Appears in *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97),* pp.1227-1232, Nagoya, Japan, August 1997

Learning to Improve both Efficiency and Quality of 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 research in learning for planning has concentrated on efficiency gains. Another important goal is improving the quality of final plans. Learning to improve plan quality has been examined by a few researchers, however, little research has been done learning to improve both efficiency and quality. This paper explores this problem by using the SCOPE learning system to acquire control knowledge that improves on both of these metrics. Since SCOPE uses a very flexible training approach, we can easily focus its learning algorithm to prefer search paths that are better for particular evaluation metrics. Experimental results show that SCOPE can significantly improve both the quality of final plans and overall planning efficiency.

1 Introduction

A considerable amount of planning and learning research has been devoted to improving *planning efficiency*, also known as "speedup learning" [Minton, 1989; Leckie and Zuckerman, 1993; Estlin and Mooney, 1996; Kambhampati *et al.*, 1996]. These systems construct domain-specific control rules that enable a planner to find solutions more quickly. Another aim of planning and learning research, which has received much less attention, is to improve the *quality of plans* produced by a planner [Pérez, 1996; Iwamoto, 1994]. In this type of learning, control rules guide the planner towards *better* solutions. Generating high-quality plans is an essential feature for many real-world planning systems, as is generating these plans efficiently [Chien *et al.*, 1996]. Improving plan quality and planner efficiency can often be accomplished by using similar learning methods. However, little research has been done in acquiring control knowledge to improve both these metrics.

To investigate this issue, we employ the SCOPE[1] learning system, which uses a combination of machine learning techniques to acquire control rules for a partial-order planner. SCOPE has previously been shown to significantly improve efficiency of a version of the well-known UCPOP planner [Estlin and Mooney, 1996]. SCOPE employs a flexible learning algorithm that can be trained to focus on different evaluation metrics. The primary focus of previous experiments was to avoid backtracking, and thus, improve planning efficiency. However, by using a particular training method, SCOPE can be easily modified to learn rules that guide the planner towards only "high-quality" solutions.

The remainder of this paper is organized as follows. Section 2 further introduces the issue of plan quality and why it is important. Section 3 describes how SCOPE is used to construct control rules, and in Section 4 we discuss how SCOPE was used to learn rules for improving plan quality. Section 5 presents experimental results that show SCOPE can improve both the quality of plans and planner efficiency. Finally, Section 6 discusses related work, Section 7 presents ideas for future research and Section 8 presents our conclusions.

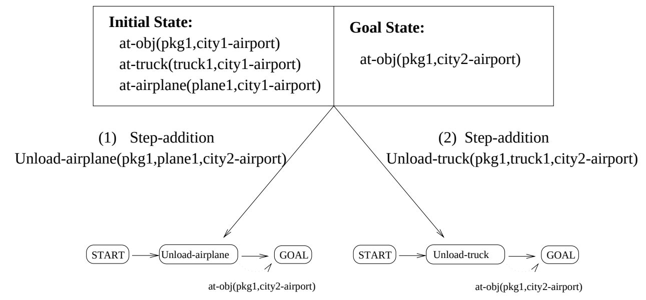
2 Plan Quality

In many real-world planning systems the quality of a plan may be just as important (if not more) than the time it takes to generate the plan. For instance, it may be vital in a manufacturing domain for a planner to produce plans with low resource consumption, or with the least number of possible steps. There are a variety of notions about what makes a good plan. Some of the more common quality metrics are listed below:

- The length of the plan (or the total number of steps)
- The execution time of the plan
- The resource consumption required
- The robustness of the plan

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

[1]**S**earch **C**ontrol **O**ptimization of **P**lanning through **E**xperience



Initial State:
at-obj(pkg1,city1-airport)
at-truck(truck1,city1-airport)
at-airplane(plane1,city1-airport)
Goal State:
at-obj(pkg1,city2-airport)
(1) Step-addition
Unload-airplane(pkg1,plane1,city2-airport)
(2) Step-addition
Unload-truck(pkg1,truck1,city2-airport)
START
Unload-airplane
GOAL
at-obj(pkg1,city2-airport)
START
Unload-truck
GOAL
at-obj(pkg1,city2-airport)

Figure 1: Two refinement candidates for achieving the goal `at-obj(pkg1,airport2)`.

Depending on the domain, different quality metrics will have varying importance. In this paper, we focus on improving the first quality metric, minimizing the number of plan steps.

3 The Scope Learning System

Scope was designed to learn search-control rules for planning decisions that might lead to failure (i.e. might be backtracked upon). Figure 1 illustrates an example from the logistics transportation domain [Veloso, 1992] where control knowledge could be useful. Here, there are two possible refinement candidates for adding a new action to achieve the goal `at-obj(pkg1,city2-airport)`. Only the first will actually lead to a solution, since in this domain, only planes can be used to transport packages between cities. For each set of refinement candidates, Scope learns control rules in the form of selection rules that define when each refinement should be applied. For example, shown next is a selection rule for the first candidate (from Figure 1) which contains several control conditions.

```
Select operator unload-airplane(?X,?Y,?Z)
  to establish at-obj(?X,?Z))
  If member(at-obj(?X,?W),Init-State) ∧
     not(member(same-city(?Z,?W),Init-State)).
```

This rule states that `unload-airplane(?X,?Y,?Z)` should be selected to add `at-obj(?X,?Z)` only when it is initially true that object `?X` starts at a location which is in a different city than `?Z`. Learned control information is incorporated into the planner so that attempts to select an inappropriate refinement will immediately fail.

Scope is implemented in Prolog, which 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]. For its base planner, Scope uses a version of the UCPOP partial-order planning algorithm which has been reimplemented in Prolog.[2] Planning decision points are represented in the planner as clause-selection problems (i.e. each refinement candidate is formulated as a separate clause).

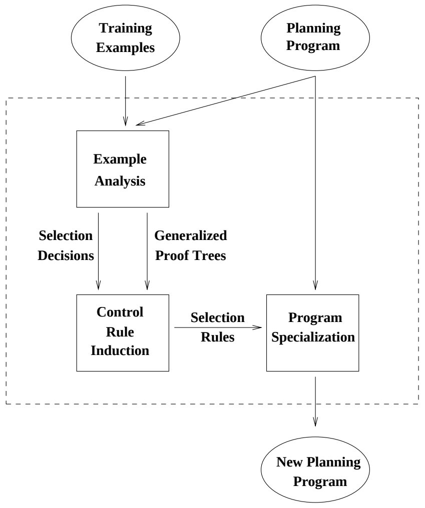


Figure 2: Scope's High-Level Architecture

By analyzing a set of training examples, Scope learns refinement-selection rules which are incorporated into the original planner in the form of clause-selection heuristics. As shown in Figure 2, Scope's algorithm has three main phases which are are briefly presented in the next few sections. A more detailed description can be found in [Estlin, 1996].

3.1 Example Analysis

In the example analysis phase, training examples are solved using the original planner and two main outputs are produced: a set of selection-decision examples and a set of *generalized* proof trees. A "selection decision" is a planning subgoal that was solved by applying a particular plan refinement, such as adding a new action. Selection-decisions record successful and unsuccessful applications of plan refinements. To collect these decisions, a trace of the planning decision process used to solve each training example is stored in a proof tree. Then, Scope uses the proof trees to extract examples of correct and incorrect refinement-selection decisions.

The second output of this phase is a set of generalized proof trees. Standard explanation-based generalization (EBG) [Mitchell *et al.*, 1986; DeJong and Mooney, 1986] techniques 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.

[2]The main difference between our planner and UCPOP is that UCPOP normally employs a best-first search strategy while our Prolog planner operates using a depth-first backtracking search. As shown in Section 5, our Prolog planner actually performs better than the standard Lisp implementation of UCPOP on the sample problem sets used to test the learning algorithm.

```
find-new-op((at-obj(X,Loc),Aid),Steps,Agenda,unload-airplane(X,P,Loc)) :-
      find-init-state(Steps,Init),
      member(at-obj(X,Loc2), Init),
      not(member(same-city(Loc,Loc2),Init)),
      member(airport(Loc), Init).

find-exist-op((at-truck(T,Loc),Aid),Steps,Agenda,init-state)) :-
      find-init-state(Steps,Init),
      member-pred(at-truck(T,Loc),Init),
      not(member((Aid2,drive-truck(T,Loc,Loc2)),Steps),Aid≠Aid2).
```

Figure 3: Learned control rules for the logistics domain

Generalized proofs provide a background context that explains the success of all correct planning decisions. Information from these trees is used in the next phase to construct control rules.

3.2 Control Rule Induction

The goal of the induction phase is to produce an operational definition of when it is useful to apply a planning refinement. SCOPE employs a version of the FOIL algorithm [Quinlan, 1990] to learn control rules through induction. FOIL attempts to learn a control definition that is composed of a set of Horn clauses. This definition covers all of the positive examples of when to apply a refinement, and none of the negatives. The selection-decision examples collected in the example analysis phase provide the sets of positive and negative examples for each refinement.

Individual clause construction is done by using a general-to-specific hill-climbing search. FOIL adds antecedents to the developing clause one at a time. At each step FOIL evaluates all literals that might be added and selects the one which maximizes an information-based gain heuristic. One drawback to FOIL 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.[3] SCOPE circumvents this search problem by utilizing 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. SCOPE also considers several other types of control rule antecedents during induction. These include negated proof tree literals, determinate literals [Muggleton, 1992], variable codesignation constraints, and relational clichés [Silverstein and Pazzani, 1991].

3.3 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 planning refinement with the selection information. This forces a refinement application to fail quickly on planning subgoals to which the refinement should not be applied. Figure 3 shows two learned rules for the logistics transportation domain. The first rule selects the new action `unload-airplane(X,P,Loc)` to achieve the goal `at-obj(X,Loc)` when `X` is found to be initially located in a different city than the goal location and when the goal location is an airport. The second rule uses the initial state to achieve the goal `at-truck(T,Loc)` if there does not exist another action in the plan `drive-truck(T,Loc,Loc2)` which moves the truck to a new location.

4 Focusing SCOPE on Plan Quality

By learning rules which avoid search paths that lead to backtracking, SCOPE has been shown to significantly improving the efficiency of a planner [Estlin and Mooney, 1996]. However, by modifying the method used to collect training data, it can easily improve other planning metrics as well.

In order to improve plan quality, we trained SCOPE on only high-quality solutions. To improve plan lengths, SCOPE was given the shortest solution plans for all training problems. This causes SCOPE to collect positive and negative examples of when to apply a plan refinement based on finding the optimal plan. Thus, it learns rules that not only avoid dead-end paths, but also that avoid paths that lead to sub-optimal solutions.

In order to train SCOPE on high-quality solutions, the search method of depth-first iterative deepening (DFID) [Korf, 1985] was employed to solve the training problems. This method ensured that the shortest possible solutions were always returned. To improve upon other quality metrics, different training methods may have to be employed that return optimal (or near-optimal) solutions based on other evaluation functions.

5 Evaluation

5.1 Experimental Setup

The logistics transportation domain [Veloso, 1992] was used to evaluate SCOPE's ability to improve both quality and efficiency. In this domain, packages must be delivered to different locations in several cities. Trucks are used to transport packages within a city, and planes are used to transport packages between different cities. Training and test problems were produced by generating random initial and final states. Problems contained one and two packages, two trucks and two planes, which were distributed among two cities.

As explained above, depth-first iterative deepening was used to solve the training problems. SCOPE was trained on separate example sets of increasing size. Five

[3]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.

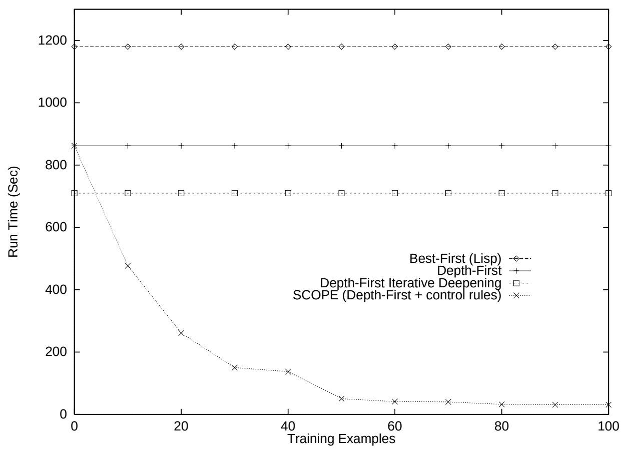


Figure 4: **Efficiency Performance**

trials were run for each training set size, after which results were averaged.

For each trial, a test set of 100 independently generated problems was used to evaluate performance. No time limit was used during testing, but a depth bound was given that ensured all test problems could be solved. For comparison purposes we used several different search methods to solve the test examples. These include depth-first search, depth-first search + learned control information, depth-first iterative deepening search, and best-first search. The best-first search tests were done using the standard Lisp implementation of UCPOP [Barrett *et al.*, 1995]. To utilize all of these search methods, only test problems with solutions under a certain length were used. This ensured all problems could be solved by all methods in a reasonable amount of time.

5.2 Experimental Results

Figure 4 shows improvement in planning efficiency. 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. The best performance occurred when the planner utilized the learned control information. In these tests, SCOPE was able to produce a new planner that was an average of 28 times faster than the original depth-first planner. Depth-first iterative deepening performs better than depth-first but not nearly as well as SCOPE. Best-first search, on the other hand, performed worse than depth-first.[4]

Figure 5 represents how SCOPE improved final plan quality. The lengths shown in the graph represent the average solution lengths returned for the test problems. The depth-first iterative deepening line shows the average length of all optimal solutions. In this experiment, SCOPE was able to produce a new planner that returned optimal solutions, and returned significantly shorter solutions than those returned by depth-first alone. Best-first search also generated near-optimal solutions.

[4]Note that the best-first tests were done using the Lisp implementation of UCPOP. Thus this result could be partially due to implementation differences between Prolog and Lisp.

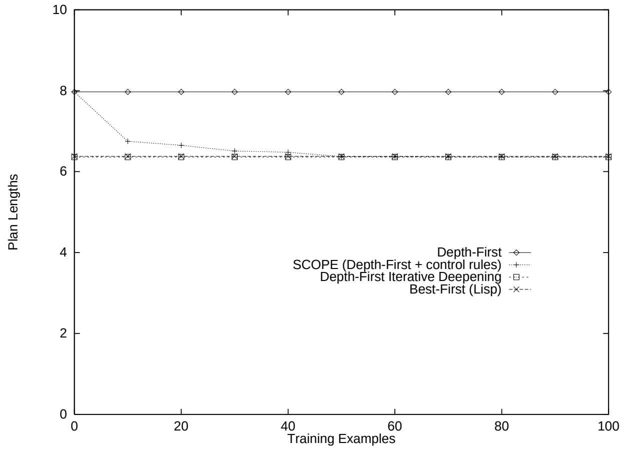


Figure 5: **Quality Performance.** Here, depth-first iterative deepening and best-first have almost identical performances.

Overall, these results indicate that is it possible to learn control rules that improve both planning efficiency and plan quality. By using depth-first iterative deepening to solve the training problems, SCOPE can be trained on optimal solutions. SCOPE can then learn control rules that not only help the planner find solutions quickly, but that also lead to high quality solutions.

5.3 Scalability

One other set of experiments was performed to test how SCOPE performed on harder problems. Using depth-first iterative deepening, SCOPE was trained on 100 training problems from the same distribution explained above. Four different test sets of increasing complexity were then used to evaluate SCOPE's performance. During testing, a time limit of 500 seconds was used. No limits on solutions length were imposed on test problems in these experiments, however, a depth bound was used that ensured all test problems could be solved. Five trials were run for these tests, after which results were averaged.

The results are shown in Table 1. The first two columns of the table show the percentage of test problems that could be solved within the time limit (using depth-first search) before and after learning. The next set of columns report the planning time required. The last two sets of columns contain results on plan quality. In the first set, we show the average solution lengths of all solved test problems before and after learning. In the last set, we show the average solution lengths (and optimal solution lengths) for problems that could be solved

Test Sets		**Problems Solved** *(%)*		**Solution Time** *(Secs)*		**Plan Lengths**		**Plan Lengths** *(Only problems solvable by original planner)*		
Num. of Goals	*Num. of Problems*	*W/o Rules*	*With Rules*	*W/o Rules*	*With Rules*	*W/o Rules*	*With Rules*	*W/o Rules*	*With Rules*	*Optimal Lengths*
1	100	100	100	5632	7	8.1	5.6	8.1	5.6	5.6
2	100	18	90	41000	4992	6.7	9.9	6.7	6.7	6.7
5	100	0	58	50000	27935	-	20.1	-	-	-
10	100	0	9	50000	45518	-	33.7	-	-	-

Table 1: Efficiency and quality results on increasingly complex test problems in the logistics domain.

by the original planner (without rules). This last set of columns show any improvements made in solution length by the learned rules.

For all four test sets, Scope was able to improve planning efficiency, and when possible, increase the percentage of problems solved. For instance, on the first set of problems (containing 1 goal), Scope was able to create a new planner that was an average of 800 times faster than the original. Unfortunately, it was difficult to gather data on whether plan quality was improved in all test sets. For many examples, the original planner could not find a solution under the time limit, and thus we were often not able to compare solution lengths. (Additionally, neither DFID or best-first search could find solutions for any problems in the two larger test sets.) For the first test set, Scope was able to significantly improve final solution quality and always generated optimal solutions. For the second test set, only 18 problems could originally be solved under the time limit and these solutions were already at optimal length.

6 Related Work

Most systems that learn control knowledge for planning have been directed at improving planning efficiency. In this regard, the most closely, related system to Scope is UCPOP+EBL [Kambhampati *et al.*, 1996]. In contrast to Scope, which uses a combination of EBL and induction, UCPOP+EBL uses a purely explanation-based approach to construct control rules in response to planning failures. This system has been shown to improve planning efficiency, however, Scope has outperformed UCPOP+EBL in previous experiments using the blocksworld domain [Estlin and Mooney, 1996]. Most other research on learning control rules to improve planning efficiency has been conducted on linear, state-based planners. [Minton, 1989; Etzioni, 1993; Leckie and Zuckerman, 1993; Bhatnagar and Mostow, 1994].

Very little research has been done in learning rules that improve plan quality. One learning mechanism for improving the quality of plans was introduced by [Pérez, 1996] and is built on top of the PRODIGY nonlinear planner [Carbonell and et al., 1992]. It uses EBL and an input quality evaluation function to explain why a higher quality solution is better than a lower quality one and converts this information into control knowledge. This method has been successfully used to improve solution quality in the process planning domain [Gil, 1991]. In contrast, Scope uses a combination of learning techniques to learn control rules that cover good planning decisions and rule out bad ones. Also, Scope has been focused on improving **both** quality and efficiency.

Another learning method for improving quality, which was also runs on the PRODIGY nonlinear planner, was developed by [Iwamoto, 1994]. This technique uses EBL to acquire control rules for near-optimal solutions in LSI design. This method is similar to the one employed by Perez, however it does not make use of the quality evaluation function to build the explanation.

Most other work in plan quality has concentrated on adding features to the plan algorithm itself that prefer least-cost plans [Hayes, 1990; Williamson and Hanks, 1994] or on examining goal interactions and how they relate to solution quality [Wilensky, 1983; Foulser *et al.*, 1992].

7 Future Directions

There are several issues we hope to address in future research. First, we would like to experiment with different types of quality metrics. We are currently working on implementing a version of the logistics domain where actions have different execution costs. In this way, we can measure plan quality in terms of plan execution cost as well as plan length. We would also like to experiment with the Truckworld domain utilized by [Williamson and Hanks, 1994] where resource consumption is important, and the process planning domain [Gil, 1991], where a number of quality metrics would be applicable.

Finally, we would like to devise a method for incremental learning and training for Scope. Scope could use learned control information for smaller problems to help generate training examples for more complex problems. This type of incremental approach could help Scope scale up more effectively to solve harder problems.

8 Conclusion

This paper describes experimental results from learning control knowledge to improve both planning efficiency and plan quality. Most planning and learning research has concentrated on improving efficiency. Another important learning goal is to improve the quality of the final solution. However, little research has been done in learning rules that improve both these metrics. In this paper, we have presented results utilizing the SCOPE learning system that show both of these metrics can be improved simultaneously by focusing the learning system on avoiding both dead-end paths and paths that lead to sub-optimal solutions.

References

[Barrett *et al.*, 1995] Anthony Barrett, Dave Christianson, Marc Friedman, Chung Kwok, Keith Golden, Scott Penberthy, and Daniel Weld. UCPOP: User's manual (version 4.0). Technical Report 93-09-06d, Department of Computer Science and Engineering, University of Washington, November 1995.

[Bhatnagar and Mostow, 1994] Neeraj Bhatnagar and Jack Mostow. On-line learning from search failure. *Machine Learning*, 15:69–117, 1994.

[Carbonell and et al., 1992] J. Carbonell and et al. PRODIGY4.0: The manual and tutorial. Technical Report CMU-CS-92-150, School of Computer Science, Carnegie Mellon University, Pittsburg,PA, 1992.

[Chien *et al.*, 1996] Steve Chien, Randall Hill, XueMei Wang, Tara Estlin, Kristina Fayyad, and Helen Mortensen. Why real-world planning is difficult: A tale of two applications. In M. Ghallab and A. Milani, editors, *New Directions in AI Planning*, pages 287–298. IOS Press, Amsterdam, 1996.

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

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

[Estlin and Mooney, 1996] Tara A. Estlin and Raymond J. Mooney. Multi-strategy learning of search control for partial-order planning. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Portland, OR, August 1996.

[Estlin, 1996] Tara A. Estlin. Integrating explanation-based and inductive learning techniques to acquire search-control for planning. Technical Report AI96-250, Department of Computer Sciences, University of Texas, Austin, TX, 1996.

[Etzioni, 1993] O. Etzioni. Acquiring search control knowledge via static analysis. *Artificial Intelligence*, 60(2), 1993.

[Foulser *et al.*, 1992] D. Foulser, M. Li, and Q. Yang. Theory and algorithms for plan merging. *Artificial Intelligence*, 52:143–181, 1992.

[Gil, 1991] Y. Gil. A specification of manufacturing processes for planning. Technical Report CMU-CS-91-179, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1991.

[Hayes, 1990] Caroline Hayes. *Machining Planning: A Model of an Expert Level Planning Process.* PhD thesis, The Robotics Institue, Carnegie Mellon University, December 1990.

[Iwamoto, 1994] M. Iwamoto. A planner with quality goal and its speedup learning for optimization problem. In *Proceedings of the Second International Conference of AI Planning Systems*, Chicago, June 1994.

[Kambhampati *et al.*, 1996] Subbarao Kambhampati, Suresh Katukam, and Yong Qu. Failure driven search control for partial order planners: An explanation based approach. *Artificial Intelligence*, 88, 1996.

[Korf, 1985] R. Korf. Depth-first iterative-deepening: An optimal admissable tree search. *Artificial Intelligence*, 27(1), 1985.

[Leckie and Zuckerman, 1993] Chistopher Leckie and Ingrid Zuckerman. An inductive approach to learning search control rules for planning. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1100–1105, Chamberry,France, August 1993.

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

[Mitchell *et al.*, 1986] Tom M. Mitchell, Richard M. Keller, and Smadar T. Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1(1):47–80, 1986.

[Muggleton, 1992] S. H. Muggleton, editor. *Inductive Logic Programming.* Academic Press, New York, NY, 1992.

[Pérez, 1996] M. Alicia Pérez. Representing and learning quality-improving search control knowledge. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 382–390, Bari,Italy, July 1996.

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

[Silverstein and Pazzani, 1991] Glenn Silverstein and Michael J. Pazzani. Relational clichés: Constraining constructive induction during relational learning. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 203–207, Evanston, IL, June 1991.

[Veloso, 1992] Manuela M. Veloso. *Learning by Analogical Reasoning in General Problem Solving.* PhD thesis, School of Computer Science, Carnegie Mellon University, August 1992.

[Wilensky, 1983] Robert W. Wilensky. *Planning and Understanding: A Computational Approach to Human Reasoning.* Addison-Wesley, Reading, MA, 1983.

[Williamson and Hanks, 1994] Mike Williamson and Steve Hanks. Optimal planning with a goal-directed utility model. In *Proceedings of the Second International Conference of AI Planning Systems*, pages 176–181, Chicago, June 1994.