Machine Learning, 6, 183–195 (1991)

Technical Note
A Critical Look at Experimental Evaluations of EBL

ALBERTO SEGRE (SEGRE@CS.CORNELL.EDU)
Department of Computer Science, Cornell University, Ithaca, NY 14853

CHARLES ELKAN (ELKAN@CS.UCSD.EDU)
Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA 92093

ALEXANDER RUSSELL (ARUSSELL@CS.CORNELL.EDU)
Department of Computer Science, Cornell University, Ithaca, NY 14853

Editor: Jaime Carbonell

Abstract. A number of experimental evaluations of *explanation-based learning* (EBL) have been reported in the literature on machine learning. A close examination of the design of these experiments reveals certain methodological problems that could affect the conclusions drawn from the experiments. This article analyzes some of the more common methodological difficulties, and illustrates them using selected previous studies.

Keywords. Explanation-based learning, speedup learning, performance measures

1. Introduction

A number of experimental evaluations of *explanation-based learning* (EBL) (DeJong & Mooney, 1986; Mitchell, et al., 1986) have appeared in the machine learning literature (Eskey & Zweben, 1990; Etzioni, 1990; Knoblock, 1990; Markovitch & Scott, 1988; Minton, 1990; Mooney, 1989; O'Rorke, 1989; Shavlik, 1990; Tambe & Newell, 1988). These studies measure the performance of a learning system against the performance of a similar non-learning system. Performance is improved if the learning system can solve more problems or if similar problems are solvable after learning more efficiently than before learning. Often performance is improved in both ways.

How much can conclusions based on experimental observations be trusted? In principle, when the relevant conditions of an experiment are replicated, similar results should be observed. In practice, our confidence depends on how carefully the experiment is designed—in short, our confidence depends on sound *experimental methodology*. Close examination of experimental designs used in the past reveals certain methodological problems that can introduce unwanted experimental bias.

The general idea behind EBL is to use traces of previous problem solving activity to alter the search space explored when solving future problems. In some systems (e.g., ARMS (Segre, 1988), GENESIS (Mooney, 1990), and BAGGER (Shavlik, 1990)), EBL is used to acquire problem space macro-operators that alter the search space by compressing generalizations of previously useful subproofs into more efficiently applicable proof idioms. In these systems EBL is essentially adding redundant operators which, when integrated with

the existing operators, bias the exploration of the search space. Acquired macro-operators may lead to quick solutions, but in some circumstances an acquired macro-operator may delay the discovery of a goal node.

Other EBL systems (e.g., LEX2 (Mitchell, et al., 1983) and PRODIGY (Minton, 1990)) acquire heuristics for controlling the application of existing problem space operators.[1] These heuristics typically alter the ordering in which alternative choices are attempted. Some heuristics may reject certain operators outright, while others may select a particular operator as especially suitable in the current situation, implicitly downgrading all other applicable operators. As in the macro-operator systems, while search-control heuristics should contribute to a quicker solution of a future goal, the cost of evaluating the heuristics can slow down the search instead.

Experimental evaluations of EBL attempt to draw conclusions about changes in the performance of a problem solving system by measuring a predefined performance metric over several trials. Each trial consists of solving selected problems using a different version of the system. The objective of any experiment of this type is to produce results that can be used to predict the future behavior of the different versions of the system. The purpose of this paper is to discuss some of the more common methodological difficulties that can limit the usefulness of the experimental results.

2. Resource limit bias

Past reports on EBL experiments commonly conclude that the *utility problem* is significant (Minton, 1990; Mooney, 1989; Tambe & Newell, 1988). This problem is that the addition of an EBL component can impose a performance penalty on a problem solver. Inspired by experimental results, the utility problem has also been studied using analytic models (Greiner & Likuski, 1989).

Questions other than the utility problem can also be addressed experimentally. In particular, experiments have compared different generalization strategies (O'Rorke, 1989), protocols for when to apply learning and how to use learned knowledge (Mooney, 1989), methods for learning recursive concepts (Etzioni, 1990; Shavlik, 1990; Subramanian & Feldman, 1990), the effectiveness of abstraction hierarchies in problem solving (Knoblock, 1990), and whether learned knowledge should eventually be overwritten or discarded (Markovitch & Scott, 1988).

Traditional EBL cannot endow a problem solver with new domain knowledge. Rather, EBL improves a problem solver's performance by modifying how the search space implicitly defined by the existing domain knowledge is explored. EBL adapts a problem solver to a particular distribution of problems, reducing the time needed to solve problems similar to those in the training set. If the problem solver is already capable of solving most problems presented to it, then EBL can make it faster. If the resource limit imposed on the problem solver—whether explicit or implicit—is too low to allow most problems to be solved initially, then EBL can increase the range of soluble problems. Regardless of which type of performance improvement is being tested, the manner in which a resource bound is imposed on a problem solver can adversely influence the reliability of experimental conclusions.

2.1. Setting resource limits

Consider a hypothetical experiment comparing two versions of the same problem solver, where the second version is augmented with an EBL component. The experiment meters the performance of the two problem solvers on the same sequence of five problems over two trials. For each attempt at each problem, we record whether or not the problem is solved and if so, the time to obtain a solution. No learning takes place in the first trial; it is used as a baseline. In the second trial, the problem solver is allowed to apply its EBL component to each successful solution—the newly acquired knowledge is then available to the problem solver when solving subsequent problems in the sequence.

In this hypothetical experiment, suppose we impose a resource limit of one CPU second per problem on the problem solver and obtain the data shown in Table 1.[2] It is an experimenter's responsibility to summarize data (perhaps through the use of graphs or charts) in a way that generates understanding that can be carried over to the design of similar systems. Unfortunately, some of the analysis techniques used in previous experiments do not lead to results that can be reliably extrapolated. The following replication of some previous analyses taken from the machine learning literature shows how it is possible to reach unfounded conclusions about the utility problem.

The simplest way to summarize the data in Table 1 is to sum the amount of time used by each system for all five problems. The non-learning system consumes 2.5 CPU seconds, while the EBL system requires 2.975 CPU seconds. By this measure, using EBL entails a 19% performance penalty. We might even plot cumulative solution time against problem number as shown in Figure 1.

Table 1. Experiment 1.

Problem	1	2	3	4	5	Total
No Learning	100	200	300	900	(1000)	2500
EBL	100	275	600	(1000)	(1000)	2975

All values are CPU-milliseconds; parentheses indicate unsolved problems.

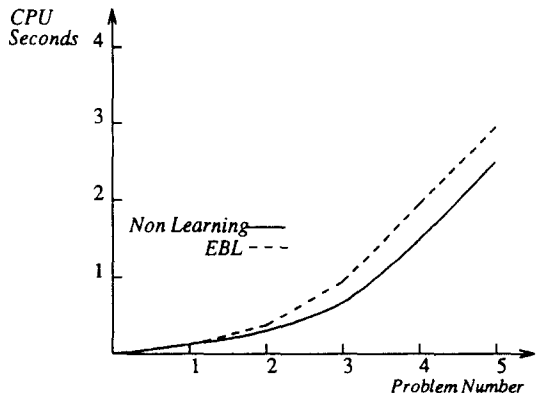

Figure 1. Cumulative time to solution.

Table 2. Experiment 2.

Problem	1	2	3	4	5	Total
No Learning	100	200	300	900	(1500)	3000
EBL	100	275	600	(1500)	1001	3476

All values are CPU-milliseconds; parentheses indicate unsolved problems.

Table 3. Experiment 3.

Problem	1	2	3	4	5	Total
No Learning	100	200	300	900	(3000)	4500
EBL	100	275	600	1560	1078	3613

All values are CPU-milliseconds; parentheses indicate unsolved problems.

Figure 1 appears quite convincing; it seems that our experiment confirms the presence of the utility problem in this domain. Is this conclusion justified? Perhaps not; let us repeat the experiment, extending the resource bound to 1.5 CPU seconds. The data for this set of trials is shown in Table 2. Now each system is able to solve four of the five problems. Comparing total CPU usage, we see the EBL system still carries a performance penalty of about 16%. It appears the predictions made previously are confirmed.

However, consider what happens when the resource limit is increased again, to three CPU seconds (see Table 3).

Now the EBL system can solve all five problems. The time to solve Problem 5 has increased slightly, presumably due to learning after solving Problem 4. Nonetheless, the non-learning system consumes a total of 4.5 CPU seconds, while the EBL system consumes only 3.613 CPU seconds; a net performance *improvement* for EBL of almost 20%.

What is the methodological flaw that engenders unreliable conclusions? Comparing cumulative resource use produces unfounded predictions, since this performance figure is dependent on the resource limit imposed. As the resource limit is increased, the apparent improvement due to EBL will also change, depending on the number and distribution of unsolved problems in each trial. Since resource limits, whether explicit or implicit, are unavoidable in empirical work, we need to find another, more reliable, method to analyze our experimental results.

2.2. Controlling for correctness

An alternative analysis procedure is to *control for correctness*. Using only those problems solved by both systems in the analysis, we ignore Problems 4 and 5 in Table 1 and obtain cumulative time values of 0.6 CPU seconds for the non-learning system and 0.975 CPU seconds for the EBL system, a 62% performance degradation for the EBL system. Table 2 yields the same figures, while Table 3 yields 1.5 and 2.535 CPU seconds for Problems 1 through 4: a 69% performance degradation penalty. In all cases, we observe a significant deterioration in performance for the EBL system.

Table 4. Experiment 4.

Problem	1	2	3	4	5	Total
No Learning	100	200	300	900	6000	7500
EBL	100	275	600	1560	1078	3613

All values are CPU-milliseconds; parentheses indicate unsolved problems.

An analysis that excludes unsolved problems is often stable across resource limits, but it is inherently biased against a learning system. EBL changes the *resource-limited competence* of a problem solver—i.e., the population of problems which can be solved within a given resource bound (sometimes termed the *resource-limited deductive closure*). Often performance decreases slightly on problems that can be solved without learning; this negative effect is, one hopes, outweighed by the usefulness of solving additional problems. If the analysis is restricted to problems that can be solved without learning, only the negative effect is likely to be observed. The greatest benefit of EBL often lies in solving problems outside the reach of a resource-limited non-learning system—precisely those problems that are excluded when controlling for correctness.

Consider extending the resource limit for our hypothetical experiment once again, this time by a margin large enough that the non-learning system can solve all five problems (Table 4). The non-learning system requires a total of 7.5 CPU seconds, while the EBL system only consumes 3.613 CPU seconds; an overall net performance improvement of almost 52% for the EBL system.

The series of examples above illustrates how an apparently minor aspect of an experiment (the use of a resource limit) can cause apparently reasonable data analysis techniques (cumulative solution time comparisons and, optionally, controlling for correctness) to produce unsubstantiated conclusions.

3. Performance metric bias

The ideal measure of the performance of a problem solver would count the number of times some unit-cost operation is executed, where the cost of this operation is the same after learning.

Traditional CPU time measurements tie an experiment to fixed hardware, as well as to aspects of a particular software implementation that have little to do with learning. While CPU time measurements are easy to obtain, there is a risk of introducing hidden bias. As an example, consider the results obtained when repeating our hypothetical experiment of Section 2 using an identical resource limit of 2 CPU seconds on the same problem solver architecture implemented in three slightly different ways (Table 5). The differing values in Table 5 arise from the use of three different indexing algorithms. The data for EBL1 assume constant-time indexing, for EBL2 a logarithmic time (in the number of database entries) indexing scheme, and for EBL3 (our original problem solver) a linear-time indexing strategy. While constant-time indexing is optimistic, logarithmic time indexing schemes are in common use. If we measure CPU time used by a problem solver, we are measuring

Table 5. Experiment 5.

Problem	1	2	3	4	5	Total
No Learning	100	200	300	900	(2000)	3500
EBL1	100	250	500	1200	770	2820
EBL2	100	260	540	1336	882	3118
EBL3	100	275	600	1560	1078	3613

All values are CPU-milliseconds; parentheses indicate unsolved problems.

not only the effect of macro-operator learning, but also the effect of low-level implementation choices. For EBL3 there is a bias *against* macro-operator learning, as EBL3 pays a premium price (linear vs. logarithmic) for each rule it adds to the domain theory.[3]

The bias due to an implementation's indexing scheme might be eliminated by using a different performance metric. Either the number of nodes expanded or the maximum depth of search could be used as a solution cost metrics. Unfortunately, these measures usually impart an unfair bias *in favor* of EBL. Expanding a node involves generating an ordered set of child nodes, an operation which can be expected to become more expensive with learning. Macro-operator learning systems will tend to increase the average number of child nodes, while systems that learn search control heuristics can be expected to take more time evaluating acquired heuristics when ordering the candidate nodes. Since expanding a node is not generally a constant time operation, the time required to search to a fixed depth or to expand a fixed number of nodes will increase. Thus a system incorporating an EBL component will consume significantly more CPU time than a non-learning system when expanding a like number of nodes (or searching to the same depth). A good performance metric must be impartial to the learning procedure used.

In summary, the problem of finding an adequate measure of resource use for evaluating EBL is not an easy one. Nonetheless, some performance metrics are less problematic than others, and it is the responsibility of an experimenter to select the most appropriate measure for his or her particular problem solving system: a performance metric that measures constant time operations in an attempt to reduce experimental bias.

4. Previous experiments and their methodologies

In Sections 2 and 3, we discussed how interactions between data analysis techniques, problem solver implementation, and experimental design could introduce experimental bias. In this section, we apply insights gained from this discussion to some previous experimental studies, and derive some general lessons about collecting useful performance data and analyzing it reliably.

Among the most complete analyses of the performance contribution of an EBL component to a problem solving system is O'Rorke's work (1987, 1989) based on a reimplementation of the early *Logic Theorist* (LT) system (Newell, et al., 1963). In these studies, a theorem prover with an EBL component was compared against non-learning and rote learning versions of the same prover on an ordered population of problems drawn from Russell and Whitehead's *Principia Mathematica* (Whitehead & Russell, 1913). For historical reasons, O'Rorke's study relies on what by today's standards is a quirky, linearly-recursive, breadth-

first theorem-proving architecture. While the impoverished design of the LT prover makes studying the effects of learning difficult, it has the advantage of being well specified; for this reason the majority of the comments below concern the LT studies, although the points made are equally applicable to other studies.

The critical choices in experimental design concern the resource limit, the performance metric, the domain theory, the problem set, the learning algorithm, and the learning protocol.

4.1. Resource limit

Resource limits are usually imposed on the same parameter that is used as the performance metric. However, some experiments have separated these two notions, while other experiments have imposed multiple resource limits (e.g., limits on both depth and nodes expanded, or on both CPU time and depth). Multiple resource limits usually arise from a need for search control. A depth limit forces a simple depth-first prover always to backtrack evenutally, thus guaranteeing termination. Using more than one resource limit makes results difficult to interpret; a better way to obtain completeness for depth-first search strategies is to use iterative deepening (Korf, 1985).

Resource limits are typically expressed in terms of CPU time, search depth, number of nodes expanded, or number of nodes generated in the search. Many of the same problems caused by using any of these parameters as a performance measure can also arise when using the parameter as a resource limit. As shown above, using search depth or number of nodes expanded as a performance metric typically introduces a bias in favor of a learning system, and a similar bias is introduced if either is used as a resource limit. Resource limits based on elapsed CPU time introduce the same extraneous implementation dependencies as CPU-based performance measures.

Imposing a limit on the number of nodes generated may impart a bias against learning systems. To understand why this is so, consider that a learning system is likely to generate more nodes (while still possibly expanding fewer nodes) than its non-learning counterparts. An EBL system that acquires perfect search control heuristics (where perfect heuristics result in an optimally directed search) might generate an arbitrary number of nodes while expanding only those nodes that lie directly on the solution path. Macro-operator learning systems suffer an analogous problem, since they may generate a large number of nodes while perhaps expanding only the nodes on the solution path.

In O'Rorke's experiments, each attempt to solve a problem is limited to a certain number of nodes generated. Normally this practice would entail a bias against EBL. However, the LT prover handles node expansion in a non-standard fashion, attempting to match child nodes to facts in the database as they are generated, rather than when they are themselves expanded later. Since the LT learning components acquire only new database facts, the cost of generating a node grows with learning. This implies a bias in favor of the EBL prover, since it is charged a fixed cost for what is actually a growing cost operation. These conflicting sources of bias make evaluation difficult.

In summary, a good resource limit should use constant time operations as its basic unit. Exactly what is or is not a constant time operation depends on the implementation of the underlying problem solver. Once a resource limit has been selected, the experimenter must be careful to avoid using analytic techniques that are sensitive to particular resource limit values.

4.2. Performance metric

The problem of choosing a metric to measure the amount of work performed by the problem solver is closely related to the problem of imposing a resource limit, with the added complexity that the performance metric must give comparable values over different versions of a problem solver and different populations of problems. Previous experimental evaluations of EBL have either held problem population constant across trials, included the cost of failed problems in the analysis, or simply ignored the issue altogether. None of these solutions are adequate.

As discussed in Section 2, controlling for correctness (totaling resource usage for successfully solved problems only) leads to a bias against learning systems. Including the cost of failed problems—i.e., incrementing the total by the resource limit for each failed problem—causes sensitivity to the initial resource bound. Allowing the problem population to differ across trials also usually results in a bias against EBL, since those problems solved by EBL which could not be solved by a non-learning system are often more difficult. As an example, consider again the hypothetical results of Tables 1 through 3. The inherent difficulty of Problem 5 (as measured by the cost of the eventual non-learning solution in Table 4) exceeds the total difficulty of Problems 1 through 4, yet Problem 5 is excluded from the analyses. Thus even an *average CPU seconds per successful solution* performance measure unfairly shows an increase in problem solving time for EBL.

O'Rorke's study takes a step in the right direction by using a so-called *average branching factor* metric (this metric is defined as the quotient of the number of nodes generated over the number of nodes expanded by each version of his problem solver). There are two problems with this approach. First, since some of the work usually performed at node-expansion time (i.e., unification with facts in the database) is frontloaded onto node generation in the LT prover, neither node expansion nor generation are fixed-cost operations. The correlation between time usage and this approximation of average branching factor is therefore unclear. Second (and most important), the denominator reflects the size of the space explored by the current problem solver and domain theory rather than the size of the space explored by a control system. This makes direct comparisons of average branching factor across problem solvers difficult.

As in the selection of a resource limit, the experimenter must adopt a performance metric whose basic unit is an implementation-independent constant time operation for the underlying problem solver. Once a performance metric is selected, the experimenter must carefully choose an unbiased procedure for analyzing data collected from trials involving different versions of the problem solver and different populations of solved problems.

4.3. Domain theory and problem set

One of the hardest problems in designing an experiment to test the usefulness of EBL is finding an adequately large corpus of problems that can be solved by a suitable domain theory. The requirements for a domain theory and problem set are necessarily vague. It seems clear that a reasonably large set of non-trivial problems is required; the problems may be randomly ordered, or placed by a teacher in a sequence intended to facilitate learning.

Some systems may improve their performance in the course of solving a single problem (by learning from solutions to subproblems), while others may learn only from problem to problem. The question of optimal problem ordering and of when to learn from subproblems has not yet been studied experimentally. Theoretical results on the difficulty of learning from random problems (Eisenberg & Rivest, 1990; Valiant, 1984) suggest that ordering can be crucial.

O'Rorke's reconstruction of the original LT experiment applies EBL to a problem set consisting of 52 propositional calculus problems drawn in their original ordering (easiest problems first) from Chapters 2 and 3 of *Principia Mathematica*.[4] Mooney (1989) repeats the experiment using a more modern performance-engine architecture. The LT domain has at least two clear advantages. First, there are many solvable problems with a relatively small (only 2 rules and 5 facts) domain theory. Second, the problems of *Principia Mathematica* were written for human consumption, and not devised with automated theorem proving or machine learning in mind. Certainly no one can claim the problem set was intentionally biased in favor of any problem solver or learning strategy.

Do conclusions based on experiments in the LT domain apply to other domains as well? Unfortunately, not all domains are equally well-suited to EBL; the LT domain only supports a specialized form of macro-operator learning that we call *generalized caching*. Since LT involves constructing proofs of properties of propositions and since there are no semantic differences between syntactically different propositions (e.g., P, Q, $\neg R$, etc.), any statement about a *particular* set of objects is always true of *every* possible set of objects. For this reason, macro-operator learning is inappropriate; instead, generalized versions of previously proven propositions are directly cached as new database facts. A sterile domain such as LT, where every object is exactly like any other object, gives EBL algorithms little room to outperform rote learning.[5]

In summary, the problem of finding adequate domains for testing EBL is still open. The LT domain forms in some sense a worst-case environment for EBL; conclusions based on experiments in the LT domain may have little bearing on EBL applications to other, semantically richer, domains.

4.4. Learning algorithm and protocol

To this day, most experiments have compared a particular learning system with a non-learning system. But not all EBL systems are equal; whether EBL is being used to derive new macro-operators or to build search-control heuristics, the effectiveness of EBL is critically dependent on any operationality pruning performed on the original proof (Elkan & Segre, 1989; Segre, 1987). Existing experiments document the effect of a particular EBL algorithm with a particular operationality criterion; a different algorithm might display dramatically different results.

Another parameter which must be taken into account is the learning protocol. This protocol determines how examples are presented and when learning actually occurs. Learning is typically applied to successful problem solutions only; however, in some experiments, unsuccessful problems are entered as facts in the database, perhaps as generalized by a rote learning algorithm.[6] Not only does rote learning of unproven (and, therefore, possibly untrue) propositions seem an unusual method of augmenting a domain theory, it makes the performance contribution of a particular learning algorithm difficult to identify.

A slightly different learning protocol has been used in other experiments. In these experiments, problems are divided into separate training and test runs; the system is allowed to learn only from training problems, while its performance is recorded only during test problems (Minton, 1990; Shavlik, 1990). This has two advantages; first, any dependency on problem ordering during the test runs is eliminated. Second, measurements taken during the test run correspond to a fixed problem solver and domain theory, rather than continuously changing versions as is the case when the results of learning are available to the problem solver in subsequent problems within the same trial.

Other changes in learning protocol include learning only from problems of greater than a certain difficulty. A special case of this heuristic is to learn only from problems that cannot be solved by instantiating domain-theory facts. Rote learning from problems harder than a fixed threshold is exactly the caching strategy already in use by some theorem provers (Elkan, 1989). Yet another difference in learning protocol occurs when constraints are placed on the use and management of learned knowledge. For example, Mooney's reconstruction of O'Rorke's LT experiments imposes a chaining constraint on learned rules or macro-operators.

The choice of learning protocol, like the expected performance gain due to EBL, is critically dependent on the problem distribution. If each new problem is completely unrelated to the preceding problems, then EBL cannot be useful. On the other hand, if only a finite number of problems are ever posed, then an optimal protocol is simply to cache solutions as they are generated for later recall. Typically, EBL systems operate in an environment somewhere between these two extremes. One might assume that problems are chosen according to some fixed probability distribution, which is unknown but invariant over time. Alternatively, one might use EBL to track a slowly changing distribution of problems over time. For this purpose, a system could manage learned rules as a fixed-size cache with various rule-management strategies. New learned rules would then cause previously learned rules to be removed from the domain theory.

In summary, whatever conclusions are drawn from experimental data are critically dependent on the particular EBL algorithm used and the learning protocol followed. It is not always clear that any generalization across protocols or algorithms is possible, although this should remain our research goal.

5. Conclusion

The message of this paper is that in order to support reliable extrapolation of experimental conclusions, experiments comparing different problem solvers must be designed with care. Previous experimenters have often made design decisions that can lead to unreliable conclusions. We do not claim that conclusions drawn from earlier experimental studies are necessarily wrong. We do claim, however, that previous studies have relied on experimental methodologies that are capable of leading to unsound conclusions.

Using a series of examples, we have illustrated how data obtained from the same problem solver, using the same learning algorithm and protocol, operating on identical problem sets, can show both a performance penalty and a performance improvement for EBL. The examples focus on three common methodological pitfalls: the effect of arbitrary resource

limits on cumulative solution time metrics, controlling for correctness as an analytical device, and the selection of inappropriate performance metrics for metering the effects of learning on problem solving. We have also described how these difficulties affect previous studies.

Many of the methodological difficulties described in this paper do not apply only to testing EBL systems, but also to empirical testing of other types of problem solving systems. A survey of the literature in logic programming, search, and automated theorem proving reveals that these communities are also struggling (Bancilhon & Ramakrishnan, 1988; Saletore & Kale, 1989; Warren, et al., 1984) to find adequate performance metrics for benchmarking systems whose search spaces are not static, such as when comparing multiprocessor and serial PROLOG implementations.

In any area of science, the conclusions drawn from an experiment depend on various assumptions. Those assumptions not tested in the experiment place limits on the predictive power of the conclusions. In designing experiments, the choice of what assumptions are left untested must depend on the directions in which one wants to extrapolate the experimental results. In other words, for empirical results to inspire justified confidence, they must be obtained following an experimental methodology designed with extrapolation in mind.

In (Segre, et al., 1990), we propose a simple yet robust methodology for experimental validations of EBL. The underlying assumptions are made explicit; the basic one is a model of problem solving as search. Our methodology is designed to produce reliable conclusions about the behavior of a given problem solver on large problems based on data about its behavior on small problems, permitting methodologically sound comparisons of different learning algorithms operating with the same problem solver, domain theory and problem population. The process of designing an experiment and analyzing the data collected within this methodological framework is illustrated with a sample reconstruction of the LT experiments. We hope that the analysis of earlier experiments presented in this paper will prompt others to take a careful look at their own experimental methodology as they continue to experiment with learning and problem solving.

Acknowledgments

We thank Jaime Carbonell, Gerry DeJong, Oren Etzioni, Craig Knoblock, Steve Minton, Ray Mooney and Paul O'Rorke for their comments on earlier drafts of this paper. Support for this research was provided by the Office of Naval Research through grant N00014-90-J-1542.

Notes

1. We use the term *search-control heuristic* to describe all learned search-control knowledge, regardless of whether or not it is deductively sound.
2. For expository purposes, the data shown in Table 1 is artificial. Real data from our reconstruction of the LT experiments (Mooney, 1989; O'Rorke, 1987; O'Rorke, 1989) is given in (Segre, et al., 1990). At least one subset of that experiment's data exhibits the same behavior as our artificial data under appropriate experimental conditions.
3. A similar phenomenon occurs when learning search control heuristics, as systems that acquire large numbers of heuristics can take advantage of clever indexing schemes for the heuristics.

4. In a later version of this same study, O'Rorke extends the problem set to a total of 92 problems from *Principia Mathematica*.
5. Since the LT domain theory consists of only two operators, learning search control heuristics could hardly fare any better.
6. This rather strange experimental procedure is a holdover from the original *Logic Theorist* work.

References

Bancilhon, F., & Ramakrishnan, R. (1988). Performance evaluation of data intensive logic programs. In J. Minker (Ed.), *Deductive databases and logic programming*. San Mateo, CA: Morgan Kaufmann Publishers.

DeJong, G., & Mooney, R. (1986). Explanation-based learning: An alternative view, *Machine Learning*, *1*, 145–176.

Eisenberg, G., & Rivest, R. (1990). On the sample complexity of PAC-learning using random and chosen examples. *Proceedings of the Third Workshop on Computational Learning Theory*. Rochester, NY.

Elkan, C. (1989). Conspiracy numbers and caching for searching and/or trees and theorem-proving. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 341–346). Detroit, MI.

Elkan, C., & Segre, A.M. (1989). *Not the last word on EBL algorithms* (Technical Report 89-1010). Ithaca, NY: Cornell University, Department of Computer Science.

Eskey, M., & Zweben, M. (1990). Learning search control for constraint-based scheduling. *Proceedings of the National Conference on Artificial Intelligence* (pp. 908–915). Boston, MA.

Etzioni, O. (1990). Why PRODIGY/EBL works. *Proceedings of the National Conference on Artificial Intelligence* (pp. 916–922). Boston, MA.

Greiner, R., & Likuski, J. (1989). Incorporating redundant learned rules: A preliminary formal analysis of EBL. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 744–749). Detroit, MI.

Knoblock, C.A. (1990). Learning abstraction hierarchies for problem solving. *Proceedings of the National Conference on Artificial Intelligence* (pp. 923–928). Boston, MA.

Korf, R. (1985). Depth-first iterative deepening: An optimal admissible tree search. *Artificial Intelligence*, *27*, 97–109.

Markovitch, S., & Scott, P.D. (1988). The role of forgetting in learning. *Proceedings of the Fifth International Machine Learning Conference* (pp. 459–465). Ann Arbor, MI.

Minton, S. (1990). *Learning search control knowledge*. Hingham, MA: Kluwer Academic Publishers.

Minton, S. (1990). Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence*, *42*, 363–392.

Mitchell, T.M., Utgoff, P.E., & Banerji, R. (1983). Learning by experimentation: Acquiring and refining problem-solving heuristics. In R.S. Michalski, J.G. Carbonell, and & T.M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 1). San Mateo, CA: Morgan Kaufmann Publishers.

Mitchell, T.M., Keller, R.M., & Kedar-Cabelli, S.T. (1986). Explanation-based generalization: A unifying view. Machine Learning, *1*, 47–80.

Mooney, R. (1989). The effect of rule use on the utility of explanation-based learning. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 725–730). Detroit, MI.

Mooney, R. (1990). *A general explanation-based learning mechanism*. San Mateo, CA: Morgan Kaufmann Publishers.

Newell, A., Shaw, J.C., & Simon, H. (1963). Empirical explorations with the logic theory machine: A case study in heuristics. In E. Feigenbaum & J. Feldman (Eds.), *Computers and Thought*. New York, NY: McGraw-Hill.

O'Rorke, P. (1987). LT revisited: Experimental results of applying explanation-based learning to the logic of *Principia Mathematica*. *Proceedings of the Fourth International Machine Learning Workshop* (pp. 148–159). Irvine, CA.

O'Rorke, P. (1989). LT revisited: Explanation-based learning and the logic of *Principia Mathematica*. *Machine Learning*, *4*, 117–160.

Saletore, V., & Kale, L.V. (1989). Obtaining first solutions faster in AND/OR parallel execution of logic programs. *Proceedings of the North American Conference on Logic Programming* (pp. 390–408). Cleveland, OH.

Segre, A.M. (1987). On the operationality/generality trade-off in explanation-based learning. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* (pp. 242–248). Milan, Italy.

Segre, A.M. (1988). *Machine learning of robot assembly plans*. Norwell, MA: Kluwer Academic Publishers.

Segre, A.M., Elkan, C., & Russell, A. (1990). *On valid and invalid methodologies for experimental evaluations of EBL* (Technical Report 90-1126). Ithaca, NY: Cornell University, Department of Computer Science.

Shavlik, J.W. (1990). Acquiring recursive and iterative concepts with explanation-based learning. *Machine Learning, 5*, 39–70.

Shavlik, J.W. (1990). *Extending explanation-based learning*. San Mateo, CA: Morgan Kaufmann Publishers.

Subramanian, D., & Feldman, R. (1990). The utility of EBL in recursive domain theories. *Proceedings of the National Conference on Artificial Intelligence* (pp. 942–951). Boston, MA.

Tambe, M., & Newell, A. (1988). Some chunks are expensive. *Proceedings of the Fifth International Machine Learning Conference* (pp. 451–458). Ann Arbor, MI.

Valiant, L.G. (1984). A theory of the learnable. *Communications of the ACM, 27*, 1134–1142.

Warren, D.S., Ahamad, M., Debray, S.K., & Kale, L.V. (1984). Executing distributed-PROLOG programs on a broadcast network. *Proceedings of the Institute of Electrical and Electronics Engineers Symposium on Logic Programming* (pp. 12–21). Atlantic City, NJ.

Whitehead, A.N., & Russell, B. (1913). *Principia Mathematica*. Cambridge, England: Cambridge University Press.