Training an Agent Through Demonstration: A Plausible Version Space Approach

Michael R. Hieb Gheorghe Tecuci[1]

Learning Agents Lab, Department of Computer Science, George Mason University
Fairfax, VA 22030-4444, U.S.A.
hieb@cs.gmu.edu tecuci@cs.gmu.edu

Abstract

This paper presents an efficient approach to training an agent to perform a complex task through demonstration, explanation and supervision. This approach is based on an integration of techniques of multistrategy and apprenticeship learning, knowledge elicitation and programming by demonstration, in a plausible version space framework, and is implemented in Agent-Disciple. Agent-Disciple addresses the complexity of the task training problem through user-agent interaction, allowing the user to specify enough knowledge and guide the agent to learn a general procedure from only a few examples. Interaction techniques allow the user to train the agent through positive or negative examples presented through the interface of the agent's application domain. Agent-Disciple is an active participant in the interaction, suggesting possible explanations of tasks and experimenting with similar tasks. Elicitation methods are integrated to allow the end user to define new relevant terms in the representation language at any point in the training process. This approach has been used to train agents for ModSAF – a virtual military simulation environment. We describe the training of a ModSAF agent, review the problems encountered, and the lesson learned from this experience.

Introduction

It has become increasingly evident that the fields of Machine Learning (ML) and Knowledge Acquisition (KA), as they have developed, have much to offer one another (Tecuci & Kodratoff 1995). Hard problems in one field can be addressed by existing techniques in the other (Tecuci & Hieb 1994). The field of Programming By Demonstration (PBD) is just as complementary by its emphasis on the problems of interaction with the end user within a graphical user interface and its integration of learning methods into existing applications for task automation (Cypher 1994).

We have developed a general system, called Agent-Disciple, that can be used to develop instructable agents for specific applications. Agent-Disciple integrates apprenticeship multistrategy learning, knowledge elicitation and innovative methods of interaction in a plausible version space framework (Tecuci 1992). We have been investigating the use of Agent-Disciple to train existing simulation agents operating in a virtual environment (Hieb 1996, Tecuci & Hieb 1996). We describe this domain in the next section. We then describe the background knowledge that is necessary prior to agent training. We next illustrate our agent instruction in terms of the underlying algorithms in the simulation domain. We conclude with a discussion of what we have learned from this implementation.

Applying Agent-Disciple to ModSAF

Computer generated forces (CGF) are virtual military simulation agents representing either entities such as soldiers, tanks, planes, etc. or higher echelon level officers such as platoon or company commanders. CGF exist as real-time computer models distributed over a network using a technology known as Distributed Interactive Simulation. CGF allow the representation of larger forces in order to improve training exercises for smaller numbers of human participants.

The virtual training simulation that we are using is the Modular Semi-Automated Forces (ModSAF) real-time environment (Ceranowicz 1994). Semi-automated forces provide facilities for a human commander to direct the performance of a group of simulated entities. One of the key elements in choosing this domain was the very sophisticated application interfaces utilized in ModSAF. These graphical interfaces stress one of the basic ideas in our approach, that of direct interaction with the end user. Our aim is to let the end user work as naturally as possible when teaching the agent, without involving a programmer to translate the end user's knowledge into procedures or subjecting the end user to analysis of internal knowledge pieces.

Using Agent-Disciple, we have developed a training system called Captain (Hieb et al. 1995) to train ModSAF agents. Each agent takes action according to its orders (such as moving to another position or defending against an enemy), and the results of these actions are represented on the screen. The initial task chosen is teaching an automated military commander how to place its subordinate units to defend an area, as shown in Figure 1. Solving this placement problem requires a very complex reasoning process about terrain, visibility, and tactics. See Tambe, et

[1]also with the Center for Machine Learning, Natural Language Processing and Conceptual Modeling, Romanian Academy, Romania

al. (1995) for an excellent discussion of ModSAF in the context of developing SOAR-based agents.

Creating an Agent Training Environment

Captain is an agent training environment built from generic modules of Agent-Disciple that contain knowledge-base management, learning and problem-solving functions. Captain also contains modules specific to ModSAF in order to interface to ModSAF's persistent object database and use ModSAF's graphical user interface.

In Captain, task-specific knowledge is represented by a semantic network and procedures. The semantic network represents information from a terrain database at a more conceptual level. Some nodes in the semantic network represent the map regions in Figure 1 as objects, for instance, hill-sector-868-1, hill-863, company-D-area-of-responsibility, while others represent general concepts like hill-sector or hill. These concepts are hierarchically organized along the IS-A relationship The objects are further described in terms of their features (such as PART-OF, IN, VISIBLE). Such a semantic network is obtained through a sequence of semantic terrain transformations applied to the ModSAF terrain database, as described in (Hille et al., 1995). The agent also maintains a correspondence between each concept in the semantic network (e.g., hill-sector-868-1) and the corresponding region on the map, so that it can interact with the user and other agents through the portion of the ModSAF interface shown in Figure 1.

Training the Agent

We illustrate how Captain is used to train a ModSAF agent by showing a training scenario where a ModSAF user teaches a company commander agent how to place its subordinate units to defend an area.

Giving the initial example

The instructor initiates the teaching session by showing the ModSAF agent a specific example of a correct placement. The instructor uses the ModSAF interface to place the three platoons of Company D (D1, D2, and D3) on the ModSAF plan view display as shown in Figure 1, a topographic map of part of Fort Knox, Kentucky. The map represents the area of responsibility of Company-D. The enemy is expected to advance along the avenue of approach, from SW to NE.

The instructor creates the ModSAF units by using the ModSAF unit creation editor. The area of responsibility and avenue of approach are drawn by the instructor on an overlay to the map. Similarly the hills and the engagement area are put in the map overlay as a result of the semantic terrain transformations (which are done by hand at present). The textual representation of the task and its solution shown in Figure 1 are in terms of the symbolic concepts from the semantic network representing the map and the forces as described above. There are five input parameters to the task, the company unit-id, a location, enemy orientation, engagement area and mission. The "solution" to the task is a *placement* of the company's platoons.

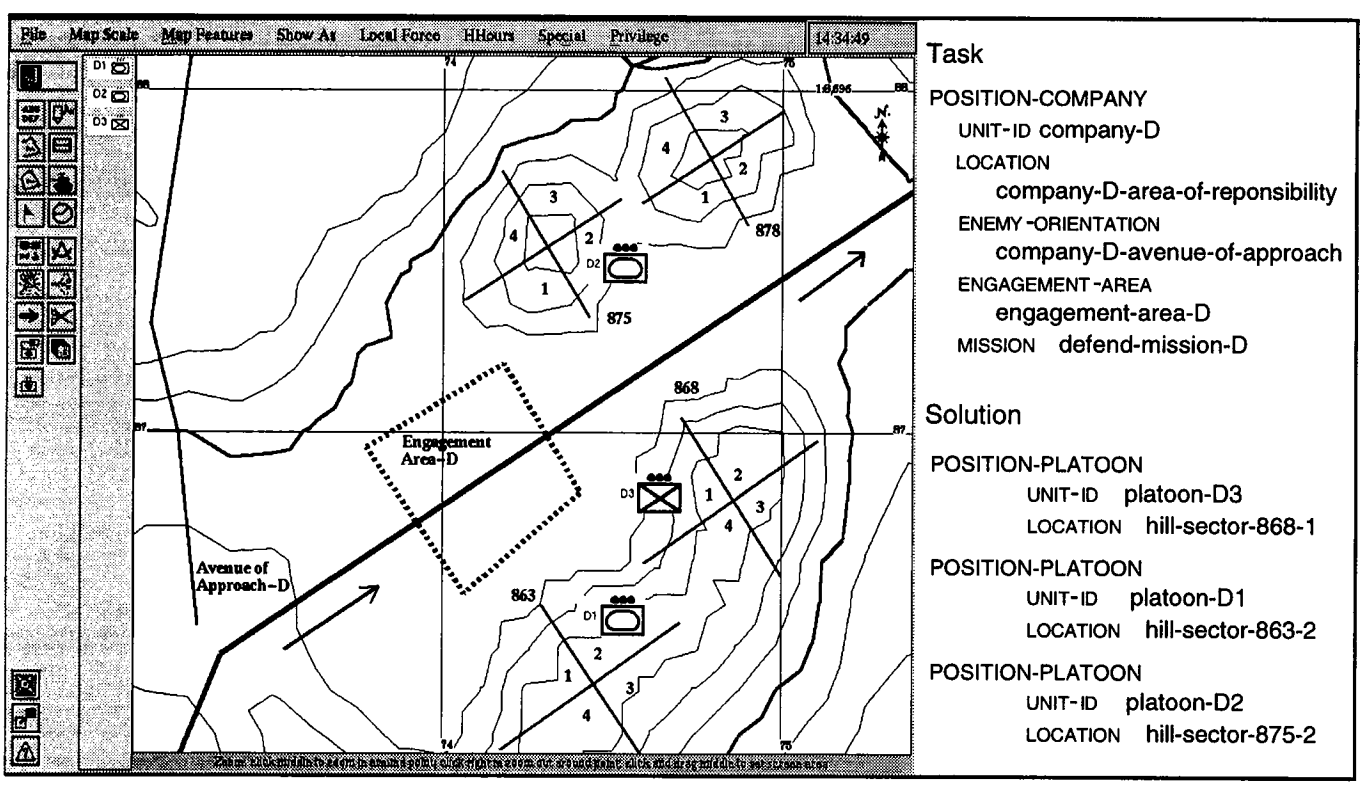

Figure 1 – Initial Placement Given by User of ModSAF Simulation

Generating Explanations

After giving the example in Figure 1, the instructor must explains why this placement is correct. An explanation may consist of a feature of some object from the example, a relationship between two objects, or some combination of features and relationships. There are currently six types of explanations:

- *Association* - a relationship between two objects in the example;
- *Correlation* - a common feature between two objects in the example;
- *Property* - a property of an object in the example;
- *Relationship* - a relationship between an object in the example and a new object;
- *Pointing* - a user-indicated object in a negative example;
- *Object Generalization* - a user-indicated generalization of an object in the example.

Rather than burdening the instructor to give precisely formatted explanations in the internal syntax of Agent-Disciple, the instructor may give a hint and focus the system's attention by asking it to generate explanations related to a certain object from Figure 1. For instance, the user may ask the agent to generate explanations (presented in a menu to the user) involving hill-sector-863-2:

hill-sector-863-2 VISIBLE engagement-area-D
hill-sector-863-2 IN company-D-area-of-responsibility
hill-sector-863-2 QUADRANT 2 &
hill-sector-875-2 QUADRANT 2
...

The first two explanations are *Associations*, the third is a *Correlation*. The instructor will select the first two explanations as relevant and may ask that other explanations be generated.

Learning a Placement Procedure

Based on these explanations and the initial example, the agent formulates a procedure like the one in Figure 2, except that the plausible lower bound corresponds to the explanation of the training example (and covers only this explanation), and the plausible upper bound corresponds to an over-generalization of the explanations (For the ModSAF domain this is obtained by generalizing the individual objects along the IS-A hierarchy and preserving the properties and relationships between these objects).

The task to be done is given in the first part under *To Accomplish*, consisting of POSITION-COMPANY. The solution of how to perform the task is given in the last part of the procedure, under *Perform*, consisting of three POSITION-PLATOON tasks. The conditions under *Verify* are conditions that need to be satisfied by the parameters of the task to be accomplished. The conditions under *Find* are conditions that need to be satisfied by the parameters of the tasks in the solution. The procedure does not have a single set of conditions but two sets, a plausible lower bound and the plausible upper bound that define a plausible version space for a hypothetical exact condition of the general placement procedure to be learned.

```
To Accomplish
  POSITION-COMPANY        UNIT-ID C
                          LOCATION AR
                          ENEMY-ORIENTATION AV
                          ENGAGEMENT-AREA E
                          MISSION M

  upper bound conditions
Verify
  (company C  (COMMANDS P2T) (COMMANDS PI)
              (COMMANDS P1T))
  (overlay-object AR)
  (overlay-object AV  (PART-OF AR))
  (engagement-area E  (PART-OF AV))
  (mission M  (WITH C)  (IN AR))
Find
  (platoon PI )
  (platoon P1T)
  (platoon P2T)
  (hill-sector HSI (DISTANCE-TO-ENGAGEMENT-AREA "close")
                   (IN AR) (VISIBLE E))
  (hill-sector HS1TP  (IN AR)  (VISIBLE E))
  (hill-sector HS2TP  (IN AR)  (VISIBLE E))

  lower bound conditions
Verify
  (company C  (COMMANDS P2T) (COMMANDS PI)
              (COMMANDS P1T))
  (area-of-responsibility AR)
  (avenue-of-approach AV (PART-OF AR))
  (engagement-area E  (PART-OF AV))
  (defend-area-mission M  (WITH C)  (IN AR))
Find
  (infantry-platoon PI )
  (armored-platoon P1T)
  (armored-platoon P2T)
  (hill-sector HSI (DISTANCE-TO-ENGAGEMENT-AREA "close")
                   (IN AR) (VISIBLE E))
  (hill-sector HS1TP  (IN AR)  (VISIBLE E))
  (hill-sector HS2TP  (IN AR)  (VISIBLE E))

Perform
  POSITION-PLATOON        UNIT-ID  PI
                          LOCATION  HSI
  POSITION-PLATOON        UNIT-ID  P1T
                          LOCATION  HS1TP
  POSITION-PLATOON        UNIT-ID  P2T
                          LOCATION  HS2TP
```

Figure 2: Procedure for ModSAF Agent

A detailed description of the plausible version space learning algorithm is given in (Tecuci, 1992). Here we will only briefly summarize it.

The plausible lower bound is a conjunctive expression that is approximately less general than the hypothetical exact condition of the procedure. By this we mean that most of the instances of the plausible lower bound are also instances of the hypothetical exact condition. The plausible upper bound is a conjunctive expression that is approximately more general than the hypothetical exact condition. The universe of instances of the procedure in Figure 2 is an n dimensional space, where n is the number

of variables from the procedure's conditions (n = 11, in this case). The learning process takes place in an incomplete representation language that may not contain all the terms necessary to learn the hypothetical exact condition. This representation language may also evolve during learning by adding such new terms to the conditions.

Experimentation

After the initial plausible version space has been defined, Captain starts an experimentation process. It generates new placement problems and solutions that are covered by the plausible upper bound, without being covered by the plausible lower bound. These examples are displayed on the ModSAF interface. The instructor judges them as correct (i.e., positive examples) or incorrect (i.e., negative examples). The instructor may stop the experimentation process at any point or may give additional positive and negative examples.

Based on these examples, the plausible lower bound and the plausible upper bound may be both generalized (i.e., extended) and specialized (i.e., contracted) to better approximate the hypothetical exact condition. Notice that this is different from the version spaces introduced by Mitchell (1978) where the upper bound is only specialized and the lower bound is only generalized. However, in spite of these incremental adjustments in both directions, the plausible bounds may not become identical because of the incompleteness of the representation language or because there are not enough examples to learn from. In fact, the procedure in Figure 2 is precisely the procedure learned by Captain, after it generated four examples of defensive placements shown to the instructor via the ModSAF graphical interface.

Knowledge Elicitation

During learning (and performing subsequent learned tasks), the agent may also accumulate negative or positive exceptions to the procedure. These are negative examples that are covered by the plausible lower bound, or positive examples that are not covered by the plausible upper bound. In such cases, the agent will attempt to elicit new knowledge from the end user to allow the agent to modify the plausible version space of the procedure such that the exceptions become examples. These knowledge elicitation techniques are described in (Tecuci and Hieb 1994; Hieb 1996).

Applying the Learned Procedure

The learned procedures are used by the agent when given a mission by a ModSAF user as follows:

1) SELECT – the agent selects a procedure such as the one in Figure 2 to accomplish the specific mission given by the user and binds the variables in the procedure's *to acomplish* task to the mission parameters.
2) VERIFY – the mission parameters are verified to meet the constrains imposed by the corresponding *verify* conditions. If this test fails the lower bound, the upper bound condition is used. The procedure execution stops if an object cannot be verified.
3) FIND – a set of objects is found corresponding to the task parameters that meet the constraints in the *find* conditions. The procedure execution stops if an object cannot be found.
4) EXECUTE – the tasks in the solution are instantiated with the set of objects from steps 2 & 3 corresponding to the parameters of the tasks and the tasks are invoked.

Lessons

The following problems have been difficult issues in the development of Captain: interfacing to the external application (ModSAF); creating an initial KB from the simulation state and the terrain database; and defining the tasks to be taught to the ModSAF agent. To achieve scalable performance, we further developed the interaction that Agent-Disciple has with a user.

Systems for automating such complex tasks must be designed so that they can be general enough to be adapted to different domains. For example, we needed to expend considerable effort to modify both the ModSAF application (which contains over 300 source libraries written in C) and Agent-Disciple to create Captain. Our goal is to make Captain one of the existing editors of ModSAF (it has many), as opposed to a separate interface. Lieberman (1994) points out that the interface between an end user and the agent training system is a crucial issue not addressed in most of the machine learning research. Our approach is to use as much of the existing application interface as possible, on the assumption that this is easier for the end user.

A disadvantage of our method is that it requires a pre-existing knowledge base and the creation of customized methods to translate the application's current state to the agent's semantic network. In the ModSAF domain we have developed terrain transformations techniques that would automatically create a substantial portion of our semantic network from the digital terrain databases (Hille et al. 1995), and provided facilities for the end user to specify additional terms in the representation language during the training process, as in Dybala & Tecuci (1995).

From our experience we have concluded that it is difficult to give the user the flexibility to define completely new complex tasks such as missions in the ModSAF domain. The missions that a ModSAF agent can perform are quite complicated, because the environment is complex and non-deterministic. We are investigating providing the end user a task template corresponding to the basic missions available to the end user. The user could then specialize or modify this template to create the task structure necessary for learning a procedure.

Much of the power of our approach comes from the multiple types of interaction between the end user and the agent being taught. Such rich interaction is rare among ML systems, but is closer to the interaction found in PBD systems (Maulsby, 1994). We believe that such interaction

is necessary to develop more powerful agents. These interactions include: specifying new terms in the representation language of the agent; giving the agent an example of a solution to a task that the agent is to learn a general procedure for; validating analogical instances of solutions proposed by the agent; explaining to the agent reasons for the validation; and being guided to provide new terms in the representation during interaction.

Conclusions

In this paper we have briefly presented an efficient approach to teaching complex behavior to a virtual commander agent through demonstration. Currently, we can teach a ModSAF agent how to solve a placement task by eliciting one example from the end user, eliciting 5 to 10 explanations and showing the end user 5 to 10 examples that the system generates. Experiments with Captain indicate that the system will scale up as it is applied to learning other tasks in this domain (Hieb 1996). This appears to be a more natural and significantly simpler approach to training ModSAF agents than the currently used approach in ModSAF where the end user manually specifies the mission of the ModSAF agents in great detail to achieve reasonable behavior in a simulation. This learning efficiency of the Disciple approach is achieved through the use of plausible version spaces and a human guided heuristic search of these spaces.

It is interesting to notice that the Disciple multistrategy and apprenticeship learning method meets the basic requirements for an ideal PBD learner, as identified by Maulsby and Witten (1995). First, the learning agent is under the user's control, who specifies the actions and features relevant to the task to be taught, gives hints and explanations to the agent, and guides its learning actions. Second, the learning agent uses various knowledge-based heuristics for performing plausible generalizations and specializations that are understandable, including plausible generalization of a single example. It also learns from a small set of examples. Third, the agent learns a task in terms of all the parameters necessary for task execution. For that reason, we believe that our learning approach is very appropriate for being used in the context of PBD systems.

Although Agent-Disciple and Captain already illustrate a synergistic combination of basic techniques from ML, KA and PBD, many improvements remain to be made, particularly in the area of general methods of designing application specific interfaces and integrating the agent training environment into applications.

Acknowledgments

The authors thank David Hille, J. Mark Pullen and Tomasz Dybala for their contributions to this work. This research was partially supported by the Advanced Research Projects Agency Grant No. N66001-95-D-8653, DMSO Contract 100-91-C-0033 and the NRC Twinning Program with Romania.

References

Ceranowicz A. (1994). ModSAF Capabilities. In *Proc. 4th Conference on Computer Generated Forces and Behavior Representation.* May. Orlando, Florida.

Cypher, A. (Ed). (1993). *Watch What I Do: Programming by Demonstration.* MIT Press, Cambridge, Mass.

Dybala T. and Tecuci G. (1995). Shared Expertise Space: A Learning Oriented Model for Cooperative Engineering Design. In *Proc. IJCAI-95 Workshop on Machine Learning in Engineering.* August. Montreal, Canada.

Hieb, M.R. (1996). Teaching Instructable Agents Through Multistrategy Apprenticeship Learning. *Ph.D. Thesis.* School of Inf. Technology and Eng., George Mason Univ.

Hieb, M.R., Tecuci, G., Pullen J.M., Ceranowicz A., & Hille D. (1995). A Methodology and Tool for Constructing Adaptive Command Agents for Computer Generated Forces. In *Proc. 5th Conf. on Computer Generated Forces and Behavioral Representation.* May. Orlando, FL.

Hille D., Hieb M.R., Tecuci G. & Pullen J.M. (1995). Abstracting Terrain Data through Semantic Terrain Transformations. In *Proc. 5th Conf. on Computer Generated Forces and Behavioral Representation.* May. Orlando, FL.

Lieberman, H. (1994). A User Interface for Knowledge Acquisition From Video. In *Proc. Eleventh Conference on Artificial Intelligence*, Morgan Kaufmann.

Maulsby, D. (1994). Instructable Agents. *PhD Thesis.* Department of Computer Science. University of Calgary.

Maulsby, D. and Witten, I.H. (1995). Learning to Describe Data in Actions. *Proc of ICML-95 Workshop on Learning from Examples vs. Programming by Demonstration*, CA.

Mitchell, T. (1978). Version Spaces: An Approach to Concept Learning. *Ph.D. Thesis,* Stanford Univ.

Tambe, M., Johnson, W.L., Jones, R.M., Koss, F., Laird, J.E., Rosenbloom, P.S. & Schwamb, K. (1995). Intelligent Agents for Interactive Simulation Environments. *AI Magazine.* 16(1), Spring.

Tecuci, G. (1992). Automating Knowledge Acquisition as Extending, Updating and Improving a Knowledge Base. *IEEE Trans. of Systems, Man and Cybernetics.* 22(6).

Tecuci, G. & Hieb, M.R. (1994). Consistency-driven Knowledge Elicitation: Using a Machine Learning-oriented Knowledge Representation to Integrate Learning and Knowledge Elicitation in NeoDISCIPLE. *Knowledge Acquisition.* 6(1).

Tecuci, G. & Kodratoff, M.R. (Eds). (1995). *Machine Learning and Knowledge Acquisition: Integrated Approaches.* Academic Press.

Tecuci, G. & Hieb, M.R. (1996). Teaching Intelligent Agents: the Disciple Approach. *International Journal of Human Computer Interaction.* To appear.