCLUSIONS AND FUTURE WORK

This paper has described a method to enable automated agents to acquire procedural skills through interaction with human instructors. The method exploits the structure and behavior of the simulation environment, by observing the effects of procedures on the state of the simulation. It thus takes advantage of infrastructure that must be created in any case, since agent-enhanced simulation environments require interactive simulations that the agents are able to observe and interact with. Demonstration and experimentation provide opportunities to employ machine learning techniques to reduce the effort required to author procedural knowledge. The agent is able to acquire a significant amount of knowledge from a small number of demonstrations, and the human instructor can further refine the agent's knowledge through interactive testing and direct specification.

This method is best suited for authoring procedural skills where the primary focus of the procedure is to achieve some desired effect in the virtual world, and where the consequences of actions are readily observed. Not all procedural skills fit this description. For example, we have chosen not to apply the Diligent method to the acquisition of clinical procedures for the Adele pedagogical agent [26]. Although diagnostic procedures in medicine have sequences of steps, the rationales for the steps are often the conclusions that they permit about the patient's condition rather than the effects they have on the patient. Furthermore, any effects on the patient are likely to be indirect and not immediately discernible. Diligent's experimental method is of more limited value in domains such as this, although demonstrations might still perform a useful role in describing procedures.

Although Diligent's method is influenced by human instructional dialog, the structure of interactions with Diligent is still very different from interactions between human tutors and students. Human tutors do not simply demonstrate procedures; they combine their demonstrations with explanations of what they are doing and why. Learners are able to ask questions, and decide what is an appropriate time to ask such questions. We believe that a fruitful area of future research would be to find ways of patterning interaction with Diligent so that it is closer to human tutorial interaction. Diligent's current capabilities provide a foundation to support such interactions.

6. ACKNOWLEDGMENTS

This work was funded by the Office of Naval Research under grant N00014-95-C-0179 and AASERT grant N00014-97-1-0598. Richard Angros thanks Raytheon for its generous support.

7. REFERENCES

[1] R. Angros, Jr. *Learning What to Instruct: Acquiring Knowledge from Demonstrations and Focussed Experimentation*. PhD thesis, Department of Computer Science, University of Southern California, Los Angeles, CA, 2000.

[2] S. Benson. Inductive learning of reactive action models. *Machine Learning: Proceedings of the 12th International Conference*, pages 47–54, 1995.

[3] R. Bindiganavale, W. Schuler, J. M. Allbeck, N. I. Badler, A. K. Joshi, and M. Palmer. Dynamically altering agent behaviors using natural language instructions. In *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 293–300, New York, 2000. ACM Press.

[4] A. Cypher, editor. *Watch What I Do: Programming by Demonstration*. MIT Press, Cambridge, MA, 1993.

[5] J. Delin, A. Hartley, C. Paris, D. Scott, and K. Vander Linden. Expressing procedural relationships in multilingual instructions. In *Proceedings of the Seventh International Workshop on Natural Language Generation*, pages 61–70, Kennebunkpurt, Maine, 1994.

[6] Y. Gil. *Acquiring Domain Knowledge for Planning by Experimentation*. PhD thesis, Carnegie Mellon University, 1992.

[7] T. R. Gruber. Automated knowledge acquisition for strategic knowledge. *Machine Learning*, 4:293–336, 1989.

[8] S. B. Huffman and J. E. Laird. Flexibly instructable agents. *Journal of Artificial Intelligence Research*, 3:271–324, 1995.

[9] W. L. Johnson, J. Rickel, R. Stiles, and A. Munro. Integrating pedagogical agents into virtual environments. *Presence: Teleoperators and Virtual Environments*, 7(6):523–546, December 1998.

[10] B. J. Krawchuk and I. H. Witten. On asking the right questions. In *5th International Machine Learning Conference*, pages 15–21. Morgan Kaufmann, 1988.

[11] C. Mellish and R. Evans. Natural language generation from plans. *Computational Linguistics*, 15(4):233–249, 1989.

[12] T. M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.

[13] T. M. Mitchell, S. Mahadevan, and L. I. Steinberg. LEAP: A learning apprentice for VLSI design. In *Machine Learning An Artificial Intelligence Approach*, volume III, pages 271–289. Morgan Kaufmann, San Mateo, CA, 1990.

[14] T. M. Mitchell, P. E. Utgoff, and R. Banerji. Learning by experimentation: Acquiring and refining problem-solving heuristics. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning An Artificial Intelligence Approach*, volume I. Morgan Kaufmann, San Mateo, CA, 1983.

[15] A. Munro, M. Johnson, Q. Pizzini, D. Surmon, and D. Towne. Authoring simulation-centered tutors with RIDES. *International Journal of Artificial Intelligence in Education*, 8:284–316, 1997.

[16] M. Nicolescu and M. Mataric. Experience-based representation construction: learning from human and robot teachers. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2001.

[17] T. Oates and P. R. Cohen. Searching for planning operators with context-dependent and probabilistic effects. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 863–868, Menlo Park, CA, 1996. AAAI Press/MIT Press.

[18] D. Pearson and J. Laird. Toward incremental knowledge correction for agents in complex environments. In S. Muggleton, D. Michie, and K. Furukawa, editors, *Machine Intelligence*, volume 15. Oxford University Press, 1995.

[19] B. W. Porter and D. F. Kibler. Experimental goal regression: A method for learning problem-solving heuristics. *Machine Learning*, 1:249–286, 1986.

[20] M. A. Redmond. *Learning by observing and understanding expert problem solving*. PhD thesis, Georgia Institute of Technology, 1992.

[21] J. Rickel and W. L. Johnson. Animated agents for procedural training in virtual reality: Perception, cognition, and motor control. *Applied Artificial Intelligence*, 13:343–382, 1999.

[22] J. Rickel and W. L. Johnson. Task-oriented collaboration with embodied agents in virtual worlds. In J. Cassell, J. Sullivan, S. Prevost, and E. Churchill, editors, *Embodied Conversational Agents*. MIT Press, Cambridge, MA, 2000.

[23] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, NJ, 1995.

[24] C. Sammut and R. B. Banerji. Learning concepts by asking questions. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume II, pages 167–191. Morgan Kaufmann, Los Altos, CA, 1986.

[25] A. M. Segre. A learning apprentice system for mechanical assembly. In *IEEE Third Conference on Artificial Intelligence Applications*, pages 112–117, 1987.

[26] E. Shaw, W. L. Johnson, and R. Ganeshan. Pedagogical agents on the web. In *Proceedings of the Third International Conference on Autonomous Agents*, pages 283–290, New York, 1999. ACM Press.

[27] W.-M. Shen. Discovery as autonomous learning from the environment. *Machine Learning*, 11(4):250–265, 1993.

[28] B. D. Smith and P. S. Rosenbloom. Incremental non-backtracking focusing: A polynomial bounded generalization algorithm for version spaces. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pages 848–853, Menlo Park, 1990. AAAI Press.

[29] D. C. Smith, A. Cypher, and J. Spohrer. KIDSIM: Programming agents without a programming language. *CACM*, 94(7):55–67, 1994.

[30] R. Stiles, L. McCarthy, and M. Pontecorvo. Training studio interaction. In *Workshop on Simulation and Interaction in Virtual Environments (SIVE-95)*, pages 178–183, Iowa City, IW, July 1995. ACM Press.

[31] G. Tecuci and M. R. Hieb. Teaching intelligent agents: the Disciple approach. *International Journal of Human-Computer Interaction*, 8(3):259–285, 1996.

[32] X. Wang. *Learning Planning Operators by Observation and Practice*. PhD thesis, Carnegie Mellon University, 1996.

[33] X. Wang. A multistrategy learning system for planning operator acquisition. In *The Third International Workshop on Multistrategy Learning*, Harpers Ferry, West Virginia, May 1996.

[34] R. M. Young. *Generating Descriptions of Complex Activities*. PhD thesis, University of Pittsburgh, 1997.