Appears in *Proceedings of the Third International Workshop on Multi-Strategy Learning (MSL-96)*
pp. 271-279, Harpers Ferry, WV, May 1996

Integrating EBL and ILP to Acquire Control Rules for Planning*

Tara A. Estlin and Raymond J. Mooney
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712
{estlin,mooney}@cs.utexas.edu

Abstract

Most approaches to learning control information in planning systems use *explanation-based learning* to generate control rules. Unfortunately, EBL alone often produces overly complex rules that actually decrease planning efficiency. This paper presents a novel learning approach for control knowledge acquisition that integrates explanation-based learning with techniques from *inductive logic programming*. EBL is used to constrain an inductive search for selection heuristics that help a planner choose between competing plan refinements. SCOPE is one of the few systems to address learning control information in the newer partial-order planners. Specifically, SCOPE learns domain-specific control rules for a version of the UCPOP planning algorithm. The resulting system is shown to produce significant speedup in two different planning domains.

Introduction

Efficient planning often requires domain-specific search heuristics; however, constructing appropriate heuristics for a new domain is a difficult, laborious task. Research in learning and planning attempts to address this important problem by developing methods that automatically acquire search-control knowledge from experience. Past systems have often employed *explanation-based learning* (EBL) to learn search-control knowledge. Unfortunately, standard EBL can frequently produce complex, overly-specific control rules that decrease rather than improve overall planning performance (Minton 1988). By incorporating induction to learn simpler, approximate control rules, we can greatly improve the utility of acquired knowledge (Cohen 1990; Leckie & Zuckerman 1993). In this paper, we describe SCOPE, a system that uses a unique combination of machine learning techniques to acquire effective search-control rules for a partial-order planner.

*This research was supported by the NASA Graduate Student Researchers Program, grant number NGT-51332.

Specifically, SCOPE (Search Control Optimization of Planning through Experience) integrates *explanation-based generalization* (EBG) (Mitchell, Keller, & Kedar-Cabelli 1986; DeJong & Mooney 1986) with techniques from *inductive logic programming* (ILP) (Quinlan 1990; Muggleton 1992; Lavrač & Džeroski 1994) to learn high-utility rules that can generalize well to new planning situations. ILP techniques have often been used in the past for inducing logic programs from a set of examples. SCOPE illustrates that these techniques can also be successfully applied for improving program efficiency.

SCOPE extends previous planning and learning research by acquiring control knowledge for the newer, more efficient *partial-order* planners. Most work in this area has been in the context of linear, state-based planners (Minton 1989; Gratch & DeJong 1992; Leckie & Zuckerman 1993). These planners are usually classified as *total-order* planners, since plans steps are maintained in a strictly ordered list. Recent experimental results, however, support that partial-order planners are more efficient than total-order planners in most domains (Barrett & Weld 1994; Minton *et al.* 1992; Kambhampati & Chen 1993). In a partially-ordered plan, some steps can remain unordered with respect to each other, thereby allowing a planner to avoid premature commitments to an incorrect ordering. Though partial-order planners are considered a more proficient planning strategy, they are not a panacea for efficient planning. Added control knowledge can still dramatically effect the their performance. However, there has been little work on learning control for current partial-order planning systems (Katukam & Kambhampati 1994).

SCOPE learns control rules for a partial-order planner in the form of selection heuristics. These heuristics greatly reduce backtracking by directing a planner to immediately select appropriate plan refinements. SCOPE is implemented in Prolog, which provides a good framework for learning control knowledge. A version of the UCPOP planning algorithm (Penberthy & Weld 1992) was implemented as a Prolog program to provide a testbed for SCOPE. Experimental results are presented on two domains that demonstrate SCOPE can significantly increase partial-order planning efficiency.

The remainder of this paper is organized as follows. In the next section, we describe the UCPOP planner and explain how control rules are implemented. Next, SCOPE's learning algorithm is described, after which experimental results are presented. Finally, we discuss related work and present ideas for future research.

Learning Control For Partial-Order Planning

Search-control information can improve the performance of a planner by guiding it to solutions quickly with minimal search. Past learning systems have usually employed either EBL (Minton 1989; Bhatnagar & Mostow 1994; Kambhampati, Katukam, & Qu 1996) or induction (Leckie & Zuckerman 1993; Langley & Allen 1991) to acquire control rules based on past planning behavior. Our approach differs from these methods by using a novel combination of analytic and inductive methods to acquire control information. In our framework, control-rule learning is viewed as a form of concept learning. SCOPE uses an inductive algorithm to learn definitions of when plan refinements should be applied. EBL focuses the inductive search so good rules are learned quickly without requiring an overly large search space.

The UCPOP Planner

The base planner we chose for experimentation is UCPOP, a partial-order planner described in (Penberthy & Weld 1992). In UCPOP, a partial plan is described as a four-tuple: $\langle \mathcal{S},\mathcal{B},\mathcal{O},\mathcal{L} \rangle$ where $\mathcal{S}$ is a set of actions, $\mathcal{O}$ is a set of ordering constraints, $\mathcal{L}$ is a set of causal links, and $\mathcal{B}$ is a set of codesignation constraints over variables appearing in $\mathcal{S}$. Actions are described by a STRIPS schema containing precondition, add and delete lists. The set of ordering constraints, $\mathcal{O}$, specifies a partial ordering of the actions contained in $\mathcal{S}$. Causal links record dependencies between the effects of one action and the preconditions of another. These links are used to detect *threats*, which occur when a new action interferes with a past decision.

UCPOP begins with a null plan and an agenda containing the top-level goals. The initial and goal states are represented by adding two extra actions to S, A_0 and A_∞. The effects of A_0 correspond to the initial state, and the preconditions of A_∞ correspond to the desired goal state. In each planning cycle, a goal is re-

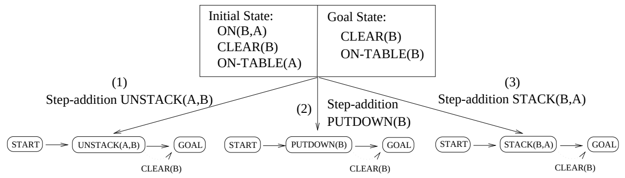

Figure 1: Three competing refinement candidates for achieving the goal *Clear(B)*.

moved from the agenda and an existing or new action is chosen to assert the goal. After an action is selected, the necessary ordering, casual link and codesignation constraints are added to $\mathcal{O}$, $\mathcal{L}$, and $\mathcal{B}$. If a *new* action was selected, the action's preconditions are added to the agenda. UCPOP then checks for threats and resolves any found by adding an additional ordering constraint. UCPOP is called recursively until the agenda is empty. On termination, UCPOP uses the constraints found in $\mathcal{O}$ to determine a total ordering of the actions in $\mathcal{S}$, and returns this as the final solution.

Control Rule Format

SCOPE learns search-control rules for planning decisions that might lead to a failing search path (i.e. might be backtracked upon). Figure 1 illustrates an example from the blocksworld domain where control knowledge could be useful. Here, there are three possible refinement candidates for adding a new action to achieve the goal *Clear(B)*. For each set of refinement candidates, SCOPE learns control rules in the form of selection rules that define when each refinement should be applied. A single selection rule consists of a conjunction of conditions that must all evaluate to true for the refinement to be used. If at least one condition fails, that refinement candidate will be rejected, and the next candidate evaluated. For example, shown next is a selection rule for the first candidate (from Figure 1) which contains several control conditions.

Select operator `Unstack(?X,?Y)` **to establish**
`goal(`*Clear(?Y)*,s_1`)`[1]
If `exists-operator(`s_2`)` $\wedge$
`establishes(`s_2, *On(?X,?Y)*`)` $\wedge$
`possibly-before(`s_2,s_1`)`.

This rule states that the operator `Unstack(?X,?Y)` should be selected to add *Clear(?Y)* only when there is an existing action s_2 that adds *On(?X,?Y)* and s_2 can be ordered before the action s_1, which requires *Clear(?Y)*.

Learned control information is incorporated into the

[1]Goals are represented in our planner by the *(Goal,Action)* structure where *Action* is the plan step that requires *Goal*.

planner so that attempts to select an inappropriate refinement will immediately fail. Control rules can consist of a conjunction of conditions (as shown above) or a disjunction of several conjunctions. SCOPE can also make selection rules deterministic or nondeterministic depending on the accuracy of the learned rule.

The Prolog programming language provides an excellent framework for learning control rules. Search algorithms can be implemented in Prolog in such a way that allows control information to be easily incorporated in the form of clause-selection rules (Cohen 1990). These rules help avoid inappropriate clause applications, thereby reducing backtracking. A version of the UCPOP partial-order planning algorithm has been implemented as a Prolog program.[2] Planning decision points are represented in this program as clause-selection problems (i.e. each refinement candidate is formulated as a separate clause). SCOPE is then used to learn refinement-selection rules which are incorporated into the original planning program in the form of clause-selection heuristics.

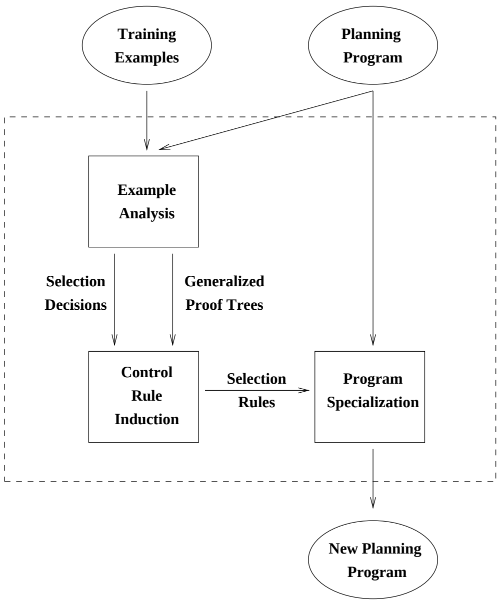

Figure 2: SCOPE's High-Level Architecture

The SCOPE Learning System

SCOPE is based on the DOLPHIN speedup learning system (Zelle & Mooney 1993), which optimizes logic programs by learning clause-selection rules. DOLPHIN has been shown successful at improving program performance in several different domains, including planning domains which employed a simple state-based planner. DOLPHIN, however, has little success improving the performance of a partial-order planner due to the higher complexity of the planning search space. In particular, DOLPHIN's simple control rule format lacks the expressibility necessary to describe complicated planning situations. DOLPHIN also has difficulty successfully generalizing explanations produced by a partial-order planner. SCOPE has greatly expanded upon DOLPHIN's algorithm to be effective on more complex planning systems.

The input to SCOPE is a planning program and a set of training examples. SCOPE uses the examples to induce a set of control heuristics which are then incorporated into the original planner. Figure 2 shows the three main phases of SCOPE's algorithm, which are explained in the next few sections. A more detailed description can be found in (Estlin 1996).

Example Analysis

In the example analysis phase, two main outputs are produced: a set of selection-decision examples and a set of *generalized* proof trees. Selection-decision examples are used to record successful and unsuccessful applications of a plan refinement. Generalized proof trees provide a background context that explains the success of all correct planning decisions. These two pieces of information are used in the next phase to build control rules.

Selection-decision examples are produced using the following procedure. First, training examples are solved using the existing planner. A trace of the planning decision process used to solve each example is stored in a proof tree, where the root represents the top-level planning goal and nodes correspond to different planning procedure calls. These proofs are then used to extract examples of correct and incorrect refinement-selection decisions. Specifically, a "selection decision" is a planning subgoal that was solved correctly by applying some plan refinement, such as adding a new action. As an example, consider the planning problem that was introduced in Figure 1. The planning subgoal represented by this figure is shown below:[3]

For $\mathcal{S}$ = (0:Start,G:Goal),
$\mathcal{O}$ = $(0 \prec \text{G})$,
$\mathcal{L}$ = $\emptyset$,
`agenda` = (*Clear(B)*,G),(*On-Table(B)*,G),
Select operator ?OP **to establish** `goal`(*Clear(B)*,G)

Selection decisions such as this one are collected for all

[2] Our Prolog planner performs comparably to the standard LISP implementation of UCPOP on the sample problem sets used to test the learning algorithm (discussed in the Experimental Evaluation section).

[3] Binding constraints in our system are maintained through Prolog, therefore, the set of binding constraints, $\mathcal{B}$, is not explicitly represented in planning subgoals.

```
UCPOP(Actions0,OrderCons0,Links0,Goals0,Solution)
        ↓
SELECT-GOAL(Goals0,goal(clear(X),0))
        ↓
MEMBER(goal(clear(X),0),Goals0)
        ↓
SELECT-OP(goal(clear(X),0),),Actions0,OrderCons0,Links0,Goals0,action(ANum1,putdown(X)))
        ↓
ADD-ACTION(action(ANum1,putdown(X)),Actions0,Actions1)
        ↓
```

Figure 3: Top Portion of a Generalized Proof Tree

competing plan refinements. Refinements are considered "competing" if they can be applied in identical planning decisions, such as the three refinement candidates shown in Figure 1. A correct decision for a particular refinement candidate is an application of that refinement found on a solution path. An incorrect decision is a refinement application that was tried and subsequently backtracked over. The subgoal shown above would be identified as a positive selection decision for candidate 2 (adding `Putdown(A)`), and would also be classified as a negative selection decision for candidates 1 and 3 (adding `Unstack(A,B)` or `Stack(B,A)`). Any given training problem may produce numerous positive and negative examples of refinement selection decisions. Selection decision examples are used later in induction to represent positive and negative examples of when to apply particular planning refinements.

The second output of the example analysis phase is a set of generalized proof trees. Standard EBG techniques (Mitchell, Keller, & Kedar-Cabelli 1986; DeJong & Mooney 1986) are used to generalize each training example proof tree. The goal of this generalization is to remove proof elements that are dependent on the specific example facts while maintaining the overall proof structure. Generalized proof information is used later to explain new planning situations. The top portion of a generalized proof tree is shown in Figure 3. This proof was extracted from the solution trace of the problem introduced in Figure 1. The top-level goal in this proof is a call to the planner that included the initial list of plan actions, operators, causal-links, and agenda as input arguments. The last argument corresponds to the output plan solution. All subsequent planning procedure calls are included in the proof structure. The generalized proof of an example provides a context which "explains" the success of correct decisions.

Control Rule Induction

The goal of the induction phase is to produce an operational definition of when it is useful to apply a refinement candidate. Given a candidate, C, we desire a definition of the concept "subgoals for which C is useful". In the blocksworld domain, such a definition is learned for each of the candidates shown in Figure 1. In this context, control rule learning can be viewed as relational concept learning. A number of systems (Quinlan 1990; Muggleton 1992; Zelle & Mooney 1994) have been designed to tackle this type of learning problem. SCOPE employs a version of Quinlan's FOIL algorithm to learn control rules through induction.

The choice of a FOIL-like framework is motivated by a number of factors. First, the basic FOIL algorithm is relatively easy to implement and has proven efficient in a number of domains. Second, FOIL has a "most general" bias which tends to produce simple definitions. Such a bias is important for learning rules with a low match cost, which helps avoid the utility problem. Third, it is relatively easy to bias FOIL with prior knowledge (Pazzani & Kibler 1992). In our case, we can utilize the information contained in the generalized proof trees of planning solution traces.

FOIL Algorithm FOIL attempts to learn a concept definition in terms of a given set of background predicates. This definition is composed of a set of Horn clauses that cover all of the positive examples of a concept, and none of the negative examples. The selection-decision examples collected in the example analysis phase provide the sets of positive and negative examples for each refinement candidate.

```
Initialization
    Definition := null
    Remaining := all positive examples
While Remaining is not empty
        Find a clause, C, that covers some examples in
            Remaining, but no negative examples.
        Remove examples covered by C from Remaining.
        Add C to Definition.
```

Figure 4: Basic FOIL Covering Algorithm

FOIL may be viewed as a simple covering algorithm which has the basic form shown in Figure 4. The "find a clause" step is implemented by a general-to-specific hill-climbing search. FOIL adds antecedents to the developing clause one at a time. At each step FOIL evaluates all possible literals that might be added and selects the one which maximizes an information-based gain heuristic.

The generation of candidate literals to add to a developing clause normally consists of trying each background predicate with all possible combinations of variables currently in the clause and any new predicate variables. SCOPE uses an intensional version of FOIL where background predicates can be defined as Prolog predicates instead of requiring an extensional representation (Mooney & Califf 1995). Any predicates that can be used as rule antecedents must be introduced as background knowledge.

One major drawback to FOIL (and other similar inductive algorithms) is that the hill-climbing search for a good antecedent can easily explode, especially when there are numerous background predicates with large numbers of arguments. When selecting each new clause antecedent, FOIL tries *all* possible variable combinations for *all* predicates before making its choice. This search grows *exponentially* as the number of predicate arguments increases. SCOPE circumvents this search problem by using the generalized proofs of training examples. By examining the proof trees, SCOPE identifies a small set of potential literals that could be added as antecedents to the current clause definition. Literals are added in a way that utilizes variable connections already established in the proof tree. This approach nicely focuses the FOIL search by only considering literals (and variable combinations) that were found useful in solving the training examples.

Building Control Rules from Proof Trees The generalized proofs of training examples can be seen as giving the context for the appropriate applications of refinements within a proof. Some nodes of a generalized proof tree contain calls to "operational" predicates. These are usually low-level predicates that have been classified as "easy to evaluate" within the problem domain, and thus can be used to build efficient concept definitions. The operational nodes of a proof represent all of the primitive conditions that had to be satisfied for the proof to succeed. SCOPE employs induction in an attempt to identify a small set of these simple tests that will provide necessary guidance in determining whether the application of a refinement is likely to lead to a solution. Since test conditions that verify a planning decision are sometimes not executed until much later, it is important to consider an entire example proof instead of just the surrounding context of a particular decision. For instance, the planner might not verify that choosing the action `Putdown(a)` to establish the goal *Clear(a)* is correct until much later in the planning process when it checks to see if some other action has asserted *Holding(a)*.

SCOPE employs the same general covering algorithm as FOIL but modifies the clause construction step. Clauses are successively specialized by considering how their target refinements were used in solving training examples. Suppose we are learning a definition for when each of the refinement candidates in Figure 1 should be applied. The actual program predicate representing this type of refinement is select-op. This predicate is defined with several arguments including the unachieved goal and an output argument for the selected operator. (Plan state information is also automatically included as arguments to any refinement predicate.) The full predicate head of this refinement is shown below.

```
select-op(Goal,Steps,OrderCons,Links,Agenda,ReturnOp)
```

For each refinement candidate, SCOPE begins with the most general definition possible. For instance, the most general definition covering candidate 1's selection examples is the following; call this clause C.

```
select-op(Goal,Steps,OrderCons,Links,Agenda,unstack(A,B)) :-
    TRUE
```

This overly general definition covers *all* positive examples and *all* negative examples of when to apply candidate 1, since it will always evaluate to true. C can be specialized by adding antecedents to its body. This is done by unifying C's head with a (generalized) proof subgoal that was solved by applying candidate 1 and then adding an operational literal from the same proof tree which shares some variables with the subgoal. For example, one possible specialization of the above clause is shown below.

```
select-op((clear(B),S1),Steps0,OrderCons,Links,Agenda,unstack(A,B)) :-
    establishes(on(B,A),Steps1,S2).
```

Here, a proof tree literal has been added which checks if there an existing plan step that establishes the goal *On(B,A)*. Variables in a newly added antecedent can be connected with the existing rule head in several ways. First, by unifying a rule head with a generalized subgoal, variables in the rule head become unified with variables existing in a proof tree. All operational literals in that proof that share variables with the generalized subgoal are tested as possible antecedents. This method initially establishes many relevant variable connections between a rule head and its antecedents.

A second way variable connections can be established is through the standard FOIL technique of unifying variables of the same type. When SCOPE tests a literal for use in a control rule, the literal may contain input parameters that are not bound by the rule head or other existing literals in the rule. If such parameters exist, SCOPE attempts to unify these parameters with terms of the same type that are already present in the rule. For example, the rule shown above has an antecedent with an unbound input, `Steps1`, which does not match any other variables in the clause. SCOPE will modify the rule, as shown below, so that the `Steps1` is unified with a term of the same type from the rule head, `Steps0`.

```
select-op((clear(B),S1),Steps0,OrderCons,Links,Agenda,unstack(A,B)) :-
    establishes(on(B,A),Steps0,S2).
```

SCOPE considers all such specializations of a rule and selects the one which maximizes FOIL's information-gain heuristic.

SCOPE also considers several other types of control rule antecedents during induction. Besides pulling literals directly from generalized proof trees, SCOPE can use negated proof literals, determinate literals (Muggleton 1992), variable codesignation constraints, and relational clichés (Silverstein & Pazzani 1991). Incorporating different antecedent types helps SCOPE learn expressive control rules that can describe partial-order planning situations.

Program Specialization Phase

Once refinement selection rules have been learned, they are passed to the program specialization phase which adds this control information into the original planner. The basic approach is to guard each refinement candidate with the selection information. This forces a refinement application to fail quickly on subgoals to which the refinement should not be applied.

A decision is also made as to whether the control information has made the planner deterministic. If a refinement rule (or rule disjunct) covers no incorrect selection decisions in the induction phase, then it is assumed that the rule is fully accurate and no other refinement candidates will need to be considered. If a refinement rule could not exclude all incorrect decisions in the previous phase, then the planner is still allowed to backtrack over the selection of that refinement. This type of rule can still substantially improve planning efficiency by preventing many inappropriate applications of that refinement.

Figure 5 shows several learned selection rules for the first two refinement candidates (from Figure 1). The first rule allows `Unstack(A,B)` to be applied only when `A` is found to be on `B` initially, and `Stack(B,C)` should not be selected instead. The second and third rule allow `Putdown(A)` to be applied only when `A` should be placed on the table and not stacked on another block.

```
select-op((clear(B),S1),Steps,OrderCons,Links,Agenda,unstack(A,B)) :-
    find-init-state(Steps,Init),
    member(on(A,B),Init),
    not(member((on(B,C),S1),Agenda),member(on-table(B),Init)).

select-op((clear(A),G),Steps,OrderCons,Links,Agenda,putdown(A)) :-
    not(member((on(A,B),G),Agenda)).

select-op((clear(A),S1),Steps,OrderCons,Links,Agenda,putdown(A)) :-
    member((on-table(A),S2),Agenda),
    not(establishes(on-table(A),S3)).
```

Figure 5: Learned control rules for two refinement candidates

Experimental Evaluation

The blocksworld and logistics transportation domains were used to test the SCOPE learning system. In the logistics domain (Veloso 1992), packages must be delivered to different locations in several cities. Packages are transported between cities by airplane and within a city by truck. In both domains, a test set of 100 independently generated problems was used to evaluate performance. SCOPE was trained on separate example sets of increasing size. Ten trials were run for each training set size, after which results were averaged. Training and test problems were produced for both domains by generating random initial and final states. In blocksworld, problems contained two to six blocks and one to four goals. Logistics problems contained up to two packages and three cities, and one or two goals. No time limit was imposed on planning in either domain, but a uniform depth bound on the plan length was used during testing that allowed for all problems to be solved. All tests were performed on a Sun SPARCstation 5.

For each trial, SCOPE learned control rules from the given training set and produced a modified planner. Since SCOPE only specializes decisions in the original planner, the new planning program is guaranteed to be sound with respect to the original one. Unfortunately, the new planner is not guaranteed to be complete. Some control rules could be too specialized and thus the new planner may not solve all problems solvable by the original planner. In order to guarantee the completeness of the final planner, a strategy used by Cohen (1990) is adopted. If the final planner fails to find a solution to a test problem, the initial planning program is used to solve the problem. When this situation occurs in testing, both the failure time for the new planner and the solution time for the original planner are included in the total solution time for that problem. In the results presented in the next section, the new planner generated by SCOPE was typically able to solve 95% of the test examples.

Figures 6 and 7 present the experimental results. The times shown represent the number of seconds required to solve the problems in the test sets after SCOPE was trained on a given number of examples. In both domains, the SCOPE consistently produced a more efficient planner and significantly decreased solution times on the test sets. In the blocksworld, SCOPE produced modified planning programs that were an average of 11.3 times faster than the original planner. For the logistics domain, SCOPE produced programs that were an average of 5.3 times faster. These results

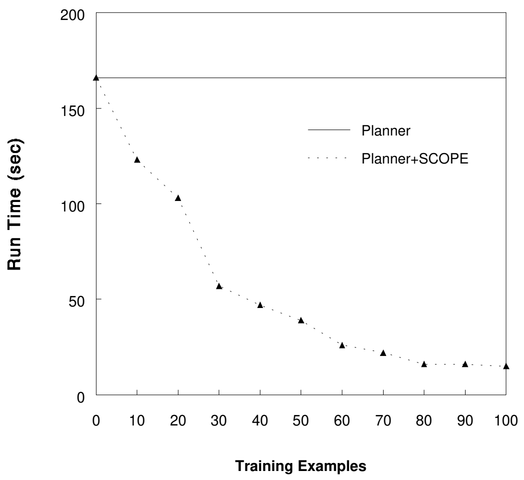

Figure 6: Performance in Blocksworld

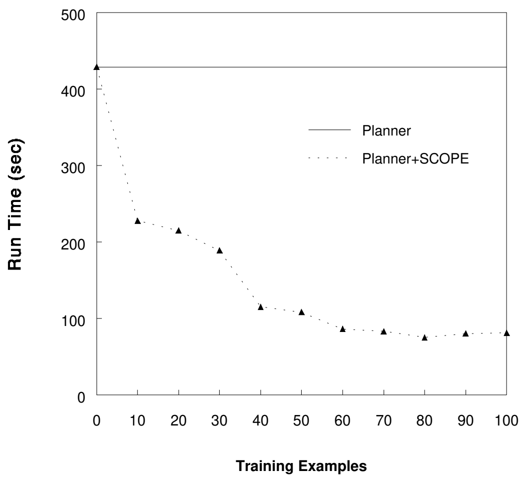

Figure 7: Performance in Logistics

indicate that SCOPE can significantly improve the performance of a partial-order planner.

Related Work

A closely related system to SCOPE is UCPOP+EBL (Kambhampati, Katukam, & Qu 1996), which also learns search control rules for UCPOP, but uses a purely explanation-based approach. Specifically, UCPOP+EBL employs the standard EBL techniques of regression, explanation propagation and rule generation to acquire search-control rules. Rules are learned only in response to past planning failures. This system has been shown to improve planning performance in several domains, including the blocksworld. To compare the two systems, we replicated an experiment used by Kambhampati, Katukam, & Qu (1996). Problems were randomly generated from a version of the blocksworld domain that contained between three to six blocks and three to four goals.[4] SCOPE was trained on a set of 100 problems. The test set also contained 100 problems and a CPU time limit of 120 seconds was imposed during testing. Data was collected on planner speedup and on the number of test problems that could be solved under the time limit. The results are shown in the following table.

System	*Orig Time*	*Final Time*	*Speedup*	*Orig %Sol*	*Final %Sol*
UCPOP+EBL	7872	5350	1.47X	51%	69%
SCOPE	5312	1857	2.86X	59%	94%

[4] In order to replicate the experiments of Kambhampati, Katukam, & Qu (1996), the blocksworld domain theory used for these tests slightly differed from the one used for the experiments presented previously. Both domains employed similar predicates however the previous domain definition consists of four operators while the domain used here has only two.

Both systems were able to increase the number of test problems solved, however, SCOPE had a much higher success rate. Overall, SCOPE achieved a better speedup ratio, producing a more efficient planner. By combining EBL with induction, SCOPE was able to learn better planning control heuristics than EBL did alone. These results are particularly significant since UCPOP+EBL uses additional domain axioms which were not provided to SCOPE.

Most other related learning systems have been evaluated on different planning algorithms, thus direct system comparisons are difficult. The HAMLET (Borrajo & Veloso 1994a) system learns control knowledge for the nonlinear planner underlying PRODIGY4.0. HAMLET acquires rules by explaining past planning decisions and then incrementally refining them. It is difficult to directly compare HAMLET and SCOPE for several reasons. First, HAMLET is directly integrated with the PRODIGY4.0 system, which does not use a partial-order planner. PRODIGY4.0 also employs some initial built-in control knowledge not given to our planner. HAMLET has successfully improved planner performance in the blocksworld and logistics planning domains. When making a rough comparison to the results reported in Borrajo & Veloso (1994b) , SCOPE achieves a greater speedup factor in blocksworld (11.3 vs 1.8) and in the logistics domain (5.3 vs 1.8).

Future Work

There are several issues we hope to address in future research. First, SCOPE should be tested on more complex domains which contain conditional effects, universal quantification, and other more-expressive planning constructs. We also plan to examine ways of using SCOPE to improve plan quality as well as planner efficiency. Borrajo (1994b) and Perez (1994) have used learned control information to guide a planner towards better solutions. SCOPE could be modified to collect positive control examples only from high-quality solutions so that control rules are focused on quality issues as well as speedup. Lastly, we want to incorporate a method that directly evaluates control-rule utility. Replacing FOIL's information-gain metric for picking literals with a metric that more directly measures rule utility could improve ultimate planning performance.

Conclusion

SCOPE provides a new mechanism for learning search control information in planning systems. This system employs a novel learning technique that contains both explanation-based and inductive learning components. Simple, high-utility rules are learned by inducing concept definitions of when to apply plan refinements. Explanation-based generalization aids the inductive search by focusing it towards the best pieces of background information. Unlike most approaches which are limited to total-order planners, SCOPE can learn control rules for the newer, more effective partial-order planners. In both the blocksworld and logistics domains, SCOPE significantly improved planner performance; SCOPE also outperformed a competing method based only on EBL.

References

Barrett, A., and Weld, D. 1994. Partial order planning: Evaluating possible efficiency gains. *Artificial Intelligence* 67:71–112.

Bhatnagar, N., and Mostow, J. 1994. On-line learning from search failure. *Machine Learning* 15:69–117.

Borrajo, D., and Veloso, M. 1994a. Incremental learning of control knowledge for nonlinear problem solving. In *Proceedings of the European Conference on Machine Learning, ECML-94*, 64–82.

Borrajo, D., and Veloso, M. 1994b. Incremental learning of control knowledge for improvement of planning efficieny and plan quality. In *AAAI-94 Fall Symposium on Planning and Learning*, 5–9.

Cohen, W. W. 1990. Learning approximate control rules of high utility. In *Proceedings of the Seventh International Conference on Machine Learning*, 268–276.

DeJong, G. F., and Mooney, R. J. 1986. Explanation-based learning: An alternative view. *Machine Learning* 1(2):145–176. Reprinted in *Readings in Machine Learning*, J. W. Shavlik and T. G. Dietterich (eds.), Morgan Kaufman, San Mateo, CA, 1990.

Estlin, T. A. 1996. Integrating explanation-based and inductive learning techniques to acquire search-control for planning. Technical report, Department of Computer Sciences, University of Texas, Austin, TX. Forthcoming. URL: http://net.cs.utexas.edu/ml/

Gratch, J., and DeJong, G. 1992. COMPOSER: A probabilistic solution to the utility problem in speed-up learning. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 235–240.

Kambhampati, S., and Chen, J. 1993. Relative utility of EBG based plan reuse in partial ordering vs. total ordering. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 514–519.

Kambhampati, S.; Katukam, S.; and Qu, Y. 1996. Failure driven search control for partial order planners: An explanation based approach. *Artificial Intelligence*. Forthcoming.

Katukam, S., and Kambhampati, S. 1994. Learning explanation-based search control for partial order planning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 582–587.

Langley, P., and Allen, J. 1991. The acquisition of human planning expertise. In *Proceedings of the Eighth International Workshop on Machine Learning*, 80–84.

Lavrač, N., and Džeroski, S., eds. 1994. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood.

Leckie, C., and Zuckerman, I. 1993. An inductive approach to learning search control rules for planning. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, 1100–1105.

Minton, S.; Drummond, M.; Bresina, J. L.; and Phillips, A. B. 1992. Total order vs. partial order planning: Factors influencing performance. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, 83–92.

Minton, S. 1988. Quantitative results concerning the utility of explanation-based learning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, 564–569.

Minton, S. 1989. Explanation-based learning: A problem solving perspective. *Artificial Intelligence* 40:63–118.

Mitchell, T. M.; Keller, R. M.; and Kedar-Cabelli, S. T. 1986. Explanation-based generalization: A unifying view. *Machine Learning* 1(1):47–80.

Mooney, R. J., and Califf, M. E. 1995. Induction of first-order decision lists: Results on learning the past tense of English verbs. *Journal of Artificial Intelligence Research* 3:1–24.

Muggleton, S. H., ed. 1992. *Inductive Logic Programming*. New York, NY: Academic Press.

Pazzani, M., and Kibler, D. 1992. The utility of background knowledge in inductive learning. *Machine Learning* 9:57–94.

Penberthy, J., and Weld, D. S. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, 113–114.

Pérez, M. A., and Carbonell, J. 1994. Control knowledge to improve the plan quality. In *Proceedings of the Second International Conference of AI Planning Systems*.

Quinlan, J. 1990. Learning logical definitions from relations. *Machine Learning* 5(3):239–266.

Silverstein, G., and Pazzani, M. J. 1991. Relational clichés: Constraining constructive induction during relational learning. In *Proceedings of the Eighth International Workshop on Machine Learning*, 203–207.

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

Zelle, J. M., and Mooney, R. J. 1993. Combining FOIL and EBG to speed-up logic programs. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, 1106–1111.

Zelle, J. M., and Mooney, R. J. 1994. Combining top-down and bottom-up methods in inductive logic programming. In *Proceedings of the Eleventh International Conference on Machine Learning*, 343–351.