AN EMPIRICAL APPROACH TO SOLVING THE GENERAL UTILITY PROBLEM IN SPEEDUP LEARNING

Anurag Chaudhry and **Lawrence B. Holder**
Learning and Planning Laboratory, Department of Computer Science Engineering
University of Texas at Arlington, Box 19015, Arlington, TX 76019-0015
Email: chaudhry@cse.uta.edu, holder@cse.uta.edu

Abstract

The utility problem in speedup learning describes a common behavior of machine learning methods: the eventual degradation of performance due to increasing amounts of learned knowledge. The shape of the learning curve (cost of using a learning method vs. number of training examples) over several domains suggests a parameterized model relating performance to the amount of learned knowledge and a mechanism to limit the amount of learned knowledge for optimal performance. Many recent approaches to avoiding the utility problem in speedup learning rely on sophisticated utility measures and significant numbers of training data to accurately estimate the utility of control knowledge. Empirical results presented here and elsewhere indicate that a simple selection strategy of retaining all control rules derived from a training problem explanation quickly defines an efficient set of control knowledge from few training problems. This simple selection strategy provides a low-cost alternative to example-intensive approaches for improving the speed of a problem solver. Experimentation illustrates the existence of a minimum (representing least cost) in the learning curve which is reached after a few training examples. Stress is placed on controlling the amount of learned knowledge as opposed to which knowledge. An attempt is also made to relate domain characteristics to the shape of the learning curve.

1 INTRODUCTION

Control knowledge is acquired and applied in machine learning systems with the goal of improving the performance of the system. However, not all of the acquired knowledge may be useful. These systems eventually learn low utility knowledge whose retention cost outweighs the performance benefits. This work tries to lay down a set of rules to limit this knowledge in order to keep the knowledge utile.

Problem solvers apply learning to improve their performance, especially to improve the speed with which they can solve problems (speedup learning). Without learning, a problem solver will solve the same problem in the same time and in the same way and will not learn from experience. Learning from experience (experiential bias) can reduce the problem solver's problem solving time by reducing the search space of the problem. There is a cost associated with applying this experience, and this experience is useful only if the benefit (acquired by utilizing this experience) exceeds the cost. As the amount of experience increases, the cost associated with applying that experience also increases which ultimately exceeds the benefits. This experience has low utility. The above discussion summarizes the utility problem [6, 12]: the eventual degradation of performance due to increasing amounts of learned knowledge.

Our approach to solving the utility problem in speedup learning requires few training examples (low learning time) and does not use utility measures. This is in contrast to other approaches which apply empirical [2] and statistical [4, 5] measures to learn control rules for which there is high certainty of utility. However, these approaches require a large number of training examples to estimate the problem distribution (implying higher learning time) and ensure utile control rules.

This work also tries to identify characteristics of domains which cause the use of a particular type of control knowledge to be beneficial. For example, control rules learned from a domain with a high percentage of rules alternatively applicable to a single goal may produce control rules which help in the selection of a domain rule for the current goal. A more general problem would be to link the semantics of the domain to the type of control knowledge. We also try to motivate a general mechanism which will limit the number of control rules by indicating when to stop learning.

The rest of the text is organized as follows. The remainder of this section gives an introduction to the utility problem and an overview of our approach. Section 2 discusses explanation-based learning (EBL) and illustrates various types of control knowledge. Section 3 summarizes some of the learning systems and compares our approach to their methodology. Section 4 describes our approach and experimental setup. Section 5 illustrates our experiments and their results, and Section 6 concludes with a discussion of these results.

1.1 Recognition of the utility problem

Both inductive and explanation-based (or speedup) learning methods suffer from the general utility problem: the eventual degradation of performance due to the increasing amount of low utility learned knowledge. For example, inductive learning methods typically use a set of training examples to generate knowledge for improving classification accuracy on unseen examples. Explanation-based learning methods use a single example to generate knowledge for improving the problem-solving speed on unseen examples.

However, all of this garnered knowledge may not be useful. There is a cost associated with applying the knowledge which may outweigh the benefits accrued via its use. For example, in a typical speedup learning system that learns macro-operators, performance improvement results from (based on a discussion in [11]):

1. *Experiential bias* or *re-ordering effect*: The learned knowledge (macro-operators) can change the path traversed to reach the goal.

2. *Decreased cost path*: Cost of reaching a goal via a macro-operator may be less than the corresponding cost of applying the sequence of primitive operators that make up the macro-operator.

Performance degradation in such a system is due to (based on a discussion in [11]):

1. *Duplication of the search space*: If macro-operators fail to achieve the goal, the system will resort to primitive operators to solve the problem thus repeating some of the work done when macro-operators were used.

2. *Redundancy by duplicate subsequences in macro-operators*: If macro-operators have primitive operators in common, some of the work done will be repeated as each of the macro-operators containing that operator is tested. So, for example, a macro-operator composed of primitive operators *op1* and *op2* will share many of the same preconditions as a macro operator composed of primitive operators *op1* and *op3*. Redundancy will be due to preconditions of *op1*.

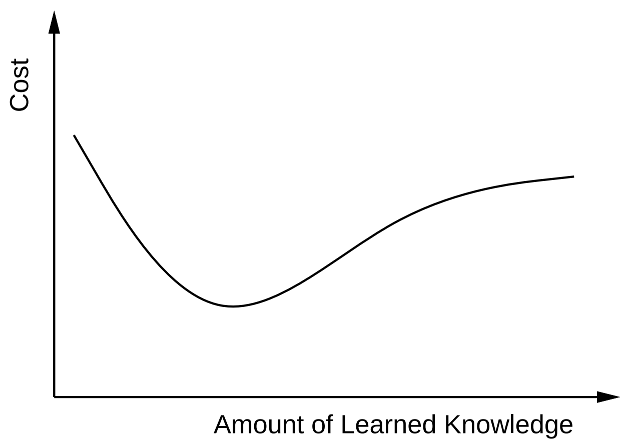

Figure 1: The relationship between cost and knowledge exhibited by a learning method suffering from the general utility problem.

In inductive methods the reason for performance degradation is overfit. Inductive methods construct a hypothesis to correctly classify not only examples from the training set but other unseen examples as well. Overfit usually occurs when the hypothesis becomes complex, because a complex hypothesis tends to be overly specific[1] and hence *explains* the training set [18]. A complex hypothesis represents trends in the training data which may not occur in unseen examples. The specificity of the hypothesis may increase due to noise in the training examples or inadequate stopping criteria of the method, the latter being more relevant to our discussion. Thus, a typical cost (classification error) curve as a function of the amount of learned knowledge will be similar to that in figure 1. The general utility problem in inductive learning has been analyzed for a number of inductive learning techniques [7]. An extensive empirical investigation has also shown that the minimum of the learning curve is frequently better than popular, statistics-based methods for addressing overfit [8].

The utility problem has been verified in several speedup learning systems [13, 19, 17, 10]; however, the underlying cause is less obvious than in inductive learning. If control knowledge is used, the expected cost to solve a representative sample (from which the control knowledge was learned) of the problems is less. This may not necessarily be true for unseen examples. In speedup learning systems using control knowledge, the performance improvement is due to reduction of problem search space. However, control knowledge may have the opposite effect if it leads to searching futile paths in the search space. As more control knowledge is generated, the cost of applying the knowledge also increases. The overall trend thus also follows the behavior in figure 1, where cost is typically problem-solving time. This degradation in problem-solving speed can ultimately be attributed to the learning of low-utility knowledge whose retention cost outweighs its benefit.

A common denominator in the strategies discussed above is that performance eventually degrades as knowledge increases. This implies that performance degradation can be avoided by controlling the amount of learned knowledge. This work aims at empirically analyzing the general utility problem in speedup learning with the ultimate goal of developing a formal model of performance response which depends on the properties of the learning task. This work suggests the existence of a relationship between domain characteristics and the shape of the learning curve and argues that fewer training examples are required, than predicted by other approaches, for avoiding the utility problem.

[1]The most complex and the most specific hypothesis would be the training set itself.

1.2 Outline of our approach

In this work, control knowledge takes the form of preference control rules with associated weights (or counters) representing the total number of times the control rule has been successfully used to solve a problem. These control rules help in preferring a database rule (representing domain knowledge) over others when many database rules are applicable to the current goal. Control rules are generated by solutions to training problems. Our approach is a greedy approach that keeps all control rules (generated by training problems). Duplicate control rules increment corresponding counters.

As indicated by results in [9] and this work, a global minimum exists in the learning curve. Few training examples (< 5) are required to reach this minimum. Thus our greedy strategy seems to be effective for the domains studied, and if results are indicative of the general trend, there is no need for a large number of training examples to estimate the problem distribution. The main advantage of our approach lies in the fact that the similarities of the response curves suggests a model which can be fitted with very few or no training examples.

2 BACKGROUND

2.1 Explanation-based learning (EBL)

Problem solvers apply learning to improve their performance, especially to improve the speed with which they can solve problems (*speedup learning*). A problem solver without learning will always solve the same problem in the same way and in about the same amount of time and will not adjust its behavior based on its experience.

Learning can produce qualitative changes in the performance of a problem-solving system. It may be possible for a problem-solver to explore all possible states in its search space, but this can prove to be combinatorially explosive for large spaces. Learning can cut down the size of the search tree by taking *big steps* in the search space (learning macro operators, or composing sequences of original operators). For example, in rule-based systems, rules can be combined to form *macro rules* by combining collections of rules. The rules

If A and B Then C

and

If C Then D

can be combined to form a macro rule

If A and B Then D

The goal D of the problem solver can be reached in one step if the macro rule is used, as opposed to two steps if the original rules are used.

The other technique for speeding up problem solvers is to learn some form of control knowledge (e.g., knowledge that can be applied to determine which operator to try next). The control knowledge can take many forms including operator selection, rejection and preference rules, evaluation functions and operator strengths [12].

- An evaluation function is a function that can be applied to a state to estimate how close the state is to the goal. Best-first search uses an evaluation function to guide the search.

- Weights can be associated with each operator. Weights quantify operator strength by indicating how successful that operator has been in the past. Weights can be used as a measure for choosing a particular operator.

- Selection, rejection and preference rules are control rules that are evaluated to decide whether a domain rule is singly-applicable, not applicable, or preferred to another rule for solving the current goal.

EBL [16, 1] systems learn by analyzing explanations of problem-solving behavior. Search control knowledge can be acquired by explaining why a particular operator, when applied to a state, leads to a successful solution (or alternatively, why it leads to an unsuccessful search path). The goals for which explanations are being constructed can be provided by the user or can be pre-programmed into the system. Explanations can be a trace of the problem solver's actions or an analysis of a record of the problem solver's actions (to decide whether it satisfies the target concept or the goal). Once produced, explanations can be generalized. Some approaches generalize the constants in explanations into constrained variables, but the structure of the graph that represents the explanation can also be generalized [3]. EBL systems save generalized versions of the solutions to specific problems under the assumption that this will speed up future problem solving. However, saving generalized solutions may be detrimental due to the cost associated with their use. This slowdown has come to be called the *utility problem* [13].

2.1.1 Macro rules

EBL begins with a high-level *target concept* and a training example for that concept. Using a set of axioms describing the domain, the system can explain why the training example is an instance of the target concept. The explanation is essentially a proof that shows how the training example satisfies the target concept.

Specification of EBL [16]:

Input:

- Target Concept: A concept definition describing the concept to be learned.
- Training Example: An example of the target concept.
- Domain Theory: A set of rules and facts to be used in explaining how the training example is an instance of the target concept.
- Operationality Criterion: A predicate over concept definitions, specifying the form in which the learned concept definition must be expressed.

Output:

- A generalization of the training example that is a sufficient concept definition for the target concept and that satisfies the operationality criterion.

Example

Input:

Target Concept:

cup(X)

Training Example:

light(obj1), color(obj1, red), part_of(handle1, obj1), handle(handle1),
bottom(b1), part_of(b1, obj1), flat(b1), concavity(c1),

```
                    part_of(c1, obj1), upward_pointing(c1)

  Domain Theory:
                    cup(X) <-
                            stable(X), liftable(X), open_vessel(X).

                    stable(X) <-
                            bottom(Y), part_of(Y, X), flat(Y).

                    liftable(X) <-
                            graspable(X), light(X).

                    graspable(X) <-
                            handle(Y), part_of(Y, X).

                    open_vessel(X) <-
                            concavity(Y), part_of(Y, X), upward_pointing(Y).

  Operationality Criterion:
                    The concept definition must be expressed in terms of the predicates
                    used to describe examples (e.g., light, upward_pointing, etc.).

Output:
                    cup(X) <-
                            bottom(Y), part_of(Y, X), flat(Y), light(X),
                            handle(Z), part_of(Z, X), concavity(W),
                            part_of(W, X), upward_pointing(W).
```

In the example shown above, cup(X) is to be expressed in terms of operational predicates: predicates used to describe examples (e.g., light, upward_pointing etc.). The training example consists of the facts used to prove the goal cup(obj1). The domain theory states that an object is a cup, if it is stable, liftable and is an open vessel. An object is stable if it has a flat bottom. An object is liftable if it is light and is graspable. An object is graspable if it has a handle, and an object is an open vessel if it has an upward_pointing concavity. The proof tree generated while proving the goal cup(obj1) is shown in figure 2. The leaves of the tree represent the operational predicates, which are ultimately used to prove the goal. This tree can be used to generate a rule, in terms of operational predicates, which states that an object is a cup if it is light and has a flat bottom, an upward pointing concavity and a handle.

The format of a rule is

consequent <- *antecedent1 antecedent2* ... .

The rule should be interpreted as

if *antecedent1* AND *antecedent2* ... then *consequent*.

where *W, X, Y* and *Z* represent variables (Prolog notation).

Re-expressing the target concept in terms of the predicates used to describe examples makes the concept operational with respect to the task of efficiently recognizing examples of the concept. The operationality criteria imposes a requirement that the learned concept definition must be not only correct, but also in a usable form before the learning is complete. The actual purpose of EBL is not to learn more about the target concept but to re-express the target concept in a more operational manner [13].

2.1.2 Control rules

Control rules extend traditional problem solving by separating search control knowledge and domain knowledge. Control rules modify the default behavior by specifying that a particular candidate (e.g., goal, operator) should be either selected, rejected or preferred over another candidate.
Examples include (based on Prodigy/EBL [13, 15]):

- Preference Rules: For the blocks-world domain, if on(X, Y) (to be read as block X on block Y) and on(Y, Z) are both goals at the current node in the search tree, then the latter goal should be achieved first, because achieving on(X, Y) first will be undone by on(Y, Z). A preference rule of the following form could be used.

  if (CANDIDATE-GOAL on(X, Y)) AND

  (CANDIDATE-GOAL on(Y, Z))

  then (PREFER GOAL on(Y, Z) TO on(X, Y))

- Failure Rules:

  if (MATCHES GOAL on(X, X))

  then (FAIL GOAL)

- Selection Rules:

  if (MATCHES GOAL on(X, Y))

  then (SELECT OPERATOR stack(X, Y))

The control rules learned in our experiments are of the form
```
        if (antecedent) then
           consequent
```
The antecedent represents a rule of the domain theory, and the consequent represents a goal (subgoal). For example, if the goal is cup(obj1) and the domain theory is the same as in the previous example, a proof trace as shown in figure 2 is produced.
The following control rules are learned from the proof tree in figure 2.

1. The consequent is:
```
            cup(obj1)
```
   The antecedent is:
```
            cup(X0) <-
                 stable(X0), liftable(X0), open_vessel(X0).
```

2. The consequent is:
```
            stable(X0)
```
   The antecedent is:
```
            stable(X2) <-
                 bottom(Y3), part_of(Y3, X2), flat(Y3).
```

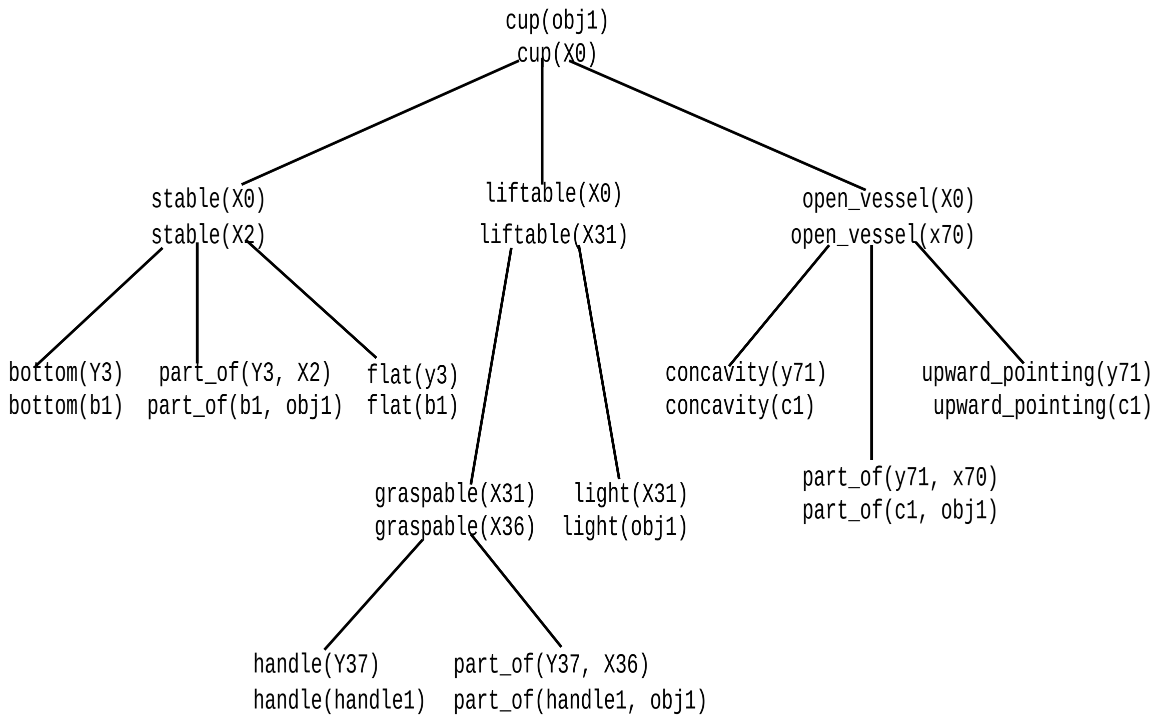

Figure 2: Proof Tree for proving cup(obj1).

3. The consequent is:

   liftable(X0)

   The antecedent is:

   liftable(X31) <–
   graspable(X31), light(X31).

4. The consequent is:

   graspable(X31)

   The antecedent is:

   graspable(X36) <–
   handle(Y37), part_of(Y37, X36).

5. The consequent is:

   open_vessel(X0)

   The antecedent is:

   open_vessel(X70) <–
   concavity(Y71), part_of(Y71, X70), upward_pointing(Y71).

If a goal (subgoal) unifies (or matches) with the consequent of the control rule, then the rule in the antecedent is preferred over other rules in the domain theory for that goal (subgoal). Thus the rules used in our experiments are basically preference rules: control rules that prefer a rule of the domain theory over other rules for proving a particular goal. If no control rules are present for a particular goal, the first rule in the current domain theory whose consequent unifies with the goal is picked. Weights are associated with each control rule. If more than one control rule are applicable, the rule having the highest weight is chosen.

2.2 Utility problem

The utility problem refers to the eventual degradation in performance due to increasing amounts of learned knowledge. With reference to our approach, selecting the highest-weight control rule from a group of control rules, applicable to the current goal, can be expensive. As the number of control rules increase, this expense also increases and may ultimately exceed the benefits of using the control rules, resulting in the generation of low utility knowledge. A possible solution to limiting this knowledge is to limit the number of control rules by stopping training when performance starts to degrade. Other solutions include using statistical and empirical evaluation of clusters of control rules to determine their utility. These are subjects of the following section.

3 RELATED WORK

Most approaches to avoiding the utility problem in speedup learning rely on training examples to empirically evaluate the utility of learned knowledge. Minton's Prodigy system [12] utilizes a utility function that evaluates control knowledge based on application cost, frequency of use and average savings. PALO [5] and Composer [4] use statistical measures to evaluate control knowledge. Several examples are needed to support an explanation with high confidence and adopt the corresponding control rules.

3.1 Prodigy

The Prodigy system [12] evaluates the utility of problem-solving control knowledge by estimating the application cost, frequency and savings afforded by the control knowledge based on the training problems. Prodigy uses explanation-based specialization to learn from a variety of phenomena including solutions, failures and goal interactions [15, 14, 13]. Explicit target concepts describe these phenomena, and each target concept is associated with a strategy for dynamically improving the performance of the problem solver. Explanations are formulated using a theory describing the domain and the Prodigy problem solver. Unlike other EBL problem-solving systems, Prodigy's target concepts are meta-level concepts, such as SUCCEEDS, FAILS and GOAL_INTERFERENCE that describe the problem-solving phenomena. A control choice (e.g., of a goal or operator) succeeds if it leads to a solution. A choice fails if there is no solution consistent with that choice. A choice results in goal interference if some condition that was previously true must be re-achieved.

The Prodigy system uses a domain-independent problem-solver and an explanation-based learning facility for acquiring search control rules from a problem-solving trace. Explanations are constructed from an axiomatized theory describing both the domain and the relevant aspects of the problem solver's architecture.

Prodigy addresses the utility problem by searching for *good* explanations that result in effective control knowledge. First, after each problem-solving episode, the system considers what to explain in the problem-solving trace, and constructs an explanation. Second, the system considers how to represent the weakest preconditions of the explanation. The resulting description becomes the left-hand side of a new control rule. Finally, the utility of the rule is measured during subsequent problem-solving to ensure usefulness.

The utility of a control rule learned by Prodigy's EBS (explanation-based specialization) process is measured in terms of speedup that results from the rule's use. Specifically, utility is given by the cost/benefit formula:

$$\textit{Utility} = (\textit{AvrSavings} * \textit{ApplicFreq}) - \textit{AvrMatchCost}$$

where *AvrSavings* is the average time savings produced when the rule is applicable due to the fact that search is eliminated, *ApplicFreq* is the probability that the rule is applicable when it is tested, and *AvrMatchCost* is the average time cost of matching the rule.

After learning the rule, Prodigy produces an initial estimate of the rule's utility based on the training example that produced the rule. Specifically, the system compares the time cost of matching the rule against the time savings that the rule would have produced by eliminating search. Only if the savings outweigh the cost is the rule included in the active set of the control rules. This estimation phase eliminates rules that are poor. After a rule is added to the system, Prodigy attempts to empirically validate the utility estimate, in order to discard any remaining rules which have negative utility.

3.2 Composer

The Composer [4] system embodies a probabilistic solution to the utility problem. Composer uses the generic utility function of a planner defined as

$$\mathit{Utility(planner)} = - \sum_{prob \epsilon Problems} \mathit{Cost(planner,prob)} \times \mathit{Pr(prob)},$$

where *Cost(planner,prob)* is the cost of solving problem *prob*, and *Pr(prob)* is the probability of occurrence of *prob*. The utility of a planner is defined as the sum of the utility of each problem in the distribution weighted by its probability of occurrence. A planner's control knowledge represents its current state. Addition of control knowledge changes the state of the planner. Utility can be associated with each state of the planner. A utile candidate control rule will change the state of the planner to one with higher utility. Thus a candidate control rule is added to the planner control knowledge if there is a high confidence that the rule will benefit the planner.

Composer is implemented within the Prodigy architecture and includes the Prodigy planner. Composer primarily utilizes selection and rejection rules. Solution traces are analyzed by the learning component of Prodigy/EBL to construct control rules for improving problem-solving time. Composer differs from Prodigy/EBL in how statistics are gathered and how control rules are introduced into the Prodigy planner. Prodigy/EBL uses a single example to learn control knowledge. Composer introduces rule interaction in the learning module of Prodigy/EBL in order to learn a utile *set* of control rules. Composer incrementally adds control rules to its control strategy. The utility of the rule depends on the current control strategy. A rule is added only after demonstrating benefit to a pre-specified confidence level. Higher confidence levels require larger numbers of examples.

After a problem is solved, Composer analyzes the trace and identifies search paths which would have been avoided by each candidate control rule. The time spent exploring these avoidable paths indicates the savings which would be provided by the rule. This savings is compared with the recorded precondition match cost, and the difference is reported as the incremental utility of the rule for that problem.

Composer's strategy of generating search control knowledge is more expensive than the heuristic approach adopted by Prodigy/EBL. This is because Composer pays the penalty of matching preconditions without acquiring any of the benefits of candidate control rules.

3.3 PALO

The PALO (Probably Approximately Locally Optimal) [5] approach adopts a hill-climbing technique that evaluates transformations[2] to the performance element (as effected by the control knowledge) using a statistical method. PALO incorporates a criterion for when to stop learning. PALO terminates learning when it has identified (with high probability) a near-local maximum in the transformation space (the learning operators collectively define the transformation space). PALO uses a set of sample queries to estimate the problem distribution and hill-climbs from an initial performance element to one that is, with high probability, close to a local optimum. PALO's results are guaranteed only if the samples are truly representative of the distribution, and the distribution is stationary. PALO provides stronger guarantees than Composer (and Prodigy/EBL) at the cost of more examples. Harmful rules are not discarded in PALO as quickly as they are in Composer. This results in a larger candidate (control rule) set in PALO which increases the cost to solve each training example. On the other hand, while Composer uses utility analysis to identify performance elements with superior performance, the analysis does not guarantee optimal performance elements.

3.4 Conclusions

All the systems discussed above have in common a function which evaluates the utility of candidate control rules. Some of these approaches have high learning times. A point to be noted is that higher utility does not entail that the planning time of any particular problem is reduced. Rather, the expected cost to solve any representative sample of problems is less. Thus the learned control strategy may not be useful in a different distribution (than from which it was learned). Therefore control knowledge that improves performance on one set of problems can degrade performance on other sets (this depends on the distribution of the problem). The actual distribution, which is needed to determine which performance element is optimal, is usually not known. The above systems depend on the training examples for the distribution of problems in the domain. Typically, a large number of training examples are necessary to accurately estimate the problem distribution and the utility of control knowledge. Moreover, the task of finding the optimal element, even knowing the distribution is intractable most of the time [5].

On the other end of the spectrum, simply limiting the *amount* of the learned knowledge (while ignoring utility) may be advantageous in terms of learning time saved. PALO tries to estimate the unknown distribution, but the learning time is extremely high. Hence, concentrating on the amount of learned knowledge rather than the utility of the learned knowledge might be more efficient. Excessive knowledge degrades performance. Limiting learned knowledge, without utility evaluation, may save learning time and eliminate degradation.

The following sections try to empirically validate this hypothesis. The next section discusses the setup for these experiments.

4 METHODOLOGY AND EXPECTATIONS

This section describes the setup used for our experiments. These experiments relate characteristics of the domain to the shape of the learning curve and empirically validate that our approach requires few training examples to learn a utile set of control rules.

[2]Learning can be viewed as a transformational process in which the learning system applies a series of transformations to a performance element.

4.1 Experimental setup

The experimental setup uses a Prolog-like deductive retrieval system with proof tree and control rule generation capabilities. The system currently supports only backward chaining.

The Prolog-like deductive retrieval system used to solve problems generates proofs for the goals. This constitutes an *explanation structure* and can be easily used to generate a generalized proof. The proof is used to generate the search control knowledge (Control Rules) for guiding the retrieval process.

The number of matches (unifications), i.e., bindings for variables, serves as a performance criteria for monitoring performance changes with the increase in amount of learned knowledge, i.e., control rules.

The rules in the database (a database consists of facts and rules which represent the domain theory) are of the form:

```
if ( (antecedent1) AND (antecedent2) ... ) then
    consequent
```

The control rules are of the form:

```
if database_rule then
    goal
```

The control rules are preference rules which choose a particular rule for solving the current goal. If the consequent of the control rule matches (unifies with) the current goal then the antecedent of the control rule is preferred over other rules to solve the current goal. The antecedent of the control rule points to a rule in the database. A weight is associated with each control rule. The weight represents the total number of times the control rule has been *successfully* used for solving problems (i.e., proving goals and subgoals). The weight is incremented by one, each time the control rule is used. If more than one control rule can help in solving the current goal then the rule with the maximum weight is chosen.

In the control rule store, control rules whose consequents have the same predicate are clustered together. The predicate of the goal (subgoal) is used to hash into the control rule store, leading to the control rule cluster having the same predicate as the goal (subgoal). Any of these rules (in the cluster) could (potentially) help in proving the goal. Hence the cost of using a control rule includes the unification cost of finding the maximum weighted rule (in this cluster), whose consequent unifies with the goal.

The average cost of using a control rule is the average number of unifications required to match a goal with the consequent of the control rules having the same predicate. The cost also includes the match cost of using wrong rules and facts from the database as a result of choosing a wrong control rule. The savings in terms of number of matches from using the right control rule is the savings resulting from not trying useless rules and facts from the database for proving the goal.

Since the cost of unification of facts (with goals) is usually less than that of rules (with the cascading effect of proving antecedents of the rule), facts are preferred over rules in our deductive retriever.

As an example of the costs associated with control-rule usage, consider the following control-rule store.

1. The consequent is:

   abcd(X, obj1)

   The weight is: 4

   The antecedent is:

   abcd(X0,Y0) <– read(X0), my(X0, Y0), lips(Y0).

2. The consequent is:

   abcd(obj2, Y)

   The weight is: 4

   The antecedent is:

   abcd(X0,Y0) <– ur(X0), so(X0, Y0), kool(Y0).

3. The consequent is:

   abcd(X, Y)

   The weight is: 5

   The antecedent is:

   abcd(X0,Y0) <– make(X0), my(X0, Y0), day(Y0).

4. The consequent is:

   abcd(obj1, Z)

   The weight is: 7

   The antecedent is:

   abcd(X0,Y0) <– quid(X0), pro(X0, Y0), quo(Y0).

The cost of using a control rule for proving a goal *abcd(obj2, obj1)* will include the cost of unification of the consequents of the rules enumerated above, with *abcd(obj2, obj1)*. Note the consequent of control rule 4 does not unify with the goal and hence the corresponding antecedent (a database rule) will not be used to prove the goal. However the cost of unifying *abcd(obj1, Z)* with *abcd(obj2, obj1)* contributes to the cost of using a control rule. The consequents of control rules 1, 2 and 3 unify with the goal (the cost of using a control rule includes this unification cost). However, the antecedent of rule 3 will be used to solve the goal because it has the highest weight. If this rule successfully solves the goal, then the savings due to this control rule includes the savings resulting from not searching (potential) futile paths – futile paths resulting from the use of antecedents of control rules 1, 2 and/or 4 and any other rule in the database whose consequent unifies with the goal. If this control rule fails to solve the goal, then the other control rules are tried (using the same procedure). The use of the control rule in this case leads to the exploration of futile paths which contributes to the cost. If all the control rules (namely 1, 2 and 3 for the example shown above) fail, a rule from the database (different from the antecedents of the control rules 1, 2 and 3 and whose consequent unifies with the goal) is chosen in the order given in the domain theory. In this situation, control rules have contributed only to the cost.
The basic learning loop is as follows:

- Control_Rule_Store = Nil;
- Solve a list of testing problems and record performance;
- While there are more training examples
  - Pick a training example and solve it;
  - Add new Control Rules to the Control_Rule_Store;
  - Solve the list of testing problems and record performance;

4.2 Expectations

This section discusses the merits and demerits of different approaches (described in this section) for our experiments and relates the shape of the learning curve to our experimental methodology by identifying the reasons contributing to the cost of the method.

Consider a domain in which rules have distinct predicate names. This implies (for our experimental setup) that for a goal to be proved, the rule choice will be the unique rule whose consequent unifies with the goal. Since this choice is unique, learning preference control rules for such a domain should be harmful since the cost associated with using the control rule outweighs the benefits (which in this case is zero, since we already know which rule to use). Hence the number of matches (our performance criteria) should increase with the increase in the number of control rules learned (i.e., it should increase with the increase in the number of training examples).

If many rules with the same predicate are present in the rule base, then a control rule can prevent the deductive retriever from exploring futile paths in the search tree (which results in savings) and hence performance could improve. However, if control rules are learned incessantly, a large number of the rules in the database may wind up in the control rule store.

For example, consider the following database of rules.

1. lmno(X,Y) <– dont(X), follow(X,Y), me(X).

2. lmno(X,Y) <– iam(X), lost(X,Y).

3. lmno(X,Y) <– a_b(X), confused_d(X,Y).

The following rules are possible specific control rules obtained using the above database.

1. The consequent is:
   lmno(obj1, obj2)
   The antecedent is:
   lmno(X,Y) <– dont(X), follow(X,Y), me(X).
2. The consequent is:
   lmno(obj3, obj2)
   The antecedent is:
   lmno(X,Y) <– dont(X), follow(X,Y), me(X).
3. The consequent is:
   lmno(obj4, obj5)
   The antecedent is:
   lmno(X,Y) <– dont(X), follow(X,Y), me(X).

The following rules are possible general control rules obtained using the above database.

1. The consequent is:
   lmno(X0,Y1)
   The antecedent is:
   lmno(X,Y) <– dont(X), follow(X,Y), me(X).
2. The consequent is:
   lmno(Z1,X2)
   The antecedent is:
   lmno(X,Y) <– iam(X), lost(X,Y).

The following rules are possible intermediate control rules obtained using the above database.

1. The consequent is:
   lmno(X0,obj1)
   The antecedent is:
   lmno(X,Y) <– dont(X), follow(X,Y), me(X).
2. The consequent is:
   lmno(obj2,X2)
   The antecedent is:
   lmno(X,Y) <– iam(X), lost(X,Y).

Either type of the general, specific or intermediate rules (or a combination) can be learned depending on the implementation.

When general control rules are learned (in our implementation), the number of rules in the control rule store cannot exceed the number of rules in the database. This is because the consequent of a database rule (or an expression which is formed by renaming the variables of the database rule) serves as the consequent of the control rule, and the antecedent of the control rule contains a pointer to the database rule. Specific control rules can exceed the number of rules in the database. Many different goals can match the consequent of the same database rule. Since the goal now represents the consequent of a control rule, the same database rule can be present as an antecedent of different control rules.

If general rules are learned, the control rule store may wind up having every rule in the database as an antecedent of a control rule after a few training examples. The control rule store may now have a large number of rule choices for the same goal template, which abets searching futile paths. In this case the retriever may *effectively* reduce to one without control rules, but with the cost of processing control rules (because every rule in the database is present as an antecedent of a control rule in the control rule store). This is specifically true if the rules are used uniformly during problem solving.

If a separate rule is learned for each goal (specific control rule), then the number of control rules in the control rule store becomes large after a few training examples. For proving a goal, the set of applicable control rules (control rule cluster for that goal predicate, i.e., control rules whose consequent has the same predicate as the goal, with possibly different arguments) is large. The cost of searching the control rule whose consequent exactly matches the goal increases drastically. Specific control rules are useful if the goals to be solved are repeated exactly. If the arguments of a goal are different from a previously solved goal with the same predicate then no control rule will be applicable, and a rule will have to be picked randomly from the database, thus contributing only to the cost.

If intermediate control rules are learned (some of the arguments of the consequent of the control rules are variables and some are constants), then the disadvantages of both the general and specific control rule cases may be circumvented. However, no deterministic way to learn these rules exists since a combinatorial number (with respect to number of arguments) of intermediate rules can be generated for each database rule. However, the degradation in performance should still occur with the increase in the number of control rules.

Control rules are helpful if there are alternative rules applicable to certain goals. Control rules have a weight associated with them. The highest-weighted rule is chosen from alternatively applicable rules since the rule seems the most conducive rule for solving the problem (the counter *loosely* estimates the problem distribution and helps in the reordering of rules within its cluster). With intermediate control rules the set of alternatively applicable rules is reduced for a certain goal (when compared to specific rules), because some of the arguments of the consequent of the control rules are variables. They also seem to have an advantage over general control rules which is illustrated by the following example.

Assume that the goal abcd(obj1,X) is solved by rule1 and abcd(obj2,Y) is solved by rule2. The control rule store may have rules which make these choices explicit. However, this is not possible for general control rules, because the consequent of the highest-weighted control rule within its cluster will be represented by abcd(X,Y) and the antecedent by either rule1 or rule2.

The cost of processing control rules increases the cost of solving the problem, which contributes to the degradation in performance. The cost of processing control rules is large if the control rule store has large clusters[3] of rules, and these clusters have either equally applicable control rules on one hand (possibly for the general control rules case) or non-applicable control rules on the other (possibly for the specific control rules case). Such large clusters represent low utility knowledge. A way to increase this utility is to reduce the cluster size (possibly to zero – which represents the no control rule situation) which will reduce the processing cost. The no control knowledge case may have an exorbitant futile search path cost associated with it. The trick is to limit the number of control rules with the intent of reducing the cost of applying the control knowledge. Experiments indicate that this can be done by limiting the number of training examples.

5 EXPERIMENTS

In this section we describe the results of our experiments to relate domain characteristics to the shape of the learning curve. We empirically show that multiple rules applicable to a single goal are necessary for control rules to be useful. We justify the use of general control rules as opposed to specific rules and show that control knowledge is beneficial to a certain point after which performance degrades and that few training examples are required to reach this point. Refer to appendix A for descriptions of the domains used in the following experiments.

5.1 Experiment 1: Relating domain characteristics to the shape of the learning curve

The aim of this experiment is to show that multiple rules applicable to a single goal are necessary for control rules to be useful. If the average number of rules alternatively applicable to certain goals is high, then the learning curve's minimum will be below the zero control rule point (thus proving the utility of control rules). Many domains were used in this experiment. General control rules were learned. Artificial domain 1 consists of 24 rules, all of which have consequents with distinct predicate names. The performance curve is shown in figure 3. Artificial domain 2 consists of 24 rules, 18 of which have consequents with distinct predicate names. The learning curve for this domain is shown in figure 4. Artificial domain 3 consists of 24 rules, 6 of which have consequents with distinct predicate names. The learning curve for this domain is shown in figure 5. Refer to appendix A for more details regarding these domains.

The following observations can be made from the results.

1. The minimum of the learning curve in figure 5 (unlike figure 4) is below the no training example (i.e., zero control rule) point. This is because the domain for figure 5 has, on average, more database rules alternately applicable to certain goals. Thus control rules learned are more utile in the sense that they help in choosing a database rule from a larger cluster of rules

[3]Clusters are implicitly defined as control rules whose consequent have the same predicate.

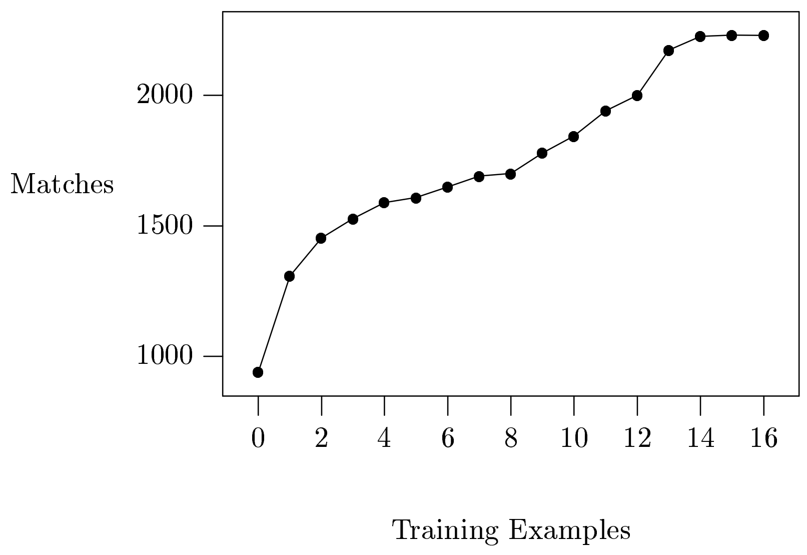

Figure 3: Artificial domain 1: Match values averaged over 10 trials consisting of 16 training and 8 testing examples sampled from 24 queries.

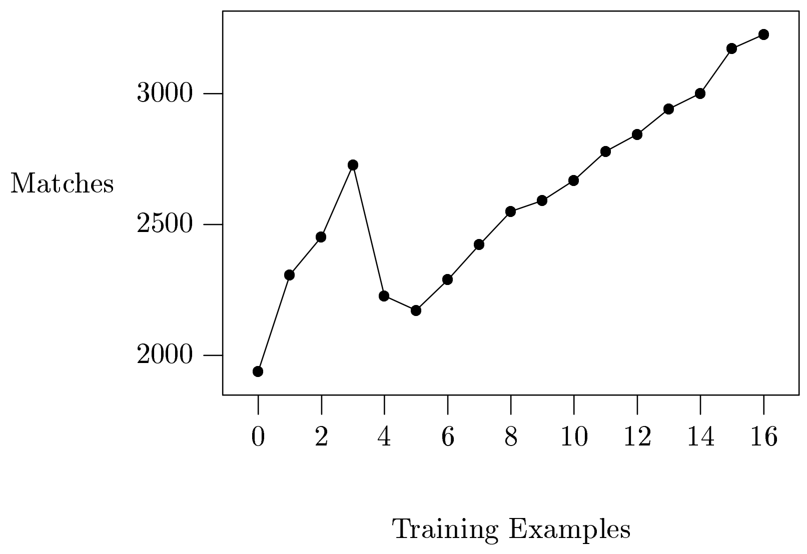

Figure 4: Artificial domain 2: Match values averaged over 10 trials consisting of 16 training and 8 testing examples sampled from 24 queries.

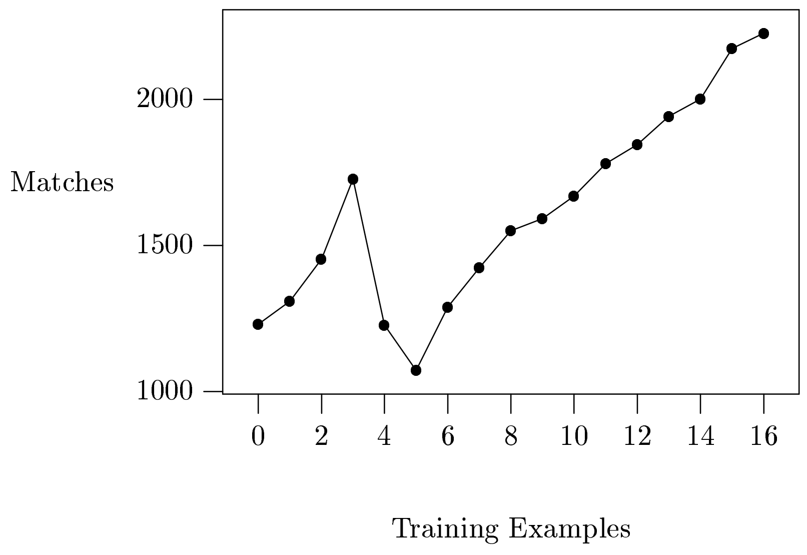

Figure 5: Artificial domain 3: Match values averaged over 10 trials consisting of 16 training and 8 testing examples sampled from 24 queries.

(when compared to the domain for figure 4). Learning was detrimental for artificial domains 1 and 2.

2. There is a monotonic degradation in performance (i.e., increase in the match cost) as the number of training examples increase (i.e., as more control rules are learned) above a certain number.

3. Figure 5 exhibits the general utility problem: the eventual degradation of performance due to increasing amount of learned knowledge. The figure shows that learning control knowledge is beneficial to a certain point after which the cost of using the control knowledge exceeds its benefits.

4. Figure 3 is a monotonically increasing curve. This is to be expected since the control rules will only add to the cost (because the database rule choice is unique).

5. The cost at the minima of the learning curve in figure 4 is more than the initial cost (without control rules). This is because a majority of the rules (75%) have distinct predicate names and hence do not actually require control rules for their selection.

The results indicate that control rules are helpful if there are alternative rules applicable to certain goals. If the percentage of rules alternately applicable to certain goals is high, then a control rule can reduce the cost by choosing the rule which shows the greatest potential for solving the problem. In such cases the savings due to control rules is better than with a lower percentage of alternately applicable rules.

5.2 Experiment 2: General vs. specific control rules

The purpose of this experiment is to justify the learning of general control rules as opposed to specific and intermediate rules in our experimental setup. A secondary aim is to show that learning too many control rules (as in the case of specific rules) is harmful. Two domains were used in this experiment. Artificial domain 4 contains 24 rules for determining family relationships combined with 21 artificial rules increasing the number of alternative rules applicable to certain goals. The sentence domain consists of 14 rules implementing a simple natural language parser. Refer to appendix A for more details regarding these domains.

Figure 6 shows the cost (averaged over 10 trials) of solving 9 testing problems in the sentence domain after learning control rules from each of 18 training problems sampled randomly from a set of 28 problems (queries). Figure 7 shows the cost (averaged over 10 trials) of solving 9 testing problems in artificial domain 4 after learning control rules from each of the 18 training problems sampled randomly from a set of 27 problems. The three curves represent the cost when specific, general and intermediate control rules are learned. Intermediate control rules perform better than the general control rules. The reason is that some rule choices are not possible with general control rules. These choices can be learned with intermediate control rules. An example illustrating this is present in section 4.2. The cost of testing whether a control rule is applicable is high for the specific control rule case, because a large number of control rules, whose consequents have the same predicate as the goal (thus increasing the match cost of selecting a control rule in this cluster), are generated. All the curves exhibit the utility problem with the learning curve of intermediate control rules having the best characteristics.

In future experiments curves will be shown only for the general case since there is no simple way of generating intermediate control rules. The number of intermediate control rules for a database rule is exponential with the number of arguments of the consequent of the control rule. In these experiments, intermediate rules were learned by asking queries having variables as arguments. The correct but *expensive* way to learn intermediate control rules would be to learn specific rules and condense their consequents by some generalizing mechanism to reduce the cluster size. For example, if the following specific rules are learned:

abcd(obj1, obj2) <- rule1
abcd(obj3, obj2) <- rule1

then these rules can be combined to form the intermediate rule

abcd(X, obj2) <- rule1.

This assumes that the domain of the first argument of *abcd* has only *obj1* and *obj2* as members. Building this mechanism is tricky, because it requires complete knowledge about the domain of each predicate. There will also be a high cost associated with building these control rules for the same reason.

5.3 Experiment 3: Too much control knowledge can be harmful

The goal of this experiment is to show that control rules are beneficial to a certain point after which they cause degradation in performance. This experiment lays ground for implementing a general mechanism which limits the number of control rules by indicating when to stop learning (at the minimum of the learning curve). The experiment also shows that a global minimum exists in the learning curve.

Many domains were used in this experiment. The sentence domain consists of 14 rules for parsing simple sentences. Artificial domain 4 contains 45 rules, 24 of which determine family relationships and the remaining increase the number of alternative rules applicable to certain goals. The blocks domain contains 8 rules for transferring blocks and building towers. Artificial domain 5 contains 21 artificial rules having a high percentage of rules alternatively applicable to goals. These domains are listed and described in appendix A.

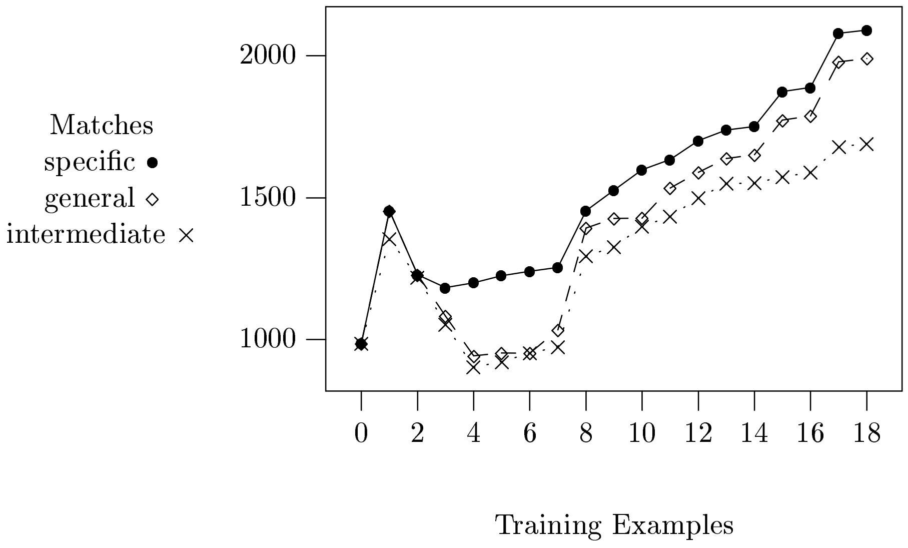

Figure 6: Sentence domain: Match values averaged over 10 trials consisting of 18 training and 9 testing examples sampled from 28 queries.

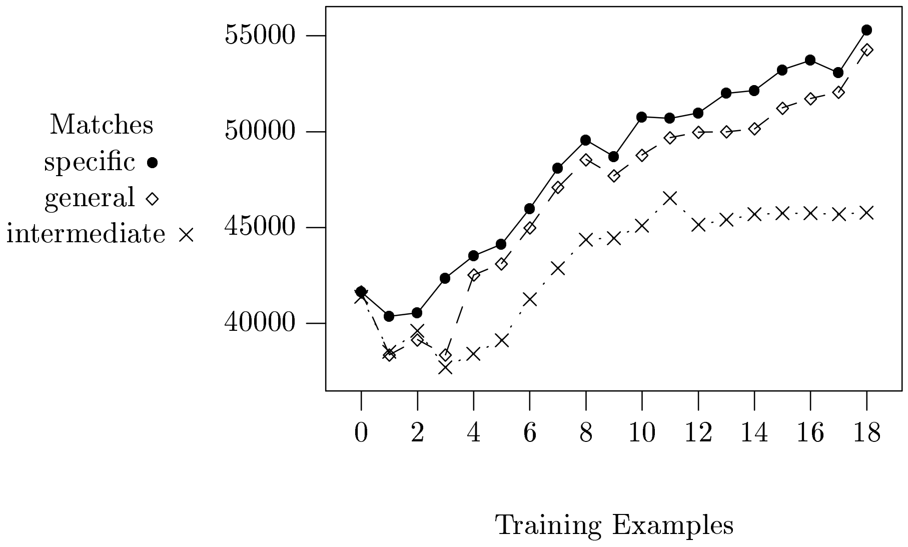

Figure 7: Artificial domain 4: Match values averaged over 10 trials consisting of 18 training and 9 testing examples sampled from 27 queries.

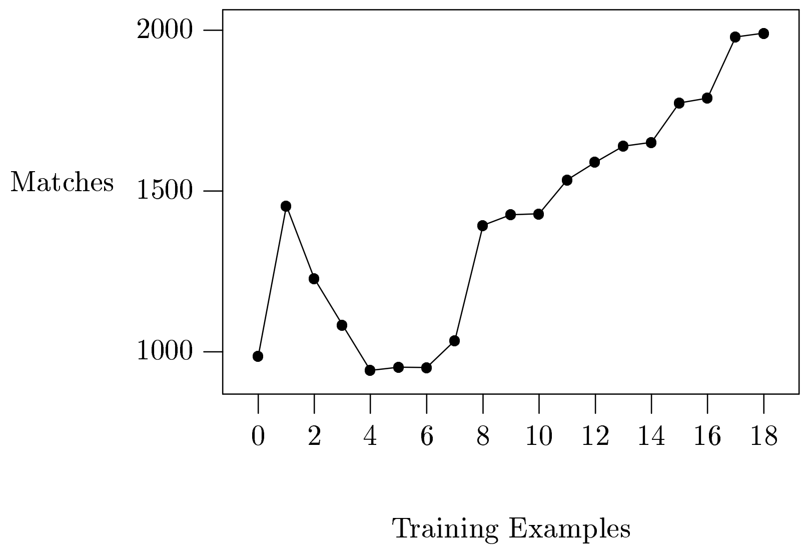

Figure 8: Sentence domain with general control rules. Match values averaged over 90 trials consisting of 18 training and 9 testing examples sampled from 28 queries.

Figure 8 shows the cost (averaged over 90 trials) of solving 9 testing problems in the sentence domain. Control rules were learned from each of 18 training problems sampled randomly (with replacement) from a set of 28 queries (problems). Control rules learned from the first training problem increased the cost, but the cost gradually decreased till a minimum was reached below the cost of the initial rules. With more training examples, the cost increased steadily following the trend of figure 1. The minimum cost occurred after the fourth training example.

Figure 9 shows the cost (averaged over 90 trials) of solving 9 testing problems in artificial domain 4. Control rules were learned from each of 18 training problems sampled randomly (with replacement) from a set of 28 queries (problems). The learning-cost curve follows the trend of figure 1. The minimum cost occurred after the third training example.

Figure 10 shows the cost (averaged over 30 trials) of solving 5 testing problems in the blocks domain. Control rules were learned from each of 10 training problems sampled randomly (with replacement) from a set of 15 queries (problems). The problems all involved building towers of height 2 from 6 blocks initially on the table. The minimum of the learning curve occurred after solving the first training problem and the cost remained fixed thereafter. This is because the queries were essentially the same, each building towers of height 2, and hence the necessary control rules were learned after the first training example.

Figure 11 shows the cost (averaged over 30 trials) of solving 10 testing problems in the blocks domain. Control rules were learned from each of 20 training problems sampled randomly (with replacement) from a set of 30 queries (problems). The 30 queries consisted of building 18 towers of height 2, 9 towers of height 3, and 3 towers of height 4. Once again, the learning-cost curve follows the trend of figure 1. The minimum cost occurred after the first training example.

Figure 12 shows the cost (averaged over 70 trials) of solving 6 testing problems in artificial domain 5 after learning control rules from each of 12 training problems sampled randomly from a set of 18 problems (queries). The familiar trend of the general utility problem is evident once again.

There is a minimum in the learning curves as is evident from figures 8, 9, 11 and 12. Initially

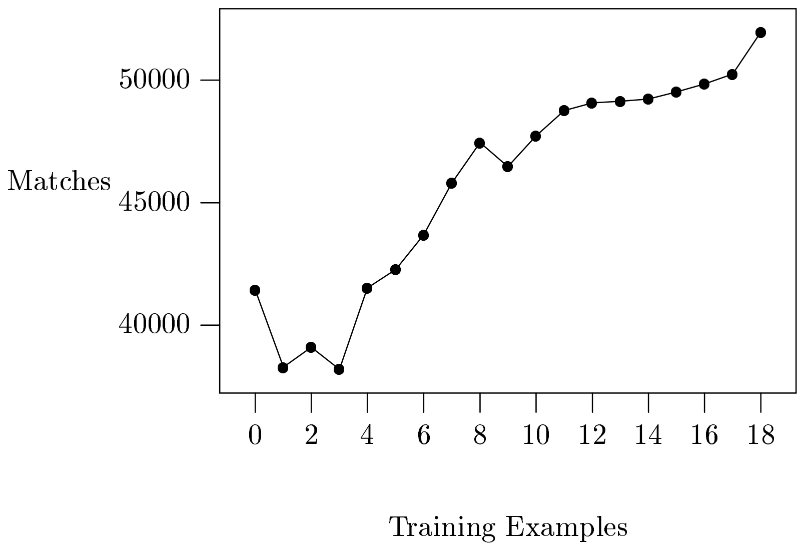

Figure 9: Artificial domain 4 with general control rules. Match values average over 90 trials consisting of 18 training and 9 testing sampled from 28 queries.

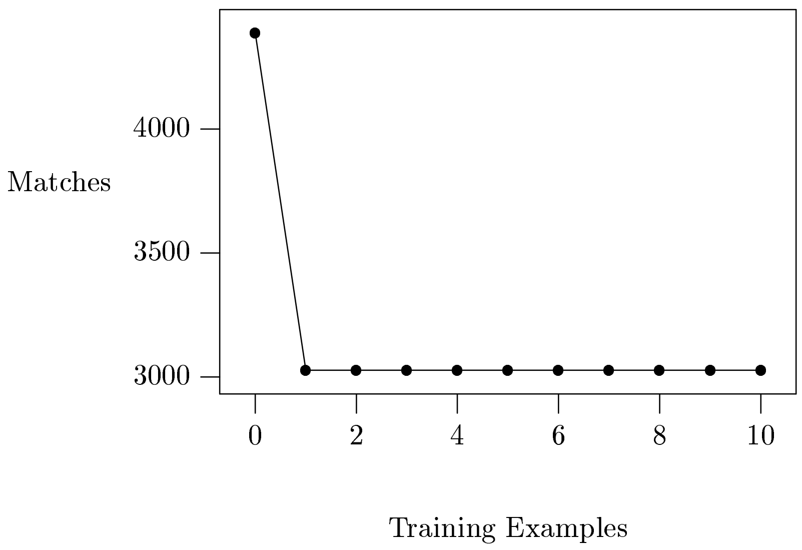

Figure 10: Blocks domain with general control rules. Match values average over 30 trials consisting of 10 training and 5 testing examples sampled from 15 queries consisting of towers of height 2.

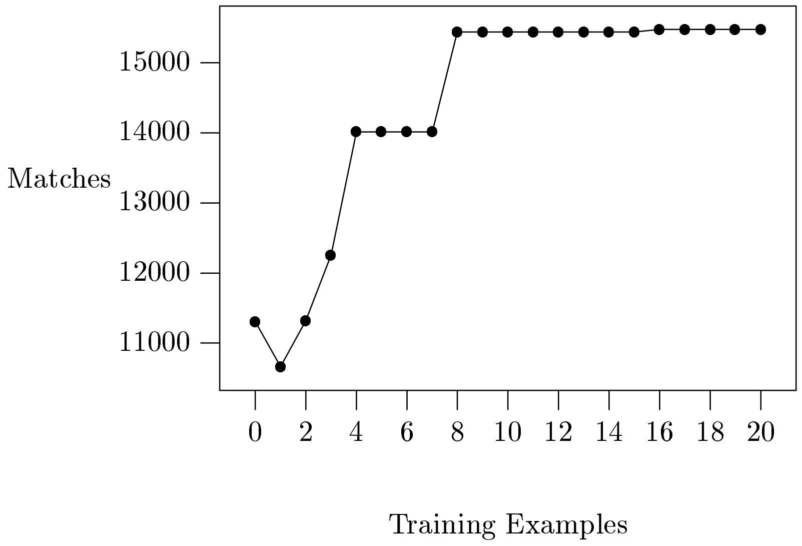

Figure 11: Blocks domain with general control rules. Match values average over 30 trials consisting of 20 training and 10 testing examples sampled from 30 queries consisting of towers of height 2 (18), height 3 (9) and height 4 (3).

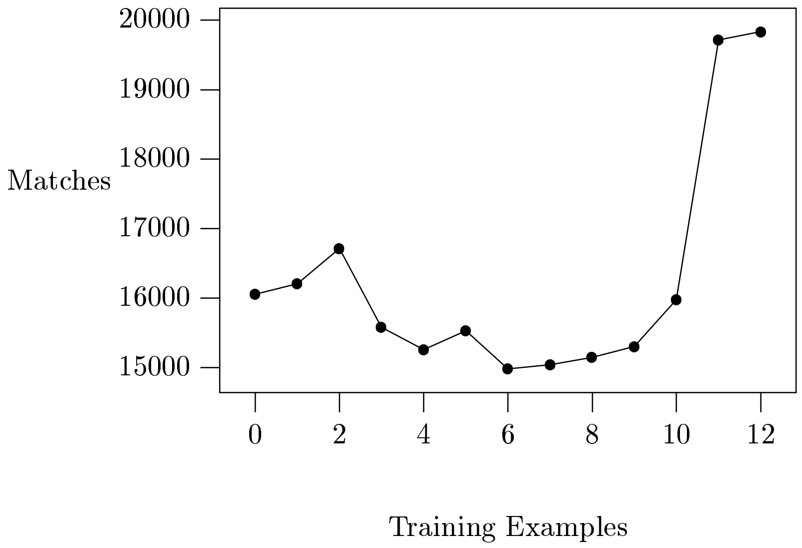

Figure 12: Artificial domain 5 with general control rules. Match values average over 70 trials consisting of 12 training and 6 testing examples sampled from 18 queries.

as the system learns control rules generated from randomly-sampled training problems, cost may increase slightly. However, the cost quickly decreases but is eventually driven up. The initial rise may be due to inclusion of low-utility control knowledge. However, the learning curve quickly turns downward as control rules are learned from training problems containing goals that are prevalent in the problem distribution. Eventually, after the utile rules have appeared, subsequent rule learning follows statistically insignificant trends in the problem distribution that drive up the cost of solving the testing problems. These factors combine to form a minimum in the learning cost curve.

We also observe that very few training examples are necessary to learn a utile set of control rules, i.e., converge to the minimum of the learning-cost curve which is below the zero control rule point (no training example). This leads us to hypothesize that this approach has a lower learning time than systems like COMPOSER and PALO which involve utility evaluation and which require a large number of training examples to estimate the distribution.

6 CONCLUSIONS

Cost associated with the use of control rules can be attributed to the time spent in testing the applicability of the control rules and following futile paths in the search space not explored by the original domain theory. Time savings associated with the use of control rules is due to the avoidance of futile paths explored by the original domain theory. These factors contribute to the existence of a global minimum in the learning-cost curve.

The difficulties in identifying the global minimum result from the presence of a local minimum as evident from figure 9 and coarse control (several control rules are learned per training example). Finer control may be possible by limiting the number of control rules instead of training examples. This hampers the prediction of the number of training examples corresponding to the minimum of the learning-cost curve.

Our empirical results indicate that few training examples are required to reach the minimum of the learning-cost curve. The testing set, representing the problem distribution, empirically determines this point. However, no theory is available to predict the number of training examples corresponding to this minimum. An approach in this direction could be to relate domain characteristics (e.g., size and shape of the search space, size of the problem space, recursive versus non-recursive domain theories) to the probability of seeing a majority of training problems that follow a certain, highly-efficient path through the search space that is also followed by a large number of other problems prevalent in the problem distribution [9]. This approach as opposed to statistical approaches could require a smaller number of training examples.

In our experiments we have demonstrated the ubiquity of the general utility problem in speedup learning. We have shown that a global minimum exists in the learning curve. With this in mind a mechanism can be incorporated in the system to stop learning at the point represented by the minimum. A set of problems can be solved to obtain the cost of solving these problems as a function of the number of training examples. From our experimental results (specifically those of experiment 3) we observe that the shape of the learning curve, for different domains, emulates the trend in figure 1. A curve can be interpolated through these points (based on the general trend), and the number of training examples corresponding to the minimum can be approximated. The main advantage of our approach lies in the fact that the similarities of the response curves suggest a model which can be fitted with very few or no training examples. This represents a simple yet efficient (or inexpensive) way to limit knowledge since the approach does not involve a utility evaluation function.

Thus our simple control-rule selection strategy lies at the opposite end of the spectrum from approaches to the utility problem dependent upon large numbers of training problems to estimate the problem distribution. Therefore we can intuitively argue that this approach has a lower learning time than systems like Composer and PALO which perform utility evaluation. Future work could involve validating this intuition and comparing the performances of these systems under identical conditions. Empirical results, for the domains listed in appendix A, indicate that few training problems are needed to learn a utile set of control rules (corresponding to the minimum of the learning-cost curve). If these results are indicative of the behavior in other domains, there should be no need for large numbers of training problems, and a set of utile control rules can be learned with less cost [9].

If the distribution of queries changes, the control strategy of the system needs to be re-evaluated. This re-evaluation will be cheaper for our approach, which requires fewer training examples to reach the minimum of the learning curve.

Our experiments (specifically experiment 1) indicate that control rules are useful only if they help in favoring database rules which save time by not following futile paths in the search space. For this to be true there should be multiple rules alternatively applicable to single goals. As evident from experiment 1, a greater percentage of alternatively applicable rules implies a lower minimum of the learning curve (implying lower cost at the minimum). Furthermore, this minimum moves to the right as the percentage of alternatively applicable rules increase. This is because more control rules are needed to prefer a larger percentage of alternatively applicable rules and hence more training examples are required. These observations throw light on the relationship between the percentage of alternatively applicable control rules, number of training examples, and the shape of the learning curve (specifically the depth of the concavity and the location of the minimum). For a higher percentage of alternatively applicable rules, the minimum represents a lower cost and occurs at a higher number of training examples. This analysis gives a general relationship, but more rigorous formal and empirical analysis is required to accurately predict the necessary number of training problems based on domain characteristics.

A DOMAINS

This appendix lists the various domains used in our experiments. The format should be interpreted as follows:

1. Facts are represented as
   (*predicate argument1 argument2 ...*).

2. Rules are represented as
   (<- *consequent antecedent1 antecedent2 ...*).

3. *consequents* and *antecedents* have the same format as facts.

4. A variable X is represented as ?X.

Thus, a Prolog rule *abcd(X, Y) :- iam(X), sokool(Y)* would be represented as *(<- (abcd ?X ?Y) (iam ?X) (sokool ?Y))*.

A.1 Artificial domains 1, 2 & 3

These domains are variations of artificial domain 4 with 0%, 25% and 75% rules alternatively applicable to goals. Artificial domain 1 consists of 24 rules, all of which have consequents with distinct predicate names. Artificial domain 2 consists of 24 rules, 18 of which have consequents with distinct predicate names. Artificial domain 3 consists of 24 rules, 6 of which have consequents with distinct predicate names. Refer to artificial domain 4 for more details.

A.2 Artificial domain 4

Artificial domain 4 contains 24 rules for determining family relationships combined with 21 artificial rules increasing the number of alternative rules applicable to certain goals.

```
(
 (male john) (male tom) (male fred) (male harry) (male jack)
 (bb cons2 cons3) (bc dons2 dons3) (bb fons2 fons3) (bc fons1 fons2)
 (male rich) (male mike) (male steve) (male scott) (female mary)
 (female alice) (female linda) (female jane) (ca cons1 cons2)
 (cb cons2) (female rachel) (female valerie) (female barbara)
 (female cindy) (female donna) (married john mary) (parent john tom)
 (ca fons2 fons1) (cb fons1) (cc dons2) (parent mary tom)
 (parent john linda) (parent mary linda) (married tom alice)
 (married linda steve) (parent alice valerie) (parent tom valerie)
 (cd dons1) (cc eons1) (cd eons2) (da cons1 cons11) (db cons11 cons12)
 (dc cons12 cons13) (dd cons13 cons2) (parent alice barbara)
 (parent tom barbara) (parent linda jack) (parent steve jack)
 (parent steve rich) (parent linda rich) (dd cons1 cons2)
 (db fons2 fons11) (dc fons11 fons12) (dd fons12 fons1)
 (dc fons2 fons11) (dd fons11 fons1) (ee cons2 cons21) (ef cons21)
 (ee fons1 fons11) (ef fons11) (married barbara scott)(parent scott cindy)
 (parent barbara cindy) (i eons1 eons3) (l eons21 eons1) (l eons21 eons3)
 (l eons2 eons21) (i dons1 dons3) (parent jack mike) (married jack donna)
 (parent donna mike) (l dons21 dons1) (l dons21 dons3) (l dons2 dons21)
 (bb ions2 ions3) (bc jons2 jons3) (married valerie fred)
 (parent fred jane) (parent valerie jane) (bb lons2 lons3) (bc lons1 lons2)
 (ca ions1 ions2) (cb ions2) (ca lons2 lons1) (married rich rachel)
 (parent rich harry) (parent rachel harry) (cb lons1) (cc jons2) (cd jons1)
 (cc kons1) (cd kons2) (da ions1 ions11) (db ions11 ions12)
 (dc ions12 ions13) (dd ions13 ions2) (dd ions1 ions2) (db lons2 lons11)
 (dc lons11 lons12) (dd lons12 lons1) (dc lons2 lons11) (dd lons11 lons1)
 (ee ions2 ions21) (ef ions21) (ee lons1 lons11) (ef lons11)
 (i kons1 kons3) (l kons21 kons1) (l kons21 kons3) (l kons2 kons21)

 (<- (father ?X ?Y) (parent ?X ?Y) (male ?X))
 (<- (mother ?X ?Y) (parent ?X ?Y) (female ?X))
 (<- (husband ?X ?Y) (married ?X ?Y) (male ?X))
 (<- (aa ?X ?Y ?Z) (ba ?X ?Y) (bb ?Y ?Z))
 (<- (aa ?X ?Y ?Z) (ba ?X ?Y) (bc ?Y ?Z))
 (<- (wife ?X ?Y) (married ?X ?Y) (female ?X))
 (<- (son ?X ?Y) (parent ?Y ?X) (male ?X))
 (<- (daughter ?X ?Y) (parent ?Y ?X) (female ?X))
 (<- (aa ?X ?Y ?Z) (bd ?X ?Y ?Z))
 (<- (aa ?X ?Y ?Z) (ba ?X ?Z) (bb ?Y ?Z) (bc ?X ?Y))
 (<- (ba ?X ?Y) (ca ?X ?Y) (cb ?Y))
 (<- (ba ?X ?Y) (ca ?Y ?X) (cb ?X))
 (<- (ba ?X ?Y) (cc ?X) (cd ?Y))
 (<- (ba ?X ?Y) (cc ?Y) (cd ?X))
 (<- (sibling ?X ?Y) (father ?F ?X) (father ?F ?Y) (mother ?M ?X) (mother ?M ?Y))
 (<- (brother ?X ?Y) (sibling ?X ?Y) (male ?X))
 (<- (sister ?X ?Y) (sibling ?X ?Y) (female ?X))
 (<- (ca ?X ?Y) (da ?X ?W) (db ?W ?U) (dc ?U ?V) (dd ?V ?Y))
 (<- (ca ?X ?Y) (db ?X ?U) (dc ?U ?V) (dd ?V ?Y))
 (<- (ca ?X ?Y) (dc ?X ?V) (dd ?V ?Y))
 (<- (sister_in_law ?X ?Y) (brother ?B ?Y) (married ?X ?B))
 (<- (brother_in_law ?X ?Y) (sister ?S ?Y) (married ?X ?S))
 (<- (mother_in_law ?X ?Y) (mother ?X ?S) (married ?S ?Y))
 (<- (father_in_law ?X ?Y) (father ?X ?S) (married ?S ?Y))
 (<- (ca ?X ?Y) (dd ?X ?Y))
 (<- (cb ?Y) (ee ?Y ?Z) (ef ?Z))
 (<- (uncle ?X ?Y) (parent ?P ?Y) (brother ?X ?P))
 (<- (uncle ?X ?Y) (parent ?P ?Y) (sister ?S ?P) (husband ?X ?S))
 (<- (aunt ?X ?Y) (parent ?P ?Y) (sister ?X ?P))
 (<- (cb ?Y) (ee ?Z ?Y) (ef ?Z))
 (<- (bd ?X ?Y ?Z) (fa ?X ?Y ?Z))
 (<- (aunt ?X ?Y) (parent ?P ?Y) (brother ?B ?P) (wife ?X ?B))
 (<- (cousin ?X ?Y) (parent ?P ?X) (parent ?O ?Y) (sibling ?P ?O))
 (<- (grandmother ?X ?Y) (parent ?P ?Y) (mother ?X ?P))
 (<- (fa ?X ?Y ?Z) (ga ?X ?Y ?Z))
 (<- (ga ?X ?Y ?Z) (h ?X ?Y) (i ?X ?Z) (j ?Y ?Z))
 (<- (h ?X ?Y) (k ?X ?Z) (l ?Z ?Y))
 (<- (h ?X ?Y) (k ?Y ?Z) (l ?Z ?X))
 (<- (k ?X ?Z) (l ?X ?Z))
 (<- (j ?Y ?Z) (h ?Y ?Z))
 (<- (grandfather ?X ?Y) (parent ?P ?Y) (father ?X ?P))
 (<- (ancestor ?X ?Y) (parent ?X ?Y))
 (<- (ancestor ?X ?Y) (parent ?P ?Y) (ancestor ?X ?P))
 (<- (descendant ?X ?Y) (ancestor ?Y ?X))
 (<- (married ?X ?Y) (married ?Y ?X))
)
```

A.3 Artificial domain 5

Artificial domain 5 contains 21 artificial rules having a high percentage of rules alternatively applicable to goals.

```
(
 (bb cons2 cons3) (bc dons2 dons3) (bb fons2 fons3) (bc fons1 fons2)
 (ca cons1 cons2) (cb cons2) (ca fons2 fons1) (cb fons1) (cc dons2)
 (cd dons1) (cc eons1) (cd eons2) (da cons1 cons11) (db cons11 cons12)
 (dc cons12 cons13) (dd cons13 cons2) (dd cons1 cons2) (db fons2 fons11)
 (dc fons11 fons12) (dd fons12 fons1 ) (dc fons2 fons11) (dd fons11 fons1)
 (ee cons2 cons21) (ef cons21) (ee fons1 fons11) (ef fons11)
 (i eons1 eons3) (l eons21 eons1) (l eons21 eons3) (l eons2 eons21)
 (i dons1 dons3) (l dons21 dons1) (l dons21 dons3) (l dons2 dons21)
 (bb ions2 ions3) (bc jons2 jons3) (bb lons2 lons3) (bc lons1 lons2)
 (ca ions1 ions2) (cb ions2) (ca lons2 lons1) (cb lons1) (cc jons2)
 (cd jons1) (cc kons1) (cd kons2) (da ions1 ions11) (db ions11 ions12)
 (dc ions12 ions13) (dd ions13 ions2) (dd ions1 ions2) (db lons2 lons11)
 (dc lons11 lons12) (dd lons12 lons1 ) (dc lons2 lons11) (dd lons11 lons1)
 (ee ions2 ions21) (ef ions21) (ee lons1 lons11) (ef lons11) (i kons1 kons3)
 (l kons21 kons1) (l kons21 kons3) (l kons2 kons21)
 (<- (aa ?X ?Y ?Z) (ba ?X ?Y) (bb ?Y ?Z))
 (<- (aa ?X ?Y ?Z) (ba ?X ?Y) (bc ?Y ?Z))
 (<- (aa ?X ?Y ?Z) (bd ?X ?Y ?Z))
 (<- (aa ?X ?Y ?Z) (ba ?X ?Z) (bb ?Y ?Z) (bc ?X ?Y))
 (<- (ba ?X ?Y) (ca ?X ?Y) (cb ?Y))
 (<- (ba ?X ?Y) (ca ?Y ?X) (cb ?X))
 (<- (ba ?X ?Y) (cc ?X) (cd ?Y))
 (<- (ba ?X ?Y) (cc ?Y) (cd ?X))
 (<- (ca ?X ?Y) (da ?X ?W) (db ?W ?U) (dc ?U ?V) (dd ?V ?Y))
 (<- (ca ?X ?Y) (db ?X ?U) (dc ?U ?V) (dd ?V ?Y))
 (<- (ca ?X ?Y) (dc ?X ?V) (dd ?V ?Y))
 (<- (ca ?X ?Y) (dd ?X ?Y))
 (<- (cb ?Y) (ee ?Y ?Z) (ef ?Z))
 (<- (cb ?Y) (ee ?Z ?Y) (ef ?Z))
 (<- (bd ?X ?Y ?Z) (fa ?X ?Y ?Z))
 (<- (fa ?X ?Y ?Z) (ga ?X ?Y ?Z))
 (<- (ga ?X ?Y ?Z) (h ?X ?Y) (i ?X ?Z) (j ?Y ?Z))
 (<- (h ?X ?Y) (k ?X ?Z) (l ?Z ?Y))
 (<- (h ?X ?Y) (k ?Y ?Z) (l ?Z ?X))
 (<- (k ?X ?Z) (l ?X ?Z))
 (<- (j ?Y ?Z) (h ?Y ?Z))
)
```

A.4 Sentence domain

The sentence domain consists of 14 rules for parsing simple sentences.

```
(
 (verb ate) (verb sat) (verb crushed ) (verb killed) (verb cleaned) (verb read)
 (verb wrote) (noun book) (noun table) (noun apple) (noun orange) (noun sofa)
 (noun banana) (noun grapes) (noun man) (noun woman) (noun boy) (noun girl)
 (noun cat) (noun dog) (noun mat) (prep on) (prep below) (prep under)
 (prep above) (prep inside) (prep outside) (det a) (det the) (det an) (adj big)
 (adj tall) (adj small) (adj tiny) (adj huge) (adj large) (conj and) (conj or)

 (<- (sent ?A ?B ?C ?D ?E)
     (np ?A ?B) (vp ?C ?D ?E))
 (<- (sent ?A ?B ?C ?D ?E ?F)
     (np ?A ?B) (vp ?C ?D ?E ?F))
 (<- (sent ?A ?B ?C ?D ?E ?F ?G)
     (np ?A ?B ?C) (vp ?D ?E ?F ?G))
 (<- (sent ?A ?B ?C ?D ?E ?F)
     (np ?A ?B ?C) (vp ?D ?E ?F))
 (<- (sent ?A ?B ?C ?D ?E ?F)
     (np ?A ?B) (vp ?C ?D ?E ?F))
 (<- (sent ?A ?B ?C ?D ?E ?F ?G)
     (np ?A ?B) (vp ?C ?D ?E ?F ?G))
 (<- (sent ?A ?B ?C ?D ?E ?F ?G ?H)
     (np ?A ?B ?C) (vp ?D ?E ?F ?G ?H))
 (<- (sent ?A ?B ?C ?D ?E ?F ?G)
     (np ?A ?B ?C) (vp ?D ?E ?F ?G))
 (<- (np ?A ?B)
     (det ?A) (noun ?B))
 (<- (np ?A ?B ?C)
     (det ?A) (adj ?B) (noun ?C))
 (<- (vp ?C ?D ?E)
     (verb ?C) (np ?D ?E))
 (<- (vp ?C ?D ?E ?F)
     (verb ?C) (np ?D ?E ?F))
 (<- (vp ?C ?D ?E ?F)
     (verb ?C) (prep ?D) (np ?E ?F))
 (<- (vp ?C ?D ?E ?F ?G)
     (verb ?C) (prep ?D) (np ?E ?F ?G))
)
```

A.5 Blocks domain

The blocks domain contains 8 rules for a situational calculus implementation consisting of one operator for transferring blocks and building towers.

```
(
 (noteq a b) (noteq b a) (noteq a c) (noteq c a) (noteq a d) (noteq d a)
 (noteq a e) (noteq e a) (noteq a f) (noteq f a) (noteq b c) (noteq c b)
 (noteq b d) (noteq d b) (noteq b e) (noteq e b) (noteq b f) (noteq f b)
 (noteq c d) (noteq d c) (noteq c e) (noteq e c) (noteq c f) (noteq f c)
 (noteq d e) (noteq e d) (noteq d f) (noteq f d) (noteq e f) (noteq f e)
 (block a) (block b) (block c) (block d) (block e) (block f)
 (clear a s0) (clear b s0) (clear c s0) (clear d s0) (clear e s0) (clear f s0)
 (on a table s0) (on b table s0) (on c table s0) (on d table s0)
 (on e table s0) (on f table s0) (achievable s0)

 (<- (achievable (do (transfer ?X ?Y) ?S))
     (clear ?X ?S) (block ?X) (clear ?Y ?S) (noteq ?X ?Y) (achievable ?S))

 (<- (clear ?Z (do (transfer ?X ?Y) ?S))
     (on ?X ?Z ?S) (block ?Z) (noteq ?Z ?Y))

 (<- (clear ?X (do (transfer ?X ?Y) ?S))
     (achievable (do (transfer ?X ?Y) ?S)))

 (<- (clear ?A (do (transfer ?X ?Y) ?S))
     (clear ?A ?S) (noteq ?A ?X) (noteq ?A ?Y))

 (<- (on ?X ?Y (do (transfer ?X ?Y) ?S))
     (achievable (do (transfer ?X ?Y) ?S)))

 (<- (on ?A ?B (do (transfer ?X ?Y) ?S))
     (on ?A ?B ?S) (noteq ?A ?X))

 (<- (tower (cons ?X (cons ?Y NUL)) (do (transfer ?X ?Y) ?S))
     (on ?Y table ?S) (block ?Y) (achievable (do (transfer ?X ?Y) ?S)))

(<- (tower (cons ?X (cons ?Y ?Z)) (do (transfer ?X ?Y) ?S))
     (tower (cons ?Y ?Z) ?S) (achievable (do (transfer ?X ?Y) ?S)))
)
```

References

[1] G. F. DeJong and R. J. Mooney. Explanation-based learning: An alternative view. *Machine Learning*, 1(2):145–176, April 1986.

[2] O. Etzioni and S. Minton. Why EBL produces overly-specific knowledge: A critique of the PRODIGY approaches. In *Proceedings of the Ninth International Conference on Machine Learning*, pages 137–143, 1992.

[3] N. S. Flann and T. G. Dietterich. A study of explanation-based methods for inductive learning. *Machine Learning*, 4(2):187–226, 1989.

[4] J. Gratch and G. DeJong. COMPOSER: A probabilistic solution to the utility problem in speed-up learning. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 235–240, 1992.

[5] R. Greiner and I. Jurisica. A statistical approach to solving the EBL utility problem. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 241–248, 1992.

[6] L. B. Holder. The general utility problem in machine learning. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 402–410, 1990.

[7] L. B. Holder. Empirical analysis of the general utility problem in machine learning. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 249–254, 1992.

[8] L. B. Holder. Intermediate decision trees. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1056–1062, 1995.

[9] L. B. Holder and A. Chaudhry. Simple selection of utile control rules in speedup learning. In *Proceedings of the Third International Workshop on Knowledge Compilation and Speedup Learning*, pages 77–82, 1993.

[10] S. Markovitch and P. D. Scott. Utilization filtering: A method for reducing the inherent harmfulness of deductively learned knowledge. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 738–743, 1989.

[11] S. Minton. Selectively generalizing plans for problem-solving. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 596–599, 1985.

[12] S. Minton. *Learning Search Control Knowledge: An Explanation-Based Approach.* Kluwer Academic Publishers, 1988.

[13] S. Minton. Quantitative results concerning the utility of explanation-based learning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 564–569, 1988.

[14] S. Minton and J. Carbonell. Strategies for learning search control rules: An explanation-based approach. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, volume 1, pages 228–235, 1987.

[15] S. Minton, J. Carbonell, C. Knoblock, D. Kuokka, O. Etzioni, and Y. Gil. Explanation-based learning: A problem solving perspective. In *Artificial Intelligence*, volume 40, pages 63–118, 1989.

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

[17] R. J. Mooney. The effect of rule use on the utility of explanation-based learning. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 725–730, 1989.

[18] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[19] M. Tambe and P. Rosenbloom. Eliminating expensive chunks by restricting expressiveness. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 731–737, 1989.