Proceedings of the Ninth National Conference on Artificial Intelligence, AAAI Press, Menlo Park, CA, 1991

Integrating Abstraction and Explanation-Based Learning in PRODIGY*

Craig A. Knoblock
Carnegie Mellon University
School of Computer Science
Pittsburgh, PA 15213
cak@cs.cmu.edu

Steven Minton
Sterling Federal Systems
NASA Ames Research Center
Mail Stop: 244-17
Moffett Field, CA 94035
minton@pluto.arc.nasa.gov

Oren Etzioni
University of Washington
Department of Computer Science
and Engineering, FR-35
Seattle, WA 98195
etzioni@cs.washington.edu

Abstract

This paper describes the integration of abstraction and explanation-based learning (EBL) in the context of the PRODIGY system. PRODIGY's abstraction module creates a hierarchy of abstract problem spaces, so problem solving can proceed in a more directed fashion. The EBL module acquires search control knowledge by analyzing problem-solving traces. When the two modules are integrated, they tend to complement each other's capabilities, resulting in performance improvements that neither system can achieve independently. We present empirical results showing the effect of combining the two modules and describe the factors that influence the overall performance of the integrated system.

Introduction

Artificial intelligence has traditionally favored a reductionistic approach to the study of intelligent systems, where methods for reasoning, learning, knowledge representation, etc., are studied separately. More recently, researchers have begun to seriously consider the issues involved in integrating the individual components into a single system, uncovering issues that were not considered when the components were designed. In this paper, we study the integration of abstraction and explanation-based learning (EBL) in the context of the PRODIGY system [Carbonell *et al.*, 1991]. PRODIGY consists of a central planner, and a set of distinct modules for abstraction generation, explanation-based learning, static problem-space analysis, analogical reasoning, etc. PRODIGY's EBL module acquires search control knowledge by analyzing problem-solving traces. ALPINE, PRODIGY's abstraction module, creates a hierarchy of abstract problem spaces so that problem solving can proceed in a more directed fashion. The purpose of both modules is to increase the efficiency of the problem solver's search, but they do so using different methods.

EBL and abstraction are natural candidates for integration. ALPINE creates a hierarchy of abstract problem spaces but provides no control guidance within each space, a role tailor-made for EBL's control rules. EBL, on the other hand, can often produce more effective control rules in an abstract problem space than in the original space. This occurs because the abstract space is generally simpler for EBL to analyze, so the resulting control rules are both more general and less expensive to match.[1]

The first part of this paper reviews the abstraction and EBL mechanisms in PRODIGY. Then we describe the integration of the two modules and illustrate the effect of combining the modules using the Tower of Hanoi puzzle. On this example, abstraction alone only provides some reduction in search, and EBL alone is ineffective in the original problem space due to the difficulties in analyzing Tower of Hanoi traces. Yet, the combination of the two modules completely eliminates search in this problem space. This example illustrates the synergistic effect that can occur when the two methods are combined. We then describe an empirical study in a machine-shop scheduling domain, which illustrates the synergy achieved by the combined system in a more complex problem space over a large set of examples. Finally, we compare the sources of power used by the two mechanisms in order to describe why this synergistic effect occurs, and identify when we can expect the two modules to work well together.

*The first author was supported by an Air Force Laboratory Graduate Fellowship through the Human Resources Laboratory at Brooks AFB. The third author was supported by an AT&T Bell Labs Ph.D. Scholarship. This research was sponsored in part by the Avionics Laboratory, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U.S. Air Force, Wright-Patterson AFB, OH 45433-6543 under Contract F33615-90-C-1465, Arpa Order No. 7597.

[1] PRODIGY also includes a problem space compiler, STATIC [Etzioni, 1990], that carries out a similar, but more restricted, version of EBL's analysis. While we have not yet explored the integration of STATIC and abstraction, we would expect the use of abstraction to similarly benefit STATIC.

The PRODIGY System

PRODIGY is an integrated system for planning and learning. PRODIGY's basic reasoning engine is a general-purpose means-ends analysis problem solver that searches for sequences of operators to accomplish a set of goals. In order to solve problems in a particular problem space, PRODIGY must first be given a specification of that problem space, consisting of a set of operators and inferences rules. Search in PRODIGY can be guided by *control rules* that apply at its decision points: selection of which node to expand, of which goal to work on, of which operator to apply, and of which objects (operator bindings) to use. At each decision point, the control rules can select or reject alternatives or prefer some alternatives over others.

In this paper the Tower of Hanoi puzzle is used for illustration. The picture at the top of Figure 1 shows the state space for the three-disk puzzle. As shown in the picture, the problem is to find a sequence of operators to move the three disks from the first peg to the third peg.

Abstraction in PRODIGY

PRODIGY's abstraction module, ALPINE, takes an initial problem-space specification and automatically generates a hierarchy of abstraction spaces [Knoblock, 1990, Knoblock, 1991]. Each abstraction space in the hierarchy is formed by dropping conditions from the original problem space. An abstraction space is defined by a set of abstract operators and states. In the Tower of Hanoi, for example, an abstract problem space can be formed by dropping all of the conditions referring to the smallest disk from both the operators and states. The resulting abstract space can be further simplified by dropping all the conditions referring to the medium-sized disk. Figure 1 shows the original state space and the two reduced state spaces for the Tower of Hanoi.

PRODIGY uses the abstraction hierarchies produced by ALPINE to perform hierarchical problem solving. The use of the abstractions in problem solving allows PRODIGY to focus on the more difficult aspects of a problem first and thereby reduce the overall search. Given a problem, PRODIGY first solves it at the highest level of abstraction. To do this, a problem is mapped into an abstract problem by dropping the conditions that are not relevant to that abstraction level. The abstract problem is then solved in the abstract problem space. The resulting abstract solution then serves as a skeleton for the solution at the next abstraction level, where additional operators are inserted into the plan to achieve the conditions that were ignored in the higher-level space. The problem solver refines the plan at each successive level in the hierarchy and ultimately produces a plan that solves the original problem.

ALPINE forms abstraction hierarchies based on the *ordered monotonicity* property, which requires that the truth value of a condition introduced at one level is never changed at a lower level. This property guarantees that the preconditions that are achieved in an abstract plan will not be deleted (clobbered) while refining that plan. To construct abstraction hierarchies with this property, ALPINE analyzes the preconditions and postconditions of the problem space operators to determine potential interactions. If a plan for achieving condition C_1 can change the truth value of condition C_2, then condition C_1 cannot be in a lower abstraction level than condition C_2. The potential interactions define a set of constraints on the final abstraction hierarchy that are sufficient to guarantee the ordered monotonicity property. The detailed algorithm is described in [Knoblock, 1991].

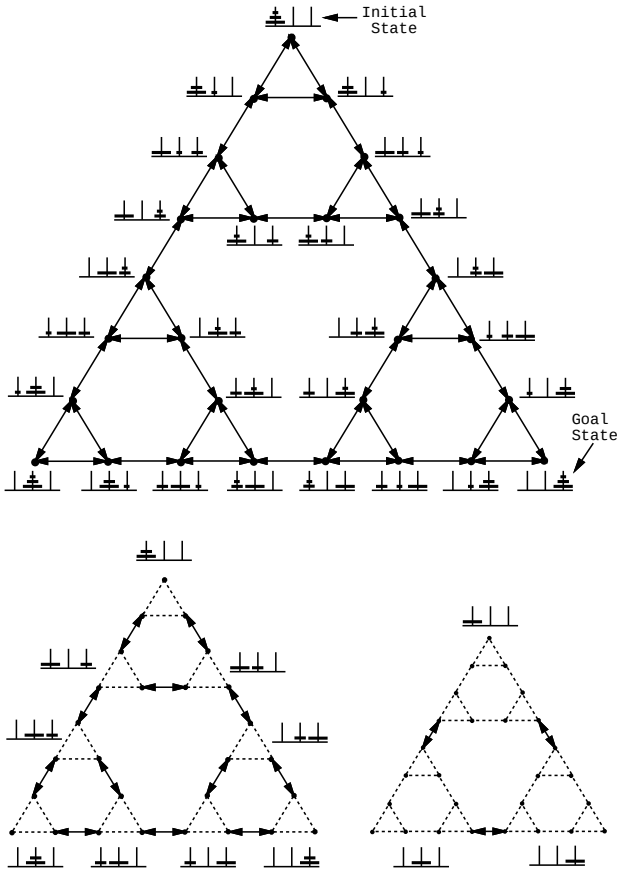

Figure 1: Original and Reduced State Spaces for the Tower of Hanoi.

ALPINE automatically constructs an abstraction hierarchy for the Tower of Hanoi by partitioning the conditions for each of the three different-sized disks into three abstraction levels. The largest disk is placed in the most abstract level, the medium disk is added at the second level, and the third level contains all three disks. This hierarchy has the ordered monotonicity property, since once an abstract plan is produced for a given disk, one can plan how to move the smaller disks

without clobbering the conditions achieved in the abstract plan.

Explanation-Based Learning in PRODIGY

PRODIGY's explanation-based learning module produces search control knowledge by analyzing the planner's experiences [Minton, 1988]. After each planning episode, the EBL module examines the control choices that were made, as recorded in a trace produced by the planner. By explaining why a control choice was appropriate or inappropriate, the system can learn a general search control rule. The learned rule enables the planner to make the right choice if a similar situation arises during subsequent problem solving. The EBL module is able to analyze several types of experiences, including success, failure and goal interference, as defined below. In EBL terminology these are referred to as the *target concepts*.

1. SUCCESS: A control choice succeeds if it leads to a solution.
2. FAILURE: A choice fails if there is no solution consistent with that choice.
3. GOAL-INTERFERENCE: A choice results in goal interference if in every solution consistent with that choice, a precondition in the solution is clobbered. (A precondition is clobbered if it is asserted, deleted, and then re-asserted).

After each problem solving episode, the system examines the problem-solving trace to find examples of the target concepts. The system uses a set of user-provided heuristics to pick out examples that it estimates would result in useful rules being learned. For each example, the system explains why the target concept is satisfied by the example. For instance, given an example of a failure where the wrong operator is chosen, the system explains why that operator failed. Explanations are constructed using a theory that describes the problem-space operators and the relevant aspects of the PRODIGY planner. To form a control rule, the system finds the weakest preconditions of the explanation, and these become the antecedent of the rule. The antecedent specifies the conditions under which the rule is applicable. The consequent of the control rule is determined by the target concept and specifies the effect of the rule.

Consider a two-disk tower of Hanoi problem that illustrates how PRODIGY learns from an example of goal interference. The problem is to move the two disks (a large disk and a small disk) from the first peg to the third peg. In order to move the large disk, the subgoal of removing the small disk from the first peg must first be achieved. There are two available alternatives, since the small disk can be moved to either the second or third pegs. However, the third peg is a bad choice since the large disk cannot then be moved there (unless the small disk is moved again). PRODIGY explains that moving the small disk to the third peg clobbers a precondition of moving the large disk to that peg, and it learns a control rule to make the correct control decision in the future. The control rule states that when the current goal is to move the small disk from a peg, and this is a subgoal of moving the large disk to some peg x, then the system should prefer moving the small disk to a peg other than peg x.

Integrating Abstraction and EBL

While both of the learning methods described in the previous sections are effective at reducing search, each approach has some limitations. ALPINE takes the initial problem space and forms a hierarchy of abstract problem spaces. However, the use of abstraction provides no guidance about how to solve a problem *within* each of the abstract problem spaces, so search is still required. The EBL module learns search control rules by analyzing the problem solving traces, however, as the complexity of the training problems increases, the EBL module tends to produce less effective control rules.

A natural approach to addressing these limitations is to integrate the use of abstraction and explanation-based learning. The EBL module can be applied within each abstract problem space to learn search-control rules. This provides the guidance needed to solve problems within an abstraction space. In addition, the abstract problem spaces provides simpler problems and a simpler theory for the EBL module, enabling it to learn better control rules. The learned rules can also be partitioned across the abstraction hierarchy, which results in fewer, more general rules to consider at each node in the search space.

Implementation

The use of abstraction in PRODIGY divides a problem into a number of simpler subproblems. To combine the use of abstraction and EBL, the EBL module is used to learn control rules on each of these subproblems. This is implemented by first solving a problem using hierarchical problem solving and recording all of the individual subproblems that arise and then running the EBL module on each of the subproblems to produce a set of control rules.

Instead of applying the learned control rules at every level of abstraction, the learned control rules are only considered in the most abstract space in which all the conditions of the rule are relevant. The ordered monotonicity property partitions the possible goals and subgoals into distinct levels in the abstraction hierarchy. Since each control rule is specific to a goal, it follows that once a rule is considered at a given level, it need not be considered at any lower levels. Thus, before problem solving begins, each control rule is associated with a particular abstraction level.

Integration in the Tower of Hanoi

The Tower of Hanoi problem space illustrates the potential synergy between abstraction and EBL. If ab-

straction is used alone, solving the Tower of Hanoi still requires some search. At each abstraction level the system must determine where to move the disk introduced at that level so that it does not interfere with the disk that was moved at the next higher level. While the use of abstraction reduces the total amount of search, it by no means eliminates it. In addition, the use of abstraction produces shorter solutions, but not optimal ones.

Similarly, if EBL is used alone, the system will not learn the control rules required for efficiently solving the three-disk problem or any larger problem. The problem is that the EBL module avoids learning about an observed goal interference if *all* ways to solve the problem involve interferences, as is the case with the Tower of Hanoi when three or more disks must be moved. (Every solution involves at least one goal that is clobbered and then reachieved). This heuristic is used to prevent the system from learning conflicting preference rules at decision points where all choices result in goal interference. Conflicting preferences would provide no guidance and merely slow the system down.

Combining abstraction and EBL in the Tower of Hanoi reduces the problem to one that can be solved completely deterministically (i.e., the correct decision is made at every choice point). As explained previously, the abstraction simplifies the problem such that the only search involves moving a disk out of the way of another disk that needs to be moved. There are only two places to move the disk, one of which is the "right" place and the other of which will interfere with the placement of another disk. Thus, within each abstraction level, there is an interference-free solution. Consequently, EBL can learn the required control rules within each abstraction level, and in fact, this is exactly the type of learning situation illustrated in the last section for the two-disk example. Thus, the EBL system can learn a set of rules that allow the problem solver to make the correct choices at each level in the hierarchy.

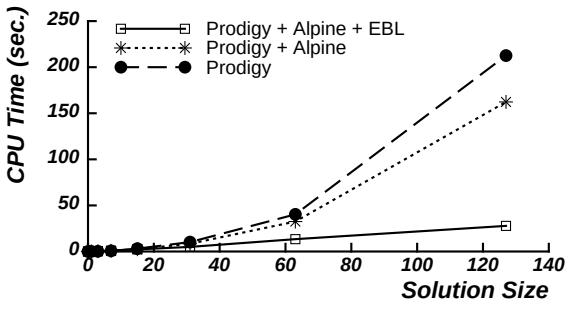

Figure 2: Integration in the Tower of Hanoi.

As shown in Figure 2, the combination of the two techniques in the Tower of Hanoi produces performance improvements that neither system can achieve independently. The use of ALPINE's abstractions alone produces only a small reduction in problem-solving search. The EBL module alone is unable to learn a useful set of rules, so it performs the same as PRODIGY without any control knowledge. In contrast, the combination of the two approaches eliminates all search from the problem and produces optimal solutions.

Results

This section compares the performance of PRODIGY individually and in combination with the abstraction and EBL modules in a machine-shop scheduling domain [Minton, 1988]. The various configurations were each run on one-hundred randomly-generated problems that were originally used for testing the EBL system.

Comparing the results of the different configurations on the set of test problems is complicated by the fact that some of the problems cannot be solved within the 600 CPU second time limit. Since the choice of a time bound can affect the relative performance of the different configurations [Segre *et al.*, 1991], Figure 3 shows the total time expended for each configuration as the time bound increases. The slope at each point in a given curve indicates the relative portion of the problems that remain unsolved. A slope of zero, for example, means that all of the problems have been solved (no more time is required to solve any of the problems). As shown by the graph, PRODIGY combined with either abstraction or EBL performs better than the basic PRODIGY system, and the combination of abstraction and EBL performs better than either system alone.[2] These results appear to be robust across different time bounds.

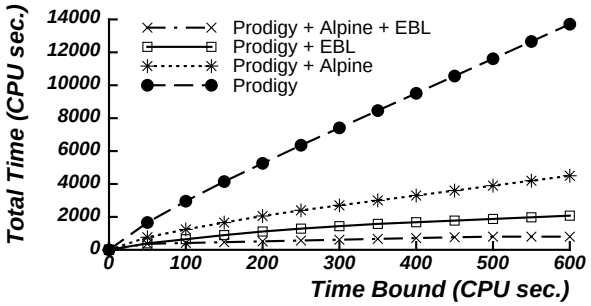

Figure 3: Integration in the Machine-Shop Domain.

Discussion

Methods for integrating abstraction and EBL have only been explored in a few other systems. Sacerdoti [1974] briefly discussed the potential uses for learned macro operators in the ABSTRIPS system. More recent work has explored the use of simplified or in-

[2] Minor syntactic variations on the problem space specification, required to integrate the two modules, degrade ALPINE's performance slightly relative to the results reported in [Knoblock, 1990].

complete explanations to address the intractable theory problem in EBL [Chien, 1989, Tadepalli, 1989, Bhatnagar and Mostow, 1990]. The most closely related work is by Unruh and Rosenbloom [1989], who developed an automatic abstraction mechanism in SOAR. While their method for generating abstractions is quite different from the method used in PRODIGY, their overall approach is similar in that SOAR employs an EBL-like "chunking" process to learn search-control knowledge. Unruh and Rosenbloom showed an example in which abstraction increased the generality of a learned chunk, but they did not explore in detail how their abstraction method affects the utility of learning.

In the integration of the abstraction and EBL systems in PRODIGY, there are several complex factors that determine the overall performance of the integrated system. It is relatively clear why abstraction and EBL can perform better than abstraction alone: EBL can learn control rules to guide search within each abstraction level. A more interesting question is why abstraction can help the EBL module to learn a more useful set of rules than it would in the original problem space? There are two different ways in which the abstraction module provides an advantage for EBL. First, the use of abstraction can simplify EBL's explanations. Second, abstraction can increase the utility of EBL's rules by reducing their match cost. The following sections explore these issues in more detail.

The Impact of Abstraction on Explanation Generation

In general, the main difficulty for EBL in PRODIGY is that the problem-solving trace may be very long and complex. The EBL module relies on the problem-solving trace for learning; if the trace is too complex, then it can lead the EBL module astray. This can happen in two ways. First, the heuristics that the EBL module uses to identify useful training examples may be confounded, as in our Tower of Hanoi example. In the three-disk Tower of Hanoi, PRODIGY cannot identify how to solve the problem without a goal interference, so it does not learn to avoid goal interferences. Second, the EBL module may identify a training example that appears useful, but the explanation may be overly complex, and thus overly-specific.

Consider a simplified example from the machine-shop domain, where `Part1` must be painted red at `Time-Slot1`. There are two possible operators: `Spray-Paint` and `Immersion-Paint`. Suppose that when trying `Spray-Paint`, PRODIGY finds that the `Spray-Paint` machine is unavailable at `Time-Slot1`, so the operation fails. When trying the alternative, `Immersion-Paint`, PRODIGY finds that there is no red paint left, so that choice fails too. The resulting explanation lists the observed reasons for failure, that is, painting an object fails if the `Spray-Paint` machine is unavailable at that time *and* there is no paint of that color left. However, this explanation is overly specific because, in fact, `Spray-Paint` will also fail if there is no red paint left *regardless of whether the* `Spray-Paint` *machine is busy or not.* The EBL system produces an overspecific explanation because it relies on the failure conditions identified by the problem solver, and the problem solver simply reports the failure conditions it finds first, not the most concise or general set of failure conditions. Thus, while a human can easily see that a simpler and more succinct explanation for the failure exists, EBL does not.

Abstraction can improve EBL's analysis because there tends to be less search in an abstract space than in the original problem space. Consequently, EBL's explanations will have fewer conditions, so the resulting control rules will be both more general and less expensive to match. Consider explaining the painting failure described above. Since ALPINE automatically separates scheduling from process planning in the machine-shop domain by dropping the conditions that involve scheduling, PRODIGY first orders the machining operations and only then assigns time slots to its operations. When time is abstracted away, painting will still fail, but the sole reason for the failure is that the required paint is not available, yielding the more succinct explanation we are after.

One important special case of this occurs when abstraction partitions the problem space so that recursion is restricted to certain partitions. Because the cost of matching control rules learned from recursive explanations is exponential in the depth of the recursions encountered, and because such rules are recursion-depth specific, EBL is often ineffective when its explanations are recursive [Etzioni, 1990]. Abstraction can help EBL overcome this "problem of recursive explanations" by partitioning a recursive problem space into a nonrecursive abstract space, and a lower-level recursive space. Thus, EBL will at least perform well at the abstract space. In the machine-shop scheduling domain, for example, recursion (or, more precisely, iteration) arises when considering successive time slots in a schedule. Since the abstractions push time slot considerations to the lowest abstract space, the more abstract problem spaces are nonrecursive and hence more manageable for EBL. In the Tower of Hanoi problem space, recursion arises in explaining goal interference. Since ALPINE's abstraction hierarchies separate the placement of each disk into distinct abstract problem spaces, the recursion is completely removed from the traces EBL analyzes.

Abstraction will not *necessarily* help EBL. Specifically, ALPINE may sometimes remove conditions critical to explaining failure and goal interference. Although ALPINE guarantees the ordered monotonicity property, so that the conditions established by the abstract plan will never be undone by problem solving at a lower level, it does not guarantee that every abstract plan is *realizable*. This means that an abstract plan may be produced that cannot be refined into a plan

in the original problem space. Thus, PRODIGY can produce an abstract solution, and find itself unable to completely refine this solution, in which case it must backtrack across abstraction levels to find a different abstract solution. In such cases EBL will not be able to form useful control rules in the abstract problem space. The information necessary to explain the appropriate control choices is "hidden" by the abstraction. Since, ALPINE usually produces "good" abstractions in the problem spaces studied [Knoblock, 1991], this scenario is atypical. Nonetheless, to the extent that the planner does backtrack across abstraction levels, it can cause EBL to perform poorly (relative to its performance in the original space). One solution to this problem is to enable EBL to explain failures, etc., across abstraction levels; this is a topic for future work.

The Impact of Abstraction on the Utility of Control Rules

Once EBL acquires a learned rule, it is indexed so that during subsequent problem solving, it is considered only at the most abstract space in which it could possibly be relevant. The partitioning of control rules into different abstraction levels reduces the cost of matching the learned control rules for several straightforward reasons. First, since each control rule is only matched at the abstraction level in which all of its conditions are relevant, the control rules are matched less frequently than in standard problem solving (where they are matched at every decision point). Second, the matching process itself is slightly faster at abstract levels because the reduced states may contain fewer literals that are relevant to the rule.

On the other hand, abstraction can also reduce the potential benefit of the learned rules. Since less time is spent at any given abstraction level, the amount of search that is saved by the application of a control rule will be reduced. However, our empirical results show that the benefits outweigh the costs.

Conclusion

In this paper we have shown that explanation-based learning and abstraction can be integrated so that each complements the other's performance. In some cases, as in the Tower of Hanoi, the individual modules may be relatively ineffective by themselves and yet the combined system can produce significant performance improvements due to their synergistic interaction. The integrated system improves on the individual components in more complex domains as well, such as the machine-shop scheduling domain. As VanLehn [1989] points out, human learners can utilize a variety of different learning methods in the course of a single learning episode. The integration of abstraction and EBL brings PRODIGY one step closer towards this goal.

References

[Bhatnagar and Mostow, 1990] Neeraj Bhatnagar and Jack Mostow. Adaptive search by explanation-based learning of heuristic censors. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 895–901, Boston, MA, 1990.

[Carbonell *et al.*, 1991] Jaime G. Carbonell, Craig A. Knoblock, and Steven Minton. PRODIGY: An integrated architecture for planning and learning. In Kurt VanLehn, editor, *Architectures for Intelligence*, pages 241–278. Lawrence Erlbaum, Hillsdale, NJ, 1991.

[Chien, 1989] Steve A. Chien. Using and refining simplifications: Explanation-based learning of plans in intractable domains. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 590–595, Detroit, MI, 1989.

[Etzioni, 1990] Oren Etzioni. *A Structural Theory of Explanation-Based Learning.* PhD thesis, School of Computer Science, Carnegie Mellon University, 1990. Available as Technical Report CMU-CS-90-185.

[Knoblock, 1990] Craig A. Knoblock. Learning abstraction hierarchies for problem solving. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 923–928, Boston, MA, 1990.

[Knoblock, 1991] Craig A. Knoblock. *Automatically Generating Abstractions for Problem Solving.* PhD thesis, School of Computer Science, Carnegie Mellon University, 1991. Available as Technical Report CMU-CS-91-120.

[Minton, 1988] Steven Minton. *Learning Search Control Knowledge: An Explanation-Based Approach.* Kluwer, Boston, MA, 1988.

[Sacerdoti, 1974] Earl D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5(2):115–135, 1974.

[Segre *et al.*, 1991] Alberto Segre, Charles Elkan, and Alex Russell. A critical look at experimental evaluations of EBL. *Machine Learning*, 6(2), 1991.

[Tadepalli, 1989] Prasad Tadepalli. Lazy explanation-baserd learning: A solution to the intractable theory problem. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 694–700, Detroit, MI, 1989.

[Unruh and Rosenbloom, 1989] Amy Unruh and Paul S. Rosenbloom. Abstraction in problem solving and learning. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 681–687, Detroit, MI, 1989.

[VanLehn, 1989] Kurt VanLehn. Discovering problem solving strategies: What humans do and machines don't (yet). In *Proceedings of the Sixth International Workshop on Machine Learning*, pages 215–217, Ithaca, NY, 1989.