LT Revisited: Explanation-Based Learning and the Logic of Principia Mathematica[1]

PAUL O'RORKE (ORORKE@ICS.UCI.EDU)

Department of Information & Computer Science
University of California, Irvine 92717

(Technical Report Number 88-29)
November 14, 1988

Last Revised: July, 1989
Printed: July 10, 1989

ABSTRACT

This paper describes an explanation-based learning (EBL) system based on a version of Newell, Shaw and Simon's LOGIC-THEORIST (LT). Results of applying this system to propositional calculus problems from *Principia Mathematica* are compared with results of applying several other versions of the same performance element to these problems. The primary goal of this study is to characterize and analyze differences between not learning, rote learning (LT's original learning method), and EBL. Another aim is to provide a characterization of the performance of a simple problem solver in the context of the *Principia* problems, in the hope that these problems can be used as a benchmark for testing improved learning methods, just as problems like chess and the eight puzzle have been used as benchmarks in research on search methods.

[1] To appear in *Machine Learning.* Preliminary versions of some of the results presented here were originally published in the Proceedings of the Fourth International Conference on Machine Learning. This paper is based on work supported in part by the National Science Foundation under grant NSF IST 83-17889, by a Cognitive Science/AI Fellowship from the University of Illinois, by a McDonnell-Douglas University External Relations contract, and by an Irvine Faculty Fellowship from the University of California, Irvine Academic Senate Committee on Research.

TABLE OF CONTENTS

1. Introduction

There is widespread agreement that explanations can form the basis for powerful learning strategies. A new subfield of machine learning called *explanation-based learning* (EBL for short) has begun to exploit this idea (see, e.g., [2, 4, 16]). To date, however, there have been few experiments on EBL involving more than a few examples.[1] Experiments involving the application of complete EBL systems to large sets of problems from challenging domains are needed in order to demonstrate claims about the advantages of EBL and to discover the limitations of proposed EBL methods.

This paper reports on an EBL experiment in a logical domain. The experiment involves complete machine learning systems built around one of the earliest, simplest AI problem-solvers: Newell, Shaw, and Simon's LOGIC-THEORIST (LT). Results of a comparison of the performances of three versions of LT (corresponding to *non-learning*, *rote learning*, and *explanation-based learning*) on a large number of problems are given.

2. The Domain: Principia Mathematica

The domain of this experiment is the propositional calculus of Alfred North Whitehead and Bertrand Russell's *Principia Mathematica* [31]. The calculus deals with a set of expressions or well-formed propositional formulae built upon a set of variables that are supposed to stand for arbitrary propositions such as "Agrippina killed Claudius." Complex propositions are built up by using connectives such as NOT ($\bar{}$), AND ($\wedge$), OR ($\vee$), and IMPLIES ($\supset$).

Valid propositions are called theorems. It turns out that the theorems of the propositional calculus can be built up from an initial set of theorems called axioms. An example of an axiom is Principia 1.2 $P \vee P \supset P$. This axiom, called the principle of tautology, states that if either P is true or P is true, then P is true. An example of a theorem is Principia 2.01 $(P \supset \bar{P}) \supset \bar{P}$. This states that if P implies its own falsehood then P is false. It is called the "principle of the reductio ad absurdum" or reduction to absurdity.

Theorems are derived from the axioms by applying rules of inference. *Detachment*, or modus ponens, is a rule of inference that allows one to infer B if one has A and $A \supset B$. Another rule allows substitution of any expression for a variable in any theorem. Other rules allow replacement of definitions for defined connectives (e.g., $\bar{P} \vee Q$ for $P \supset Q$). A derivation of a theorem is called a proof. A proof of a desired theorem can be written as a sequence ending in that theorem where each step in the sequence is either an axiom or else follows from previous steps by a rule of inference. Alternatively, the proof can be depicted as a tree whose root is the desired theorem where each node in the tree is either an axiom or follows from its offspring by an inference rule.

[1] Minton's work on MORRIS [13] and on PRODIGY [14, 15] are noteworthy exceptions.

Complete lists of the axioms and problems used in our experiments are provided in appendix 2. Note that Whitehead and Russell use a labeling scheme (originally due to Peano) involving numbers with integer and decimal parts. The integer part indicates the chapter of Principia containing the axiom or theorem. The axioms and theorems used in the experiments are from the first, second, and third chapters of part one of *Principia.* The decimal indicates the ordering within a chapter and was used to make insertion easy, avoiding relabelling. Note that, under the decimal notation used to sequence the theorems in Principia, 2.11 comes before 2.2 (even though 11 is greater than 2).

One of the advantages of propositional logic is an advantage of mathematics in general, namely *high levels of abstraction and generality.* When one learns arithmetic one is learning something that will apply to an extremely wide range of tasks, from the mundane (e.g., choosing best buys at the grocery store) to the esoteric (e.g., performing basic calculations in the astrophysics of black holes). Similarly, propositional logic is an abstract, general theory of deduction. One can apply logic in infinitely many definite, specific domains by specifying propositions that provide details about particular subjects. Any learning that takes place at the logical level can thus be applied in many domains.

Another advantage of propositional logic is that the number of problem solving operators required for complete coverage of a large set of problems is small. This feature facilitates the execution of large scale experiments because it is not necessary to hand code new operators in order to introduce new examples.

The particular logical system of *Principia Mathematica* has the added advantage that a very simple and natural theorem prover exists, which can be used as the problem-solving performance element in learning experiments.

3. The Performance Element: LT, the Logic Theorist

According to Donald Loveland's *Automated Theorem-Proving: A Quarter-Century Review* [12], Newell, Shaw, and Simon's *The Logic Theory Machine* [20] (hereafter referred to as LT) was the first publication reporting results of a computer program that proved theorems. LT solved problems (proved theorems) in the propositional calculus of *Principia Mathematica.* This section is a brief description of how a simple version of LT works, in the terminology of schema-based problem solving.

3.1. Schemata

In this context, a *schema* is merely a collection of related descriptions. Each LT schema has three descriptions (in this case, descriptions are well-formed formulae of propositional calculus) related by a logical dependency. The dependency states that one description, (the *consequence* of the schema) is true if both of the others (the *antecedents*) are true. LT uses only two schemata: a *detachment* schema and a *chaining* schema (see Figure 1). The detachment schema captures *modus ponens* and is comprised of X, Y, and $X \supset Y$, where Y depends on $X \supset Y$ and X. The chaining schema captures the *transitivity of implication*, and is comprised of $X \supset Y$, $Y \supset Z$, and

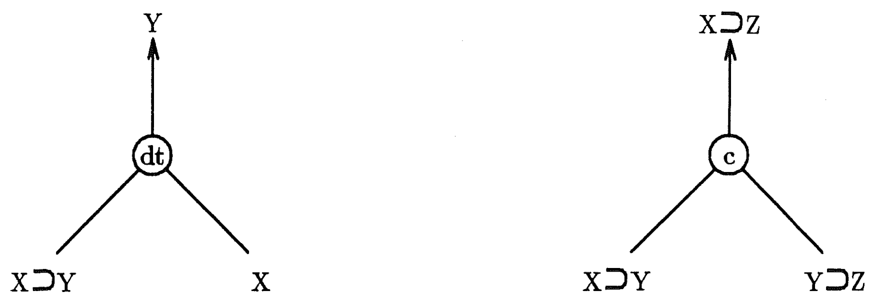

Figure 1. The Detachment and Chaining Schemata

X⊃Z, where X⊃Z is true if both X⊃Y and Y⊃Z are true.

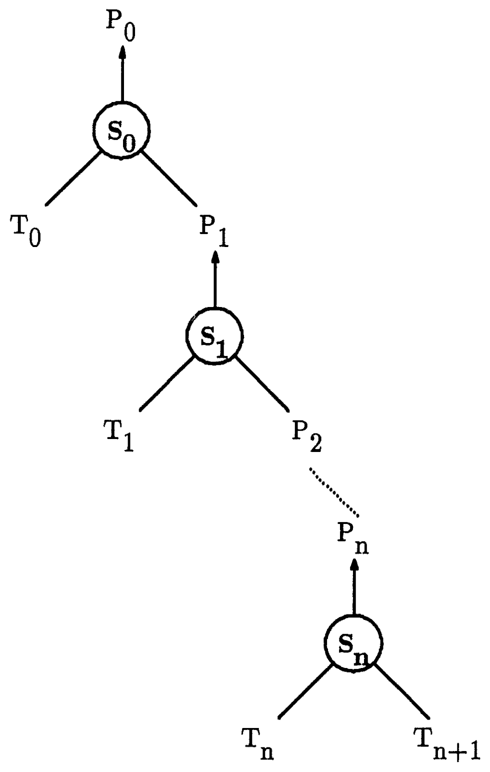

Figure 2. The Structure of LT's Proofs

3.2. Problem Solving

LT's mission is to construct a proof of a given conjecture, building it out of detachment and chaining schemata, axioms, and previously proven theorems. LT's proofs are always linear trees similar to the one shown in Figure 2. The leaves (labelled $T_0 \ldots T_{n+1}$) are all taken from the set of known theorems. Each schema (labelled S_i) corresponds to a step in the proof and can be considered to reduce a problem (P_i) to a simpler one (P_{i+1}) as in Figure 3.

Given a problem, LT first considers whether the problem is an instance of a known theorem. If not, the problem is matched against the conclusion of a schema such as detachment. If the match succeeds, the resulting substitution is applied to the conclusion and to both antecedents of the schema. The result of applying the substitution to one of the antecedents is then matched against known theorems until a match succeeds. The resulting substitution is applied to the other antecedent, and it becomes a new subproblem.

Figure 3 shows the details. If a match between a known theorem and the current problem P_i succeeds, the problem is considered to be solved immediately by substitution. Otherwise, LT chooses a schema S_i and a known theorem T_i, such that the problem is an instance of the schema's conclusion C_i and the theorem T_i is compatible with one of the schema's antecedents A_i. If θ is the most general unifier (MGU) [8, 25] of C_i with P_i and of A_i with T_i, the result of applying θ to the other

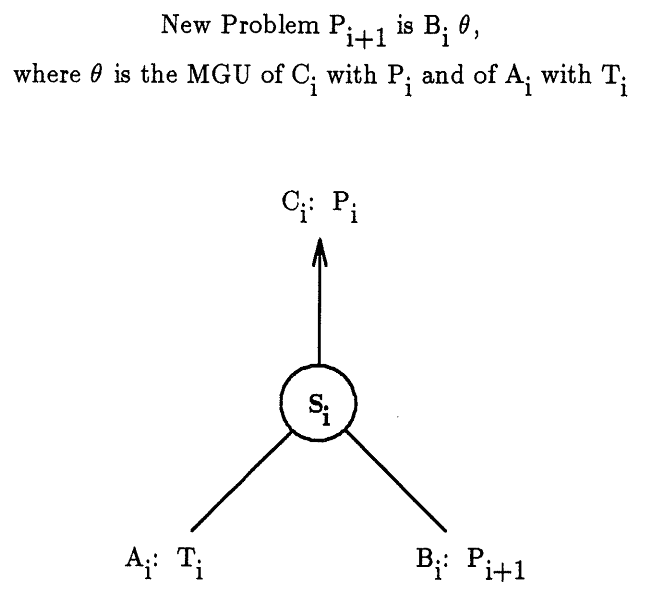
New Problem P_{i+1} is $B_i\,\theta$,
where θ is the MGU of C_i with P_i and of A_i with T_i
C_i: P_i
S_i
A_i: T_i
B_i: P_{i+1}

Figure 3. One Step in the Construction of a Proof

antecedent B_i becomes the new problem P_{i+1}.

3.2.1. Substitution and Matching

In attempting to use known theorems to solve problems, our version of LT uses matching (one-way unification [8]) in order to determine whether a given problem is an instance of a known theorem. If it is, the matcher returns a substitution that specifies the variable bindings that map the known theorem into the problem.

In order to avoid solving a special case of a problem instead of solving the problem in its full generality, the matcher treats variables in the initial problem as if they were constants. For this reason, while the matcher is able to specialize a variable P in a theorem to a pattern $\bar{Q}$ in a problem, it is unable to bind the expression $\bar{Q}$ in a theorem to the expression P in a problem. This is in contrast with full (two-way) unification, which could return substitutions instantiating variables from *both* the known theorem and the problem expressions.

It is important to note that the matcher and unifier incorporated in our version of LT go a bit beyond simple unification. The matcher has some information about the meaning of the logical connectives, but other information is reserved for the logic of *Principia*. The matcher decodes defined symbols such as AND and IMPLIES, so it can recognize that $P \supset Q$ should match $\bar{p} \vee q$. However, the matcher does not recognize that variables can be replaced by their negations without altering the validity of a theorem, and it fails to match P and $\bar{\bar{P}}$ because information about double negation is unavailable to it. The logic itself, rather than the matcher, deals with most logical relationships. In conjunction with the detachment inference rule, the fact that double negations can be added or stripped away is encoded in Principia-2.12 $P \supset \bar{\bar{P}}$ and

New Problem P_{i+1} is X θ
where θ is the MGU of Y with P_i and of X$\supset$Y with T_i

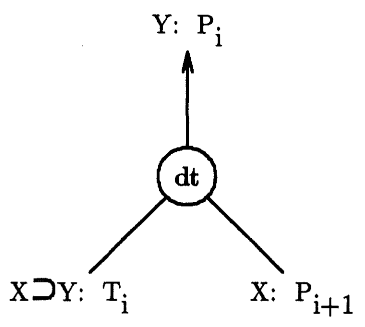

Figure 4. A Detachment Step

Principia-2.14 $\bar{\bar{P}} \supset P$. The fact that P is logically equivalent to $\bar{\bar{P}}$ is encoded in Principia-4.13.

In the analysis of the initial experiments reported later in this paper, it became apparent that restrictions on matching that were included in Newell, Shaw and Simon's LT and in all our initial versions of LT had a significant effect on the initial experimental results. These restrictions required that matches associated with the detachment and chaining operators only be done on known theorems in the form of implications, so we call them "IMPLIES restrictions."

3.2.2. Detachment

LT extends a proof by means of a detachment schema in only one way: given a problem, LT looks for a known implication[2] whose conclusion subsumes the problem. If such a theorem is found, its antecedent becomes the new problem (see Figure 4). This amounts to applying a detachment *operator* corresponding to modus ponens run backward.

3.2.3. Chaining

The chaining schema can be used in two ways to extend the proof, so in effect LT has two chaining operators corresponding to transitivity of implication inference rules. Both operators attempt to prove an implication of the form $X \supset Z$. *Chaining forward*, (Figure 5), involves trying to show that an immediate consequence of X implies Z. In contrast, *chaining backward* (Figure 6) tries to show that X implies an immediate antecedent of Z.[3]

3.3. An Example: Principia-2.17

For a concrete example, consider the operation of LT on Principia-2.17: $(\bar{Q} \supset \bar{P}) \supset (P \supset Q)$. This is the *only if* part of an important tautology called *contraposition* [5], which states that the contrapositive $\bar{Q} \supset \bar{P}$ holds if and only if the implication $P \supset Q$ holds.

[2] To be more specific, LT scans the known theorems looking for theorems that have IMPLIES as the top level connective — matching their conclusions against the problem. This strategy contains IMPLIES restrictions carried over from the original LT. The alternative is to match the problem against a variable Y representing the conclusion of the detachment schema. This match yields a substitution θ for Y. Next, the *entire* list of known theorems, including patterns whose top level connective is not IMPLIES, is scanned — matching known theorems against $X \supset Y\theta$. Since the matcher decodes defined symbols such as IMPLIES, it is possible for known theorems that are not expressed as implications to match the implication associated with the detachment schema, so the IMPLIES restrictions inhibit some applications of the detachment operator that would otherwise occur.

[3] The IMPLIES restrictions associated with chaining are as follows. In the original LT (and in our initial versions) chaining was not invoked unless the problem to be solved had IMPLIES as its top level connective. Furthermore, known theorems used in chaining were required to have IMPLIES as their top level connective.

New Problem P_{i+1} is $Y \supset Z\ \theta$,
where θ is the MGU of $X \supset Y$ with T_i and of $X \supset Z$ with P_i

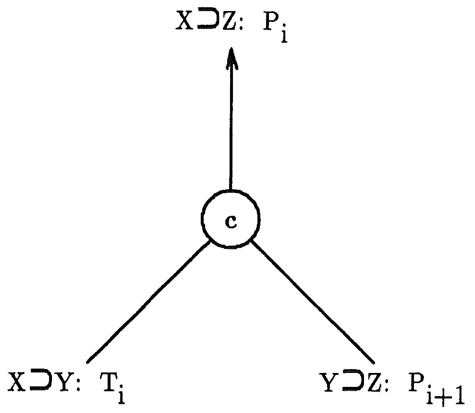

Figure 5. A Chaining Forward Step

New Problem P_{i+1} is $X \supset Y\ \theta$,
where θ is the MGU of $Y \supset Z$ with T_i and of $X \supset Z$ with P_i

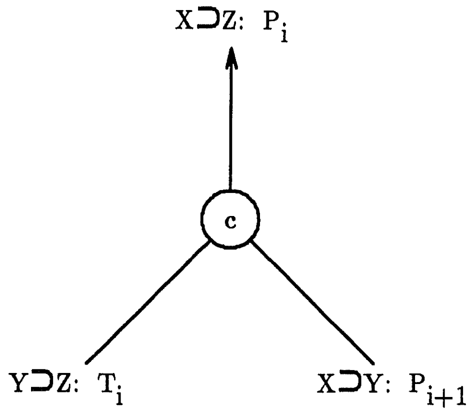

Figure 6. A Chaining Backward Step

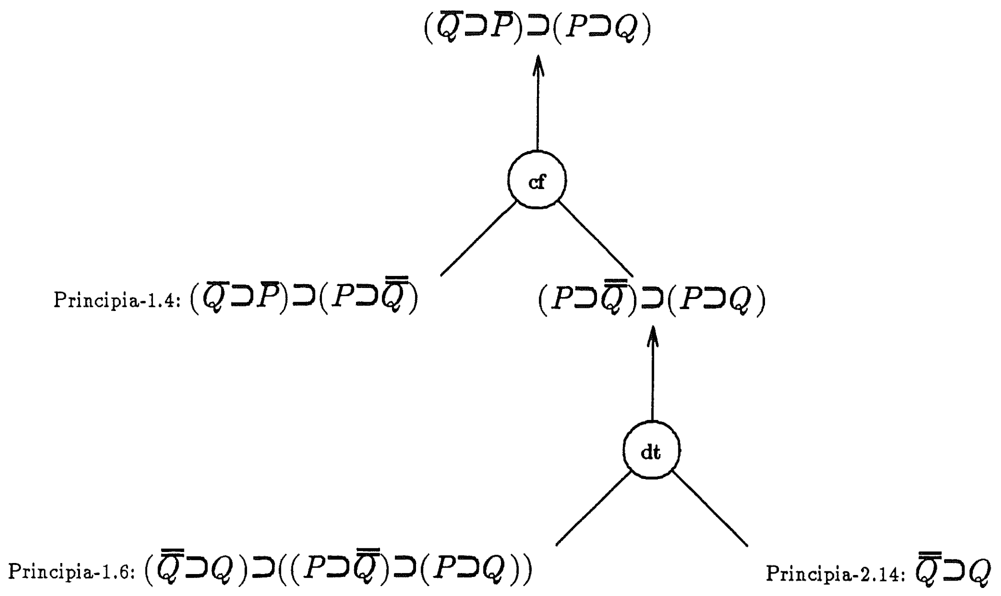
$(\bar{Q}\supset\bar{P})\supset(P\supset Q)$
cf
Principia-1.4: $(\bar{Q}\supset\bar{P})\supset(P\supset\bar{\bar{Q}})$
$(P\supset\bar{\bar{Q}})\supset(P\supset Q)$
dt
Principia-1.6: $(\bar{\bar{Q}}\supset Q)\supset((P\supset\bar{\bar{Q}})\supset(P\supset Q))$
Principia-2.14: $\bar{\bar{Q}}\supset Q$

Figure 7. The Proof of Principia-2.17

LT proves Principia-2.17 by chaining forward (see Figure 7): it proves that the contrapositive $\bar{Q}\supset\bar{P}$ implies an intermediate result that eventually leads to the original implication $P\supset Q$. The first link of the chain is supplied by an instance of the axiom Principia-1.4: $(A \vee B)\supset(B \vee A)$. With A bound to $\bar{\bar{Q}}$, and B bound to $\bar{P}$ this yields $(\bar{\bar{Q}} \vee \bar{P})\supset(\bar{P} \vee \bar{\bar{Q}})$. By the definition of the implication connective, $\bar{P} \vee \bar{\bar{Q}}$ is the same as $P\supset\bar{\bar{Q}}$; this serves as the intermediate step in the chain from $\bar{Q}\supset\bar{P}$ to $P\supset Q$. Chaining forward transforms the initial problem into the problem of proving $(P\supset\bar{\bar{Q}})\supset(P\supset Q)$.

Next, the detachment operator is used. The detachment operation amounts to performing modus ponens backwards, so a theorem is needed whose conclusion subsumes the current subproblem. The conclusion of axiom Principia-1.6: $(A\supset B)\supset((C \vee A)\supset(C \vee B))$ meets this requirement. With C bound to $\bar{P}$, A bound to $\bar{\bar{Q}}$, and B bound to Q, the axiom becomes $(\bar{\bar{Q}}\supset Q)\supset((P\supset\bar{\bar{Q}})\supset(P\supset Q))$. Detachment on Principia-1.6 transforms the problem of proving $(P\supset\bar{\bar{Q}})\supset(P\supset Q)$ into the problem of proving $\bar{\bar{Q}}\supset Q$. Assuming Principia-2.14 is known, $\bar{\bar{Q}}\supset Q$ is the final subproblem because it is an instance of Principia-2.14.

The example just discussed may leave the impression that no search is involved in finding proofs. Of course this is not correct. Theorem provers are notorious for searches involving high branching factors, and LT is no exception. LT performs breadth first search: when it uses its operators and known theorems to expand the unsolved subproblem in a partial proof, it generates a number of new partial proofs each with one new subproblem. These subproblems are checked to see whether they are instances of known solutions and if not, they are added to the end of a queue of incomplete proofs to be worked on later.

4. The Learning Methods

The goal of this paper is to present results of computational experiments aimed at evaluating the performance of several versions of the problem solver described in the previous section. In particular, the problem solver and the problems discussed previously are used to compare two learning strategies — rote and explanation-based learning. The following sections describe the details of these learning strategies.

4.1. LT Plus Rote Learning

In *Human Problem Solving* [19], Newell and Simon describe experiments on LT augmented with simple forms of learning. They mention that perhaps the simplest learning method is to make LT's list of known theorems variable, modifying LT so as to add new theorems onto the list, so that "... in proving a sequence of theorems, LT would gradually become more capable" [19].

When problems encountered are simply added to memory one is left with the distinct impression that no "understanding" or "thinking" is taking place during learning. For this reason, one can consider this sort of learning strategy to be a form of *rote learning*. Our version of LT incorporating this form of learning is called ROTE LT, or just ROTE.

There are a number of different variations of rote learning, many of them corresponding to different answers to questions about the storage of learned results. In general, our initial approach was to duplicate as closely as possible the original learning method used by Newell, Shaw, and Simon so that we could validate our initial implementation by comparing overlapping results. As we gained experience with the particular version of ROTE used in both the original and the present studies we noticed that this version has two major features that have some impact on experimental results. These features correspond to ROTE's answers to the questions "what is to be stored?" and "how are learned results to be stored?".

What is to be stored? The particular version of ROTE learning used here allows ROTE to use all prior theorems in its attempts on each new theorem, whether it had succeeded in proving them or not. In this strategy, since all problems encountered are stored, each new problem is reduced to one that has been seen before.

This strategy appears to be appropriate for the *Principia* problem sequence because this sequence has been carefully constructed so that all of the problems in it are solvable.[4] In addition, the problems appear to form a sequence in order of

[4] We ran our version of the problem set through a truth table validity tester in order to check our translations of Whitehead and Russell's arcane, obsolete notation. This validity tester verified that the problems are all theorems, so the problems are all solvable in the sense that each theorem is provable.

increasing difficulty. The solutions to later problems often depend on solutions to earlier problems. In other words, the strategy of using all prior problems in attempts on each new problem appears to be appropriate here because the problems present LT with a controlled learning situation such as one might encounter in *learning from a teacher*. One can imagine relatively uncontrolled learning situations where this strategy would be less appropriate, especially if previous problems were not solved. For example, in *learning by discovery* conjectures are made that may turn out to be false and problems are posed that may not be solvable even in principle. In such situations, it might be more appropriate to try a variation of ROTE in which only solved problems are stored. In this alternative strategy, each new problem is reduced to one that has been solved before.

How are learned results to be stored? The ROTE learning version of LT adds new theorems to the *end* of a list of known theorems. In order to make comparisons simpler, all our learning systems (including the EBL version of LT described in the next section) add learned theorems to the end of a list. This will be seen to have important consequences for the performance of the learning systems (see, e.g., section 9 *On the Effects of Adding Instances in Rote Learning*).

4.2. LT Plus Explanation-Based Learning

One problem with rote learning systems is that they tend to be very sensitive to the particular form of the examples they observe. Extraneous details of specific examples tend to be retained as well as essential facts. This can cause failure to recognize that a solution to one problem can also be used for another problem because the problems differ in trivial ways. One would prefer to forget about the extraneous details and to remember only the essentials of an example. The basic idea of *explanation-based learning* is that one can do this by constructing and using explanations. When one conducts an analysis and constructs an explanation of how and why a solution solves one particular problem, one is better prepared to see the general class of problems that the method can be successfully applied to.

LT can be augmented with EBL by focusing on explanations instead of focusing on problems. The explanations are LT's proofs, considered as structures built out of schemata. The EBL version of LT ignores the specific theorem that gave rise to a proof. Each proof is considered in its full generality in order to compute the most general theorem that can be concluded on the basis of the proof. The generalized conclusion of an LT proof can be defined recursively as follows. The generalized conclusion of an elementary proof (a proof that states that the desired conclusion is an instance of a known theorem) is simply the known theorem itself. The generalized conclusion of a complex proof is the result of simultaneously unifying one antecedent with a known theorem and the other antecedent with the generalized conclusion of the subproof (see Figure 8).

In particular, the generalized conclusion of a proof by detachment is the result of simultaneously unifying one of its antecedents ($X \supset Y$) with some known theorem (T_i) and the other antecedent X with the generalized conclusion (G_{i-1}) of the subproof.

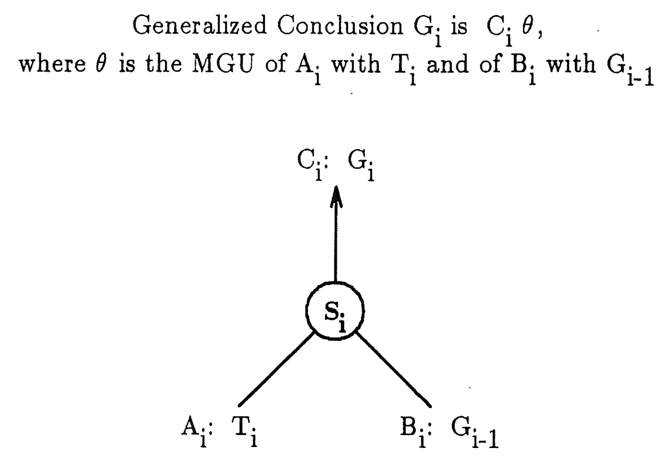
Generalized Conclusion G_i is $C_i\,\theta$,
where θ is the MGU of A_i with T_i and of B_i with G_{i-1}
C_i: G_i
S_i
A_i: T_i
B_i: G_{i-1}

Figure 8. The Generalized Conclusion of One Step of a Proof

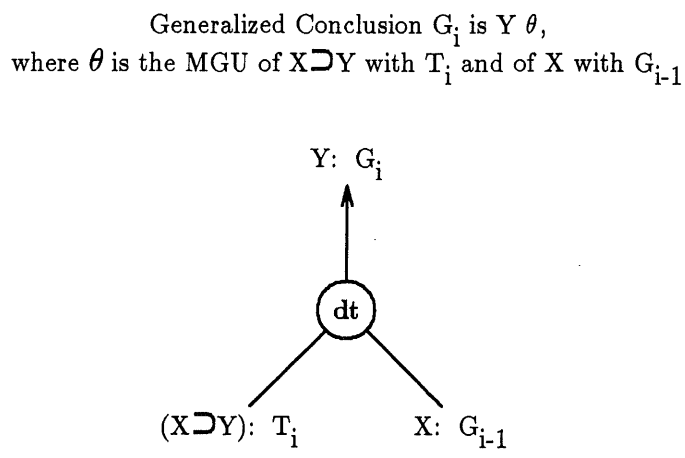
Generalized Conclusion G_i is $Y\,\theta$,
where θ is the MGU of $X \supset Y$ with T_i and of X with G_{i-1}
Y: G_i
dt
$(X \supset Y)$: T_i
X: G_{i-1}

Figure 9. The Generalized Conclusion of a Detachment Proof

The simultaneous unification results in a substitution which is then applied to the conclusion Y of the detachment schema to produce the generalized conclusion of the detachment proof. (See Figure 9.) In forward chaining, $X \supset Y$ is unified with a known theorem while $Y \supset Z$ is unified with the generalized subconclusion to obtain a substitution that is applied to $X \supset Z$. In backward chaining, the roles of $X \supset Y$ and $Y \supset Z$ are reversed.

For a concrete example, reconsider the proof of Principia-2.17, $(\overline{Q}\supset\overline{P})\supset(P\supset Q)$. Recall that the proof involved two steps: chaining forward on Principia-1.4 reduced the original problem to a subproblem that was solved by detachment on Principia-1.6 and Principia-2.14. Computing the generalized conclusion also involves two steps: first the generalized conclusion of the detachment subproof must be computed, then the generalized conclusion of the overall chaining forward proof can be determined.

The generalized conclusion of the detachment step is computed by unifying Principia-2.14 $\overline{\overline{D}}\supset D$ with the antecedent $A\supset B$ of Principia-1.6, $(A\supset B)\supset((C\vee A)\supset(C\vee B))$. This unification binds A to $\overline{\overline{D}}$, and B to D. The conclusion of Principia-1.6, in the context of this substitution, becomes the generalized conclusion of the detachment step, namely $(C\vee\overline{\overline{D}})\supset(C\vee D)$. The generalized conclusion of the entire proof is computed by chaining forward on Principia-1.4 $(E\vee F)\supset(F\vee E)$ and the generalized conclusion of the subproof $(C\vee\overline{\overline{D}})\supset(C\vee D)$.

The conclusion of Principia-1.4 plays the role of the middleman, and is unified with the antecedent of the generalized conclusion of the detachment. That is, $F\vee E$ is unified with $C\vee\overline{\overline{D}}$: F is bound to C while E is bound to $\overline{\overline{D}}$. The generalized conclusion of the chaining forward step kicks out the middleman and goes directly from $E\vee F$, the antecedent of Principia-1.4, to $C\vee D$, the conclusion of the generalized conclusion of the detachment step, *in light of the unifications in force.* Substituting

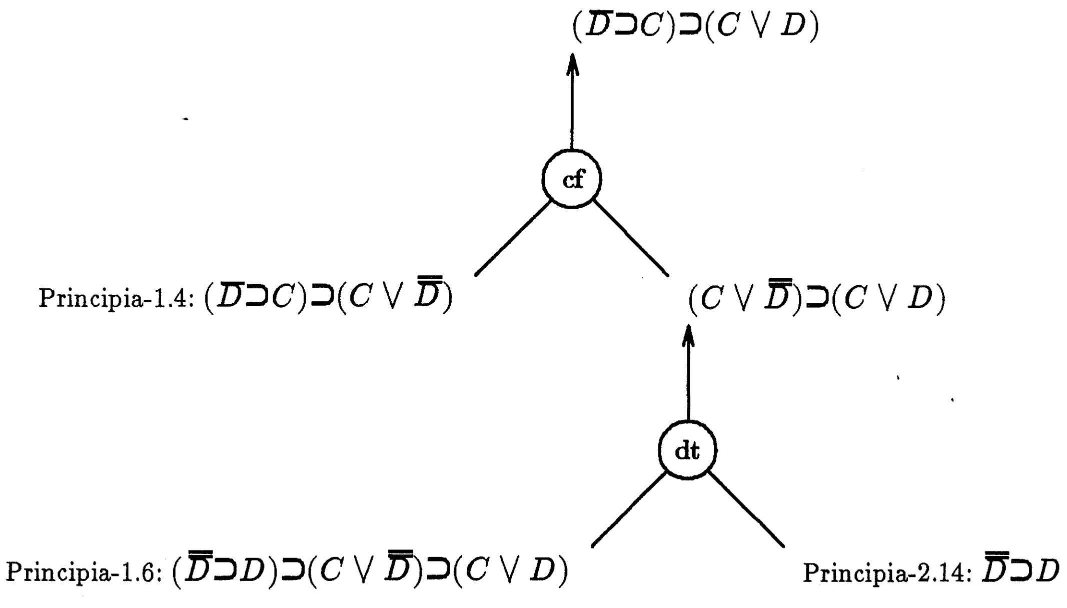
$(\overline{D}\supset C)\supset(C\vee D)$
cf
Principia-1.4: $(\overline{D}\supset C)\supset(C\vee\overline{\overline{D}})$
$(C\vee\overline{\overline{D}})\supset(C\vee D)$
dt
Principia-1.6: $(\overline{\overline{D}}\supset D)\supset(C\vee\overline{\overline{D}})\supset(C\vee D)$
Principia-2.14: $\overline{\overline{D}}\supset D$

Figure 10. The Generalized Conclusion of the Proof of Principia-2.17

C for F and $\overline{D}$ for E in $(F \vee E) \supset (C \vee D)$, the generalized overall conclusion is $(\overline{D} \vee C) \supset (C \vee D)$. By the definition of implication, this can also be seen as $(\overline{D} \supset C) \supset (C \vee D)$ as shown in Figure 10.

For a discussion of differences in the generality of learning in EBL versus rote learning please see subsection 5.3 on *comparing the generality of learned solutions* in the section on *Methodology*

5. Methodology

The experiments reported here involve the application of three different versions of LT to propositional logic problems from *Principia Mathematica*. All three versions of LT start with the same axioms (the same initial set of known theorems). However, the first (*non-learning*) version of LT is not allowed to learn from its successes or failures. No new theorems are added to the list of known theorems. The non-learning LT attempts to solve each new problem by reducing it to one of the original axioms. The second version is allowed a form of *rote learning*: problems are added to the list of known theorems whether they are solved or not. The rote learning LT attempts to solve new problems by reducing them to problems it has seen before. The third version augments the basic LT by the simple form of explanation-based learning described earlier. Of course EBL is useless when the search for a solution (i.e. proof) fails, so rote learning is resorted to in this case. However, when a novel theorem is proved, the generalized conclusion of the proof is added to the list of known theorems.[5]

The main questions asked are:

- What are the number and quality of the solutions found?
- What is the nature of the search for solutions?
- What is the quality of the learning?

The following subsections give details of measurements taken in an attempt to answer these questions. The measurements are intended to make the questions more concrete and well defined so that they can be answered by experiment.

5.1. Evaluating the Results of Problem Solving

For each problem in *Principia*, a record is made concerning whether each version of LT solves or fails to solve the problem in a limited search. Total numbers of problems solved and not solved are computed; methods that solve more problems are judged to be better. Detailed analysis is done in case one method fails to solve a problem solved by another method contrary to expectations.

[5] It is important to note that the same basic problem solving machinery (LT) is being used in each version. LT is admittedly primitive by modern standards of problem solving and theorem proving but our main interest is in differences in performance between no learning, rote learning and EBL systems, not in the particular performance element. All the learning and non-learning systems are handicapped by LT's lack of sophistication. In particular, all systems use the same matcher. Simplicity of the shared performance element is an advantage in debugging and analysis.

If a proof of a theorem is found, a measure of the quality of the proof is recorded. Short (shallow) proofs are considered to be higher quality than long (deep) proofs. Thus, the quality of a proof is measured by computing the depth of the proof. When a problem is an instance of a known theorem, the proof is by substitution and is considered to be of depth zero. Proofs requiring one application of detachment or chaining are considered to have depth one, and so on.

5.2. Characterizing the Search for a Solution

In this section, we describe search measures used in this study to evaluate the effectiveness of knowledge acquired by different learning strategies. The measures are intended to address the issue of whether the knowledge added by learning reduces or increases search.

LT does breadth first search, maintaining a queue of untried problems. Counts are maintained of the number of problems generated and attempted. Before a newly generated subproblem is added to the queue, it is checked to see whether it is an instance of a known theorem. Problems are dequeued and attempted by applying LT's operators: first the detachment operator is applied, followed by chaining forward and then chaining backward. Each of these operators may produce new subproblems which may be added to the queue of untried problems. Search is restricted by limiting the number of problems attempted.

Whether the problem is solved or not, the total number of subproblems generated and the number of problems attempted in each search for a solution are recorded and used to compute the *average branching factor* for that problem. This is a simple ratio — namely, the number of subproblems generated divided by the number of problems attempted. This average branching factor measure reflects work done in exploring many search paths, not just the one that ultimately led to a solution.

One of our aims in using these measures is to avoid any possible dependence of our results on the particular implementation platform we are using, partly because we want to be free to run experiments on different classes of machines. The experiments discussed in this paper have been run on Xerox Lisp Machines, various Suns and other Unix machines such as Sequents. In fact, the experiments have been carried out with programs written in different dialects of LISP and PROLOG. Using machine and language independent measures of performance has the practical advantage that it facilitates comparisons of results: old experiments need not be re-run when new experiments are done using new implementations.

More importantly, we distrust alternative measures such as CPU time because we believe they are highly dependent on details of the implementation that have little to do with the methods under study. It is not just that the absolute values of measures such as CPU time depends on the particular machine; we are more concerned that differences in machines may manifest themselves in different relative orderings of the values obtained for machine dependent measures. Different implementation platforms often have different strengths and weaknesses. We want our measures of performance to focus on the relative strengths and weaknesses of different learning methods and to be immune to differences in hardware. We also want our measures to be independent of software such as operating systems, programming languages, interpreters, and compilers.

The numbers of subproblems generated and attempted can be thought of as machine-independent, "absolute" measures of the amount of search done by LT in solving or attempting to solve a particular problem. The number of problems attempted includes the original problem and any subproblems taken off the breadth-first search queue. When a problem is taken off the queue, operators are tried in an effort to generate new subproblems. If a subproblem is not an instance of a known solution it is placed at the end of the queue to be attempted later. The number of problems generated is generally larger (sometimes much larger) than the number attempted; typically many problems remain on the queue when a problem is solved or search is abandoned.

The average branching factor can be thought of as a "relative" measure of the amount of search done in attempting to solve a given problem. Since it is the ratio of the number of subproblems generated divided by the number attempted, a larger branching factor indicates that a given method produces more new subproblems for each subproblem attempted. It is important to note, however, that one search method can easily generate and attempt many more subproblems than another search method applied to the same problem, while still yielding the same average branching factor. For example, if method A generates 10 problems in 1 attempt and method B generates 100 problems in 10 attempts, they both have the same average branching factor: 10/1 = 100/10 = 10. In fact, the absolute measures of search can be much higher, while the relative measure is lower. Method C might have a branching factor of 5 because it generates 500 subproblems in attempting 100. In absolute terms, method C has done more work than methods A and B, but since the average branching factor is a ratio, it is lower for method C.

Since one method may be superior with respect to the relative measure (average branching factor) yet inferior to another method with respect to the absolute measures (goals attempted and subgoals generated), or vice versa, in general both the absolute and the relative measures of "amount of search" are important. All of these measures are taken and the results are plotted and analyzed in each of our experiments. However, it turns out that in the experiments reported on here, when one method is superior in that it generates fewer subproblems per problem attempted, this method is also superior in the absolute sense; it also attempts fewer problems and generates fewer subproblems. In the interests of economy of presentation, we have chosen to present only graphs of average branching factors in order to reduce the amount of data presented in this paper. The average branching factor is used rather than the alternative search measures because it has the advantage that it combines information from the other measures (it is a ratio of the subproblems generated and the number of problems attempted). In presenting only the relative measure, great care was taken to ensure that this measure is not misleading. In particular, graphs of each of the measures taken in each experiment were inspected to ensure that relationships that hold between the average branching factors of competing methods also hold for the corresponding absolute measures.

5.3. Comparing the Generality of Learned Solutions

In addition to measuring the quality of search and the quality of the solutions found, measurements of the quality of learned results are taken. Generality is taken to be an indication of quality, although it is really an empirical question whether more general learning leads to higher performance.

Some measures used for making comparisons between rote and explanation-based learning have been discussed previously (see the previous subsection (5.2) on *characterizing the search for a solution*). These measures are indirect in the sense that they measure problem solving performance to gauge the effects of learning. The differences between ROTE and EBL are measured more directly by recording and comparing the things that are learned. The theorems learned by EBL degenerate to the input theorem in cases when no proof is found. When a non-trivial proof (i.e., a proof with non-zero depth) is found, the generalized conclusion is computed and the generality of the result is compared to the generality of the corresponding result of rote learning.

A purely syntactic measure of generality is used for the comparison. Please see [24] for a fully detailed description of this measure.

In brief, generality is measured by using one-way matches between the theorems learned by EBL and the theorems learned by ROTE. The theorems are considered as terms, and we focus on the syntax of these terms. The one-way matches determine whether the theorem learned by ROTE is an instance of the theorem learned by EBL and vice versa. If a one-way match returns a substitution showing how to bind variables in term E so as to get term R, then we say R is an instance of E, and E is *as general as* R. A priori, one can see that when EBL works at all, it provides theorems that are at least as general as the original problems. *The original problems are always instances of the generalized conclusions of their proofs.* So if R is the result of ROTE learning and E is the corresponding result of EBL, a one-way match will always succeed, yielding a substitution showing how to specialize E by binding its variables so as to produce R.

The question is whether the substitution mapping the result of EBL into the result of ROTE simply changes the names of the variables. If both one-way matches comparing two terms succeed, then the terms are said to be *variants*. If they are variants, one differs from the other only because it has different names for the other's variables. In this case they are instances of each other and so they are equally general. If they are not variants, the theorem learned by EBL is strictly more general than the corresponding theorem learned by ROTE and the substitution list computed by the successful one-way match (call it θ) shows how to specialize the EBL result to get the one learned by ROTE.

Note that the measure of relative generality used here is not quantitative in the sense that numbers are not assigned to generality. Instead the measure specifies a partial ordering on terms. As will be seen in examples given later, a qualitative understanding of the relative generality of two terms can be gained by looking at difference substitutions mapping one term into another. For example, when three variables in a term learned by EBL are replaced by a single variable in the corresponding term learned by ROTE, it is clear that three things which could vary independently collapse to a single variable with some loss of generality. Note also that the measure is syntactic in the sense that it ignores the semantics of the functions and predicates involved. For example, the fact that P can be replaced by $\overline{P}$ everywhere in a theorem without changing its validity is ignored, as are other logical relationships such as the fact that P and $\overline{\overline{P}}$ are logically equivalent.

As an example, consider the results of rote and explanation-based learning on Principia-2.17:

$$\text{Principia-2.17: } (\overline{Q}\supset\overline{P})\supset(P\supset Q)$$
$$\text{Generalized Conclusion: } (\overline{D}\supset C)\supset(C \vee D)$$

The ROTE LT simply adds Principia-2.17 to its list of known theorems. The EBL LT computes the generalized conclusion shown above and adds it. A one-way match shows that these two learned results are not equally general. Matching yields the substitution $\theta = \{D/Q, C/\overline{P}\}$. The generalized conclusion is strictly more general than the problem because the problem contains an extraneous negation. The NOT in $\overline{P}$ is not really necessary. A detailed comparison of the generality of some results learned by ROTE and EBL is presented in subsection 8.2 on *quality of learning.*

Increases in the generality of learned results make a difference in future performance. Recall that when LT attempts to use known theorems in solving problems, it does one-way matches, treating the variables in the problem as if they were constants in order to avoid solving a special case of a problem instead of solving the problem in its full generality.

For an artificial example of how differences between EBL and ROTE can make a difference in future performance, imagine that $(\overline{q}\supset p)\supset(p \vee q)$ is given as a new problem, after Principia-2.17 has been solved. The EBL LT would immediately recognize it as an instance of the generalized conclusion above. But a one-way match between this new problem and Principia-2.17 would fail, because the p in the problem cannot be specialized to $\overline{P}$. As a result, EBL would solve the problem immediately, while ROTE might be forced to regenerate the proof of Principia-2.17. For uncontrived examples of improvements in performance due to increased generality of learned results, please see appendix 2 on *examples of improvements in performance due to ebl.* This appendix gives actual examples taken from the results of experiments described in the following sections.

6. Limited Search With the IMPLIES Restrictions

The main goal of all our experiments is to find and explain differences in problem solving performance between non-learning (NL), rote learning (ROTE), and explanation-based learning (EBL) versions of LT. It was known from the work of Newell, Shaw, and Simon that rote learning improves LT's performance dramatically over not learning. Before EBL experiments were conducted, one researcher (a well-known partisan of empirical approaches to learning) conjectured that generalization and performance would not improve in going from ROTE to EBL versions of LT on the *Principia* problems because these problems are so general to begin with.[6] EBL advocates expected some improvement in performance. The only question for advocates of EBL was: *how much* would performance improve?

The experiments reported in this section involve severely limited search. To be exact, the number of subproblems LT is allowed to attempt in its effort to solve each Principia problem is limited to 15.[7] Limited search experiments are interesting because people do not seem to search much; when they fail to find a solution fairly quickly in problem solving, they tend to give up and go on. Another (stronger) reason for our interest in limited search is that advantages of one method over another may be magnified. For example, if one method requires more search than another to find a solution to a particular problem, it may not find a solution at all when the search is limited, so the difference in the percentage of problems solved may increase as search is limited. The quality of the learned results and the search performance on later problems will also be affected.

6.1. Results on the Number and Quality of the Solutions

LT solves far more problems with learning than without, and the learning versions are roughly comparable in this respect. Of the 92 problems from Chapters 1-3 of *Principia* attempted, the NL system solves 22 problems (22/92 = 24%), including one problem not solved by ROTE (problem 31, Principia 2.41).[8] ROTE solves 70 problems (76%), including 49 problems not solved by NL. EBL solves everything that ROTE solves and more (73 problems in all, or 79%). (EBL picks up problem 31 and thus also solves everything NL solves as well.)

Turning to the quality of the solutions found by LT, recall that quality is measured in terms of the depth of the solution. Depth zero solutions involve recognizing that a problem is an instance of a known theorem; these solutions are considered the best. Depth one solutions involve one application of a schema such as detachment, depth two solutions require two schemata, etc.

[6] Pat Langley, personal communication at the Seventh Conference of the Cognitive Science Society, Irvine, California, August 1985.

[7] The number 15 was chosen because this small number severely restricts search, but no special reason existed for choosing 15 over, say, 10 or 20.

[8] An analysis shows that ROTE fails to solve 2.31 in limited search because it misses an early proof due to the IMPLIES restrictions. This, and several other anomalies in the results of this section motivate the experiment reported in the next section. In the next section, it will be shown that when the IMPLIES restrictions are lifted, this and several other odd results disappear.

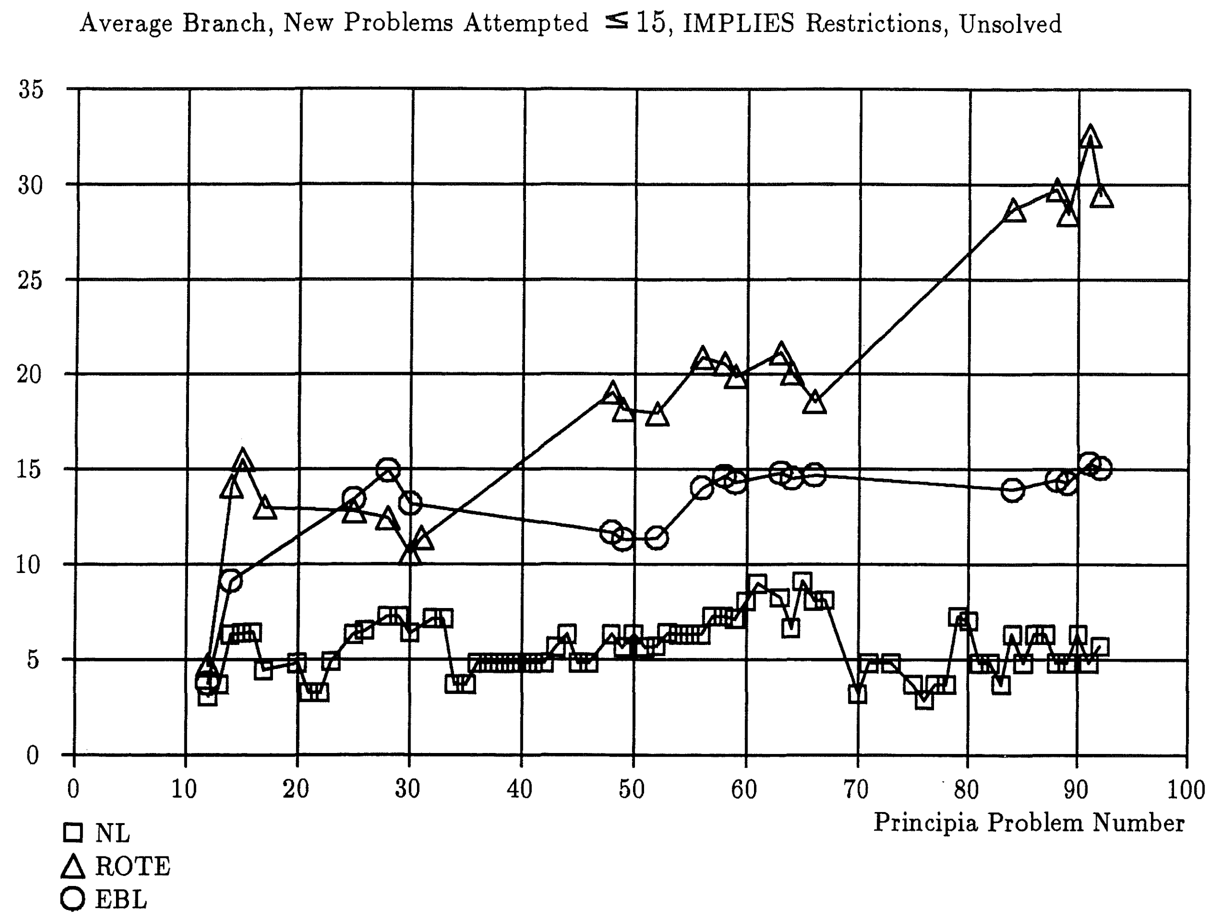

Figure 11. Limited Search Performance on Unsolved Problems

With the single exception of problem 31, the proofs discovered by the ROTE LT are always at least as short as the proofs discovered by the non-learning LT, and often shorter. The ROTE and EBL proofs are of comparable quality, the same proofs being found in most cases. Sometimes (viz Principia-2.49, 2.56, and 2.8) EBL produces shorter proofs. EBL's proofs of these theorems are of lengths 0, 1, and 0, while ROTE's are of length 1, 2, and 1, respectively. However, in other cases (namely Principia-2.68, 3.24, and 3.41) the ROTE proofs are shorter. EBL's solutions are of length 2, 3, and 2, respectively, while ROTE's are of length 1. In other words, the differences in quality between ROTE and EBL are mixed. They often get the same proof, but sometimes

ROTE finds a shorter one, sometimes vice versa.[9]

The average depth of the 22 proofs found by NL is 26/22 or approximately 1.18. The average depth of the 70 proofs found by ROTE is 48/70 or about 0.69. The average depth of the 73 proofs found by EBL is 54/73 or about 0.74. Overall, ROTE's superior solutions give it a slight advantage in solution quality over EBL, if we ignore the problems solved by EBL but not by ROTE.

6.2. Results on Search Performance

The results on search performance are easier to understand if solved and unsolved problems are considered separately. Figure 11 shows the search behavior of three versions of LT on the problems they fail to solve. Each point on the graph corresponds to an unsolved problem. The figure gives the "average branching factor", the number of problems generated in the search divided by the number of problems attempted. In the case of unsolved problems, the number of problems attempted is a constant, since this parameter is limited to 16 (15 subproblems and the original problem) and search has to be abandoned at that point when LT fails to find solutions. Thus, the average branching factor is proportional to the number of problems generated in this case.

The fact that the ROTE curve is generally above the EBL curve and both are always above the NL curve indicates that, in most cases, ROTE generates more subproblems than EBL, which generates more subproblems than NL. When both systems solve a problem, ROTE never attempts more subproblems than NL, and ROTE almost always generates fewer (or the same number of) subgoals before a solution is found.

In general, EBL does less search than ROTE, as measured in terms of problems attempted, subproblems generated, and in terms of average branching factors (as shown in Figure 12).[10] However, there are a significant number of exceptions to this rule, e.g., problems 86 and 87. Note also the relative flatness of the NL and EBL "curves" for both solved and unsolved problems. This indicates that the number of subproblems generated per problem attempted does not increase markedly with the number of problems tried by NL and EBL, while it does increase in ROTE. [11]

6.3. Discussion of Limited Search Under the IMPLIES Restrictions

This subsection contains interpretations and explanations of the experimental results just described. Some of the explanations are hypothetical. Some are alternative

[9] As will be explained more fully in the discussion of the results, this is another anomaly due to the IMPLIES restrictions, motivating the next experiment.

[10] Note that the graphs of measurements on solved problems are presented in a different graphical format than that used for unsolved problems. In order to facilitate comparison of corresponding solutions, vertical lines connect corresponding points in the graph.

[11] The relative flatness of the EBL curve turns out to be an anomaly. The next experiment lifts the IMPLIES restrictions. The results will show that the EBL curve is relatively flat here only because the IMPLIES restrictions prevent many results learned by EBL from being used in the search for solutions.

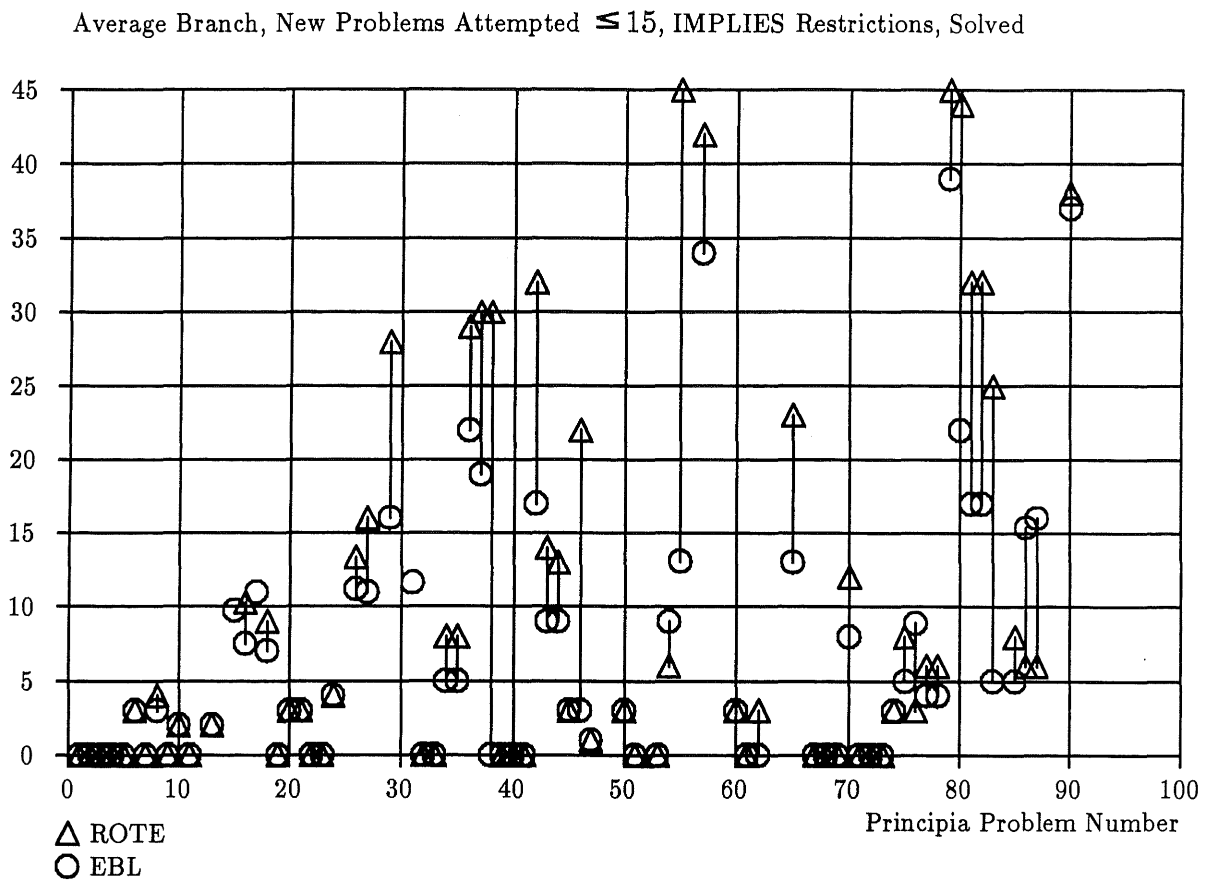

Figure 12. ROTE vs EBL on Solved Problems in Limited Search

explanations of the same effects. Experiments in later sections of this paper are designed to tease apart the magnitudes of the contributions of competing but not necessarily incompatible alternatives.

In general, problems solved by the non-learning LT are also solved by the ROTE version and problems solved by the ROTE LT are also solved by the EBL version. In general, the solutions provided by EBL are superior to the solutions found by ROTE and these are superior to the solutions found without learning.

The fact that ROTE proofs are as short as or shorter than proofs produced without learning is not surprising; it is a consequence of the fact that LT does breadth-first search and the ROTE LT needs only to reduce a problem to a previously seen problem, whereas the non-learning LT has to reduce it to one of the original axioms. ROTE has the advantage that the steps used in the proofs of the learned theorems are not counted against it when the results of learning are used to solve later problems.

Looking at the number of subproblems generated and attempted by each version of LT, one sees increases in these "amounts of search" in going from non-learning to learning in some cases and drops in other cases. The drops indicate that some learned theorem is useful in solving a problem efficiently. The increases are due to the fact that learned theorems increase the number of possible next steps in proofs. The learning versions of LT generally have more ways of attempting to solve a problem. When one of the early attempts succeeds, the problem is solved immediately and less search is done. Otherwise, more attempts are made before a solution is found or search is abandoned.

Relationships between the average branching factors are relatively easy to see in the unsolved problems. The branching factors are relatively low for the non-learning system, much higher in both learning systems, but significantly lower in EBL than in ROTE. They appear to be relatively constant in NL, but increase with learning, more quickly in ROTE than in EBL.

It may seem odd that the no learning performance is poor (it solves far fewer problems) as compared to the learning systems when it seems to have a less difficult search space (lower branching factor) to contend with. There are several reasons for this. First, learned theorems enable the learning versions of LT to "see more deeply into the search space." Search is limited and there are problems that cannot be solved without learning just because the required search exceeds the limit. The learning versions of LT may be able to effectively exceed the limit because search done in constructing the proof of learned solutions is not counted against searches that apply these earlier solutions to solve later problems.

Another possible reason for the drastic improvement in performance in learning as compared to non-learning has to do with LT's limited control strategy. LT is restricted to producing linear proofs: each operator (detachment and chaining) uses a known theorem in order to reduce a problem to a new subproblem. However, the learning systems add to the initial axioms theorems that follow from the initial axioms by one or more operations. This has the effect of allowing the learning versions of LT to break out of this constraint so that the search for a solution is taking place in a radically different search space, one which contains solutions that cannot be generated by the non-learning version of LT.

Figure 13 shows an example of a proof that is within the search space of the learning systems but denied to the NL LT. While NL does manage to construct an equivalent proof, using chaining forward as a mirror image of chaining backward, this is done at the cost of extra search.

An additional source of the improvement in performance in learning is that learned solutions can increase the set of problems that can be solved. It is known that LT is an incomplete theorem prover; in other words there are theorems that it cannot prove in principle (even ignoring any limitations of the amount of search allowed). For example, *Principia* problem 2.13 cannot be solved by LT. Adding such problems to the list of known results covers for incompleteness in the theorem prover and leads to solutions of

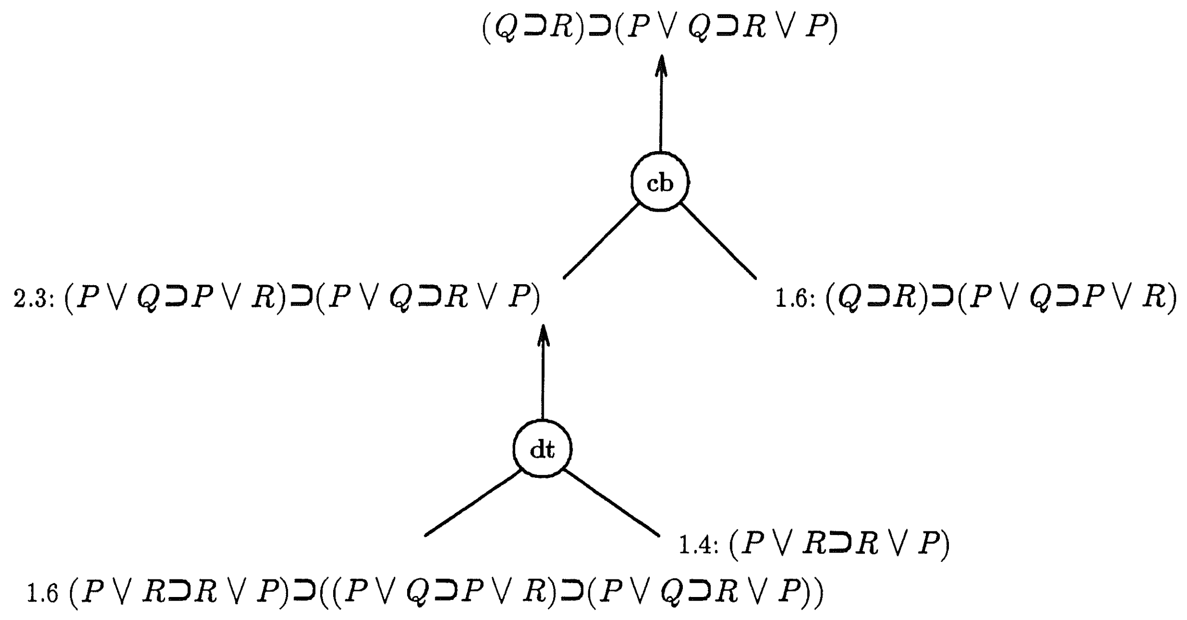
$(Q \supset R) \supset (P \vee Q \supset R \vee P)$
cb
2.3: $(P \vee Q \supset P \vee R) \supset (P \vee Q \supset R \vee P)$
1.6: $(Q \supset R) \supset (P \vee Q \supset P \vee R)$
dt
1.4: $(P \vee R \supset R \vee P)$
1.6 $(P \vee R \supset R \vee P) \supset ((P \vee Q \supset P \vee R) \supset (P \vee Q \supset R \vee P))$

Figure 13. Rote-Learning and EBL Proof of Principia-2.36

problems that otherwise could not be solved.

Focusing on search performance differences between the learning systems, we note that sometimes problems are solved by the EBL version alone, (for example, in this experiment, Principia-2.16 and 2.18). Also, it is often the case that EBL finds proofs with less search than ROTE, measuring the amount of search in terms of problems attempted, subproblems generated, and in terms of average branching factor. One possible reason for improvement in performance in EBL as opposed to ROTE is the improved generality of the results of explanation-based learning.

Another possible reason for improvements in branching factors in going from ROTE to EBL is that the ROTE LT adds instances of known theorems to the end of the list of known theorems. Our initial ROTE LT did this because it is a simple, natural way to do rote learning, but also because of historical reasons. The original rote learning version of Newell, Shaw, and Simon's LT added every problem, regardless of whether it was solved as an instance of a known theorem. This was not done in the EBL version of LT because it violates a general principle of explanation-based learning that might be paraphrased: *only novel solutions to problems are worth remembering* [3]. Even without invoking EBL, however, there is no point in adding instances of known theorems to the list of known theorems because any instance of an instance X of Y is also an instance of Y. Or to put it another way, any "indirect" instance is also a "direct" instance. Adding instances hurts by increasing the branching factor of the search but provides no benefits, since the instances are added to the end of the list of known theorems, rather than the beginning. Section 9, *On the Effects of Adding Instances in Rote Learning*, reports on an experiment aimed at determining how much of the difference between ROTE and EBL is accounted for by the fact that one adds instances and the other does not. This experiment will show that the obvious way of augmenting LT with ROTE learning is not the most effective way to do so. It turns out that the obvious way of augmenting LT with EBL is not the most effective way, for a different reason.

This was discovered when the examples where EBL missed superior solutions found by ROTE were viewed as possible anomalies and explanations were sought. It was hypothesized that the increased generality of results learned by EBL might get in the way of finding a proof found by ROTE or NL. A detailed analysis, however,

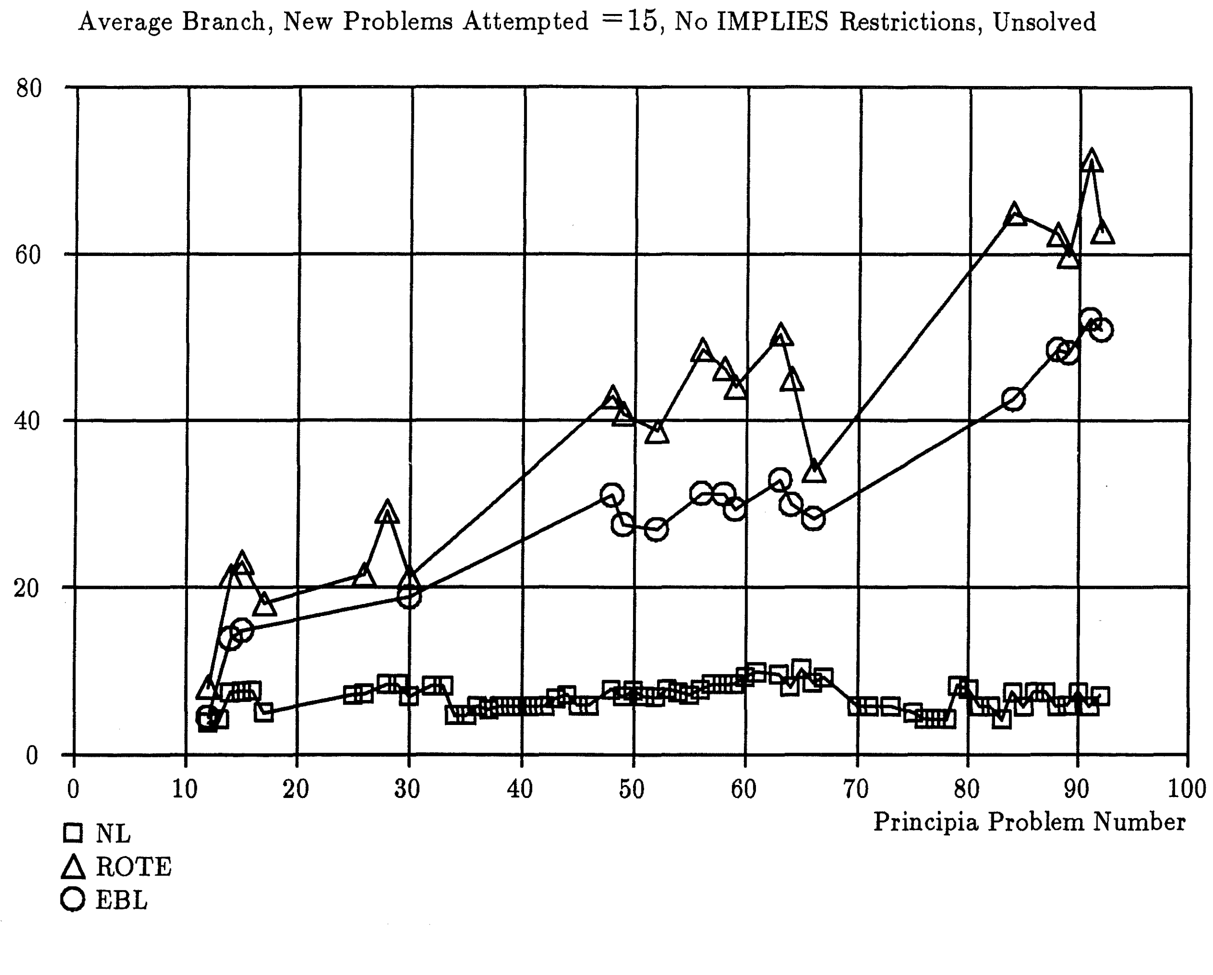

Figure 14. Search Behavior Without the IMPLIES Restrictions

revealed that LT had some restrictions in its use of problem solving operators and that these restrictions seemed to interact with learning and to prevent some good proofs from being found. The details of these restrictions were presented in footnotes in the descriptions of the detachment and chaining operators in the section on LT, the performance element. Briefly, LT requires subproblems to have IMPLIES as their top level connective before it will attempt to reduce them using forward or backward chaining. In addition, the known theorem used must also have IMPLIES as its top level connective ([19], [30] page 24).[12] These restrictions may have been incorporated in the original LT in an effort to reduce the branching factor of the search. In the present study, however, these IMPLIES restrictions were identified as culprits in some cases where EBL missed solutions found by ROTE and it was hypothesized that the restrictions might be responsible for a number of anomalies observed in the initial experiments. Since EBL often learns theorems that are more general than their ROTE counterparts, and many of these are not explicit implications, EBL was thought to be more vulnerable to degradations in performance under the IMPLIES restrictions than ROTE. The next experiment tests this hypothesis.

7. On the Effects of Removing the IMPLIES Restrictions

The initial experiment reported in the previous section compared improvements in the behavior of a performance element augmented in obvious ways with rote and explanation-based learning. Analysis of the results indicated that the obvious way to combine a performance element with a learning method such as ROTE or EBL may not always be the best way. It was expected that EBL would perform at least as well as ROTE, if not better, but in the experiment ROTE outperformed EBL in some cases. Analysis of the reasons for this surprise pointed to unexpected interactions between the performance and learning elements in the EBL system. It was discovered that restrictions in the performance element prevented some results learned by EBL from being used in future problem solving. In this section, the restrictions in the problem solver are lifted in order to determine the extent of their interference with EBL.

To be specific, in this section we lift LT's IMPLIES restrictions. Lifting these restrictions has effects on LT regardless of whether or how it is learning. The question is how much of an effect? How will the relationships between non-learning and the competing learning strategies change?

7.1. Results on Number and Quality of Solutions and Search

With the IMPLIES restrictions lifted, NL solves 26 of the 92 problems (24%), ROTE solves 71 (77%), and EBL solves 74 (80%). EBL solves everything solved by ROTE and ROTE solves everything solved by NL. ROTE no longer gets superior solutions; EBL gets two solutions of length zero where ROTE has solutions of length one.

[12] The resulting subproblem is, by definition, an explicit implication, but it may ground in (match with) a known theorem that is not an implication.

As in the first experiment, it is easier to understand the search results by considering the solved and unsolved problems separately. Figure 14 shows the search behavior of three versions of LT on unsolved problems. Lifting the IMPLIES restrictions has substantial impact. The branching factors are much higher, especially in the learning systems. EBL's average branch now seems to increase with learning at nearly the same rate as ROTE, whereas under the IMPLIES restrictions its branching factor curve seemed relatively flat. In addition, the relationships between the various systems are simpler: the ROTE curve is now always above the EBL curve and it in turn is always above the NL curve.

Turning to solved problems, Figure 15 shows the search behavior of ROTE versus EBL. EBL does less search than ROTE on every problem now, as measured by problems generated, subproblems attempted, and by average branching factors. The exceptions observed in the initial experiments no longer occur once the IMPLIES

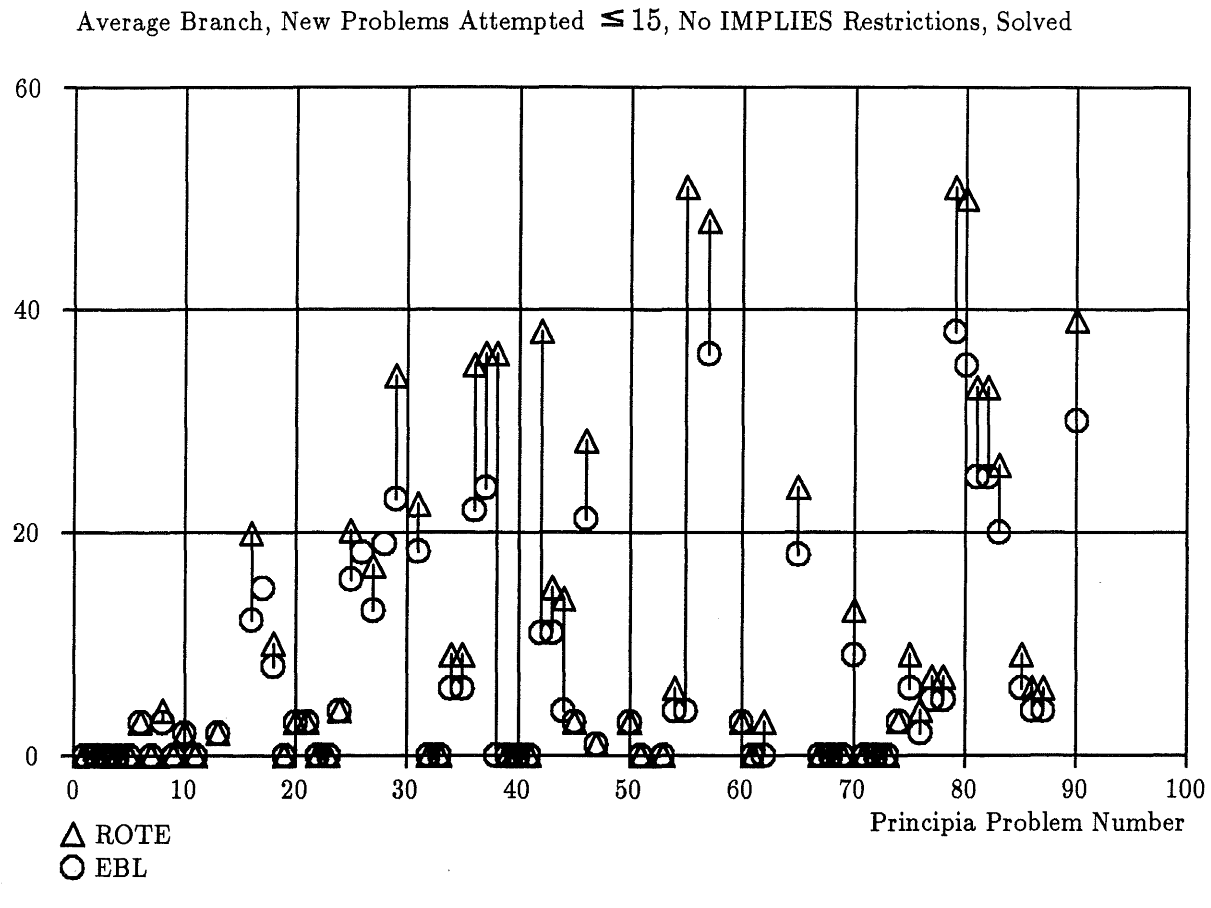

Figure 15. ROTE vs EBL Without the IMPLIES Restrictions

restrictions are lifted.

7.2. Discussion on the Effects of Lifting the IMPLIES Restrictions

Lifting the IMPLIES restrictions tends to increase branching factors, as reflected in the numbers of problems generated in limited attempts on the unsolved problems. However, this is offset by the fact that the new subproblems generated sometimes lead to early proofs. This sometimes makes the difference, in limited search, between solving or not solving a problem. In other cases it means that a shorter proof is found.

Since problems must be reduced to the initial axioms when learning is disallowed, changes in behavior noted in the non-learning system are due solely to the fact that chaining is allowed to work on subproblems that are no longer required to be implications (the initial axioms are always implications). In ROTE, changes are due to both types of the IMPLIES restrictions, but the effect of the requirement that known theorems be implications is muted by the fact that almost all of the Principia problems are implications. Thus, once they are learned, they can be used even under the IMPLIES restrictions. In the EBL system, however, lifting the IMPLIES restrictions leads to much more pronounced changes in performance, because many of the theorems learned are not implications. Many of the theorems learned by EBL are disjunctions more general than implications; they can be specialized to implications.

EBL does significantly more search than ROTE in some cases due to the fact that the IMPLIES restricted LT fails to put the results of explanation-based learning to full use. LT only uses the chaining schema to solve problems when they have the form of implications. In addition, it only uses known implications in chaining, in order to transform problems into new subproblems. These restrictions effectively prevent the EBL LT from finding some legitimate proofs by preventing LT from using some of the theorems learned by EBL. *Thus, in the remaining experiments discussed in this paper, the IMPLIES restrictions are lifted.*

The general significance of the results of this section is that one cannot expect to get a high performance learning system by simply adding an explanation-based learning element to an existing performance element. It is important to check whether the performance element is capable of making full use of the generalizations provided by explanation-based learning. Hidden restrictions in the performance element can block the effective application of knowledge acquired by EBL.

8. On the Effects of Allowing Extended Search

It is important to determine whether the relationships observed so far are dependent on the fact that search was restricted rather severely in the previous experiments. Increasing the search limits should enable each version of the problem solver to solve more problems. What else happens when the search limits are relaxed? What happens to the differences in performances of the learning systems? In this section, we loosen the limits on search. Instead of only attempting the first 15

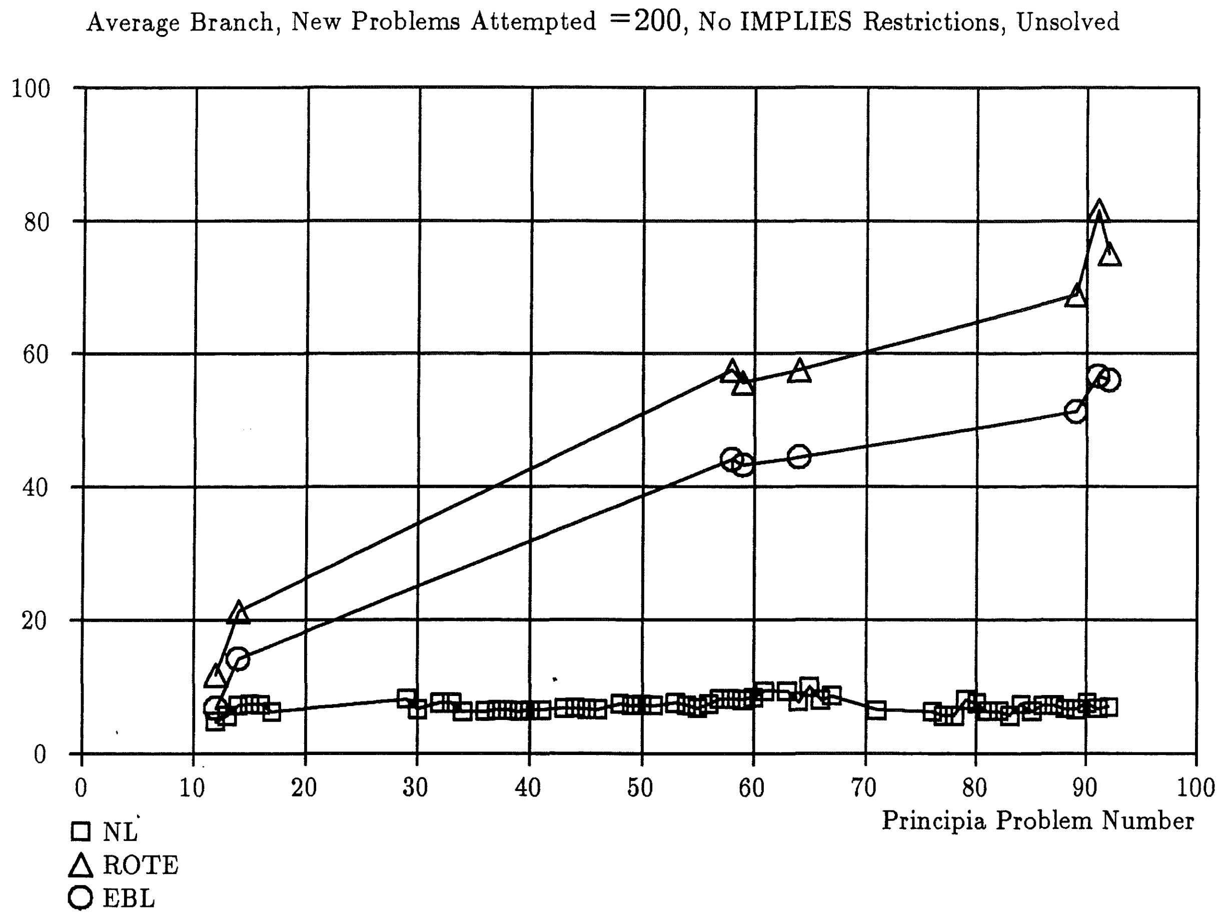

Figure 16. Search Behavior on Unsolved Problems in Extended Search

subproblems, 200[13] subproblems may now be attempted.

8.1. Number and Quality of Solutions and Search Performance

As in limited search, NL solves far fewer problems than the learning methods; all problems solved by NL are solved by ROTE, and ROTE solves many problems

[13] Note that the IMPLIES restrictions are not in force in this section. Note also that the limit of 200 was arrived at empirically and has no special theoretical significance. Several numbers much larger than 15 were tried. It was found that 200 (as opposed to a limit of, e.g., 1000) is small enough that experiments can be done in reasonable computing time. On the other hand, 200 is large enough that certain experimental results obtain. In particular, 200 provides enough search resources so that the percentage of problems solved by the learning methods exceeds 90% and the difference in the number of problems solved by the learning methods goes to zero.

unsolved by NL. NL solves 35 of 92 (38%) while the learning systems each solve the same 84 problems (91%). The difference with respect to problems solved by the learning systems goes to zero with extended search.

Figure 16 shows the search behavior of NL, ROTE, and EBL on the problems they fail to solve. The ROTE and EBL curves have points at the same X coordinates since they solve the same problems. The NL curve is denser because it solves fewer problems than the learning systems.

The branching factor observed in NL seems to be roughly constant and much lower than the sharply increasing branching factors of the learning systems. ROTE does more search as indicated by subproblems generated and attempted (not shown). EBL's branching factors are uniformly below those of ROTE (as shown in Figure 17). One important reason for this is identified in section 9 *on the effects of adding*

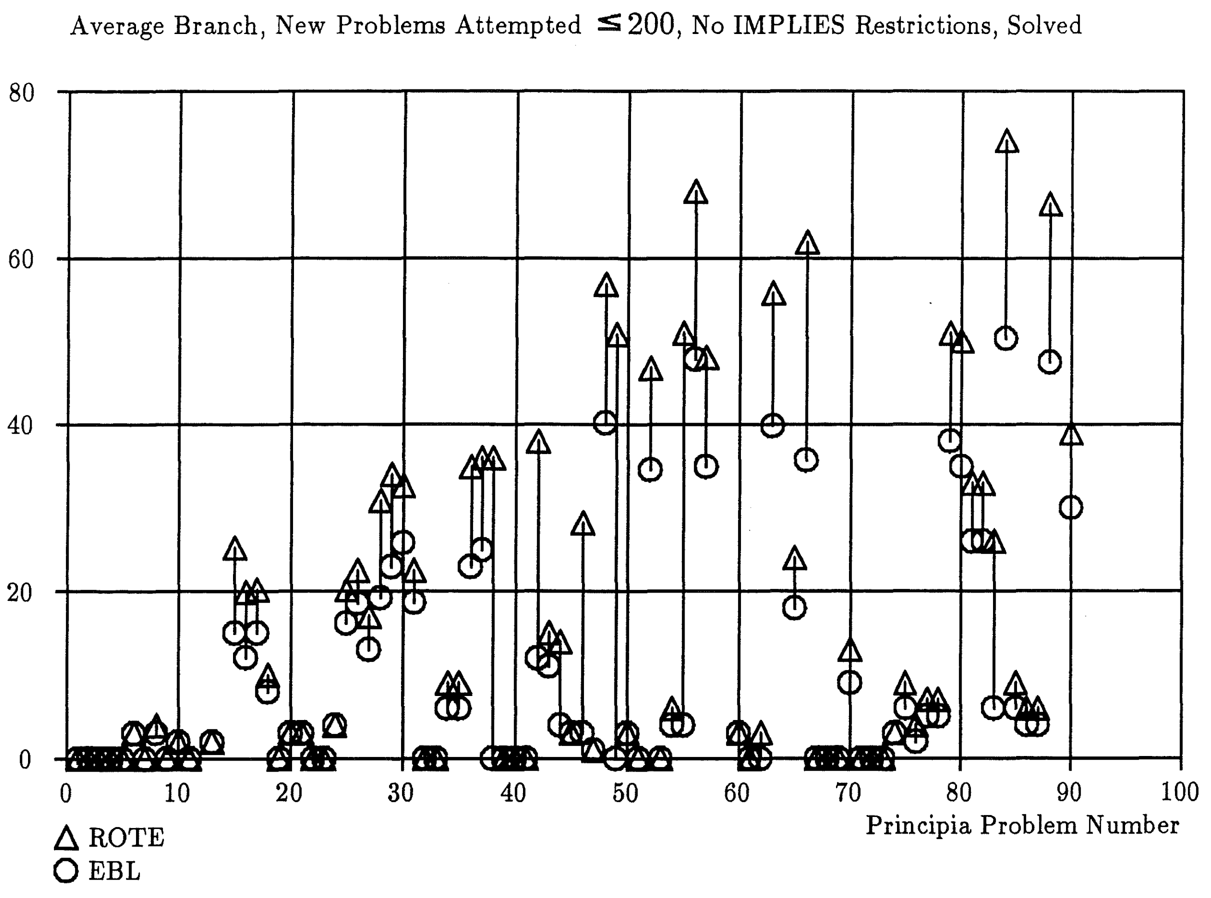

Figure 17. ROTE vs EBL on Solved Problems in Extended Search

instances in rote learning.

Another reason is the fact that EBL can learn strictly more general results than ROTE. This can lead to superior performance as shown in appendix 2: *Examples of Improvements in Performance Due to EBL.* In brief, the fact that the difference in the number of problems solved between EBL and ROTE goes to zero as the search limit goes up is due to the fact that EBL only improves efficiency over ROTE. EBL has an efficiency advantage in case the best solution of a new problem requires an instance of a more general result learned by EBL and this instance is not an instance of the corresponding result learned by ROTE. If the search limit is low enough in this situation, EBL will find a solution where ROTE fails to find one. If the search limit is increased, both will find a proof because if the search limit is high enough ROTE can always get the necessary lemma by reconstructing the original proof.

8.2. Quality of Learning

A comparison of the quality of the results learned by ROTE versus those learned by EBL is shown in Table 1.[14] Recall from the discussion in the subsection on *Comparing the Generality of Learned Results* in the section on *Methodology* that EBL results are always at least as general as the results of ROTE learning, and that EBL can yield strictly more general results. Note that EBL is not attempted in many cases, specifically in cases where the original problem is an instance of a known theorem and in cases where no proof is found. The following comparison focuses on the theorems learned by EBL versus ROTE in the remaining cases.

The column labelled "Problem" in the table shows the problem numbers from *Principia.* The column labelled "EBL vs ROTE" shows an "=" if EBL and ROTE learn equally general theorems; it shows ">" if EBL learns a strictly more general result. The next column shows the theorem learned by EBL. When EBL learns a strictly more general result than ROTE, the column labelled "θ" shows a difference substitution; applying this substitution to the theorem learned by EBL yields the theorem learned by ROTE. The substitution θ may be interpreted as showing the overspecialization which results from using ROTE rather than EBL.

In the results of this particular experiment, one sees that while many theorems (25 of 56, or about 45%) learned via EBL are no more general than the given problems, in many cases (31 of 56, about 55%) the theorem added by EBL is strictly more general than the theorem added by ROTE.

How are the results of EBL more general? In many cases (e.g., Principia-2.06) EBL produced a variant of the original problem, minus an extraneous negation. The EBL version of LT acquires the theorem $\overline{P \vee Q} \vee (\overline{Q \supset R} \vee (P \vee R))$ from the solution

[14] These results are from the experiment with extended search (the number of new problems attempted was limited to 200). Similar results on the quality of learning obtained in other experiments have been omitted for brevity. These particular results were chosen for presentation because more problems are solved and hence more explanation-based learning occurs in relatively extended search, so there are more opportunities to compare the different learning methods.

Table 1: Comparison of Theorems Learned by EBL vs ROTE.

Problem	EBL vs ROTE	Theorem Learned by EBL	θ
2.06	>	$\overline{P \vee Q} \vee (\overline{\overline{Q} \vee R} \vee (P \vee R))$	$\{P/\overline{p}, Q/q, R/r\}$
2.08	=	$P \supset P$	
2.11	=	$P \vee \overline{P}$	
2.14	=	$\overline{\overline{P}} \supset P$	
2.16	>	$P \vee Q \supset \overline{\overline{Q}} \vee P$	$\{P/\overline{p}, Q/q\}$
2.17	>	$\overline{\overline{P}} \vee Q \supset Q \vee P$	$\{P/p, Q/\overline{q}\}$
2.18	=	$(\overline{P} \supset P) \supset P$	
2.2	=	$P \supset P \vee Q$	
2.24	>	$P \vee (\overline{P} \vee Q)$	$\{P/\overline{p}, Q/q\}$
2.25	=	$P \vee (P \vee Q \supset Q)$	
2.3	=	$P \vee (Q \vee R) \supset P \vee (R \vee Q)$	
2.31	=	$P \vee (Q \vee R) \supset (P \vee Q) \vee R$	
2.32	=	$(P \vee Q) \vee R \supset P \vee (Q \vee R)$	
2.36	=	$(P \supset Q) \supset (R \vee P \supset Q \vee R)$	
2.37	=	$(P \supset Q) \supset (P \vee R \supset R \vee Q)$	
2.38	=	$(P \supset Q) \supset (P \vee R \supset Q \vee R)$	
2.4	=	$P \vee (P \vee Q) \supset P \vee Q$	
2.41	=	$P \vee (Q \vee P) \supset Q \vee P$	
2.45	=	$\overline{P \vee Q} \supset \overline{P}$	
2.46	=	$\overline{P \vee Q} \supset \overline{Q}$	
2.47	>	$\overline{P \vee Q} \supset \overline{P} \vee R$	$\{P/p, Q/q, R/q\}$
2.48	>	$\overline{P \vee Q} \supset R \vee \overline{Q}$	$\{P/p, Q/q, R/p\}$
2.521	>	$\overline{\overline{P \vee Q}} \vee (\overline{Q} \vee R)$	$\{P/\overline{p}, Q/q, R/p\}$
2.53	=	$P \vee Q \supset (\overline{P} \supset Q)$	
2.54	=	$(\overline{P} \supset Q) \supset P \vee Q$	
2.55	=	$\overline{P} \supset (P \vee Q \supset Q)$	
2.56	=	$\overline{P} \supset (Q \vee P \supset Q)$	
2.6	>	$P \vee (\overline{Q} \vee (R \vee Q))$	$\{P/\overline{\overline{p}}, Q/q, R/\overline{\overline{p} \vee q}\}$
2.61	>	$P \vee Q \supset \overline{\overline{P} \vee Q} \vee Q$	$\{P/\overline{p}, Q/q\}$
2.621	=	$(P \supset Q) \supset (P \vee Q \supset Q)$	

Table 1: Comparison of Theorems Learned by EBL vs ROTE (ctd.)

Problem	EBL vs ROTE	Theorem Learned by EBL	θ
2.64	=	$P \vee Q \supset (P \vee \overline{Q} \supset P)$	
2.67	>	$\overline{\overline{P \vee Q} \vee R} \vee (\overline{P} \vee R)$	$\{P/p, Q/q, R/q\}$
2.68	>	$\overline{\overline{\overline{P} \vee Q} \vee R} \vee (P \vee R)$	$\{P/p, Q/q, R/q\}$
2.69	>	$\overline{\overline{P} \vee Q} \vee R \supset \overline{\overline{R} \vee P} \vee P$	$\{P/p, Q/q, R/q\}$
2.73	=	$(P \supset Q) \supset ((P \vee Q) \vee R \supset Q \vee R)$	
2.76	=	$P \vee (Q \supset R) \supset (P \vee Q \supset P \vee R)$	
2.81	=	$(P \supset (Q \supset R)) \supset (S \vee P \supset (S \vee Q \supset S \vee R))$	
2.83	>	$\overline{P \vee (Q \vee R)} \vee (\overline{P \vee (\overline{R} \vee S)} \vee (P \vee (Q \vee S)))$	$\{P/\overline{p}, Q/\overline{q}, R/r, S/s\}$
2.85	>	$\overline{P \vee Q} \vee (R \vee S) \supset R \vee (\overline{Q} \vee S)$	$\{P/p, Q/q, R/p, S/r\}$
3.12	>	$P \vee (Q \vee \overline{P \vee Q})$	$\{P/\overline{p}, Q/\overline{q}\}$
3.21	>	$P \vee (Q \vee \overline{Q \vee P})$	$\{P/\overline{p}, Q/\overline{q}\}$
3.22	>	$\overline{\overline{P \vee Q}} \vee \overline{Q \vee P}$	$\{P/\overline{p}, Q/\overline{q}\}$
3.24	>	$\overline{\overline{P \vee \overline{P}}}$	$\{P/\overline{p}\}$
3.26	>	$\overline{\overline{\overline{P} \vee Q}} \vee P$	$\{P/p, Q/\overline{q}\}$
3.27	>	$\overline{\overline{P \vee \overline{Q}}} \vee Q$	$\{P/\overline{p}, Q/q\}$
3.3	>	$\overline{P \vee Q} \vee R \supset P \vee (Q \vee R)$	$\{P/\overline{p}, Q/\overline{q}, R/r\}$
3.31	>	$P \vee (Q \vee R) \supset \overline{\overline{P \vee Q}} \vee R$	$\{P/\overline{p}, Q/\overline{q}, R/r\}$
3.33	>	$\overline{\overline{P \vee Q} \vee \overline{\overline{Q} \vee R}} \vee (P \vee R)$	$\{P/\overline{p}, Q/q, R/r\}$
3.34	>	$\overline{\overline{\overline{\overline{P} \vee Q} \vee \overline{R \vee \overline{P}}}} \vee (R \vee Q)$	$\{P/p, Q/q, R/\overline{r}\}$
3.35	>	$\overline{\overline{P \vee \overline{P \vee Q}}} \vee Q$	$\{P/\overline{p}, Q/q\}$
3.37	>	$\overline{P \vee Q} \vee R \supset \overline{\overline{P \vee \overline{\overline{R}}}} \vee Q$	$\{P/\overline{p}, Q/\overline{q}, R/r\}$
3.4	>	$\overline{\overline{P \vee \overline{Q}}} \vee (R \vee Q)$	$\{P/\overline{p}, Q/q, R/\overline{p}\}$
3.41	>	$\overline{\overline{P} \vee Q} \vee (\overline{\overline{\overline{P} \vee R}} \vee Q)$	$\{P/p, Q/q, R/\overline{r}\}$
3.42	>	$\overline{\overline{P} \vee Q} \vee (\overline{\overline{R \vee \overline{P}}} \vee Q)$	$\{P/p, Q/q, R/\overline{r}\}$
3.43	>	$\overline{\overline{\overline{P \vee Q} \vee \overline{P \vee R}}} \vee (P \vee \overline{\overline{Q} \vee \overline{R}})$	$\{P/\overline{p}, Q/q, R/r\}$
3.45	>	$P \vee Q \supset \overline{\overline{P \vee R}} \vee \overline{\overline{Q} \vee R}$	$\{P/\overline{p}, Q/q, R/\overline{r}\}$

of 2.06. The problem, $(p\supset q)\supset((q\supset r)\supset(p\supset r))$, is an instance of this theorem with Q bound to q, R bound to r, and P bound to $\bar{p}$. (This is the meaning of the difference substitution in the theta column of Table 1.) The ROTE LT demands a negation that is not required by the proof.

The EBL version of LT offers only modest improvements over ROTE in such examples because it is well known that one can reverse the sign of a literal everywhere in a theorem to get a new theorem. One could easily modify the ROTE LT to take advantage of the fact that whenever a literal appears only negatively in a *Principia* problem, one can safely delete the negative sign to obtain a logically equivalent but syntactically more general problem.

In a number of cases, however, there is no such simple fix that the ROTE LT could use to obtain theorems as general as those acquired by the EBL version. For example, in a number of cases, ROTE unnecessarily collapses two or more variables into one. Problem 36, Principia-2.47, is $\overline{p\vee q}\supset(p\supset q)$. The EBL version of LT acquires $\overline{P\vee Q}\supset(\overline{P}\vee R)$ from the proof. The problem is obtained as an instance by substituting p for P and by binding both Q and R to q.

In other cases rote learning results in more interesting overspecializations. In these examples, variables are not simply collapsed by ROTE; instead they are required to be related logically in complicated ways when they really should be completely independent.

In Problem 42, one variable is made to be the negation of another when they should be independent and neither need be a negation. The problem (Principia-2.521) is $\overline{p\supset q}\supset(q\supset p)$. The generalized conclusion of the proof is $\overline{P\vee Q}\vee(\overline{Q}\vee R)$. These match with bindings of P to $\bar{p}$, Q to q, and R to p.

In Problem 47, three independent variables are specialized by effectively making one into a negation of an implication between the others and by requiring one to be a double negative. The problem (Principia-2.6) is $\bar{p}\supset(q\supset((p\supset q)\supset q))$ and the generalized conclusion of the proof is $P\vee(Q\supset(R\vee Q))$. These results of rote and explanation-based learning match with P bound to $\bar{\bar{p}}$, Q bound to q, and R bound to $\overline{p\supset q}$.

With LT's IMPLIES restrictions loosened, the improved generality afforded by EBL leads to performance that is superior to that of the ROTE version of LT.[15] Sometimes the EBL system solves problems that could not be solved by the ROTE system due to limits on search. Sometimes both ROTE and EBL solve the problem but the EBL solution is found earlier in the search. In some of these cases, the EBL solution is of higher quality than the solution provided by ROTE. The quality of solutions found by the learning systems in extended search is not discussed in this section. Instead, this comparison will take place in the next section, in the context of a comparison between EBL and a version of rote learning that produces the same solutions as the present ROTE system using a more efficient search.

[15] For detailed discussions of examples of how EBL improves performance, see the appendix on "Examples of Improvements in Performance Due to EBL."

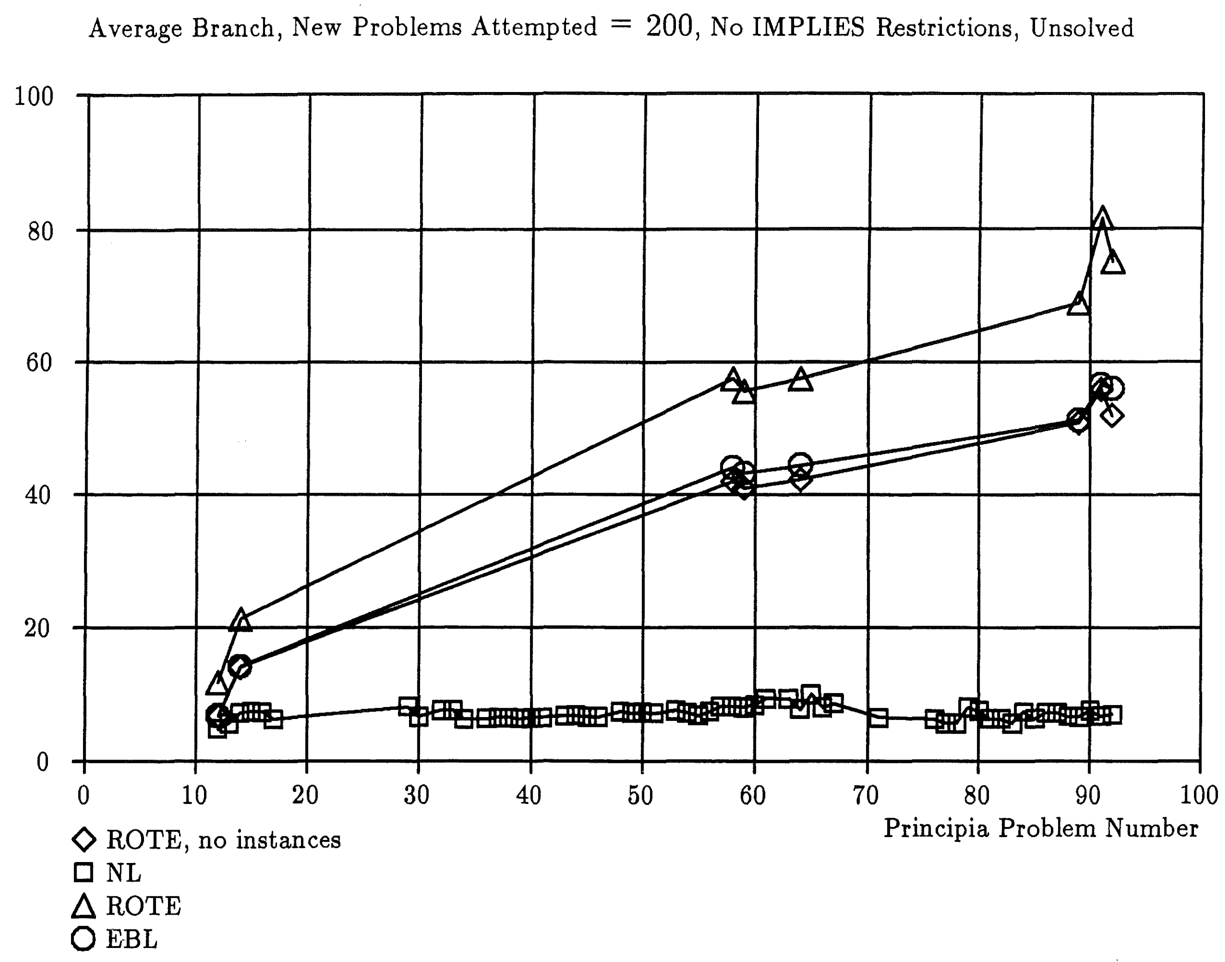

Figure 18. The Effects of Adding Instances During ROTE Learning

9. On the Effects of Ignoring Instances

Two sources of the improved performance of the EBL over ROTE have been identified. The improved generality of learned solutions helps, as does the fact that EBL does not add instances. In this section, we report on an experiment that isolates these sources of improvement by allowing the rote learning LT to avoid learning instances of known theorems.

The results on quantity and quality of solutions for improved ROTE are the same as for ROTE because the improved ROTE gets exactly the same solutions, it just gets them more efficiently. ROTE should never produce shorter proofs than EBL because our performance element does breadth-first search and the theorems learned by EBL are always at least as general as those learned by ROTE. Some (5/84, or roughly 6%)

of the 84 proofs discovered by EBL are shorter than the proofs provided by improved ROTE. On the five problems where they find solutions at different depths, ROTE had depths of 2, 1, 2, 2, and 1 while EBL had depths of 1, 0, 1, 0, and 0, respectively. The average depth of the 35 NL proofs was 61/35 (1.74), the average depth of the ROTE proofs was 77/84 (0.92), and the average of the EBL proofs was 71/84 (0.84).

Figure 18 shows the search behavior of ROTE (without adding instances) on unsolved problems superimposed on that of the systems previously studied. The numbers of problems attempted and subproblems generated by improved ROTE are both substantially reduced by not adding instances. Figure 19 shows the resulting differences in average branching factors between improved ROTE and EBL on solved problems.

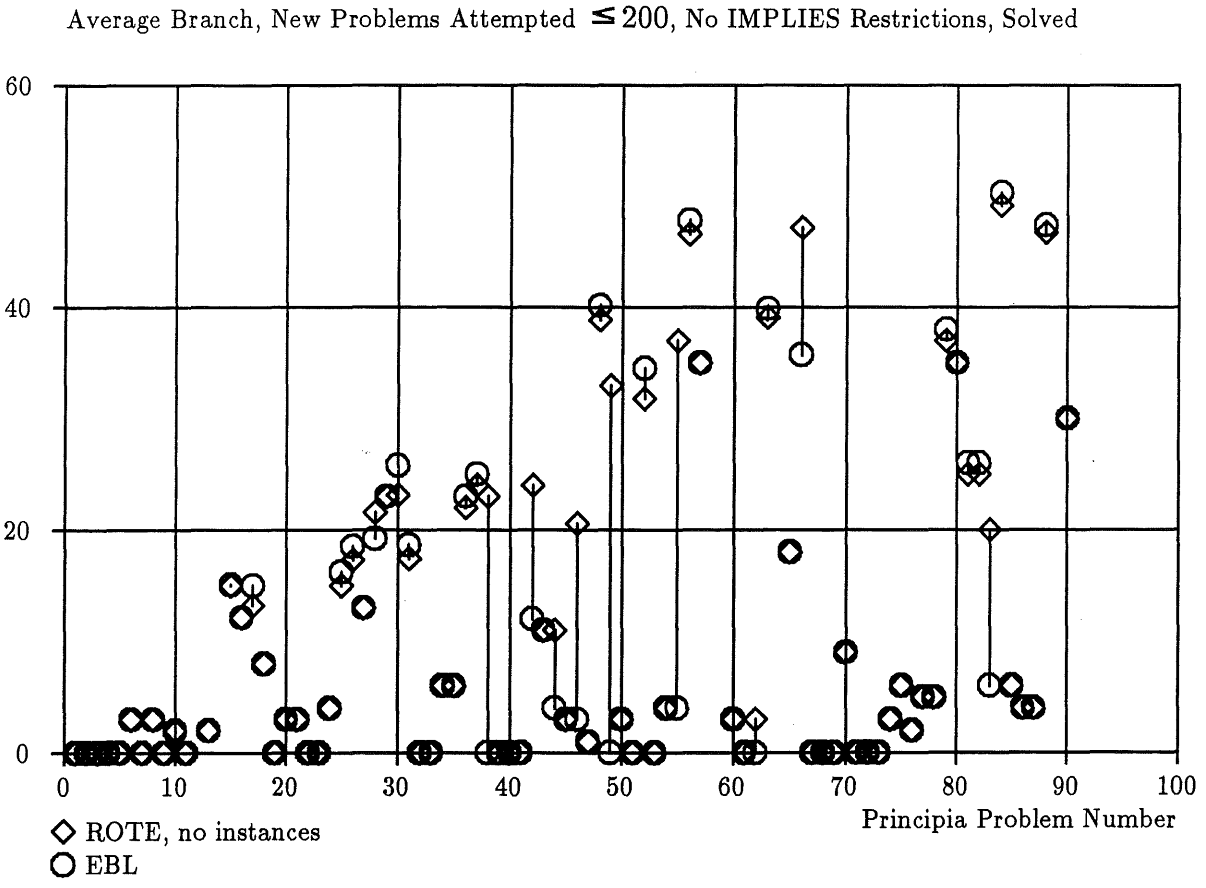

Figure 19. Improved ROTE vs EBL on Solved Problems in Extended Search

The results on search behavior indicate that adding instances accounts for much of the decrease in branching factors that occurs in going from the original ROTE to EBL. While EBL does uniformly less search than the original ROTE system on both solved and unsolved problems, EBL tends to do slightly more search than the improved ROTE method. This is always the case on the unsolved problems, and often the case in solved problems as well. However, exceptions in the solved problems occur when EBL pays off by enabling the problem solver to hit upon a solution earlier in the search. As a result, EBL generates 1850 fewer subproblems than the improved ROTE method and attempts 67 fewer problems overall on the problems solved by both methods. Since ROTE with no instances generates 17485 subgoals and attempts 565 problems, EBL still yields additional improvements of roughly 11% and 12% respectively overall.

The bottom line seems to be that, for a small increase in search on most problems, one can buy large decreases in search on some problems and improved solutions on some problems by using EBL rather than the improved rote learning procedure.

10. Summary

This section summarizes the contributions of this study toward an empirical understanding of explanation-based learning (EBL). Much of the early work on EBL involved partially implemented learning systems and most of the early EBL systems worked on only a small set of examples. Yet the superiority of EBL over alternative learning strategies (including no-learning) often seemed to be taken for granted. The experiments reported here were originally undertaken in an attempt to treat the relationship of EBL to alternative learning methods as a question to be decided empirically. The initial EBL version of LT was one of the first complete EBL systems to run on more than a handful of problems. Our aim in analyzing EBL's performance and in comparing it to other learning systems on the same set of problems was to get results that would eventually facilitate the construction of useful AI systems that learn. It was hoped that tests involving large numbers of problems would help determine the effectiveness of EBL and help determine whether the encouraging results of initial prototypes scaled up to problem solvers that used EBL to tackle long sequences of problems.

The results on the *Principia* problems presented in the present paper are part of an effort to provide a baseline characterization of the performance of learning methods such as rote and basic EBL on these problems. It is hoped that this will make it possible to use the *Principia* problems as a benchmark for testing improved learning methods, just as problems like the eight puzzle have been considered the *Drosophila* or fruit fly of research on search in AI [11].

10.1. Characteristics of the Domain, Test Problems, and Performance and Learning Elements

It might be nice to make grand general claims of the form "learning method X is universally superior to all other forms of learning." No such claim can follow from experiments, however, because one can only test a given learning method against a limited set of competitors, in the context of a limited set of performance elements, on limited sets of test data. It would make no sense to claim that the performance element used in these experiments is a "representative" performance element. It would make even less sense to claim that the problems used in this study are "typical" problems. Performance elements and problem domains vary dramatically. Thus the interpretation of the results of this study must be carefully qualified. The aspects of the domain, test problems, performance element, and learning strategies that seem most relevant to our results are:

- The domain is a highly abstract, purely logical one and the problems are very general, involving variables for propositions rather than concrete propositions like "Agrippina killed Claudius."

- The problems form a carefully constructed sequence arranged in order of increasing difficulty so that solutions to later problems follow naturally from solutions to earlier problems. All of the problems are solvable (though not necessarily by LT).

- The performance element is a schema-based problem solver. The schemata are comprised of propositional terms related by logical dependencies. A small number of schemata are required, namely two.

- The performance element uses breadth first search. The solutions (proofs) have a linear structure.

- The results of learning are added to the end of a list of known solutions. The performance element is allowed to use all prior problems in ROTE learning.

- EBL reverts to ROTE when problems are not solved. No subgoal learning is tried (either within or between trials). EBL learns generalized conclusions rather than condition/action rules. There is no postprocessing of the results of EBL.

10.2. Summary of Experiments and Results

The overall goal of the four experiments reported in this paper was to compare no learning, rote learning, and explanation-based learning (EBL) versions of LT, focusing in particular on the relationship between EBL and ROTE. The experiments systematically vary three parameters corresponding to whether the performance element was allowed to perform extensive amounts of search, whether certain restrictions were involved in determining whether operators should be applied, and whether instances were learned by ROTE.

The first experiment involved severely constrained search. The search was limited partly because people do not seem to attempt large numbers of subproblems during problem solving and EBL was originally developed with human performance and learning in mind. Another reason search was initially limited was that we hypothesized that differences in learning methods might be magnified because improved search efficiency can make the difference between success and failure when search resources are limited.

The results of the limited search experiment showed that the learning methods improved the performance of the problem solver. The learning systems solved more than three times as many problems as the non-learning system, with EBL solving slightly more problems than ROTE. The no learning system had a roughly constant and relatively low branching factor. EBL's branching factor was relatively flat but higher. ROTE learning had a branching factor increasing roughly linearly in the number of learned results. The no learning system produced lower quality solutions requiring more operators than both learning systems, but the comparison of the learning systems gave mixed results: sometimes EBL was better, sometimes ROTE.

These results surprised us, in that they appeared to reverse some of the earlier results reported in [22]. The experiments reported in the present paper include all problems from chapter 3 of *Principia*, thus going beyond the 52 problems from chapter 2 used in Newell, Shaw, and Simon's work and in our own earlier work. We were surprised to find that in the new experiments ROTE occasionally outperformed EBL, finding superior solutions.[16] One possible explanation of the surprising inferior performances was that the increased generality of the results learned by EBL might be hurting by increasing branching factors. However, a detailed analysis of some of the anomalous cases revealed that the increased generality hurt in an unanticipated way. Restrictions built into the performance element prevented operators from being applied in some cases when their application would have led to earlier, better solutions. These restrictions on critical matching operations involved in determining whether operators were applicable forced problems and known theorems to have IMPLIES as their top level connective. These IMPLIES restrictions may have been included in Newell, Shaw, and Simon's original problem solver in order to improve search performance; in any case they were carried over to our own initial versions of LT during the process of re-implementing it according to descriptions in the literature and were not deliberately included in the initial experiments. Once the restrictions were seen to be affecting the coupling between learning and performance, we decided to determine the magnitude of their effect.

The second experiment was performed with limited search but without the IMPLIES restrictions. The major difference in the results was that EBL now solved all problems solved by ROTE and produced solutions that were at least as good as

[16] Worse, in some experiments not reported here involving larger search limits using the new problems, EBL's *overall* performance under the IMPLIES restrictions was worse than ROTE's in terms of numbers of problems solved, costs incurred during search, and quality of solutions.

ROTE's. In addition, EBL's branching factor, while still less than ROTE's, was seen to increase with learning in the same way (roughly linearly with the number of learned results) without the IMPLIES restrictions.

This experiment demonstrated conclusively that the restrictions embedded in the original performance element had a large effect on performance under learning and that the restrictions rendered many of the results of explanation-based learning unusable (non-operational) in future problem solving. These restrictions affected EBL more strongly than ROTE, so it was found in some cases that they caused EBL to be *less effective* than ROTE learning. The lesson of the first and second experiments is that one should not simply add EBL to an existing performance element without considering the coupling between learning and performance. The striking change from relatively flat average branching factor curves to curves increasing with learned results indicates that LT under the IMPLIES restrictions ignored many of the results learned through EBL. One cannot assume that a given performance element will make effective use of the results of EBL.

The author was concerned that the first and second experiments involved two conditions that might tend to favor EBL: the extremely limited search involved may have magnified any advantage EBL had over ROTE, and the eradication of the IMPLIES restrictions improved EBL's performance more than ROTE's. The two remaining experiments may be viewed as efforts to be fair to ROTE.

The third experiment allowed much more extensive search. All systems solved more problems and nearly all the problems were solved by the learning systems. The learning systems still solved many more problems than the non-learning system. Measurements of the relative generality of learned results taken in this experiment showed that EBL learned equally general results roughly half the time and strictly more general results than ROTE roughly half the time. EBL still exhibited advantages in search efficiency and quality of solutions over ROTE as measured by numbers of problems attempted, subproblems generated, average branching factors, and sizes (depths) of solutions. However, each learning method solved exactly the same problems.

The small advantage in solved problems that EBL has over ROTE in limited search disappears when the search limit is relaxed. This is because EBL's advantage over ROTE is basically an efficiency advantage. EBL can solve problems more efficiently than ROTE by using previously learned results that are more general than the corresponding results of ROTE learning. Under limited search, this advantage can enable EBL to find a solution where ROTE fails.[17] Under extended search, however, ROTE can compensate for the fact that it failed to learn all it could from earlier problems by simply reconstructing the solutions it needs.

The fourth experiment was motivated by the idea that, since EBL learns results that are at least as general as results learned by ROTE, one might expect EBL to have

[17] Please see Appendix 2 on *examples of improvements in performance due to ebl.*

higher branching factors than ROTE. It was hypothesized that the major reason why EBL's branching factors were uniformly better than ROTE's was the fact that EBL did not add instances. ROTE learning, as implemented in our initial ROTE LT and in the original due to Newell, Shaw, and Simon, added instances to the end of the list of learned results.

The fourth experiment was carried out to test the hypothesis that ROTE's performance would improve dramatically without the counterproductive learning of instances and to determine whether this improvement in ROTE would surpass EBL. The results were similar to the experience with EBL. The first and second experiments showed that the obvious way to add EBL to our performance element turned out to be flawed because the results of EBL were not being used effectively. The third and fourth experiments show that the obvious way of adding ROTE learning to the performance element was flawed as well. The results show conclusively that when instances were being added to the end of the list of known theorems they were not contributing to finding proofs more quickly and indeed were hurting the search by increasing ROTE's branching factors substantially above EBL's. When ROTE is improved by throwing away instances of known theorems instead of storing them it has slightly lower branching factors than EBL on most problems because EBL yields more general patterns which sometimes match where the patterns learned by ROTE do not. It turned out, however, that while these more general patterns usually amounted to an extra bit of overhead, they sometimes contributed to finding higher quality solutions substantially more quickly, so that in this final experiment EBL's overall performance in terms of quality of solutions and work done during search was slightly better than the performance of the improved ROTE.

11. Relation to Other Work

This paper is part of the growing body of results on Explanation-Based Learning. In this section, we first describe relationships between this work and other existing work, then we describe suggestions for future work.

11.1. Prior and Parallel Work

The papers by Mitchell, et al [16] and DeJong and Mooney [2, 17] provide overviews and pointers into the EBL literature. The present paper is basically an experimental study of EBL versus no learning and rote learning in a particular "domain". Other experimental studies of EBL involving relatively large numbers of examples have been done recently in a number of other task domains, including planning [14, 15]. Mooney's Ph.D. thesis [18] reports on experiments with a general EBL system solving examples from a number of task domains such as planning, recognition, etc. Shavlik's Ph.D. thesis [29] contains the results of a number of experiments on planning and physical reasoning tasks and Segre's Ph.D. thesis [27, 28] contains experimental work on applications of EBL in robotics.

Viewing the EBL versions of LT studied in our experiments as examples of the general model proposed in [16], the *operational concepts* are members of LT's list of known theorems. The IMPLIES restrictions narrow this set of operational concepts to a subset including only the implications among the known theorems. Our results suggest that including restrictions like the IMPLIES restrictions among the operationality criteria is a bad idea and that, in general, it is dangerous to restrict notions of operationality unnecessarily.

It should be noted that the way explanation-based generalization contributes to future problem solving in the present study is a bit unusual. In contrast to EBG macro-learning systems based on PROLOG [7, 23] and MRS [6], only generalized conclusions are learned in the EBL LT. The leaves of the "generalized" proof tree are thrown away; they are not needed because they are (always true) theorems. With this reservation, the present study can be considered to be related to other work on learning macro-operators [10], and can be viewed as a step in the direction of viewing theorem proving as a process that can benefit from the acquisition of macro-operators (as suggested, for example, by Korf in [9]).

Another way to view the work presented here is as evidence bearing on the relative value of methods for generalizing examples by turning constants into variables versus EBL methods of generalizing examples by specializing existing general knowledge. Our results may be viewed as providing experimental evidence that conversions from methods based on turning constants to variables like ROTE to methods like EBL should yield improved performance. For example, since the chunking method originally used in SOAR bears some similarities to ROTE learning, our results suggest that converting SOAR's chunking to an EBL approach as described in [26] should lead to improved performance.

11.2. Future Work

In the summary section, a number of distinguishing *characteristics of the domain, test problems, and performance element* that seemed most relevant to the experimental results were listed. Several possible extensions to the experiments reported here — aimed at determining how sensitive our results are to these distinguishing features — suggest themselves.

There is some concern that our results comparing EBL and ROTE may depend on the nature of the problem sequence used. This sequence appears to have been carefully constructed so that solutions to later problems follow naturally from solutions to earlier problems. In addition, all the problems in the sequence are solvable. In new experiments aimed at determining how changes in the order of the *Principia* problems affect the relative performances of the competing learning methods, we reverse and randomly permute the problem set. In another new experiment, we disallow learning of unsolved problems, in order to address concerns that this strategy, while it may be appropriate for highly structured learning situations, is not appropriate in general. These experiments are still in progress but the author hopes to present them in a follow-on paper.

The particular performance element used may also limit the scope of our experimental results. The scope of our results could be better delimited by conducting experiments aimed at answering questions such as *how much of the improvement in going from non-learning to rote learning and then to EBL is due to the limited control structure of LT requiring that proofs be linear*?

While a number of variations in the performance element and learning methods were tried here, there are still other variations that might yield valuable information. The version of EBL used in this study is just one of several possible versions. Trying alternative approaches could be informative.

For example, some EBL systems learn from the achievement of subgoals and not just when top level goals are achieved (see, for example [21, 23]). It would be simple to augment the EBL version of LT with subgoal learning. This could yield results on how much improvement occurs in *between trial learning* (where learning from one problem facilitates solution of later problems). The original LT would have to be modified to produce non-linear proof trees in order to get results on the effectiveness of *within trial learning* (where learning from a subgoal helps in solving another subgoal within the same problem).

The present domain also seems like a good place for studies of proposed improvements in storing and accessing learned results. It was noted here that it is a mistake to add instances to the end of a list of known theorems in rote learning. Can they make a positive contribution if they are added to the beginning of the list rather than the end? Unlike the EBL LT discussed in this paper, EBL systems typically try the most recently learned solutions first. This strategy may make especially good sense when facing highly structured sequences of problems where new problems often build upon their immediate predecessors as in *Principia.* Indeed, it would be interesting to try out more sophisticated storage and retrieval mechanisms in this highly organized domain. Rather than looking down linear lists as was done in this study, discrimination nets [1] could be used. This could affect the efficiency of the search for solutions and might alter the effectiveness of learning strategies. Perhaps more powerful methods for organizing learned knowledge can be used so as to benefit EBL.

Evidence that carefully controlled "forgetting" will play an important role in the management of memories containing learned results is beginning to accumulate. One such method is based on tracking the usefulness of learned results in terms of measurements of search utility. It would be interesting to see whether Minton's results on search utility in planning [14] can be replicated in the purely logical domain of *Principia.*

In particular, an EBL version of LT could learn general macro operators which would play roles similar to detachment and chaining. These operators could be constructed by ignoring not just the specific problem proved by a composition of schemata, but also by ignoring the specific known theorems used to ground the leaves of the proof tree. For example, recall that Principia-2.17 was proved using detachment and chaining. Treating the detachment schema as an operator going from A and $A \supset B$ to B and the chaining schema as an operator going from $C \supset D$ and $D \supset E$ to $C \supset E$, the alternative approach to EBL in this example would be to paste these two operators together to form a new problem solving operator. Unifying the conclusion B of the detachment step with the antecedent $D \supset E$ of the chaining forward step yields a schema corresponding to a macro-operator taking three antecedents $C \supset D$, $A \supset (D \supset E)$ and A to the conclusion $C \supset E$. Of course, the search control strategy in the performance element would have to be altered in order to make use of these new operators.

If one looks at the theorems learned by EBL in the present study, it is obvious that they could stand some improvement. Many of these theorems are still less general than they could be (e.g., some contain double negations). A future experiment could determine how performance changes in this domain when basic EBL is augmented by a system designed to transform learned expressions into more general forms.

12. Conclusion

One virtue of this study is that it explores the performance of a complete EBL system against competing learning methods on a significant number of examples. The examples used in the study are uncontrived in the sense that the *Principia* problem set was designed for entirely different purposes three quarters of a century ago — long before electronic computers existed and with no consideration of machine learning experiments in mind. During the study a number of interactions between performance elements, learning elements, and the problem set were observed. Some of these interactions were quite unexpected.

The results reported here show that both EBL and rote learning are much better than no learning at all on the *Principia* problems. The experiments focus on the difference between EBL and rote learning in an abstract, purely logical setting, using very general problems, where neither learning method is allowed an advantage in "turning constants to variables." Before these experiments were done, it was hypothesized that generalization and performance would not improve in going from rote to EBL on purely logical problems because neither learning system is allowed to make inductive leaps from concrete propositions like "It is raining" or "Today is Monday." It seems harder for EBL to "win big" on the *Principia* problems because any improvements of EBL over rote learning are forced to occur at a very high level of abstraction. Indeed, the experiments can be interpreted as supporting this hypothesis, because rote learning is "roughly comparable" to EBL on the *Principia* problems in the sense that the difference between the learning systems is small when compared with the large differences between non-learning and learning.

Ignoring non-learning, however, the experimental data focusing on the differences between the learning systems shows that EBL is significantly more effective than rote learning even in highly abstract settings. The size of EBL's advantage over ROTE is magnified when search is extremely limited. The experiments and analysis show that even on very general problems, EBL learns strictly more general results quite frequently and that *provided the performance element is able to exploit it* EBL's superior generalization contributes to superior problem solving performance.

When one considers the goals of the designers of a logical system such as the one found in *Principia Mathematica*, it seems surprising that the EBL LT performs significantly better than rote learning on the *Principia* problems. As we show in detail in appendix 2 on *examples of improvements in performance due to EBL*, when EBL learns a result more general than a given problem learned by ROTE, it is in a position to get superior proofs of later problems. Since the authors of *Principia* did not need to be as concerned as we have been with the efficiency of the process of finding proofs, they had even more reason to ensure that each problem was as general as possible. Imagine Whitehead and Russell creating their sequence of theorems. Assume they have proposed a certain problem as the next theorem in the sequence. If they found a proof for this problem that actually proved a more general theorem than the problem they began with, they could have crossed out the proposed theorem, replacing it with the more general one, thus ensuring shorter (more comprehensible) proofs of later theorems. The EBL LT proves Principia 2.18 in only one step and requires only two axioms while Whitehead and Russell's proof is substantially longer. With this in mind, one can view the superior generalizations produced by the EBL version of LT as suggesting improvements on the logical system of *Principia Mathematica*.

In our opinion, however, the most important lessons of our experiments transcend the particular performance engine and domain used. For example, the caveat that it is important to ensure that a performance element makes good use of the more general results learned by EBL is critical. The performance element should be examined carefully to determine whether the operationality criteria imposed by the performance element include any restrictions that might prevent the effective application of the results of EBL. In our experiments, restrictions on matching limited the application of operators, severely hampering EBL.

This lesson can be generalized still further. In our experiments, the initial integration of the performance element and ROTE learning also turned out to be flawed. Instances of known solutions were added during learning, but in a manner that prevented them from contributing to finding earlier or better solutions. The experience of finding serious, unexpected flaws in our initial rote learning and explanation-based learning systems has convinced us that the coupling between learning and performance elements is critically important.

In general, one cannot simply add a learning method to a performance engine and assume that the results of learning will be used effectively. It is important to be aware of the potential for unforseen interactions between the learning and performance elements. Such interactions can prevent or interfere with the use of learned results. Worse, learned results can be a hindrance instead of facilitating performance.

Another lesson of this paper is that, even when they are deliberately simplified, machine learning systems are complex and tricky beasts deserving careful experiments and analysis. It is often not enough to do one experiment and produce graphs that indicate that learning method X is better than learning method Y. As we discovered, a number of causes typically underlie initial results. The interactions of the underlying causes can be complex and this can make initial data misleading. The goal of advancing machine learning research as rapidly as possible is best served by taking the additional steps necessary to determine *why* method X is better than Y.

Acknowledgements

Some of the results presented here were originally published in the Proceedings of the Fourth International Conference on Machine Learning [22]. The research presented here grew out of a discussion with Pat Langley at the Cognitive Science Conference held at the University of California, Irvine in the summer of 1985. Early work was carried out by the author when he was a member of Gerald DeJong's Explanation-Based Learning Research Group at the University of Illinois at Champaign-Urbana. Special thanks to Gerald DeJong and Scott Bennett at Illinois, to Ray Mooney (now at the University of Texas at Austin), and also to Pat Langley, Michael Pazzani, Tony Wieser, and Heping He at Irvine. Wieser provided a great deal of research assistance to the author, including reimplementation of the author's LISP systems in PROLOG, as well as having discovered that the IMPLIES restrictions forced the EBL LT's performance to deteriorate in the extended experiments first reported here. Heping He provided programming support and assisted in the analysis and presentation of the data. At the Machine Learning Journal, Jack Mostow and three anonymous reviewers contributed very useful suggestions for improvements. Thanks also to Bernhard Pfahringer of the Austrian Research Institute for AI, for finding several errors in our original translations of the Principia problems. This research was supported in part by the National Science Foundation under grant NSF IST 83-17889, by a Cognitive Science/AI Fellowship from the University of Illinois, and by a McDonnell-Douglas University External Relations contract.

Appendix 1: Axioms and Problems from Principia

This appendix contains the propositional logic axioms and problems used in experiments with non-learning, rote-learning and explanation-based learning versions of LT. The axioms corresponding to numbers 1.2 through 1.6 in Whitehead and Russell's *Principia Mathematica* are shown in Table 2. Note that Principia-1.1 is not included because it corresponds to a rule of inference, an operator in LT.

Table 2: Axioms from Principia

Designation in Principia	Axiom
1.2	$p \vee p \supset p$
1.3	$p \supset q \vee p$
1.4	$p \vee q \supset q \vee p$
1.5	$p \vee (q \vee r) \supset q \vee (p \vee r)$
1.6	$(p \supset q) \supset (r \vee p \supset r \vee q)$

The problems are shown in Table 3. They are the first 92 theorems from chapters two and three of part one of *Principia*. The axioms and problems are listed in an abbreviated format using operators for (in order of syntactic precedence) *logical equivalence*, *implication*, *disjunction*, *conjunction*, and *negation*. Machine readable versions of these (and other) *Principia* axioms and problems may be obtained by writing the author (preferably via electronic mail).

Table 3: Problems from Principia

Problem	Theorem	Problem	Theorem
2.01	$(p \supset \bar{p}) \supset \bar{p}$	2.6	$\bar{p} \supset (q \supset ((p \supset q) \supset q))$
2.02	$q \supset (p \supset q)$	2.61	$(p \supset q) \supset ((\bar{p} \supset q) \supset q)$
2.03	$(p \supset \bar{q}) \supset (q \supset \bar{p})$	2.62	$p \vee q \supset ((p \supset q) \supset q)$
2.04	$(p \supset (q \supset r)) \supset (q \supset (p \supset r))$	2.621	$(p \supset q) \supset (p \vee q \supset q)$
2.05	$(q \supset r) \supset ((p \supset q) \supset (p \supset r))$	2.63	$p \vee q \supset (\bar{p} \vee q \supset q)$
2.06	$(p \supset q) \supset ((q \supset r) \supset (p \supset r))$	2.64	$p \vee q \supset (p \vee \bar{q} \supset p)$
2.07	$p \supset p \vee p$	2.65	$(p \supset q) \supset ((p \supset \bar{q}) \supset \bar{p})$
2.08	$p \supset p$	2.67	$(p \vee q \supset q) \supset (p \supset q)$
2.1	$\bar{p} \vee p$	2.68	$((p \supset q) \supset q) \supset p \vee q$
2.11	$p \vee \bar{p}$	2.69	$((p \supset q) \supset q) \supset ((q \supset p) \supset p)$
2.12	$p \supset \bar{\bar{p}}$	2.73	$(p \supset q) \supset ((p \vee q) \vee r) \supset q \vee r)$
2.13	$p \vee \bar{\bar{\bar{p}}}$	2.74	$(q \supset p) \supset ((p \vee q) \vee r) \supset p \vee r)$
2.14	$\bar{\bar{p}} \supset p$	2.75	$p \vee q \supset (p \vee (q \supset r) \supset p \vee r)$
2.15	$(\bar{p} \supset q) \supset (\bar{q} \supset p)$	2.76	$p \vee (q \supset r) \supset (p \vee q \supset p \vee r)$
2.16	$(p \supset q) \supset (\bar{q} \supset \bar{p})$	2.77	$(p \supset (q \supset r)) \supset ((p \supset q) \supset (p \supset r))$
2.17	$(\bar{q} \supset \bar{p}) \supset (p \supset q)$	2.8	$q \vee r \supset (\bar{r} \vee s \supset q \vee s)$
2.18	$(\bar{p} \supset p) \supset p$	2.81	$(q \supset (r \supset s)) \supset (p \vee q \supset (p \vee r \supset p \vee s))$
2.2	$p \supset p \vee q$	2.82	$((p \vee q) \vee r) \supset (((p \vee \bar{r}) \vee s) \supset ((p \vee q) \vee s))$
2.21	$\bar{p} \supset (p \supset q)$	2.83	$(p \supset (q \supset r)) \supset ((p \supset (r \supset s)) \supset (p \supset (q \supset s)))$
2.24	$p \supset (\bar{p} \supset q)$	2.85	$(p \vee q \supset p \vee r) \supset p \vee (q \supset r)$
2.25	$p \vee (p \vee q \supset q)$	2.86	$((p \supset q) \supset (p \supset r)) \supset (p \supset (q \supset r))$
2.26	$\bar{p} \vee ((p \supset q) \supset q)$	3.1	$p \wedge q \supset \overline{\bar{p} \vee \bar{q}}$
2.27	$p \supset ((p \supset q) \supset q)$	3.11	$\overline{\bar{p} \vee \bar{q}} \supset p \wedge q$
2.3	$p \vee (q \vee r) \supset p \vee (r \vee q)$	3.12	$\bar{p} \vee (\bar{q} \vee (p \wedge q))$
2.31	$p \vee (q \vee r) \supset (p \vee q) \vee r$	3.13	$\overline{p \wedge q} \supset \bar{p} \vee \bar{q}$
2.32	$(p \vee q) \vee r \supset p \vee (q \vee r)$	3.14	$\bar{p} \vee \bar{q} \supset \overline{p \wedge q}$
2.36	$(q \supset r) \supset (p \vee q \supset r \vee p)$	3.2	$p \supset (q \supset p \wedge q)$
2.37	$(q \supset r) \supset (q \vee p \supset p \vee r)$	3.21	$q \supset (p \supset p \wedge q)$
2.38	$(q \supset r) \supset (q \vee p \supset r \vee p)$	3.22	$p \wedge q \supset q \wedge p$
2.4	$p \vee (p \vee q) \supset p \vee q$	3.24	$\overline{p \wedge \bar{p}}$
2.41	$q \vee (p \vee q) \supset p \vee q$	3.26	$p \wedge q \supset p$
2.42	$\bar{p} \vee (p \supset q) \supset (p \supset q)$	3.27	$p \wedge q \supset q$
2.43	$(p \supset (p \supset q)) \supset (p \supset q)$	3.3	$(p \wedge q \supset r) \supset (p \supset (q \supset r))$
2.45	$\overline{p \vee q} \supset \bar{p}$	3.31	$(p \supset (q \supset r)) \supset (p \wedge q \supset r)$
2.46	$\overline{p \vee q} \supset \bar{q}$	3.33	$(p \supset q) \wedge (q \supset r) \supset (p \supset r)$
2.47	$\overline{p \vee q} \supset \bar{p} \vee q$	3.34	$(q \supset r) \wedge (p \supset q) \supset (p \supset r)$
2.48	$\overline{p \vee q} \supset p \vee \bar{q}$	3.35	$p \wedge (p \supset q) \supset q$
2.49	$\overline{p \vee q} \supset \bar{p} \vee \bar{q}$	3.37	$(p \wedge q \supset r) \supset (p \wedge \bar{r} \supset \bar{q})$
2.5	$\overline{p \supset q} \supset (\bar{p} \supset q)$	3.4	$p \wedge q \supset (p \supset q)$
2.51	$\overline{p \supset q} \supset (p \supset \bar{q})$	3.41	$(p \supset r) \supset (p \wedge q \supset r)$
2.52	$\overline{p \supset q} \supset (\bar{p} \supset \bar{q})$	3.42	$(q \supset r) \supset (p \wedge q \supset r)$
2.521	$\overline{p \supset q} \supset (q \supset p)$	3.43	$(p \supset q) \wedge (p \supset r) \supset (p \supset q \wedge r)$
2.53	$p \vee q \supset (\bar{p} \supset q)$	3.44	$(q \supset p) \wedge (r \supset p) \supset (q \vee r \supset p)$
2.54	$(\bar{p} \supset q) \supset p \vee q$	3.45	$(p \supset q) \supset (p \wedge r \supset q \wedge r)$
2.55	$\bar{p} \supset (p \vee q \supset q)$	3.47	$(p \supset r) \wedge (q \supset s) \supset (p \wedge q \supset r \wedge s)$
2.56	$\bar{q} \supset (p \vee q \supset p)$	3.48	$(p \supset r) \wedge (q \supset s) \supset (p \vee q \supset r \vee s)$

Appendix 2: Examples of Improvements in Performance Due to EBL

The superior generality of EBL contributes to superior problem solving performance in two main ways. Sometimes it enables the problem solver to solve problems that could not be solved before. Alternatively, when both rote-learning and EBL systems solve a problem, the EBL solution is sometimes found more quickly and is sometimes simpler than that provided by rote-learning.

For an example of an EBL system solving more problems as a result of improved generality note that the EBL version of LT found a proof for Problem 17 (Principia-2.18) while the rote-learning version failed to find a proof in a small search (with the subproblems attempted limited to 15). The proof found by EBL involves the theorem learned from Problem 16 (Principia-2.17). The proof was obtained by chaining forward on the learned theorem and axiom Principia-1.2 (see Figure 20). The generalized conclusion of this proof of Principia-2.18 is $(\bar{A} \supset A) \supset A$. The proof is not constructed by rote-learning because of the extraneous NOT in Principia-2.17.

Chaining forward on the result of rote-learning on Principia-2.17 yields the less general conclusion $(\bar{\bar{A}} \supset \bar{A}) \supset \bar{A}$, (Figure 21). Principia-2.18 $(\bar{P} \supset P) \supset P$ is not an instance of this conclusion.

Examples of EBL constructing simpler solutions as a result of improved generality also occurred in the experiments (see Figure 22). While both learning versions of LT solve Problem 38 (Principia-2.49), the EBL version recognizes it as an instance of the class of problems solved by a previous solution while the rote-learning version has to regenerate that solution. The problem is $\overline{P \vee Q} \supset (\bar{P} \vee \bar{Q})$, an instance of the generalized conclusion of the proof of Problem 36 (Principia-2.47), namely $\overline{A \vee B} \supset (\bar{A} \vee C)$. This proof is not constructed during rote-learning because Principia-2.47 identifies B and C but Q and $\bar{Q}$ are incompatible. It turns out that in order to prove Principia-2.49, the rote-learning LT winds up having to prove the theorem that the explanation-based learning LT extracted from Principia-2.47. That is, it

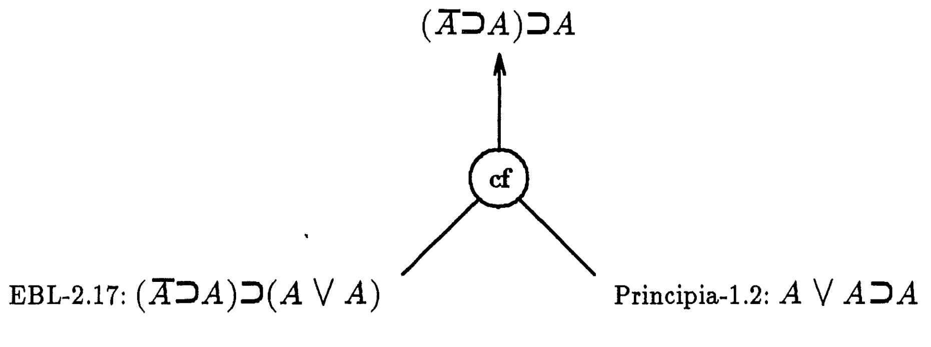
$(\bar{A} \supset A) \supset A$
cf
EBL-2.17: $(\bar{A} \supset A) \supset (A \vee A)$
Principia-1.2: $A \vee A \supset A$

Figure 20. The Proof of Principia-2.18

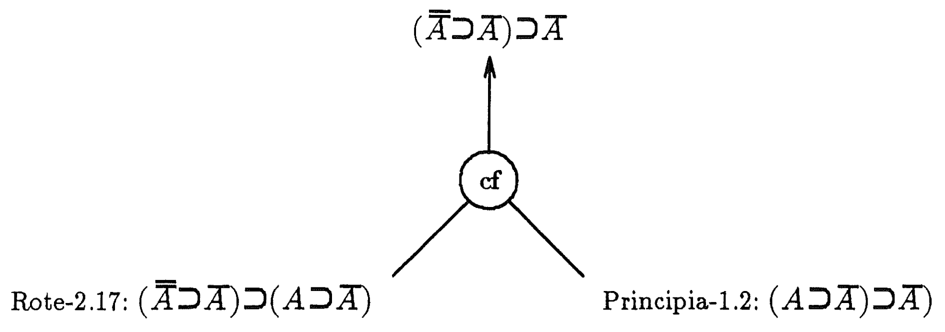
$(\bar{\bar{A}}\supset\bar{A})\supset\bar{A}$
cf
Rote-2.17: $(\bar{\bar{A}}\supset\bar{A})\supset(A\supset\bar{A})$
Principia-1.2: $(A\supset\bar{A})\supset\bar{A})$

Figure 21. Inferiority of Rote Learning on Principia-2.18

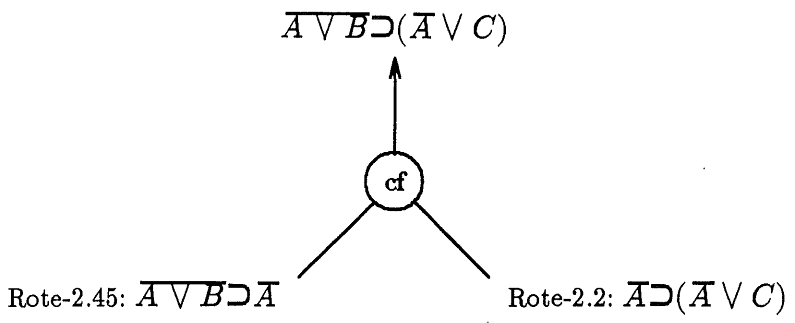
$\overline{A \vee \bar{B}}\supset(\bar{A} \vee C)$
cf
Rote-2.45: $\overline{A \vee \bar{B}}\supset\bar{A}$
Rote-2.2: $\bar{A}\supset(\bar{A} \vee C)$

Figure 22. Rote-Learning Proof of Principia-2.47 and 2.49

regenerates the same proof that it used before on Principia-2.47 because it failed to learn all it could from this proof. This is a very clear case of inferior performance on the part of rote learning due to the fact that it simply stores problems instead of generalizing and computing the class of problems that a novel solution solves.

REFERENCES

1. E. Charniak and D. McDermott, *Introduction to Artificial Intelligence Programming*, Lawrence Erlbaum Associates, Hillsdale, N.J., 1986.

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

3. G. F. DeJong, "An Approach to Learning from Observation," in *Machine Learning: An Artificial Intelligence Approach, Vol.* 2, Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell (ed.), Morgan Kaufmann, Los Altos, California, 1986, 571-590.

4. G. F. DeJong (Ed.), Proceedings of the American Association for Artificial Intelligence Spring Symposium on Explanation-Based Learning, , March 22-24, 1988.

5. H. B. Enderton, *A Mathematical Introduction to Logic*, Academic Press, Inc., New York, 1972.

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

7. S. T. Kedar-Cabelli and L. T. McCarty, "Explanation-Based Generalization as Resolution Theorem Proving," *Proceedings of the Fourth International Workshop on Machine Learning*, Los Altos, CA, June 1987, 383-389.

8. K. Knight, "Unification: A Multidisciplinary Survey," *Association for Computing Machinery Computing Surveys* 21, 1 (March 1989), 93-124.

9. R. E. Korf, *Learning to Solve Problems by Searching for Macro-Operators*, Morgan Kaufmann Publishers, Inc., Palo Alto, CA, 1985.

10. R. E. Korf, "Macro-Operators: A Weak Method for Learning," *Artificial Intelligence* 26, 1 (April 1985), 35-77.

11. R. E. Korf, "Search: A Survey of Recent Results," in *Exploring Artificial Intelligence: Survey Talks from the National Conferences on Artificial Intelligence*, Howard E. Shrobe (ed.), Morgan Kaufmann, San Mateo, CA, 1988, 197-237.

12. D. W. Loveland, "Automated Theorem Proving: A Quarter-Century Review," in *Proceedings of the Special Session on Automatic Theorem Proving at the American Math. Soc. Annual Meeting*, Denver, Colorado, January 1983.

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

14. S. Minton, "Quantitative Results Concerning the Utility of Explanation-Based Learning," *Proceedings of the Seventh National Conference on Artificial Intelligence*, Saint Paul, Minnesota, August 1988, 564-569.

15. S. Minton, "Learning Effective Search Control Knowledge: An Explanation-Based Approach," Ph.D. Thesis Carnegie Mellon University, Computer Science Department Carnegie Mellon University, Pittsburgh, PA, March 1988.

16. T. M. Mitchell, R. M. Keller and S. T. Kedar-Cabelli, "Explanation-Based Generalization — A Unifying View," *Machine Learning* 1, 1 (1986), 47-80, Kluwer Academic Publishers.

17. R. Mooney and S. Bennett, "A Domain Independent Explanation-Based Generalizer," *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, Morgan Kaufmann Publishers, Inc..

18. R. J. Mooney, "A General Explanation-Based Learning Mechanism and its Application to Narrative Understanding," UILU-Engineering-87-2269, Coordinated Science Laboratory University of Illinois at Urbana-Champaign, Urbana IL, December 1987.

19. A. Newell and H. A. Simon, "The Logic Theorist: An Example," in *Human Problem Solving*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1972, 105-140.

20. 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*, Edward Feigenbaum and Julian Feldman (ed.), Robert E. Krieger Publishing Company, Inc., Malabar, Florida, 1981, 109-133. (This article was also published in the *Proceedings of the Joint Computer Conference* (pp 218-230) in 1957. The original edition of *Computers and Thought* was published in 1963 by McGraw-Hill, Inc.)

21. P. O'Rorke, "Generalization for Explanation-Based Schema Acquisition," *Proceedings of the National Conference on Artificial Intelligence*, Austin, TX, August 1984, 260-263.

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

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

24. J. C. Reynolds, "Transformational Systems and the Algebraic Structure of Atomic Formulas," in *Machine Intelligence* 5, Bernard Meltzer and Donald Michie (ed.), Edinburgh University Press/ American Elsevier, New York, 1970, 135-151.

25. J. A. Robinson, "A Machine-Oriented Logic Based on the Resolution Principle," *Journal of the Association for Computing Machinery* 12, 1 (January 1965), 23-41.

26. P. S. Rosenbloom and J. E. Laird, "Mapping Explanation-Based Generalization onto SOAR," *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, Morgan Kaufmann Publishers, Inc..

27. A. M. Segre, "Explanation-Based Learning of Generalized Robot Assembly Tasks," UILU-Engineering-87-2208, Coordinated Science Laboratory University of Illinois at Urbana-Champaign, Urbana IL, January 1987.

28. A. M. Segre, *Machine Learning of Robot Assembly Plans*, Kluwer Academic Publishers, Dordrecht, the Netherlands, 1988.

29. J. W. Shavlik, "Generalizing the Structure of Explanations in Explanation-Based Learning," Ph.D. thesis, Coordinated Science Laboratory University of Illinois at Urbana-Champaign, Urbana IL, January 1988.

30. E. Stefferud, "The Logic Theory Machine: A Model Heuristic Program," Memorandum RM-3731-CC, The Rand Corporation, Santa Monica, CA, June, 1963.

31. A. N. Whitehead and B. Russell, *Principia Mathematica*, Cambridge University Press, London, 1913. (also available in a paperback edition to *56, published in 1962)