Learning by Observation and Practice: Towards Real Applications of Planning Systems

Xuemei Wang **Jaime Carbonell**[*]
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
wxm+, jgc+@cs.cmu.edu

Abstract

Acquiring knowledge from experts for planning systems is a rather difficult knowledge engineering task, but is essential for any applications of planning systems. This work addresses the issue of automatic acquisition of planning operators. Operators are learned by observing the solution traces of experts agents and by subsequently refining knowledge in a learning-by-doing paradigm. It is domain-independent and assumes minimal requirements for *a priori* knowledge and expert involvement in order to reduce the burden on the knowledge engineerer and domain experts. Planning operators are learned from these observation sequences in an incremental fashion utilizing a conservative specific-to-general inductive generalization process. In order to refine the new operators to make them correct and complete, the system uses the new operators to solve practice problems, analyzing and learning from the execution traces of the resulting solutions or execution failures. We describe techniques for planning and plan repair with incorrect and incomplete domain knowledge, and for operator refinement through a process which integrates planning, execution, and plan repair. Our learning method is implemented on top of the PRODIGY architecture(Carbonell, Knoblock, & Minton 1990; Carbonell *et al.* 1992) and is demonstrated in the extended-strips domain(Minton 1988) and a subset of the process planning domain(Gil 1991).

Introduction

Acquiring knowledge from experts for planning systems is a rather difficult knowledge engineering task, but is essential for any applications of planning systems. Most of the work on automatic acquisition of planning knowledge have concentrated on learning *search control knowledge* (Fikes, Hart, & Nilsson 1972; Mitchell, Utgoff, & Banerji 1983; Minton 1988; Veloso 1992). The work described in this paper addresses the issues of automatic acquisition of *domain knowledge*, i.e. planning operators.

Previous work on acquiring planning operators mostly requires a partially correct domain knowledge before learning(Gil 1992). Such initial knowledge may be difficult to acquire sometimes. This work aims at further reduce the burdern of the knowledge engineerer and domain experts by assuming minimal requirements for *a priori* knowledge and expert involvement. Our learning system no longer requires a large chuck of inital knowledge about the planning domain. Instead, the learning system is given at the outset only the description language for the domain, which includes the types of objects and the predicates that describe states and operators. Motivated by the fact that it is usually much easier for the domain experts to give a solution to the problems than to write down the operators they use to solve the problems, we have developed a framework for learning planning operators by observing the solution traces of expert agents and subsequent knowledge refinement in a learning-by-doing paradigm(Anzai & Simon 1979). Thus the experts involvement is reduced to simply given solutions to problems.

Learning by observation and by practice provides a new point in the space of approaches for planning and learning. In addition to the problems faced in previous work such as numerous irrelevant features in the state when the operators are applied (Gil 1992), our learning system also needs to plan with not only incomplete, but also incorrect operators, plan repair is necessary when the initial plan cannot be completed because of some unmet necessary preconditions in the operators in the plan; planning and execution must be interleaved and integrated. This paper describes our approach to address these problems. Empirical results in the process planning domain (Gil 1991) demonstrates the effectiveness of this approch. We also discuss some extensions that are essential for applying our technique to real world domains.

Learning Planning Operators

We use the PRODIGY architecture(Carbonell, Knoblock, & Minton 1990; Carbonell *et al.* 1992) as our test-bed for the learning system. PRODIGY's domain knowledge is represented using an extended form of STRIPS-like oper-

[*]This research is sponsored by the Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, and the Advanced Research Projects Agency (ARPA) under grant number F33615-93-1-1330. Views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing official policies or endorsements, either expressed or implied, of Wright Laboratory or the United States Government.

ators (Fikes & Nilsson 1971). The goal of this work is to learn the preconditions and the effects of the operators. We assume for now that the operators have conjunctive preconditions and no conditional effects, everything in the state is observable, and there is no noise in the state.

The architecture for learning by observation and practice in planning includes *the observation module*, *the learning module*, *the planning module*, and *the plan execution module*, as illustrated in Figure 1.

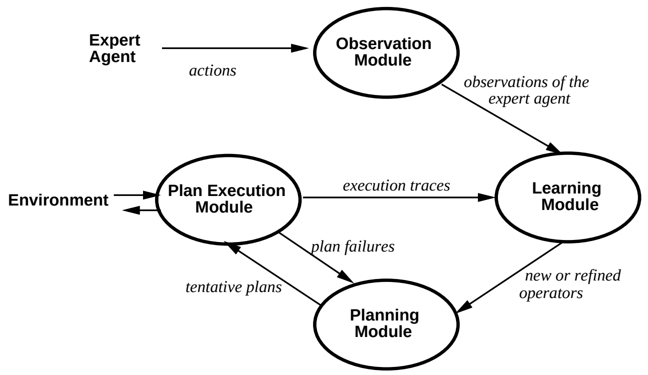

Figure 1: Overview of the data-flow among different modules of the learning system. The observation module provides the learning module with observations of the expert agent. The learning module formulates and refines operators in the domain. The planning module is essentially the PRODIGY planner modified to use some heuristics for planning with incomplete and incorrect operators as well as for plan repair, it generates tentative plans to solve practice problems. The plan execution module executes the plans, providing the learning module with the execution traces, and passing the plan failures to the planning module for plan repair.

Learning Operators from Observation

Operators for the domain are learned from these observation sequences in an incremental fashion utilizing a conservative specific-to-general inductive generalization process. When an operator is observed for the first time, we create the corresponding operator such that its precondition is the parameterized *pre-state*, and its effect is the parameterized *delta-state*. The type for each variable in the operator is the most specific type in the type hierarchy that the corresponding parameterized object has[1]. This is the most conservative constant-to-variable generalization. Generalizing the observation in Figure 2 yields the operator shown in Figure 3.

New observations are used to further refine both the preconditions and the effects of the operators: the preconditions are generalized by removing facts that are not present in the *pre-state* of the new observation[2]; the effects are augmented by adding facts that are in the *delta-state* of the observation.

[1]Objects that do not change from problem to problem are treated as constants and are not parameterized, such as the object ROBOT in the extended-strips domain.

[2]A precondition is removed iff the predicate of the precondition is not present in the new *pre-state*.

```
Pre-state:

(connects dr13 rm1 rm3) (connects dr13 rm3 rm1)
(dr-to-rm dr13 rm1) (dr-to-rm dr13 rm3)
(inroom robot rm1) (inroom box1 rm3) (pushable box1)
(dr-closed dr13) (unlocked dr13) (arm-empty)

Delta-state: add: (next-to robot dr13)
```

Figure 2: An observation of the state before and after the application of the operator GOTO-DR

```
(operator goto-dr
  (preconds ((<v1> door) (<v2> object) (<v3> room) (<v4> room))
   (and (inroom robot <v4>)
        (connects <v1> <v3> <v4>)
         (connects <v1> <v4> <v3>)
         (dr-to-rm <v1> <v3>)
         (dr-to-rm <v1> <v4>)
         (unlocked <v1>)
         (dr-closed <v1>)
         (arm-empty)
         (inroom <v2> <v3>)
         (pushable <v2>)))
  (effects nil ((add (next-to robot <v1>)))))
```

Figure 3: Learned operator GOTO-DR when the observation in Figure 2 is given to the learning module

For example, given the observation in Figure 4, the operator GOTO-DR is refined: (dr-closed <v1>) is removed from the preconditions, and the effects become:

```
(effects ((<v5> Object))
  (add (next-to robot <v1>)) (del (next-to robot <v5>)))
```

```
Pre-state:

(connects dr12 rm2 rm1) (connects dr12 rm1 rm2)
(dr-to-rm dr12 rm1) (dr-to-rm dr12 rm2)
(next-to robot a) (dr-open dr12) (unlocked dr12)
(inroom robot rm1) (inroom c rm2) (inroom b rm1)
(inroom a rm1) (pushable c) (pushable b) (arm-empty)

Delta-state: add: (next-to robot dr12) del: (next-to robot a)
```

Figure 4: The second observation of the operator GOTO-DR

Refining Operators with Practice

The operators acquired by the learning module are themselves incorrect or incomplete. They can be further refined during practice. To do so, PRODIGY's search algorithm is augmented with some domain-independent heuristics to generate plans that are possibly incomplete or incorrect. These plans are executed in the environment to further generate learning opportunities.

Planning with incorrect operators PRODIGY's standard planner assumes a correct action model. But before our learner acquires a complete model of the domain, the operators can be incomplete or incorrect in the following ways: (a) *over-specific preconditions:* when a learned operator has unnecessary preconditions. In this case, the system has to achieve its unnecessary preconditions during planning. Not only does this force the system to do unnecessary search, but also it can make many solvable problems unsolvable because it may not be possible to achieve some unnecessary preconditions. (b) *incomplete effects:* when some effects of an operator are not learned because they are present in the *pre-state* of the operator. While searching for a plan, PRODIGY applies in its internal model an operator that is missing some effects, the state becomes incorrect.

These problems make it very difficult to generate a correct plan for a problem. Therefore, we relax the assumption that all the plans generated by the planner are correct. The system intentionally generate plans that only achieve a subset of the preconditions of the operators. Of course, this introduces the possibility of incomplete or incorrect plans in that a necessary precondition may be unsatisfied, necessitating plan repair. But if an operator with unmet preconditions is executed in the environment, are unmet preconditions can be removed from the preconditions, thus the operator is further refined.

Plan repair Plan repair is necessary when a planned operator fails to execute in the environment because of unmet necessary preconditions. Since there can be multiple unmet preconditions, and since some of them may be unnecessary preconditions, it is not necessary to generate a plan to achieve all of them. Therefore, the plan repair mechanism generates a plan fragment to achieve one of them at a time. The unmet preconditions are ranked according to whether or not they have they contain variables that appear in the effects of the operators. Ties are breaked arbitrarily. If the system fails to achieve any of the unmet preconditions, or the operator still cannot be executed in the environment even all its preconditions have been achieved, the plan repair mechanism proposes a different operator.

Learning critical preconditions As mentions in the previous section, the system only achieves a subset of the preconditions of the operators. This subset of preconditions are called *critical preconditions*. Critical preconditions are incrementally learned during practice. They can be learned in the following two ways:

- If all but one precondition P of an operator *OP* are satisfied in the state, and *OP* fails to execute in the environment, precondition P is learned as a *critical precondition*.
- As discussed before, if an operator *OP* fail to execute in the environment because of unmet preconditions, the system repairs the plan by generating additional plans steps to achieve one of the preconditions. If the system executes the additional steps, and after achieving P, *OP* becomes applicable, then precondition P is learned as a *critical precondition*.

The more critical precondition learned by the system, the less likely will the plans generated fail. The critical preconditions thus learned cannot be proved to be the necessary preconditions assuming there may be negated preconditions. But in practice, they are highly accurate.

Learning negated preconditions The learned operators are used to solve practice problems. During practice, if an operator is *un-executed* even when all the preconditions of the operators are satisfied, the learning module discovers that some negated preconditions are missing. The negations of all the predicates that are true in the current state when the operator is *un-executed*, but are not true in the observations of this operator are conjected as potential negated preconditions, and are added to the preconditions of the operator. Some of the added negated preconditions may be unnecessary, and they will be removed in the same way as all other preconditions with more observations and practices.

Empirical Results

The learning algorithm described in this paper has been tested in the extended-STRIPS(Minton 1988) domain and a subset of the process planning domain(Gil 1991). It also easily learns operators in more simple domains such as the blocksworld domain and the logistics domain. The following results are from the process planning domain. The learning system is first given the solutions of 30 problems to observe, then given 25 problems to practice. 15 test problems are used. All the above problems are randomly generated.

Figure 5 shows how the amount of executions (i.e. the sum of successfully and unsuccessfully executed operators) to solve test problems decrease with more practice. Figure 6 shows how the amount of planning (i.e. the total number of nodes generated) to solve the test problems decrease with more practice problems. In both cases, the solid lines show the performance of the bassic prodigy without control knowledge. We see that observation and practice, the learning system has acquired the necessary operators to solve problems effectively.

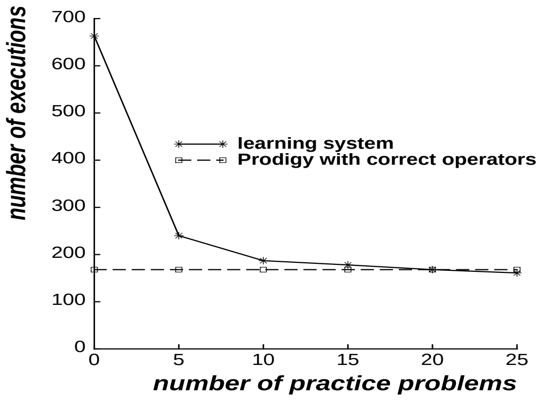

Figure 5:

Summary

In summary, learning, planning, plan repair, and plan execution are tightly integrated in our learning system, and operators are learned through a specific-to-general inductive generalization process. (Wang 1994) provides more details and examples of the learning algorithm. The empirical results demonstrate the effectiveness of our approch.

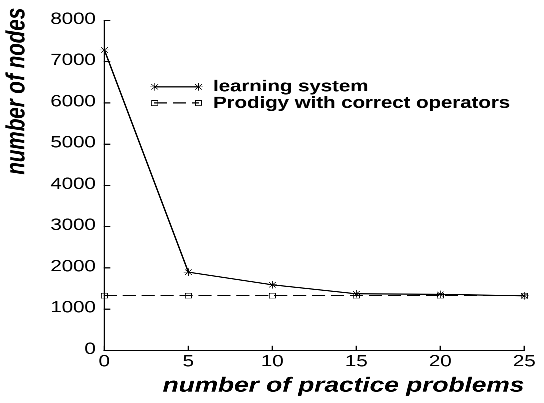

Figure 6:

On to Real Applications: Extending the Current Framework

The framework described so far has many characteristics that are essential for applications on real problems. These characteristics include the incremental learning of operators, and the ability to use partially correct domain knowledge to solve problems through an integration of planning, plan repair, execution, and learning. The current framework is being further extended to deal with other important issues in real world problems. In this section, we discuss different possible extensions, such as handling imperfect perceptions of the states, and dealing with variability in experts as teaching agents.

Imperfect Perception of the States

In real world domains, the perception of the domain is rarely perfect. Consider the a set of predicates that describe the operators and states. We have thus far assumed all predicates are directly observable. For example in the process planning domain (Gil 1991), most predicates, such as SHAPE-OF, HAS-CENTER-HOLE, etc, are observable. But a subset of the predicates are unobservable, e.g. HOLDING-WEAKLY. To learn under this circumstance, we assume that these unobservables are determinable by the system while it is executing its own plans, but not by observations of the expert. Take the HOLDING-WEAKLY as an example, although the system cannot observe the other agent is holding weakly or strongly, while it is applying the action itself, holding-weakly is known to the system, i.e. observable. Another example would be HEAVY. The weight of an object is not determinable by remote observation, but by actual manipulation.

Having unobservable facts in the state of the other agent means that the observed state is a subset of the real state, and therefore, the operators we learn may miss some preconditions and effects that involve the unobservable facts. Assuming that these unobservable facts can be determined in the system's own state when it is executing plans, these missing preconditions and effects may be learned as follows: when an operator is executed by the system during its practice, if some unobservable facts of the state are present in the *pre-state*, or *post-state*, they can be added to the preconditions or effects of the operator; or when an operator fails to execute even though all its preconditions are satisfied, then the planner knows that it is missing some preconditions. The system can propose some unobservable facts as the preconditions of this operator, and add them to the operator The relevance of the added preconditions will be determined in later executions and retained or not accordingly, until a version of the operator that executes correctly is formulated.

There may also be random noise in the state so that a fact may be incorrectly observed as true in state when it is not, or vice versa. To handle such noise, we can have a threshold for each precondition so that a predicate is removed from the preconditions only when it is absent from the state more often than the threshold.

Teaching Agent Variability

While it is important to alleviate the burden on the experts in the process of knowledge engineering, the variability of experts as teaching agent affects the speed (e.g. the number of observations needed to learn the operators) and quality (e.g. the percentage of unnecessary preconditions in the learned operators) of knowledge engineering. While in general it is difficult for the expert to write the exact preconditions and effects of the operators, it is easier for them to help the learning systems in other ways, such as maximizing the state diversity in which the operators are applied, maximizing different ways to solve similar problems etc, as illustrated roughly in Figure 7.

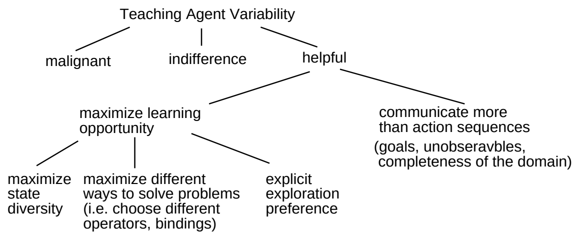
Teaching Agent Variability
malignant
indifference
helpful
maximize learning opportunity
communicate more than action sequences
(goals, unobseravbles, completeness of the domain)
maximize state diversity
maximize different ways to solve problems (i.e. choose different operators, bindings)
explicit exploration preference

Figure 7: Teaching agent variability

We are currently exploring the impact of teaching agent variability in our learning system. We have implemented a preliminary version of a helpful teacher that increases learning opportunity by increasing state diversity and by trying different operators to solve problems. The operators learned by observation and practice under the helpful teacher have a higher accuracy in terms of the percentage of unnecessary preconditions in the learned operators. Because of the noise in the observations in real world domains, it is especially critical to explore more helpful teacher in order to reduce the number of observations needed to learn the operators. We plan to explore other aspects of helpful teachers, such as learning control knowledge assuming the experts can tell the learning system the top level goals that they are achieving.

Related Work and Conclusions

Learning by observation and practice is a for of learning from the environment (Shen 1994). LIVE (Shen 1994) is a system that learns and discovers from environment. It integrates action, exploration, experimentation, learning, problem solving.

EXPO(Gil 1992) is a learning-by-experimentation module for refining incomplete planning operators. Learning is triggered when plan execution monitoring detects a divergence between internal expectations and external observations. Our system differs in that: 1) The initial knowledge given to the two systems is different. EXPO starts with a set of operators that are missing some preconditions and effects. Our system starts with no knowledge about the preconditions or the effects of the operators. 2) EXPO only copes with incomplete domain knowledge, i.e. over-general preconditions and incomplete effects of the operators, while our system also copes with incorrect domain knowledge, i.e. operators with over-specific preconditions. 3) Operators are learned from general-to-specific in EXPO, while they are learned from specific-to-general in our system.

Other work on learning from observations:??

Generalizing the preconditions and effects of the operators is similar to inductive logic programming (ILP), which constructs quantified definite clause theories from examples and background knowledge. Efficient algorithms such as FOIL (Quinlan 1990), GOLEM (Muggleton & Feng 1990) have been developed within the framework of ILP. Unfortunately, our language for describing operators cannot be represented using first-order logic. Although the preconditions of our operators are a subset of first-order logic, both FOIL and GOLEM learns only a restricted subset of first-order language that is not sufficient to describe the preconditions of our operators.

On the planning side, some planning systems do plan repair (Simmons 1988; Wilkins 1988; Hammond 1986; Kambhampati 1990). However, all these plan repair systems use a correct domain model. In our system, on the other hand, a correct domain model is not available, therefore it is necessary to plan and repair plan using incomplete and incorrect domain knowledge.

In conclusion, we have described a framework for learning planning operators by observation and practice and have demonstrated its effectiveness in several domains. In our framework, operators are learned from specific-to-general through an integration of plan execution as well as planning and plan repair with possibly incorrect operators.

References

Anzai, Y., and Simon, H. A. 1979. The Theory of Learning by Doing. *Psychological Review* 86:124–140.

Carbonell, J.; J.Blythe; O.Etzioni; Gil, Y.; Joseph, R.; Kahn, D.; Knoblock, C.; Minton, S.; Pérez, M. A.; Reily, S.; Veloso, M.; and Wang, X. 1992. PRODIGY 4.0: The Manual and Tutorial. Technical report, School of Computer Science, Carnegie Mellon University.

Carbonell, J. G.; Knoblock, C. A.; and Minton, S. 1990. PRODIGY: An Integrated Architecture for Planning and Learning. In VanLehn, K., ed., *Architectures for Intelligence*. Hillsdale, NJ: Erlbaum.

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence* 2(3,4):189–208.

Fikes, R. E.; Hart, P. E.; and Nilsson, N. J. 1972. Learning and Executing Generalized Robot Plans. *Artificial Intelligence* 3:251–288.

Gil, Y. 1991. A Specification of Manufacturing Processes for Planning. Technical Report CMU-CS-91-179, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Gil, Y. 1992. *Acquiring Domain Knowledge for Planning by Experimentation*. Ph.D. Dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Hammond, K. 1986. *Case-Based Planning: An Integrated Theory of Planning, Learning, and Memory*. Ph.D. Dissertation, Yale University, New Haven, CN.

Kambhampati, S. 1990. A Theory of Plan Modification. In *Proceedings of the Eighth National Conference on Artificial Intelligence*.

Minton, S. 1988. *Learning Search Control Knowledge: An Explanation-Based Approach*. Kluwer Academic Publishers.

Mitchell, T.; Utgoff, P.; and Banerji, R. 1983. Learning by Experimentation: Acquiring and Refining Problem-solving Heuristic. In *Machine Learning, An Artificial Intelligence Approach, Volume I*. Palo Alto, CA: Tioga Press.

Muggleton, S., and Feng, C. 1990. Efficient Induction of Logic Programs. In *Proceedings of the First Conference on Algorithm Learning Theory*.

Quinlan, R. 1990. Learning logical definitions from relations. *Machine Learning* 5:239–266.

Shen, W.-M. 1994. *Autonomous Learning from the Environment*. Computer Science Press, W.H. Freeman and Company.

Simmons, R. 1988. A Theory of Debugging Plans and Interpretations. In *Proceedings of the Sixth National Conference on Artificial Intelligence*.

Veloso, M. 1992. *Learning by Analogical Reasoning in General Problem Solving*. Ph.D. Dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Wang, X. 1994. Learning Planning Operators by Observation and Practice. In *Proceedings of the Second International Conference on AI Planning Systems*.

Wilkins, D. 1988. *Practical Planning: Extending the Classical AI Planning Paradigm*. Morgan Kaufmann. chapter 11.