Learning Approximate Control Rules Of High Utility

William W. Cohen
Computer Science Department
Rutgers University
New Brunswick, NJ 08903

Abstract

One of the difficult problems in the area of explanation based learning is the *utility problem*; learning too many rules of low utility can lead to *swamping*, or degradation of performance. This paper introduces two new techniques for improving the utility of learned rules. The first technique is to combine EBL with inductive learning techniques to learn a better set of control rules; the second technique is to use these inductive techniques to learn approximate control rules. The two techniques are synthesized in an algorithm called *approximating abductive explanation based learning* (AxA-EBL). AxA-EBL is shown to improve substantially over standard EBL in several domains.

1 Introduction

One of the difficult problems in the area of explanation based learning is the *utility problem.* The *utility* of a rule is its contribution to performance improvement; the utility is directly proportional to the *coverage* of a rule and inversely proportional to the *match cost* of a rule, where *coverage* is defined as the percentage of the time that the rule is used in problem solving, and *match cost* is simply the time needed to determine if a rule is applicable.

The utility problem often arises in learning rules intended to improve the performance of a problem solver. Learning too many rules of low utility can lead to *swamping*, or degradation of performance: in the worst case, a problem solver can be slower after learning than before learning. Previous research on the utility problem has focused on detecting and discarding rules of low utility [Minton, 1988; Markovitch and Scott, 1989], lowering the match cost of rules using partial evaluation or other simplification techniques [Prieditis and Mostow, 1987; Minton, 1988; Tambe and Rosenbloom, 1989], and constraining the use of learned rules [Mooney, 1989].

This paper introduces two new techniques for improving the utility of learned rules. The first technique is to combine EBL with *inductive learning techniques* to learn a better set of control rules. The second technique is to use these inductive techniques to learn *approximate control rules*: that is, to learn rules which are approximations of the control rules which would be learned by standard EBL techniques. Although use of inductive techniques alone does not always lead to rules of higher utility, the combination of these techniques is shown to substantially improve the performance of EBL across a variety of domains.

Combining EBL with inductive learning. Flann and Dietterich have noted that in the process of learning control rules, many EBL systems make inductive leaps. An example given in [Flann and Dietterich, 1989] is a rule learned by LEX2 in the symbolic integration domain:

If the current problem matches
$\int cx^r dx (r \neq -1)$
Then use the operator
$\int cf(x)dx \Rightarrow c\int f(x)dx$

Forming this rule is an inductive leap; this is witnessed by the fact that the rule recommends the *wrong* action on the problem $\int 0x^4dx$ (multipying out the zero to get $\int 0dx$ leads to a shorter solution). Another case in which inductive decisions are made is when only one of several possible control rules is learned; for example, PRODIGY might learn either a macro-operator or a set of operator preference rules from a trace [Minton, 1988].

However, the standard implementation of EBL, in which control rules are learned incrementally as needed, is relatively slow as an inductive learner. Combining EBL with more powerful inductive learning techniques can improve the rate at which learning converges to an adequate set of control rules. A crucial point is that on average, *improving the rate of convergence will also improve the coverage of the individual rules learned.* This is true because in order to improve the learning rate, it necessary to find *fewer* rules with *better* coverage.[1]

[1] It must be emphasized that we are *not* advocating use of pure inductive learning techniques (*e.g.*, version spaces) to learn control rules. Explanation-based techniques are clearly more appropriate in learning control rules because there is a strong theory.

Using approximate rules. The first technique alone, however, is not enough to avoid swamping. The problem is that in learning control rules, one is trying to maximize the utility of the set of rules. To do this, one must simultaneously *maximize total coverage*, and *minimize total match cost.* Standard inductive learning techniques, however, maximize coverage without regard to match cost.

Viewed in this light, *learning useful control rules is a multiple-resource optimization problem*, in which the two antagonistic resources of coverage and match cost must be simultaneously optimized. A standard technique for solving such problems is to set an artificial constraint on one resource, and optimize the other resource subject to this constraint. This technique can be applied to the control rule learning problem by constraining the inductive learner to only consider control rules with match cost below a certain fixed cutoff.

Unfortunately, the control rules learned by explanation based techniques are almost always very complex; hence if a reasonable match-cost cutoff were chosen, very few rules would be available for the inductive learning method to consider. This problem can be remedied by allowing the learner to also consider *approximations* to expensive control rules.

In the remainder of the paper, we first discuss the control rule learning problem used to test these ideas, and then the learning algorithm used. We then describe the domains used for experimentation. Finally, we present and interpret our experimental results and draw some conclusions.

2 The learning problem

To evaluate the learning algorithms discussed in this paper, we used as a testbed the problem of learning *clause selection rules* for Prolog programs. This learning problem has several advantages.

- Commercial Prologs are highly optimized, and hence the initial problem solver has a fairly low overhead. This encourages careful experimentation, and also means that comparisons of performance before and after learning are a reasonable test of the overall effectiveness of learning.
- Many different problem solving strategies (for example, state space search, problem decomposition, means-end analysis, etc.) can be easily coded as Prolog programs in such a way that learning clause selection rules significantly improves performance.

Our learning testbed consists of two routines. The first is a *critic* which analyzes a set of Prolog traces and extracts examples of correct and incorrect clause selection decisions. A *correct* decision is any decision which leads to a solution; an *incorrect* decision is any decision which is eventually retracted by Prolog's backtracking strategy. Along with each correct decision, a proof of its correctness is recorded. The theory used to derive this proof is very simple: it merely contains the clauses of the initial program as axioms, and also an axiom of the form

$$\text{useful}(c_i, A) \leftarrow B_1, \ldots, B_q$$

for each clause $c_i = (A \leftarrow B_1, \ldots, B_q)$ in the initial program. This 'usefulness theory" is an extension to the problem of Prolog clause selection of the theory used in the LEX/2 system [Mitchell, 1982] for operator selection.

The second routine is a *control rule compiler* which, given a set of control rules and an initial program, generates a new Prolog program which incorporates the learned control rules. The routine takes as input a set of control rules of the form

$$\text{useful}(c_i, A) \leftarrow \mathcal{U}(A)$$

which can be read as saying "if $\mathcal{U}(A)$ is true, then it is useful to apply clause c_i to the goal A". $\mathcal{U}(A)$ is typically a complex conjunctive condition involving the variables of the goal A. For each such rule, the post-processor constructs a copy of clause c_i to which $\mathcal{U}(A)$ has been added as an additional "filtering condition"; in other words, if $c_i = (A \leftarrow B_1, \ldots, B_q)$, then a clause of the form

$$A \leftarrow \mathcal{U}(A), !, B_1, \ldots, B_q$$

is constructed. Notice that the cut (!) makes selection of this clause a *committed choice*: no backtracking will be done if this choice is incorrect.

One advantage of incorporating control rules in this way is that matching is done by Prolog's unification procedure; directly using the implementation language reduces the overhead of pattern matching.

If there is more than one control rule for c_i, then multiple copies of c_i are generated: *i.e.*, multiple control rules are interpreted disjunctively. A copy of the original clause with no filtering condition is retained if and only if some decisions to use that clause are not explained by any control rule. Clauses in the new program are ordered first according to the order of the clauses in the initial program to which they correspond (since this ordering may represent important control information), and then according to the ordering of control rules.

No simplification or compression is done.

Retaining the original clauses when some control decisions are unexplained means that some performance improvement can be gained even if control rules can only be learned for some clause selection decisions. This makes the learning system less brittle; however, it also means that some backtracking may still be done in the learned program. This is the motive for discarding the original clauses when all control decisions have been explained, and requiring that learned control decisions are not backtracked over; otherwise, in backtracking, all of the solutions considered by the original program would still have to be examined and rejected, limiting the degree to which performance can be improved. However, discarding clauses also means that the learned program may fail on problems which are solvable by the initial program. When this happens the original program is invoked on the top-level goal. Thus, in spite of using approximate control rules and committed choice control decisions, the learned program always solves at least as many problems as the original program did.

3 The learning algorithms

Three learning algorithms were experimentally compared. The first two are strawmen, corresponding to a conventional EBL control-rule learning system, and to an extended EBL learning system which uses inductive learning techniques to improve coverage, but which ignores match cost. The third algorithm uses inductive techniques to search a space of approximate rules, and is the main focus of this research.

One important difference between these algorithms and conventional control-rule learning algorithms is that they are *non-incremental*. Each learning algorithm takes as input a set of examples of correct and incorrect control decisions, which are generated by applying the critic to a large set of traces.

Standard EBL (EBL): Standard EBL starts with an empty list of control rules, and examines each correct control decision in the order in which they were made. If there are no existing control rules which account for the decision, then EBG is applied to the proof of the correctness of the control decision, and the resulting rule is added to the end of the list of learned control rules.[2]

Standard EBL is a batch simulation of a SOAR-like incremental learning system which uses a learned clause selection rule if one is available, and which otherwise first uses search to determine the right clause, then forms a clause selection rule which summarizes the results of the search process.

Abductive EBL (A-EBL): Abductive EBL also starts with an empty list of control rules, but repeatedly adds to the end of the list the control rule R which a) is *consistent*, *i.e.*, does not explain any incorrect control decisions, and b) maximizes the ratio of the number of unexplained decisions explained by R to the size of R.[3] R is chosen from a *candidate pool* of rules which consists of all rules which could be generated by applying EBG to some correct control decision.

The main loop in the A-EBL algorithm implements a greedy set cover of the control decisions; in other words, A-EBL looks for a minimal-size set of control rules to explain all the control decisions. The greedy set cover technique has also been used in Haussler's algorithm for learning disjunctive boolean formulae. A-EBL has been used with some success on inductive learning tasks [Cohen, 1989; Cohen, 1990]. Its main advantage over standard EBL techniques for such tasks is that it can be used even on an "abductive" domain theory — one which generates multiple inconsistent explanations. A-EBL has been shown to satisfy Valiant's criterion of pac-learnability, and to have near-optimal sample complexity [Cohen, 1989].

Approximating abductive EBL (AxA-EBL): Approximating abductive EBL works exactly like A-EBL, except that the candidate pool consists of all *k-bounded approximations* to a rule generated by applying EBG to some correct control decision. A k-bounded approximation is formed by dropping all but j conditions from the body of the rule, for some $j \leq k$.

A k-bounded approximation rule will have low match cost, unless the operational conditions are expensive to test. For example, if the truth of operational conditions is tested by database lookup in a database of n facts, then the match cost of a k-bounded approximation is $O(n^k)$.

AxA-EBL is more computationally expensive than EBL; it runs in time polynomial in the total size of the proofs of the control decisions, but exponential in k. Hence, only small values of k can be used. In the current implementation, two techniques are used to prune the search space of approximate rules: Prolog mode declarations are used to eliminate ill-formed approximations, and an admissible heuristic search is used to find the optimal rule from the candidate pool. These tricks substantially reduce learning time; the current implementation of AxA-EBL takes an average of 15.5 CPU minutes on a Sun/4 to process 120 traces from the STRIPS robot-world domain described below. (It takes about 20 minutes to solve the 120 problems using the original means-ends analysis planner.)

A-EBL is also computationally expensive, but for a different reason: to find the optimal rule in the candidate pool, each candidate must be tested against every control decision. This is very expensive if rules have a high match cost.

Standard EBL, A-EBL, and AxA-EBL share two properties which appear to be crucial for success on this learning task. First, all three algorithms can be constrained to produce rules which contain only operational features. Second, like standard EBL, both A-EBL and AxA-EBL usually learn *several* control rules which determine when a particular clause should be selected: in other words, the concept of "usefulness" for each clause can be *disjunctively* defined. The latter property is important because many Prolog programs contain clauses which are useful in several different situations. Usually, several control rules, interpreted disjunctively, will be needed to determine when such a clause is useful.

A-EBL and AxA-EBL differ in these respects from other inductive extensions to EBL, in particular mEBG and IOE [Flann and Dietterich, 1989].

4 Experimental results

4.1 Description of the domains

Experimentation has been done with these three learning algorithms in several domains. The domains are summarized in Table 1.

μLEX: A simplified version of symbolic integration using state-space search with iterative deepening. The training and test problem sets and the operator set are

[2] Experimental studies [Shavlik, 1987] suggest that this ordering is most beneficial.

[3] The size of a rule is defined to be the number of nodes in the explanation structure used to form the rule.

Table 1: Summary of domains used in experiments

Domain	Description	Search Strategy	Number of Operators	Training Set Sizes	Test Set Size
μLEX	simplified LEX	state-space	36	4,8,12,16,19	13
RW	STRIPS robot world	means-ends	10	5,15,25,45,80,120	50
BW	Nilsson blocks world	means-ends	4	5,15,25,45,80,120	100
GRID	graph search	depth-first	3	5,10,15,25	100

taken from [Keller, 1987].

RW: The STRIPS robot world of [Fikes *et al.*, 1972], with a means-ends analysis planner. Problems for the training and test set are generated by selecting a floorplan randomly from the three given in [Minton, 1988], randomly placing 3 blocks and the robot, and then taking a random walk of bounded length in state space.

BW: The blocks world of [Nilsson, 1987], with the identical means-ends analysis planner. Problems are generated by constructing an initial scene with 20 randomly placed blocks, and then taking a random walk of bounded length in state space.

GRID: Depth-bounded depth-first search of an artificial graph. The graph is a 15x15 grid, with 75 randomly placed obstacles, and 5 randomly placed "cities", which are tightly connected by a network of five "interstate highways". There are three kinds of operationally different connections in the graph, corresponding to increasing or decreasing the value of a coordinate by 1 or following an "interstate". Problems are selected by randomly picking start and end points, and asking for a path between the points.

BW and RW problems can be quite difficult, so the planner includes two resource bounds: first, a bound on the maximal length of the plan, and second, a time bound. The time bound was set at 60 seconds for these experiments: almost all of the blocks world problems and about 6/7 of the robot world problems were solvable in this time bound.

The domains are listed roughly in order of their difficulty for standard EBL techniques. For instance, symbolic integration is well suited to use of EBL for operator selection; however, unconstrained use of EBL in the blocks world domain can lead to swamping [Minton, 1988; Mooney, 1989]. Graph searching is a very difficult problem for standard EBL techniques because the match cost of learned rules is very high, and the cost of problem solving without any learned rules is low; matching the rules learned by EBL is equivalent to solving the NP-complete subgraph isomorphism problem [Minton, 1988], whereas depth-first search only requires time linear in the size of the graph.

4.2 Comparison of EBL and AxA-EBL

A series of experiments were designed to determine how the performance of a learned program varied as a function of the number of training examples used in learning. In each experiment, first a *test set*, of the size indicated in table 1, was selected. In the case of BW, RW and GRID, the test problems were randomly selected; in μLEX, the test problems were the designated test problems in [Keller, 1987] (problem sets AT and BT.) Then a *training set* was selected, again randomly for BW, RW and GRID, and from the designated training problems (problems sets A and B) for μLEX. Progressively larger subsets of the training set were then given to each learning system, again as indicated in table 1, and the control rules learned from each subset were applied to the problems in the test set. Two statistics were kept. First, the *total time* required to solve all of the test problems was recorded. The second statistic kept was the *coverage* of the set of control rules: the percentage of the test problems that the learner program solved directly, without recourse to the original program.[4] These experiments were repeated ten times, and the results were averaged. In BW, RW and GRID, each trial used a different randomly selected training set; in μLEX, each trial used a different ordering of the fixed training set.

Approximations in the domains were chosen by incrementing k until either some performance improvement occurred, or until the cost of running AxA-EBL became excessive. One of the surprises of these experiments was the expressiveness of a language of very short approximations: in all of the domains except GRID, substantial performance improvement occurred using AxA-EBL with $k \leq 2$.

The results of these experiments are summarized in table 2. The crucial column of table 2 is the last one, which shows the ratio of the time spent by the program learned by AxA-EBL in solving the problems of the test set to the time spent by the program learned by EBL. AxA-EBL is marginally slower than EBL on the μLEX domain, and substantially faster on the remaining domains; overall, the programs learned by AxA-EBL are about twice as fast as the programs learned by EBL.

The results of the experiment are given in more detail Figure 1 and Figure 2. These figures graph the average time spent by a learned program, and also the average coverage of the learned program (*i.e.*, the percentage of the time that it was *not* necessary to go back and use the original program to solve a problem) as a function of the number of training examples given.

The three learning techniques behaved essentially

[4] Recall that the program which incorporates the learned control rules may fail if not enough control rules have been learned, or if some of the control rules are incorrect. If the program fails, then the original problem solver is invoked on the problem.

μLEX Time

μLEX Coverage

GRID Time

GRID Coverage

Figure 1: Performance and Coverage Curves for μLEX and GRID

RW Time

RW Coverage

BW Time

BW Coverage

Figure 2: Performance and Coverage Curves for RW and BW

Table 2: Summary of experimental results

Domain	Average Time on Test Problems				AxA-EBL / EBL
	Nolearn	EBL	A-EBL	AxA-EBL	
μLEX	2.019	0.103	0.103	0.104	1.01
RW	9.956	2.768	0.922	1.596	0.58
BW	3.150	2.553	4.774	0.987	0.39
GRID	0.137	0.361	0.423	0.268	0.74
Average	4.320	1.447	1.555	0.739	0.51

identically on the μLEX domain. This domain is not very informative for the purpose of comparing the techniques: it is included in this paper as an additional demonstration of the generality of AxA-EBL.

In the GRID domain, AxA-EBL starts out with a very bad set of control rules — one which usually leads the problem solver on a wild goose chase across the graph — but then quickly recovers. In all ten trials, AxA-EBL converges to the same program; the programs learned by EBL and A-EBL are still getting progressively slower. Unfortunately, the program learned by AxA-EBL is still slower than the original program. This indicates that some additional utility analysis may be necessary, even when approximations are used.

In the RW and BW domains, AxA-EBL outperforms EBL by wide margins. Suprisingly, A-EBL does somewhat better than AxA-EBL in the RW domain. We interpret this result as follows. The coverage graphs show that although AxA-EBL generates control rules which have lower match cost than A-EBL,[5] AxA-EBL converges somewhat more slowly; this is probably the price of searching through a much larger space of possible control rules. In short, *AxA-EBL trades off convergence speed for lower match cost* in the planning domains. This unfortunately leads to a performance tradeoff: if matching rules is cheap and problem solving is expensive, as in the RW domain, then faster convergence will offset the poorer quality of the rules, and A-EBL will produce faster programs.

5 Related work

Use of approximations in learning control knowledge was investigated in the MetaLEX project [Keller, 1987]. The techniques used in MetaLEX, however, were specific to state-space search, and required periodic testing of programs including learned control knowledge against a benchmark. This would be prohibitively expensive in a real-world domain. Ellman [Ellman, 1988], Tadepalli [Tadepalli, 1989] and Chien [Chien, 1989] have also investigated explanation based learning of approximate theories. The search techniques described by these authors improve on Keller's by taking advantage of a partial order on approximate theories; however, they have not been applied to the problem of learning control rules.

Approximations to EBL rules have also been investigated in [Chase *et al.*, 1989] in the context of the ULS system. The approximations done in ULS are not, however, selected by an inductive learning mechanism; as a consequence, ULS is limited to conservative approximations (*e.g.*, dropping one or two conditions). ULS nonetheless has made modest improvements in planning time in the RW domain.

The use of k-bounded approximations is one way of bounding the cost of learned rules. An alternative approach is described by Tambe and Rosenbloom in [Tambe and Rosenbloom, 1989]; they describe a syntactic constraint on SOAR programs which ensures that chunks have match cost linear in their length. A corresponding constraint can be imposed on EBL rules; it is easy to see that requiring every operational predicate to have at most one solution will lead to EBL rules which can be matched in linear time. An advantage to linear-time restricted EBL is that special-purpose inductive techniques are not needed to search for correct rules; the standard incremental learning algorithm can be used.

In order to satisfy the constraint that every operational predicate has at most one solution, it is typically necessary to change the operationality predicate. In the planning domains, for instance, it is sufficient to modify the operationality predicate so that the *holds* predicate, which determines if a condition is true in a state, is marked as non-operational. Using the new operationality predicate, control rules learned by EBL are expressed in terms of constraints on elements of the list structure used to represent a state, rather than constraints on the conditions that are true or false in that state. Changing the operationality predicate in this way closely corresponds to Tambe and Rosenbloom's suggestion of replacing multi-attributes in SOAR programs with list-processing utilities.

Figure 3 shows the result of using linear-time restricted EBL (LR-EBL) in the BW and RW domain. This experiment points out a disadvantage of the tech-

[5] This is obvious in the blocks world domain, since AxA-EBL's control rules outperform A-EBL's control rules even though their coverage is usually worse. It is a little more difficult to verify in the RW domain. One way to control for the effect of coverage is to compare only sets of control rules with equal coverage; this comparison shows that AxA-EBL control rules are about 12% faster than A-EBL control rules with equivalent coverage. Another indication that the AxA-EBL control rules are faster is that the best control rules learned by AxA-EBL outperform the best control rules learned by A-EBL, requiring 1.5 seconds versus 12.9 seconds to solve the 50 problems in the test set.

BW/RW Time

BW/RW Coverage

Figure 3: Performance and Coverage Curves for BW and RW with Linear-Time Rules

nique: learned rules tend to be very specific, and hence do not often transfer to new situations. As a consequence, there is little or no performance improvement on a test set of novel problems; and so in both domains, performance improvement was far less than that attained by AxA-EBL. In the RW domain, a modest 15-20% improvement gain was achieved, whereas AxA-EBL improved performance by over 80%; in the BW domain, performance was degraded slightly, whereas AxA-EBL improved performance by 65%.[6]

6 Conclusions

The utility of a rule is its contribution to performance improvement; utility is directly proportional to the coverage of a rule and inversely proportional to the match cost. This paper introduces two new techniques for improving the utility of learned rules. The first technique is to combine EBL with *inductive learning techniques*, thus increasing the coverage of control rules; the second technique is to constrain these inductive techniques to learn *approximate control rules* that have very low match cost. These two techniques have been synthesized in the AxA-EBL algorithm. AxA-EBL has been experimentally shown to produce control rules which improve substantially over the control rules learned by standard EBL in several domains; the average improvement is about a factor of two over the domains that have been studied.

Several open problems remain. First, the techniques used are non-incremental; it would be desirable if learning could take place concurrently with problem-solving, rather than as a separate pass. Second, the current implementation of AxA-EBL must, for reasons of computational complexity, use a very small and constraining set of approximations; it would be desirable to use a larger set of approximations. Third, one cost of learning approximate control rules seems to be a slower convergence rate, relative to simpler learning systems. It would be desirable if this tradeoff could be lessened or avoided.

[6]The training and test problems used for these experiments and the experiments of Figure 2 are the same; the CPU time needed to solve the test set without learning differs because with a different operationality predicate, different domain-specific predicates can be compiled.

Acknowledgements

I am grateful to my advisor, Alex Borgida, for his advice and encouragement; to Haym Hirsh, for invaluable suggestions on experimental methodology; to Chun Liew, for comments on a draft of the paper and several useful insights; and to Susan Cohen for proofreading a late draft of this paper. The author is supported by an AT&T Fellowship.

References

[Chase *et al.*, 1989] Melissa Chase, Monte Zweben, Richard Piazza, John Burger, Paul Maglio, and Haym Hirsh. Approximating learned search control knowledge. In *Proceedings of the Sixth International*

Workshop on Machine Learning, Ithaca, New York, 1989. Morgan Kaufmann.

[Chien, 1989] Steve Chien. Using and refining simplifications: Explanation-based learning of plans in intractible domains. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Michigan, 1989. Morgan Kaufmann.

[Cohen, 1989] William W. Cohen. Abductive explanation based learning: A solution to the multiple explanation problem. Technical Report ML-TR-26, Rutgers University, 1989.

[Cohen, 1990] William W. Cohen. Learning from textbook knowledge: A case study. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, Massachusetts, 1990. MIT Press.

[Ellman, 1988] Tom Ellman. Approximate theory formation: An explanation-based approach. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, Saint Paul, Minnesota, 1988. Morgan Kaufmann.

[Fikes *et al.*, 1972] Richard Fikes, Peter Hart, and Nils Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972.

[Flann and Dietterich, 1989] Nicholas Flann and Thomas Dietterich. A study of explanation-based methods for inductive learning. *Machine Learning*, 4(2), 1989.

[Keller, 1987] Richard Keller. The role of explicit contextual knowledge in learning concepts to improve performance. Technical Report ML-TR-7, Rutgers University, 1987.

[Markovitch and Scott, 1989] Shaul Markovitch and Paul Scott. Utilization filtering: A method for reducing the inherit harmfulness of deductively learned knowledge. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Michigan, 1989. Morgan Kaufmann.

[Minton, 1988] Steven Minton. Learning effective search control knowledge: An explanation-based approach. Technical report, Carnegie-Mellon University Department of Computer Science, 1988.

[Mitchell, 1982] Tom Mitchell. Toward combining empirical and analytical methods for inferring heuristics. In *Human and Artificial Intelligence*. Lawrence Erlbaum, 1982.

[Mooney, 1989] R. J. Mooney. The effect of rule use on the utility of explanation based learning. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Michigan, 1989. Morgan Kaufmann.

[Nilsson, 1987] Nils Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann, 1987.

[Prieditis and Mostow, 1987] Armand Prieditis and Jack Mostow. PROLEARN: Towards a prolog interpreter that learns. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, Washington, 1987. Morgan Kaufmann.

[Shavlik, 1987] Jude Shavlik. Generalizing number in explanation based learning. Technical Report UILO-ENG-87-2276, Univ. of Illinois/Champaign, 1987. (PhD thesis).

[Tadepalli, 1989] Prasad Tadepalli. Lazy explanation-based learning: A solution to the intractible theory problem. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Michigan, 1989. Morgan Kaufmann.

[Tambe and Rosenbloom, 1989] Milinde Tambe and Paul Rosenbloom. Eliminating expensive chunks by restricting expressiveness. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Michigan, 1989. Morgan Kaufmann.