Learnable Representation for Real World Planning

Mihai Boicu, Gheorghe Tecuci, Bogdan Stanescu, Liviu Panait, Cristina Cascaval

Learning Agents Laboratory, Department of Computer Science, MSN 4A5, George Mason University, Fairfax, VA 22030
{mboicu, tecuci, bstanesc, lpanait, ccascava}@gmu.edu

Abstract

This paper presents a learnable representation for real-world planning systems. This representation is a significant extension of the ones used in the most recent systems from the Disciple family, the Disciple-Workaround system for plan generation, and the Disciple-COA system for plan critiquing. The representation is defined to support an integration of domain modeling, knowledge acquisition, learning and planning, in a mixed-initiative framework. It also helps to remove the current distinction between the development phase of a planning system and its maintenance phase. It provides an elegant solution to the knowledge expressiveness / knowledge efficiency trade-off, and allows reasoning with incomplete or partially incorrect knowledge. These qualities of the representation are supported by several experimental results.

1 Introduction

A great challenge in constructing a real world planning system is the selection of an appropriate representation of the domain knowledge that has to satisfy several constraints, generally regarded as being contradictory. In this paper we discuss a representation of knowledge that has several desirable properties for planning, properties that are derived primarily from the fact that the representation is learnable directly from a subject matter expert. This learnable knowledge representation is being developed as part of the evolving Disciple theory, methodology, and tools (Tecuci et al., 1998). Our long-term goal with the Disciple research is to enable users that do not have any special training in knowledge engineering and computer science, to build by themselves agents that can act as intelligent assistants. While Disciple is being developed as a general approach, applicable to a wide range of domains, planning has received considerable attention in our recent work, performed as part of the DARPA's High Performance Knowledge Bases program (Cohen et al., 1998). In this program we have developed two systems, Disciple-Workaround and Disciple-COA.

Disciple-Workaround was developed to address the workaround challenge problem which consists of rapidly developing and maintaining a knowledge-based agent that is able to plan how a convoy of military vehicles can circumvent or overcome obstacles in their path (such as damaged bridges or minefields). The primary goal of the agent is to estimate the time needed to workaround such an obstacle in a given situation (Jones, 1998). Disciple-COA was developed to address the COA challenge problem which consists of rapidly developing and maintaining a critiquing agent to evaluate military Courses Of Action that were developed as hasty candidate plans for ground combat operations (Jones, 1999). Disciple-COA is able to identify the strengths and the weaknesses of a COA with respect to the principles of war and the tenets of army operations.

The learnable representation presented in this paper emerged as an extension of the ones used in Disciple-Workaround and Disciple-COA. In this paper we will concentrate on those aspects of this representation that support planning-related activities that lie outside basic plan generation. We will clarify what we mean by learnable representation; how a domain model that exists in the mind of the expert is captured at different levels of abstractions and formalization (informal for modeling of the domain, formal for actual planning, and natural language for user-agent communication); how the rationale underlying the generated plans is acquired and represented; how the reasoning complexity is managed by using different levels of details in user-agent communication; how time-ordering constraints and functions are learned and represented; and finally how this representation supports reasoning with incomplete or partially incorrect knowledge.

The rest of this paper is organized as follows. We first introduce the basic Disciple approach to agent development, and its knowledge representation, and clarify what we mean by learnable representation. Then we use an example from the workaround domain to address several knowledge representation issues that are important for real-world planning systems. We continue with presenting several experimental results that support our claims concerning this representation, and conclude the paper.

2 User-Developed Planning Agents

Disciple is a learning agent shell that allows customization for a particular domain. It consists of a learning and knowledge acquisition engine as well as a problem solving engine. It also supports building an agent with a knowledge base consisting of an ontology that defines the concepts from the application domain, and a set of task reduction rules expressed in terms of these concepts. The problem-solving engine is based on the task reduction paradigm. In this paradigm, a task to be accomplished by the agent is successively reduced to simpler tasks until the initial task is reduced to a set of elementary tasks that can be immediately performed. This problem solving paradigm applies very naturally to hierarchical non-linear planning (Tate, 1977; Wilkins, 1988).

While an ontology is characteristic to a certain domain (such as an ontology of military units, or an ontology of military equipment, in the case of the military domain), the rules are much more specific, corresponding not only to a certain type of application in that domain, but even to a specific expert, representing his or her characteristic problem-solving strategies (e.g. rules for an agent that assists a military commander in critiquing courses of action, or rules for an agent that assists in planning the repair of damaged bridges or roads). Therefore the rules and the tasks composing them have to be acquired from the expert. Based on this observation, the process of developing a Disciple agent starts with importing an initial ontology from an external knowledge server, such as CYC (Lenat, 1995). This process continues with teaching the agent how to perform various tasks, in a way that resembles how an expert would teach a human apprentice when solving problems in cooperation. During this teaching process, the agent will learn the tasks and the rules from the expert and will also extend its ontology.

3 Modeling, Planning and Learning

The process of teaching the agent integrates domain modeling, planning and learning, in a mixed-initiative framework. The expert selects or defines a planning task and Disciple tries to automatically generate a plan by applying the task reduction rules from its knowledge base. This may produce several good plans, wrong plans, or no plans at all. Each of these situations is an opportunity for learning, but let us here consider the case where no complete plan is produced. In this case the agent will help the expert to identify the most promising partial plan that needs to be extended to a complete and correct plan. The expert and the agent will enter a mixed-initiative modeling, planning and learning process, working together to complete the plan. In the same time, the agent will learn from this joint activity. While trying to reduce a current task to a set of subtasks, the expert and the agent may encounter three different situations:

A) No rule is applicable and therefore no reduction is proposed by Disciple. In this case the expert enters a modeling phase where he or she has to define the reduction of the current task. From this example Disciple will learn both a general task reduction rule and several tasks.

B) A reduction rule is applied to reduce the current task into subtasks, and the expert accepts it. In this case the rule may be automatically generalized (depending on how the reduction was generated) and the planning process continues with further reducing one of the subtasks.

C) A reduction rule is applied to reduce the current task into subtasks, but the expert rejects this reduction. In this case the applied rule is specialized (in a mixed-initiative scenario) to no longer generate the wrong reduction, and the process continues as in case A.

Figure 1 shows an example of task reduction provided by the expert, as a result of a modeling process. In order to reduce the top level task in Figure 1, the expert formulates a question that clarifies what aspects of the situation determine the solution (“What bank needs to be reduced?”). Then the expert finds and formulates the answer to the question (“Both site107 and site105”), and defines the solution (the ordered subtasks shown at the bottom of Figure 1). All these phrases are in natural language, except that whenever a reference to a concept or instance from the ontology is needed, the user has to use the name of that instance in the ontology (such as “bulldozer-unit201” or “bank”). The agent assists not only in identifying the name of an object, but also in structuring the natural language phrases representing the tasks (as shown in Figure 1). In addition, it learns general tasks patterns from these task examples, as shown in section 4.

Once the example reduction is defined, a mixed-initiative explanation generation process starts in which the agent will find the explanations of this reduction, shown in the left and right hand sides of Figure 1. This process is discussed in section 5. Then, based on the example reduction and its explanations, the agent will learn the rule from Figure 2. This rule is refined in future planning situations of type B or C when it is applicable.

4 Learning of Task Representation

The format of a task in the user provided example is a compromise between a free natural language description and a restricted formal representation. The Disciple agent helps the expert to define the tasks in a predefined structure, but inside that structure there are no additional restrictions (except that of using the object names from the ontology). For instance, the expert expresses the first subtask of the reduction in Figure 1 in natural language:

“Reduce the slope of site107, by direct cut, using bulldozer-unit201, to allow the fording of unit10”

A task example must start with a task name (an unrestricted natural language phrase that does not contain any object name) and is followed by one or more task description features. Each task description feature includes at most one relevant object from the ontology (concept or instance) or a constant (number or string). Based on this structure, automatically (or sometimes requiring some minimal help from the user), the agent will create a general task pattern that includes generic variables, formal task features, and plausible version spaces for the ranges of these features. For instance, the task pattern learned from the above task example is:

Reduce-the-slope
  of ?O1 (pub: (geographical-region) plb: (site107))
  by-direct-cut
  using ?O2 (pub: (equipment) plb: (bulldozer-unit201))
  to-allow-the-fording-by ?O3 (pub: (modern-military-organization)
                                plb: (unit10))

While the first time a task is used, the user can define it any way he or she wants, future uses of this task will have to use the same structure. These new examples of the task pattern will be used to learn better ranges for the features of the task. The structuring of the tasks is also used in reasoning about task similarities during domain modeling, planning and learning. Disciple also learns hierarchical relations between tasks that are used in domain modeling to support creative reasoning.

In previous versions of Disciple, the user had to first model the domain, then define the task features, and then define the tasks based on the defined feature. All these activities had to be done before using the task in an example. The new version of Disciple allows all these activities to take place in the same time, when the example is defined. Moreover, the user is primarily responsible only for domain modeling, because the tasks and tasks features are learned by the agent from the provided model of task reduction. In the next two sections we will briefly discuss the mixed-initiative learning of task reduction rules, starting from the provided example.

5 Mixed-Initiative Explanation Generation

To learn a rule from the example reduction, Disciple needs to understand why the reduction is correct. Finding the explanations of the task reduction is a mixed-initiative process of searching the agent's ontology, an explanation being a path of objects and relations in this ontology. This process is guided by three sources of knowledge: a set of heuristics for analogical reasoning, the Question and the Answer corresponding to the reduction to be explained, and a mixed-initiative process of hint refinement.

The heuristics for analogical reasoning are based on the similarity relations between the tasks and the features from Disciple's knowledge base, and on different types of structure similarity between the current example and the existing rules. In essence, Disciple identifies the rules that include tasks similar to those in the current example. Then it uses the explanations from which these rules have been learned as a guide to search for similar explanations of the current example. It displays the found explanations, ordered by their plausibility, and asks the expert to select the correct ones.

Guidance for explanation generation is also provided by the Question and the Answer from the example that identify the objects that should be part of the explanation, even though Disciple does not yet have the ability to understand these natural language phrases. However, in order to facilitate explanation generation, we are currently investigating techniques to partially understand the Question and the Answer.

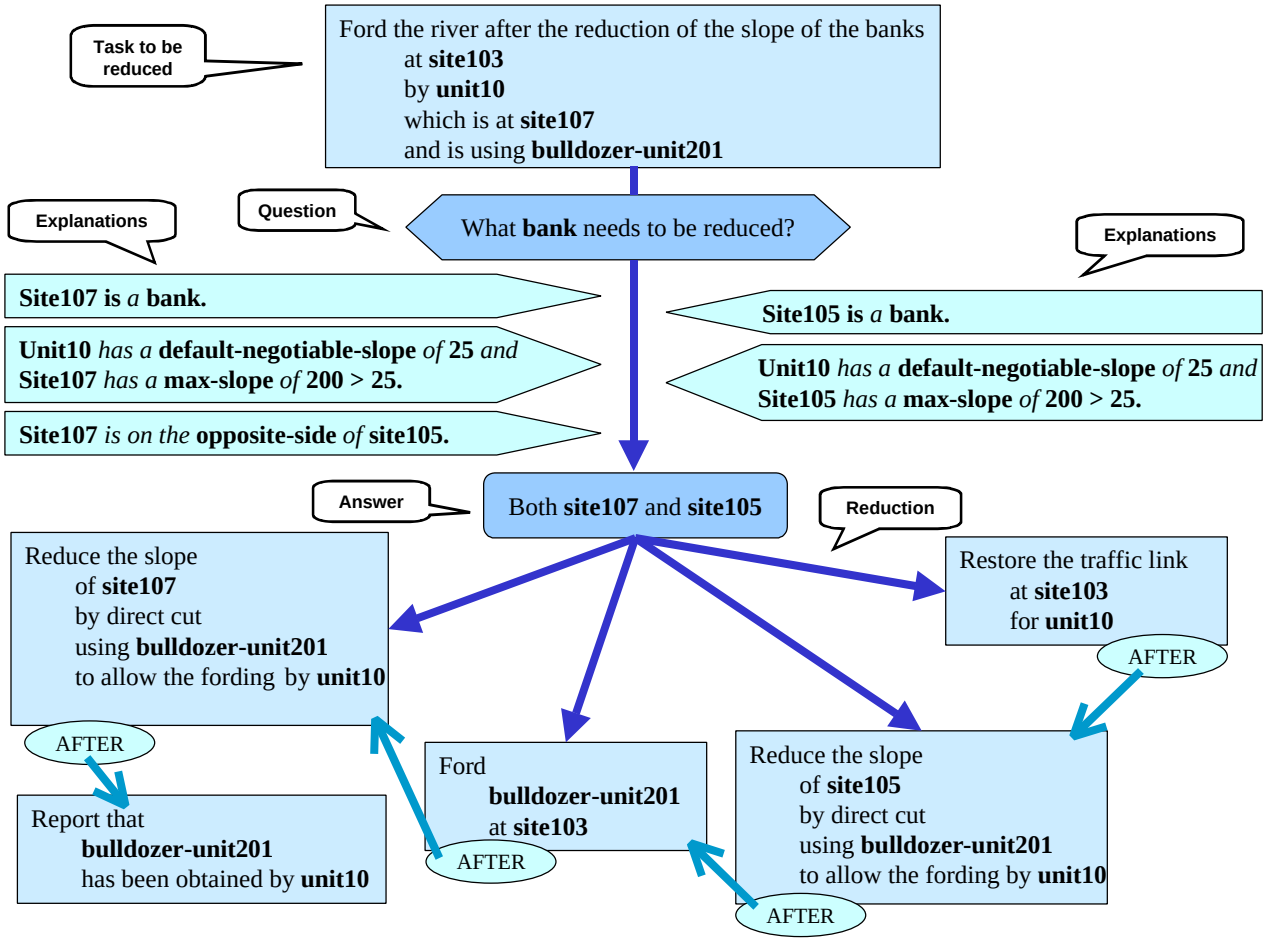

Figure 1. An example of task reduction

Finally, there is a mixed-initiative process of hint refinement. A hint might be a fragment of an explanation (such as an object or a relationship between two objects), or any other abstraction of an explanation. Both the expert and the agent can formulate an initial hint, and then each of them may propose possible hint refinements. The refinements proposed by the agent are based on an analysis of the structure of the ontology, and on the analogy with the hints from which other rules have been learned. The expert may also guide the agent in proposing various refinements of the current hint, and then may select the most promising refinements. This process continues until the hint is refined to an explanation accepted by the expert.

6 Learning of Plausible Version Space Rules

From the example task reduction and its explanations shown in Figure 1, the agent automatically generates the task reduction rule shown in Figure 2. This rule is a complex IF-THEN structure that specifies one or several conditions under which the task from the IF part can be reduced to the tasks from the THEN part. Each rule includes a main condition that has to be satisfied in order for the rule to be applicable. Partially learned rules, such as the one shown in Figure 2, do not contain exact conditions, but plausible version spaces for these conditions. Each such plausible version space is represented by a plausible upper bound condition which, as an approximation, is more general than the exact (but not yet known) condition, and a plausible lower bound condition which, as an approximation, is less general than the exact condition. In addition to the main condition, the learned rule also includes generalizations of the Explanations, and of the Question and its Answer, which basically represent the same information at higher levels of abstraction and formalization. This allows the Disciple agent to perform a wide range of reasoning processes. For instance, the Question and the Answer help the expert in domain modeling, by suggesting a certain line of reasoning, when the expert attempts to reduce a task similar with the one reduced by the current rule. They are also used by the natural language generation module of Disciple to generate the question and answer part of a task reduction step obtained by applying a rule. Representing knowledge in a rule at various levels of abstractions allows the Disciple agent to generate the justifications of its solutions at each of these levels, as has been demonstrated with Disciple-COA. The most abstract one included only the initial task, the sequence of questions and answers, and the final solution. A more detailed one also included the intermediate tasks. Finally, the most detailed one also included the explanations and the rules that generated the intermediate steps.

As the planning agent learns plausible version space rules, it can use them to propose routine, innovative or inventive solutions to the current problems. The *routine solutions* are those that satisfy the plausible lower bound conditions of the task reduction rules and are very likely to

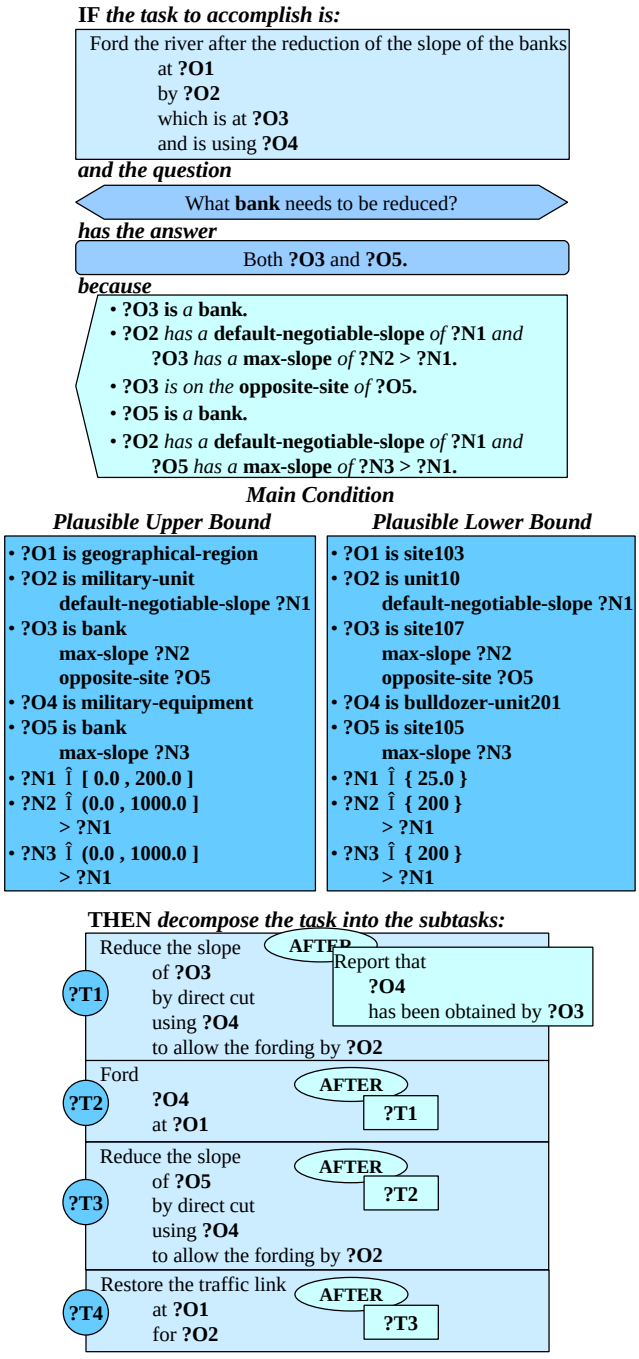

Figure 2: The rule learned from the example in Figure 1.

be correct. If such a solution is incorrect, then the corresponding rule is extended with an Except-for condition.

The *innovative solutions* are those that satisfy the plausible upper bound conditions. These solutions may or may not be correct, but each case will lead to a refinement of the task reduction rules that generated them. Correct solutions will lead to automatic generalization of the plausible lower bound condition of the rule. Incorrect solutions may lead to either the specialization of the plausible upper bound condition of the rule, or the addition or refinement of an Except-When condition. To illustrate

this last situation, let us consider that the Disciple agent generated a solution that was rejected by the expert because the bank to be reduced was made of rock. This failure explanation "site105 soil-type rock" will cause the extension of the rule in Figure 2 with the except-when plausible version space condition shown in Figure 3. In future planning situations the updated rule will only be applied if the Except-When condition will not be satisfied. During refinement, the rule may be augmented with several such Except-When conditions.

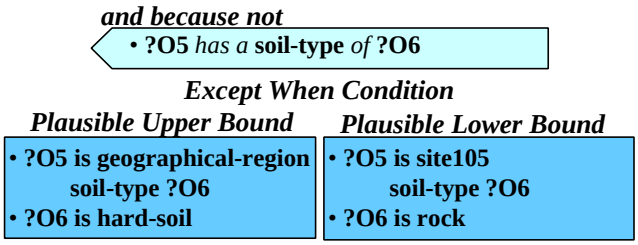

Figure 3: A learned Except When condition.

The *inventive solutions* are based on weaker forms of plausible reasoning (such as partial matching of the plausible conditions of the rules, and tasks similarity based on the structure of the ontology). An inventive task reduction step is based on several rules, and is generally a novel reduction based on tasks from these rules. From inventive solutions the agent will learn new plausible task reduction rules.

One can also notice that the general rule in Figure 2 contains generalized AFTER relations, in its right hand side. They are also generated automatically from the example in Figure 1. An important feature of our representation is that the value of an AFTER relation may be an incomplete description of a task that may appear in some other part of the plan being generated. For instance, the first task of the reduction in Figure 1 (i.e. "Reduce the slope of site107...") can only start after it has been reported that unit10 has obtained bulldozer-unit201. This is a very elegant solution to the generation of nonlinear plans that allowed us to treat task reductions as if they were independent. Consider, for instance, a situation that involves a damaged bridge that is also mined. We can reduce this task into two subtasks, de-mining the bridge, and installing an AVLB bridge, and further reduce these subtasks independently. However, in each case where an important condition is satisfied, we introduce a Report task that signals this condition. Similarly, at each point where such a condition is needed, we would introduce a corresponding AFTER relation. For instance, in the de-mining planning tree, after the action of de-mining site107, the near bank of the river, we introduce the task "Report that site107 has been de-mined..." Similarly, in the install-AVLB planning tree, the task "Reduce the slope of site107..." will include an "AFTER Report that site107 has been de-mined..." This works also in the cases when the bridge is not mined. Because the generated plan will not contain any de-mining report, the corresponding AFTER relation of the action "Reduce the slope of site105 ..." will be ignored.

The task reduction examples may also contain functions that are automatically generalized in the learned rules. The workaround domain made heavy use of functions for estimating the time required for various operations (such as reducing the slope of a bank, or emplacing a bridge).

In addition to being learned based on very simple interactions with a domain expert, the Disciple rules show a practical solution to the expressiveness/efficiency trade-off. The different types of conditions of a rule (main, except-when, for, and except-for) offer it a high level of expressiveness. Maintaining also generalized justifications at various levels of abstractions and formalizations (informal and abstract for Question and Answer; formal and abstract for the explanations; formal and compiled for the conditions), allows Disciple to provide justifications of its actions at each of these levels. On the other hand, the (compiled) conditions of the rules allow very efficient reasoning. Notice also that a partially learned rule, such as the one in Figure 2, is an incomplete and possibly partially incorrect piece of knowledge. However, as has been shown above, Disciple could use such rules in various types of reasoning, from routine to creative.

7 Robustness of the Learned Representation

In the Disciple approach, the whole process of developing the KB of the planning agent is one of creating and refining (or adapting) knowledge pieces to better represent the domain model of the teaching expert. The same operations are involved in updating the KB in response to changes in the application environment or in the goals of the planning agent, that is, in maintaining the planning agent. Therefore, in the life-cycle of a Disciple planning agent there is no distinction between KB development and KB maintenance. If we take into account that the Disciple approach significantly speeds up the process of KB development (as demonstrated in the HPKB program), and that, traditionally, KB maintenance costs are about four times the (already high) KB development costs, then we can conclude that Disciple can have a very significant positive impact on the development and maintenance costs for real world planning systems.

In addition to its ability to update the task reduction rules to account for new examples, as presented in the previous section, Disciple also implements a mechanism of rule re-generation from (a partial set of) the examples and the explanations from which it was initially learned. This is precisely what might be required during KB maintenance when, for instance, some concepts that appear in the rule might be deleted from the ontology. In such a case, the automatic regeneration of the rule, based on the updated ontology, will lead to a new rule that no longer contains the concept that was deleted.

Disciple also provides intelligent assistants for ontology maintenance, such as the delete assistant or the copy assistant (Boicu et al., 1999), that control the operations on the ontology, in order to maintain its consistency. Let us consider, for instance, the deletion of a concept from the ontology. Many knowledge elements may be affected by

this deletion, such as its subconcepts, features that have the deleted concept is their ranges or domains, rules that contain the concept in their conditions, etc. In such a case the delete assistant engages in a mixed initiative dialog with the expert in order to identify the best strategies to update its knowledge, while maintaining its consistency.

8 Experimental Results and Conclusions

Disciple-Workaround and Disciple-COA have been the subject of intense experimentation during the HPKB program, and the results obtained support the claims made in this paper about the suitability of our representation for developing and maintaining real world planning systems. Disciple-Workaround demonstrated that a knowledge engineer can rapidly teach Disciple using Military Engineering manuals and sample solutions provided by a domain expert. During the 17 days of DARPA's 1998 evaluation, the KB of Disciple was increased by 72% (from the equivalent of 5,920 simple axioms to 10,162 simple axioms) with almost no decrease in performance. Also, Disciple-Workaround was rapidly extended with new planning strategies, and was incorporated by Alphatech into a more complex application presented at EFX'98, the Air Force's showcase of the most promising technologies.

With Disciple-COA we achieved two new significant milestones. For the first time we developed the KB around an ontology created by another group (Teknowledge and Cycorp), demonstrating both the feasibility of knowledge reuse with the Disciple approach, and the generality of the Disciple rule learning and refinement methods. Moreover, the Disciple-COA agent was taught even more rapidly than the Disciple-Workaround agent, and has again demonstrated a very high performance. In this case, Disciple was taught jointly by a domain expert and a knowledge engineer, and its knowledge base increased by 46% in 8 days of evaluation, from a size of 6,229 simple axioms equivalent to a size of 9,092 simple axioms equivalent. The final knowledge base contained 801 concepts, 444 object and task features, 360 tasks and 342 task reduction rules. Also, each COA was represented with around 1,500 facts. The second milestone was the knowledge acquisition experiment performed at the US Army Battle Command Battle Lab in Fort Leavenworth, KS. In this experiment four military experts with no prior knowledge engineering experience received very limited training in the teaching of Disciple-COA and then each succeeded to significantly extend its knowledge base (with around 275 simple axioms in about 3 hours), receiving only very limited support from a knowledge engineer.

Our research has emphasized the acquisition of planning knowledge directly from a subject matter expert. However, an environment for the development of real world planners should also include a wide range of powerful tools for a knowledge engineer, such as those from the Expect system (Kim & Gil, 1999). Expect guides the user in extending and updating the KB by deriving expectations from the use of knowledge pieces and their interdependencies, and by following knowledge acquisition scripts (Tallis and Gil, 1999). We started to address this issue by developing various intelligent assistants for KB management.

In the advisable planning system developed by Myers (1996) the user may guide the plan generation by providing strategic advice on how to achieve a certain goal (such as to use a certain object or task). We are considering the integration of such a capability into Disciple, by enabling it to learn meta-rules from the expert's advice received during the mixed-initiative plan construction.

Acknowledgments. This research was supported by AFOSR and DARPA through the grants F49620-97-1-0188 and F49620-00-1-0072. Dorin Marcu, Mike Bowman, Florin Ciucu, Cristian Levcovici and Marinel Alangiu have contributed to the development of the current version of the Disciple system. This paper has also benefited from the insightful comments of Yolanda Gil.

References

Boicu M., Tecuci G., Bowman M., Marcu D., Lee S.W. and Wright K., (1999). A Problem-Oriented Approach to Ontology Maintenance, *Proc. of AAAI-99, Workshop on Ontology Management*, Orlando, Florida, AAAI Press.

Cohen P., Schrag R., Jones E., Pease A., Lin A., Starr B., Gunning D., and Burke M. (1998). The DARPA High-Performance Knowledge Bases Project, *AI Magazine*, 19(4),25-49.

Jones E. (1998). *HPKB Year 1 End-to-End Battlespace Challenge Problem Specification*, Burlington, MA.

Jones E. (1999). *HPKB Course of Action Challenge Problem Specification*, Burlington, MA.

Kim, J. and Gil, Y. (1999). Deriving Expectations to Guide Knowledge Base Creation, in *AAAI-99/IAAI-99 Proc.*, AAAI Press, CA.

Lenat, D.B. (1995). CYC: A Large-scale Investment in Knowledge Infrastructure *Comm of the ACM* 38(11):33-38.

Myers K.L. (1996). Strategic Advice for Hierarchical Planners, in Principles of Knowledge Representation and Reasoning, KR-96, San Francisco, CA: Morgan Kaufman.

Tallis, M. and Gil, Y. (1999). Designing Scripts to Guide Users in Modifying Knowledge-based Systems, in *AAAI-99/IAAI-99 Proc.*, AAAI Press, CA.

Tate, A. (1977). Generating Project Networks, In *Proceedings of IJCAI-77*, Massachusetts, pp. 888-893.

Tecuci, G. (1998). *Building Intelligent Agents: An Apprenticeship Multistrategy Learning Theory, Methodology, Tool and Case Studies.* London, England: Academic Press.

Tecuci, G., Boicu, M., Wright, K., Lee, S.W., Marcu, D. and Bowman, M. (1999). An Integrated Shell and Methodology for Rapid Development of Knowledge-Based Agents, in *AAAI-99/IAAI-99 Proc.*, AAAI Press, CA.

Wilkins, D. (1988). *Practical Planning: Extending the Classical Artificial Intelligence Paradigm*, San Mateo, CA: Morgan Kaufman