The Effect of Rule Use on the Utility of Explanation-Based Learning

Raymond Mooney
Department of Computer Sciences
University of Texas
Austin, TX 78712

Abstract

The *utility problem* in explanation-based learning concerns the ability of learned rules or plans to actually improve the performance of a problem solving system. Previous research on this problem has focused on the amount, content, or form of learned information. This paper examines the effect of the *use* of learned information on performance. Experiments and informal analysis show that unconstrained use of learned rules eventually leads to degraded performance. However, constraining the use of learned rules helps avoid the negative effect of learning and lead to overall performance improvement. Search strategy is also shown to have a substantial effect on the contribution of learning to performance by affecting the manner in which learned rules arc used. These effects help explain why previous experiments have obtained a variety of different results concerning the impact of explanation-based learning on performance.

1. Introduction

The *utility problem* in explanation-based learning concerns the ability of learned rules or plans to actually improve the performance of a problem solving system [Minton87J. The concern that too much learned information might eventually degrade problem solving performance was initially expressed in the work on learning macro-operators in STRIPS [Fikes72]. Since then, several studies have found that the unrestricted learning and use of rules or macro-operators can degrade rather than improve overall performance [Markovitch88, Minton85, Minton88a]. This decrease in performance is due to time wasted trying to apply learned rules or macro-operators in situations in which they are incapable of efficiently solving the problem at hand. However, other experiments have revealed only overall performance improvement from explanation-based learning even though no attempt was made to limit the number or form of learned rules [0'Rorke87, Shavlik88|.

* This research was supported by the Department of Computer Sciences and University Research Institute of the University of Texas at Austin.

This paper presents experimental results and informal analysis that may help explain this apparent contradiction in the existing empirical data. The proposed explanation relies on the fact that how learned rules are actually *used* in problem solving can greatly affect their tendency to improve or degrade overall performance. This focus on the use of acquired knowledge differs from previous attempts at addressing the utility problem, which have focused on forgetting or selectively retaining learned knowledge or on translating it into a more efficient form [Markovitch88, Minton85, Minton88b].

The paper is organized as follows. Section 2 describes the performance system under investigation, a Horn-clause theorem prover that learns macro-rules (compositions of existing rules). Section 3 presents the basic experimental methodology used to examine the effect of learning on performance and describes the problem sets used in the experiments. Section 4 compares two approaches to using learned rules. It shows that unrestricted use of macro-rules can degrade overall performance; however, limiting chaining on learned rules can help alleviate the negative impact of learning and lead to overall performance improvement. Section 5 shows how different search techniques such as depth-first and breadth-first can greatly affect the impact of learning on performance by changing how learned rules are used. Section 6 concludes and presents some problems for future research.

2. The Performance System

Unfortunately, comparing different explanation-based learning systems is difficult because they generally employ different underlying performance elements and knowledge representation schemes. However, a performance element that has been used in a number of systems is a backward-chaining, depth-first, Horn-clause theorem prover (like Prolog) [Hirsh87, Kedar-Cabelli87, Mooney88, Pricditis87], a general, well-understood, and popular performance system in AI.

The system used in the current experiments is a version of EGGS [Mooney86, Mooney88] that includes a Horn-clause theorem prover as a performance component. When EGGS finds a proof for a query by chaining together several existing rules, the proof is generalized using standard explanation-based techniques [DeJong86, Mitchell86] and compiled into a macro-rule. In the system's list of available rules for each predicate, learned rules are placed before the original rules in the domain theory so that using learned rules is preferred to solving a problem from scratch. However, new rules are placed at the end of the set of learned rules so that rules learned from problems encountered earlier in system's experience are tried first (since they probably represent more typical problems). This approach has been empirically found to work better than adding new rules to the beginning of the list of learned rules [Shavlik88].

The Horn clauses initially given to EGGS as a domain theory are simply assumed to be a set of declarative facts rather than a well-ordered Prolog program which is guaranteed to terminate (as in PROLEARN [Prieditis87] and PROLOG-EBG [Kedar-Cabelli87]). In order to prevent depth-first search from getting lost (possibly in an infinite recursion), the theorem prover is given a depth bound that limits the number of rules it is allowed to chain together. Another feature of the system is the ability to heuristically order the antecedents of learned rules in an attempt to make them more efficient. Antecedents that have a greater percentage of variables that are likely to be already bound (by occurring in the consequent or a previous antecedent) are positioned earlier in the list.

3. Experimental Methodology

The effect of learning on performance is empirically examined by having both learning and non-learning systems solve a sequence of problems in a particular domain and comparing their performance. The learning system learns rules from its solutions which are available for solving subsequent problems in the sequence. Since search is limited by a depth-bound, not all problems will necessarily be solved. Consequently, performance is compared by measuring both the number of problems solved and the total effort expended on the complete problem set (in terms of both CPU time and search nodes generated). Time spent on the generalization and learning process itself is generally minimal (less than 0.1% of the overall time) and is not reported. This simple methodology seems cleaner and fairer than a complicated methodology involving stages in which learning is turned on and off on different selected sets of problems (e.g. [Minton88a]). All experiments were run on a Texas Instruments Explorer II with 12 MB of main memory.

Two problem domains are used to test EGGS and compare its performance with learning to its performance without learning. One problem domain is proving theorems in propositional logic. The problem set used in this domain consists of 52 problems from *Principia Mathematica* [Whiteheadl3] that were first used in experiments with the Logic Theorist (LT) [Newell63]. Experiments on this problem set with an explanation-based learning version of LT (EBL-LT) [0'Rorke87] showed overall performance improvement due to learning even though no attempt was made to limit the number of learned rules. For use in EGGS, all *implies* in the theorems were rewritten in terms of *not* (-i) and *or* (V)) and the system was given the following domain theory for constructing proofs.

1) $\mathbf{Theorem}(\neg(x \lor x) \lor x)$
2) $\mathbf{Theorem}(\neg x \lor (y \lor x))$
3) $\mathbf{Theorem}(\neg(x \lor y) \lor (y \lor x))$
4) $\mathbf{Theorem}(\neg(x \lor (y \lor z)) \lor (y \lor (x \lor z)))$
5) $\mathbf{Theorem}(\neg(\neg x \lor y) \lor (\neg(z \lor x) \lor (z \lor y)))$
6) $\mathbf{Theorem}(\neg y \lor x) \land \mathbf{Theorem}(y) \rightarrow \mathbf{Theorem}(x)$
7) $\mathbf{Theorem}(\neg x \lor y) \land \mathbf{Theorem}(\neg y \lor z) \rightarrow$
$\mathbf{Theorem}(\neg x \lor z)$

The other problem domain used to test the system is blocks-world planning. Standard blocks-world problems can be solved with a Horn-clause theorem prover by stating them as theorems in situation calculus. The blocks-world theory used for the current experiments is a simplified version of the rules used to test the BAGGER system [Shavlik88]. Since blocks-world problems are quite difficult to solve using a theorem prover and axioms in situation calculus, the problem sets are restricted to building stacks of one to three blocks given a randomly generated scene with five blocks. The procedure described in [Shavlik88] is used to randomly generate problems and initial states. Complete problem sets are produced by randomly generating a sequence of 30 stacking problems.

Although a Horn-clause theorem prover is a weak problem solver and can perform quite poorly in certain domains (e.g. planning domains), this does not affect its suitability as a performance element for a learning system. The goal of learning is to improve performance and a weak problem solver simply has more room for improvement. The important factor is not absolute performance but the relative performance of learning and non-learning systems.

4. Full Versus Limited Use of Learned Rules

One approach to using learned rules is to treat them exactly like rules in the original domain theory. As a result, learned rules can be combined arbitrarily with domain rules and with other learned rules. This approach may seem promising since it allows for the greatest possible use of learned information. Unfortunately, it can quickly lead to a great deal of effort being spent trying to use a learned rule before eventually giving up and trying another path in the search tree. Minton has referred to the fact that, in the general case, matching the antecedents of a learned rule to a set of facts is NP-Complete [Minton88bj. Using arbitrary theorem proving to prove them is, of course, even worse (i.e. undecidable).

Consider the following case in which unrestricted use of learned rules is applied to blocks-world problems. First, given the situation shown in Figure Ia, the system is asked to build a three block tower on Table-2. It solves the problem (transfer(A,Table-2), transfer(B,A), transfer(CB)) and learns a rule for building a tower of three blocks on a table by inverting an existing tower of three blocks on another table. Next, it is given the same goal with the initial state shown in Figure Ib. Since the goal matches the consequent of the rule that was just learned, the system backward-chains on this rule and tries to achieve its antecedents. This requires constructing a tower of three blocks that can then be inverted. The system will waste a great deal of effort trying to achieve this subgoal before eventually finding it is impossible and attempting to solve the problem from

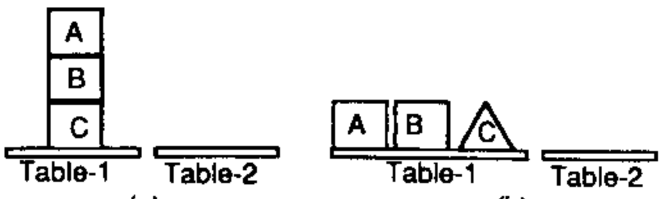

Figure 1: Blocks-World Problems

Table 1. Results on Logic Theorems			
Depth/Mode	**CPU sec**	**Nodes**	**Correctness**
Depth 3			
No-learn	348.4	33,920	30/52 (58%)
Full-use	459.0	33,662	51/52 (98%)
Limited-use	335.3	31,840	30/52 (58%)
Depth 4			
No-learn	6,033.5	480,857	48/52 (92%)
Full-use	13,752.8	721,790	52/52 (100%)
Limited-use	4,472.2	341,147	48/52 (92%)

scratch. The result is a significant decrease in performance due to learning. Since the antecedents of a learned rule frequently involve more constraints than the original goal, using arbitrary problem solving to achieve them is generally much more trouble than it is worth.

In logic theorem proving, since learned rules are simply generalized theorems (facts) rather than rules with antecedents, one might suspect that such problems do not arise. However, matching a subgoal to a learned fact may require that variables in the subgoal be bound to particular formulae. These bindings may cause problems in subsequent subgoals and cause the system to backtrack on the decision to use the learned fact (after possibly having wasted a lot of time trying to use it). For example, imagine the system has learned the fact: Theorem(-iX V (x V y)). Given the problem Theorem(PV-TP), the system might use rule 6 above to generate the subgoals: Theorem(¬x V (P V ¬P)) and Theorem(x). If the first subgoal is matched to the learned theorem, then the second subgoal becomes: Theorem(P). The system may waste a great deal of time trying to prove this impossible subgoal which was generated due to the attempt to use the learned fact. Consequently, even the use of learned facts that do not have any antecedents may cause considerable wasted effort.

In order to prevent the possibility of spending too much time trying to use learned rules, limiting the use of what is learned is perhaps a better alternative. An extreme approach is to only allow a learned rule to be used if it completely solves the problem at hand without being combined with other rules. In other words, the problem solver is not allowed to backchain on the antecedents of learned rules nor is it allowed to use a learned rule (or fact) to solve a subgoal, since both of these actions may result in a great deal of wasted effort if the learned information is not actually relevant. This is a very strict schema-based or script-based approach; a learned plan either solves a problem completely or it is simply not used.

Table 1 presents empirical data (CPU time and search nodes generated) on various versions of the EGGS system proving 52 logic theorems from the *Principia* with depth bounds of 3 and 4. The conditions shown are: no learning, learning with unrestricted use of learned facts (Full-use), and learning with no chaining with learned facts (Limited-use). Full-use learning takes more time than the non-learning version; however, it solves more problems. This is an ambiguous result with respect to the effectiveness of this method. Some clear advantage of full-use learning is shown by the fact that the full-use version with a depth-bound of three expends less than a tenth of the effort of the non-learning version with a depth-bound of four and yet solves three more problems. However, the results are hard to interpret since both search effort *and* correctness vary. On the other hand, limited-use learning is incapable of solving more problems than no-learn since, unlike full-use learning, it does not use learned rules to search even deeper into the search tree defined by the original domain theory. However, learned rules can be used to more quickly handle problems solvable by the original theory within the depth bound. The data in Table 1 shows that limited-use learning does increase speed relative to no learning without changing correctness and that its relative advantage increases with the size of the search space (i.e. the depth bound).

Correctness can be completely controlled by using only the problems that can be solved by the original domain theory within the given depth-bound. In this case, only the time to solve the problems varies and can be used as the basis of a fair comparison. Table 2 presents data in which correctness has been controlled in this manner. These results clearly show the negative effects of full-use learning and the advantage of limited-use learning. For a depth bound of three, full-use is 1.1 times as slow as no-learn while limited-use is 1.4 times as fast. For a depth bound of four, full-use is 2.4 times as slow as no-learn while limited-use is 2.0 times as fast. The fact that the speedup actually increases with the size of the search space for limited-use while it decreases for full-use underlines the conclusion that limited-use learning is best.

Table 2. Controlled Results on Logic Theorems			
Depth/Mode	**CPU sec**	**Nodes**	**Correctness**
Depth 3			
No-learn	78.5	7,441	30/30
Full-use	88.8	7,225	30/30
Limited-use	56.3	5,361	30/30
Depth 4			
No-learn	3,329.3	257,566	48/48
Full-use	7,876.8	475,516	48/48
Limited-use	1,642.7	117,856	48/48

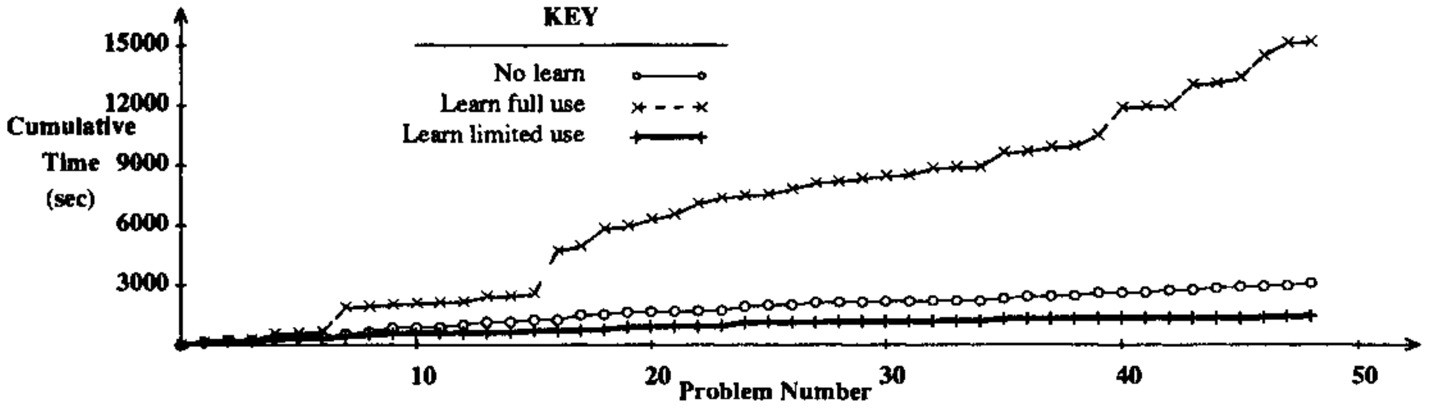

Figure 2. Cumulative Time Curves for Logic Theorems

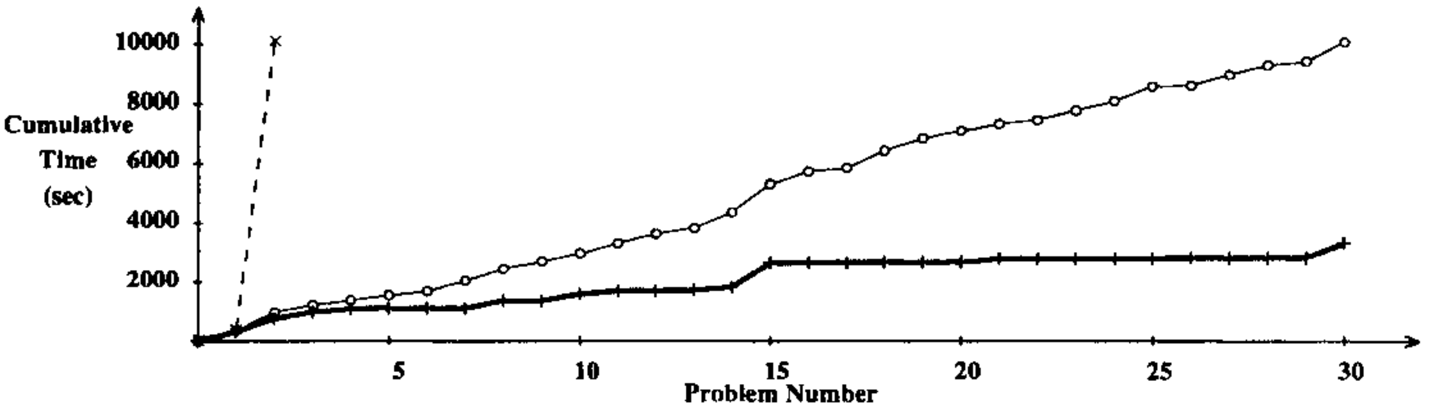

Figure 3. Cumulative Time Curves for Blocks World Problems

An issue that has been raised regarding the LT problems is that, as a sequence of problems from the *Principia,* they are explicitly ordered so that later problems build on earlier ones [Minton88b]. In order to address this issue and present a more detailed picture of the performance of the various methods, Figure 2 shows cumulative time curves for random orderings of the problems. Cumulative time is the the total time spent solving all problems up to that point. The curves are for the 48 problems solvable by all versions with a depth bound of four. Each curve is the average of five random orderings of the problems. Not surprisingly, the full-use learning system performs even poorer on randomly ordered problem sets. For the full set, full-use takes on average 4.9 times as long as no-learn while limited-use takes 0.46 as long. One would not expect problem ordering to have an effect on limited-use learning since if the rule learned from problem A is capable of completely solving problem B, then the same rule can be learned from problem B and applied to problem A if the order of the two problems is reversed.

Figure 3 shows cumulative time curves for the blocks-world domain. These curves are the average of five randomly generated sets of 30 stacking problems. All problems can theoretically be solved within the depth-bound of 7 supplied to all systems. However, due to time constraints, the full-use learning version was stopped after it had spent about 15 hours on a problem set. In this time, it never got past the third or fourth problem in the set. Backward chaining on the antecedents of a learned rule consistently caused the system to get lost in a huge search space. In order to increase the applicability of learned rules in the limited-use case, a minimal amount of chaining is allowed with learned rules. Specifically, a few efficient rules for concluding various static properties of blocks can be used to prove the antecedents of learned rules. Unlike the abysmal performance of full-use learning, on average the limited-use version solves all 30 problems in one third the time of the no learning version.[1]

Therefore, avoiding or severely limiting chaining with learned rules is clearly preferable to the unrestricted use of learned rules. The explanation-based learning experiments on blocks-worlds problems presented in |Shavlik88] also allowed only very limited chaining on learned rules. The current experiments indicate that the limited use of learned rules was crucial to the overall beneficial effects of learning obtained in these experiments.

5. The Effect of Search Methods

Although the previous section presents data that unrestricted acquisition and use of learned rules degrades performance on the *Principia* problems, experiments with EBL-LT on this problem set showed only positive effects of

[1] The average number of search nodes explored to solve all 30 problems is 1,184,512 for no-learn and 426,556 for limited-use learning.

learning [0'Rorke87]. Limiting the use of learned rules in the manner discussed in the previous section does not explain this apparent contradiction since EBL-LT treated learned theorems just like the initial axioms in the domain theory.

There are a number of differences between EBL-LT and EGGS that could account for the inconsistent results. For example, unlike EGGS, EBL-LT is only capable of constructing linear proof trees and it is allowed to learn from unproven theorems. However, a difference between the two systems that greatly affects the use of learned rules is the fact that EBL-LT uses breadth-first search while EGGS (like Prolog) uses depth-first. When a learned rule is tried with a depth-first theorem prover, the system exhaustively tries to prove its antecedents and may waste a great deal of time if they cannot be proven. A breadth-first prover, on the other hand, pursues all paths "in parallel" and does not spend an inordinate amount of time first trying to exhaust the use of learned rules before eventually resorting to the initial domain theory (like depth-first does). Like the approach described in the previous section, breadth-first search limits the use of learned rules compared to depth-first search.

A breadth-first Horn-clause theorem prover is used to test the hypothesis that switching from depth-first to breadth-first search can change the effect of learning on performance. Table 3 presents data on the performance of a breadth-first prover given the LT domain theory and problem set used in the previous experiments. In order to limit run-time, each system is only allowed to generate a certain number of search nodes for each problem. If this limit is exceeded, the system gives up and goes on to the next problem. Results are presented for limits of 1000, 3000, and 5000 nodes per problem. Unlike the results reported for depth-first search, the learning system shows significant improvement in performance compared to the non-learning version. The learning system solves about twice as many problems in 1/2 to 1/3 the time and its relative performance improves as the search space gets larger. Comparing the data in Tables 1 and 3 shows that breadth-first search with learning can solve more problems with less search than depth-first with limited-use learning. However, breadth-first takes more time per search node because of the extra memory operations required. It is also important to notice that without learning, the performance of breadth-tirst search in terms of both run-time and search is significantly worse than depth-first since it requires more memory and cannot take advantage of rule-ordering. The overall poor performance of breadth-first search on problems requiring relatively deep proofs prevented the possibility of running it on blocks-world problems.

Table 3. Breadth-First Search on Logic Theorems

Nodes/Mode	CPU sec	Nodes	Correctness
1000 limit			
No-learn	1,091.9	33,970	19/52 (37%)
Learn	564.9	16,600	40/52 (77%)
3000 limit			
No-learn	4,533.0	99,980	19/52 (37%)
Learn	1,298.2	28,588	45/52 (87%)
5000 limit			
No-learn	8,047.7	164,136	22/52 (42%)
Learn	1,977.7	33,096	47/52 (90%)

6. Conclusions and Future Research

The data and informal analysis presented in this paper demonstrate that the use of learned information can have a large impact on the utility of explanation-based learning. Previous research on the utility problem has focused on the amount or form of learned information rather than on its use in problem solving. Unconstrained use of learned rules can clearly lead to degraded performance. However, appropriately limiting the use of learned information can help avoid the negative effects of learning and help insure overall performance improvement. Various properties of the performance element can have a significant impact on how learned information is used and consequently on the effect of learning on performance. For example, allowing chaining on learned rules in a depth-first system leads to degraded performance while allowing chaining on learned rules in a breadth-first system does not. Therefore, it is unwise to generalize utility results on one performance system to different performance systems.

One important area for future research is finding a workable compromise between the extremes of full use and no-chaining on learned rules. A good compromise will allow greater use of learned knowledge while avoiding the problems with unrestricted use. Allowing a limited amount of chaining with learned rules (as in the experiments with the blocks-world) is one promising approach. In domains like logic theorem proving where learned information takes the form of facts or theorems instead of rules with antecedents, methods are needed for being able to effectively use these facts as lemmas. Sometimes a subgoal can be satisfied by a learned theorem without the risk of creating other difficult subgoals. A problem only arises if matching the subgoal to a learned theorem creates bindings for variables in the subgoal that may not satisfy the remaining antecedents (see section 4). Therefore, a good approach may be to allow a subgoal to be proven by matching a learned fact only if the match does not bind any variables occurring in the subgoal. This would allow the use of lemmas without the risk of having to eventually backtrack on this decision. Initial experiments with this approach on the LT problems have given promising results.

As shown by the above experiments, both depth-first and breadth-first search strategies have advantages and disadvantages with respect to being a successful search strategy for a system which learns macros. Depth-first iterative-deepening [Korf85] exhibits some of the advantages of both depth-first and breadth-first and may be a good strategy for a learning system. Some additional specific issues which need further study are the effect of the ordering of learned rules[2] and their antecedents and the effect of learning rules for subgoals and subsequences of solutions. Another important research area concerns the effect different domains and problem solving architectures have on the relative performance of various approaches such as learning heuristic control rules versus learning macro-operators [Minton88a]. Further investigations will hopefully result in even better ways for insuring that explanation-based learning improves overall system performance.

Acknowledgements

I would like to thank Jude Shavlik and Paul O'Rorke for helpful discussions on the utility of explanation-based learning and for contributing to the code and/or data used in the experiments.

References

[DeJong86] G. F. DeJong and R. J. Mooney, "Explanation-Based Learning: An Alternative View," *Machine Learning 1, 2* (1986), pp. 145-176.

[Fikes72] R. E. Fikes, P. E. Hart and N. J. Nilsson, "Learning and Executing Generalized Robot Plans," *Artificial Intelligence 3,* 4 (1972), pp. 251-288.

[Hirsh871 H. Hirsh, "Explanation-Based Generalization in a Logic-Programming Environment," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence,* Milan, Italy, August 1987, pp. 221-227.

[Kedar-Cabelli87] S. T. Kedar-Cabelli and L. T. McCarty, "Explanation-Based Generalization as Resolution Theorem Proving," *Proceedings of the 1987 International Machine Learning Workshop,* Irvine, CA, June 1987, pp. 383-389.

[Korf 85] R. E. Korf, "Depth-first Iterative Deepening: An Optimal Admissible Tree Search," *Artificial Intelligence 27,* 1 (1985), pp. 97-109.

[Markovitch88] S. Markovitch and P. D. Scott, 'The Role of Forgetting in Learning," *Proceedings of the Fifth International Conference on Machine Learning,* Ann Arbor MI, June 1988, pp. 459-465.

[Minton85] S. N. Minton, "Selectively Generalizing Plans for Problem-Solving," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence,* Los Angeles, CA, August 1985, pp. 596-599.

[Minton87] S. Minton, J. G. Carbonell, O. Etzioni, C. A. Knoblock and D. R. Kuokka, "Acquiring Effective Search Control Rules: Explanation-Based Learning in the PRODIGY System," *Proceedings of the 1987 International Machine Learning Workshop,* Irvine, CA, June 1987, pp. 122-133.

[Minton88a] S. Minton, "Quantitative Results Concerning the Utility of Explanation-Based Learning," *Proceedings of the National Conference on Artificial Intelligence,* St. Paul, MN, August 1988, pp. 564-569.

[Minton88b] S. Minton, "Learning Effective Search Control Knowledge: An Explanation-based Approach," Ph.D. Thesis, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1988.

[Mitchell86] T. M. Mitchell, R. Keller and S. Kedar-Cabelli, "Explanation-Based Generalization: A Unifying View," *Machine Learning I,* 1 (1986), pp. 47-80.

[Mooney86] R. J. Mooney and S. W. Bennett, "A Domain Independent Explanation-Based Generalizer," *Proceedings of the National Conference on Artificial Intelligence,* Philadelphia, PA, August 1986, pp. 551-555. (A longer updated version appears as Technical Report UILU-ENG-86-2216, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign)

[Mooney88] R. J. Mooney, "A General Explanation-Based Learning Mechanism and its Application to Narrative Understanding," Ph.D. Thesis, Department of Computer Science, University of Illinois, Urbana, IL, January 1988. (Also appears as Technical Report UILU-ENG-87-2269, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign.)

[Newell63] A. Newell, J. C. Shaw and H. A. Simon, "Empirical Explorations with the Logic Theory Machine: A Case Study in Heuristics," in *Computers and Thought,* E. Feigenbaum and J. Feldman (ed.), McGraw-Hill, New York, NY, 1963.

[0'Rorke87] P. V. O'Rorke, "LT Revisited: Experimental Results of Applying Explanation-Based Learning to the Logic of Principia Mathematica," *Proceedings of the 1987 International Machine Learning Workshop,* Irvine, CA, June 1987, pp. 148-159.

[Prieditis87] A. E. Prieditis and J. Mostow, "PROLEARN: Towards a Prolog Interpreter that Learns," *Proceedings of the National Conference on Artificial Intelligence,* Seattle, WA, July 1987, pp. 494-498.

[Shavlik88] J. W. Shavlik, "Generalizing the Structure of Explanations in Explanation-Based Learning ," Ph.D. Thesis, Department of Computer Science, University of Illinois, Urbana, IL, January 1988. (Also appears as Technical Report UILU-ENG-87-2276, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign.)

[Whiteheadl3] A. N. Whitehead and B. Russell, *Principia Mathematica,* Cambridge University Press, Cambridge, England, 1913.

[2] Some experiments on the ordering of learned rules are presented in [Shavlik88].