DOCUMENT RESUME

ED 353 956 IR 015 915

AUTHOR Gratch, Jonathan M.; DeJong, Gerald F.
TITLE Utility Generalization and Composability Problems in Explanation-Based Learning.
INSTITUTION Illinois Univ., Urbana. Dept. of Computer Science.
SPONS AGENCY National Science Foundation, Washington, D.C.
REPORT NO UILU-ENG-91-1727; UIUCDCS-R-91-1681
PUB DATE Aug 91
CONTRACT NSF-IRI-87-19766
NOTE 24p.; For related reports, see IR 015 913-914.
PUB TYPE Information Analyses (070) -- Reports - Research/Technical (143)

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS *Artificial Intelligence; *Computer System Design; Design Requirements; *Learning Strategies; Mathematical Models; *Planning; Probability; Problem Solving; *Search Strategies; Statistical Analysis; Systems Development
IDENTIFIERS *Explanation Based Learning; *Knowledge Management

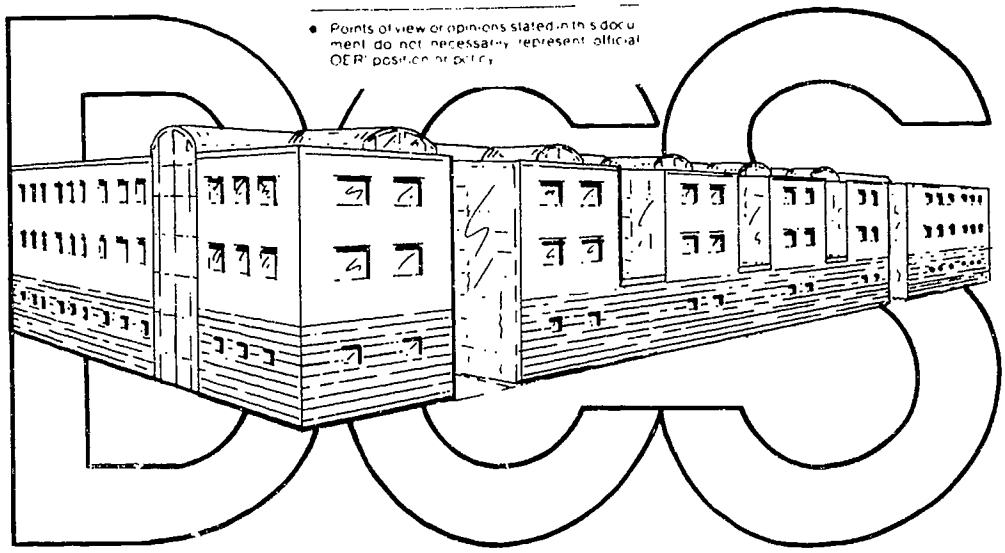
ABSTRACT

The PRODIGY/EBL system [Minton88] was one of the first works to directly attack the problem of strategy utility. The problem of finding effective strategies was reduced to the problem of finding effective rules. However, this paper illustrates limitations of the approach. There are two basic difficulties. The first arises from the fact that the utility of a control rule cannot be accurately determined from a single instance of the rule. This is a manifestation of a more basic problem which we term the utility generalization problem. The difficulty is that generalization techniques employed by speed-up learning systems are accuracy preserving but not utility preserving. The second difficulty is that control rules interact such that the utility of one control rule is a function of the other control rules in the system. This composability problem means that systems cannot reduce the problem of learning effective strategies to the problem of identifying rule utility in isolation. We document the seriousness of these problems with an example domain theory. With this theory, PRODIGY/EBL generates control strategies which are up to 17 times slower than the original planner. While this raises serious questions about the effectiveness of PRODIGY/EBL, we also claim that the utility generalization and composability problems are basic issues which are not adequately addressed by current speed-up learning techniques. We introduce an alternative technique called COMPOSER. This system is based on a sound statistical model which is validated with a series of experiments. COMPOSER successfully avoids the utility generalization and composability problems. (Contains 33 references.) (Author/ALF)

DEPARTMENT OF COMPUTER SCIENCE

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN





THE NEW ADDITION

REPORT NO. UIUCDCS-R-91-1681 UILU-ENG-91-1727

# Utility Generalization and Composability Problems in Explanation-Based Learning

by

Jonathan M. Gratch
Gerald F. DeJong



August 1991

# Utility Generalization and Composability Problems in Explanation–Based Learning

Jonathan M. Gratch and Gerald F. DeJong

Artificial Intelligence Research Group
Beckman Institute for Advanced Science and Technology
University of Illinois at Urbana–Champaign
405 North Matthews Avenue
Urbana, IL 61801

Telephone: (217) 244–1503
Internet: gratch@cs.uiuc.edu

**Topic Area: MACHINE LEARNING**

**Keywords**
*Explanation–based Learning*
*Utility Problem*
*Control Rules*

**Abstract**

The PRODIGY/EBL system [Minton88] was one of the first works to directly attack the problem of strategy utility. The problem of finding effective strategies was reduced to the problem of finding effective rules. However, this paper illustrates limitations of the approach. There are two basic difficulties. The first arises from the fact that the utility of a control rule cannot be accurately determined from a single instance of the rule. This is a manifestation of a more basic problem which we term the utility generalization problem. The difficulty is that the generalization techniques employed by speed–up learning systems are accuracy preserving but not utility preserving. The second difficulty is that control rules interact such that the utility of one control rule is a function of the other control rules in the system. This composability problem means that systems cannot reduce the problem of learning effective strategies to the problem of identifying rule utility in isolation. We document the seriousness of these problems with an example domain theory. With this theory, PRODIGY/EBL generates control strategies which are up to seventeen times slower than the original planner. While this raises serious questions about the effectiveness of PRODIGY/EBL, we also claim the the utility generalization and composability problems are basic issues which are not adequately addressed by current speed–up learning techniques. We introduce an alternative technique called COMPOSER. This system is based on a sound statistical model which is validated with a series of experiments. COMPOSER successfully avoids the utility generalization and composability problems.

## 1 INTRODUCTION

There is considerable research in machine learning into techniques to improve problem solving ability. Unfortunately, "speed–up learning" systems can result in substantial performance degradation [Etzioni90a, Minton85, Mooney89, Subramanian90, Tambe89]. Additionally, empirical claims of success are frequently shown to be sensitive to subtle changes to the experimental conditions [Gratch90, Mooney89, Segre91, Subramanian90]. It is not surprising that a basic question dominates research in this area: what is the value of knowledge?

There are two major approaches to identifying "good" knowledge. The first places syntactic restrictions on the learning mechanism such that it only generates beneficial knowledge. Researchers try to identify a set of domain independent syntactic constraints to discriminate helpful from harmful knowledge. Learning systems can then be designed to obey these constraints. We will use the term *operationality criteria* [Mitchell86] to refer to any set of domain independent syntactic constraints which limit the generation of knowledge. Many criteria have been proposed [Etzioni90a, Letovsky90, Segre87, Subramanian90].

A second approach is to compute a numeric estimate of the value of knowledge. This estimate is then used to discard harmful knowledge. The learning system implements a cost model and estimates parameters of this model through direct observation of problem solving behavior within a particular domain [Gratch91, Keller87, Leckie91, Minton88, Yoo91]. We use the term *utility analysis* for techniques which directly estimate the value of knowledge. The two approaches complement each other. Utility analysis allows an inexact operationality criteria. An accurate operationality criteria reduces the burden for utility analysis.

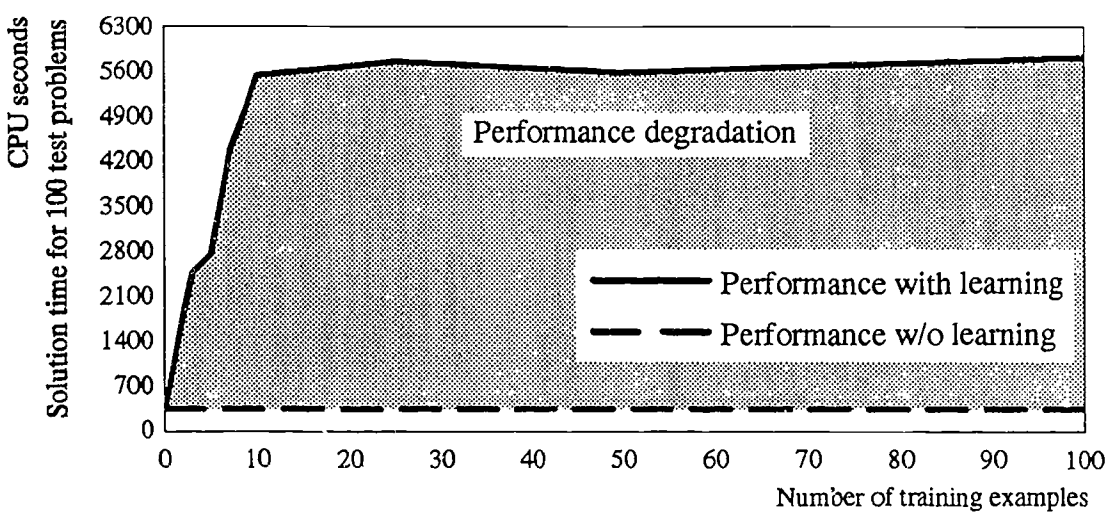



Figure 1: PRODIGY/EBL learning curve illustrating a harmful control strategy.
Results are averaged over ten trials.

In this paper we will relate an in–depth investigation of one approach to utility analysis: the utility analysis method of PRODIGY/EBL [Minton88] This is an approach which has shown empirical success on several domains. Unfortunately, this success is not guaranteed. Figure 1 illustrates a learning curve for PRODIGY/EBL on an artificial domain (described below). The learned strategy actually degrades performance by an order of magnitude. As we will show, there are many issues not adequately addressed by the PRODIGY/EBL method. We will then argue that these are fundamental problems, and are not adequately addressed by current approaches to utility analysis or operationality.

## 2 REVIEW OF PRODIGY/EBL

PRODIGY/EBL [Minton88] is a learning approach which enhances the effectiveness of an underlying STRIPS–like planner. The system uses explanation–based learning (EBL) [DeJong86, Mitchell86] to produce control rules from traces of problem solving behavior. Control rules are condition–action statements which alter the way the PRODIGY planner explores its problem spaces. By default, the planner lists all operators which unify with an unachieved goal and explores these alternatives depth–first. Control rules change the search by discarding or reordering some alternatives. Figure 2 illustrates a control rule learned by PRODIGY/EBL on the blocksworld domain. The blocksworld has several operators for clearing a block. RULE–1 asserts that in situations with an unheld block, only consider the UNSTACK operator.

RULE–1: IF *current–node* is ?n
*current–goal* at ?n is (CLEAR ?x)
(NOT (HOLDING ?x)) is true at ?n
THEN choose operator UNSTACK

Figure 2: An example control rule

### 2.1 Utility Analysis

Not all control rules increase the efficiency of planning. Control rules avoid search in the problem space, however they introduce the cost of matching their preconditions. A rule is harmful when the precondition evaluation cost exceeds the savings. PRODIGY/EBL incorporates utility analysis to avoid this situation. Minton proposes a cost model to captures the tradeoff between a control rule's savings and precondition match cost. The model associates a *utility* value with each control rule:

$$\text{UTILITY(rule)} = \text{Average\_Savings(rule)} \times \text{Success\_rate(rule)} - \text{Match\_cost(rule)} \quad (1a)$$

The utility of a control rule is the difference between the savings it produces (attenuated by the percent of time its preconditions are satisfied) and its precondition match cost. Savings, Success–rate, and Match–cost are parameters of the model which the system must estimate. Unfortunately, it is difficult to measure Average_Savings directly. To do so would require exploring the portions of the problem space which the rule avoids, nullifying the effect of the rule. To avoid this difficulty, PRODIGY/EBL is implemented with a simplified cost model:

$$\text{UTILITY}_{\text{PERCEIVED}}(r) = \text{Initial\_savings}(r) \times \text{Success\_rate}(r) - \text{Match\_cost}(r) \quad (1b)$$

This model derives Average–Savings from the savings which results on the instance from which the control rule was learned. Success–Rate and Match–cost are directly measured from subsequent problem solving experience. Minton assumes that perceived utility (Equation 1b) will be a close approximation to the true utility (Equation 1a).

### 2.1 Defining "Cost"

PRODIGY/EBL is based on an *average cost* model of utility. That is, the model considers a control strategy effective if it reduces average problem solving cost. This model does not entail that the cost of any particular problem will be reduced. Rather, the cost to solve any representative sample of problems will be less. The average cost model is ubiquitous in the speed–up learning community and we will not discuss its merits in this paper. One should be aware, however, that alternatives do exist.

Speed–up learning systems reduce the cost of problem solving. Therefore, it is paramount to define "cost" precisely. Many criteria are in use. One possibility is to emphasize solution quality; either

by guaranteeing optimality [Mostow89] or by defining cost metrics which prefer quality solutions [Eskey90]. Accuracy is another important dimension. Namely, what is the ratio of solvable to unsolvable problems. PRODIGY/EBL defines cost by CPU seconds required to solve problems. The system actually measures the time required to perform certain processes, and tries to reduce total problem solving time.

The choice of a cost criteria can have dramatic impact on system behavior. This issue is explored in detail in [Segre91]. We will briefly illustrate the difficulties in the context of PRODIGY/EBL. PRODIGY/EBL tries to reduce problem solving cost. This is easily accomplished by a single control rule which immediately fails to solve a problem. But this "fast" strategy reduces the accuracy of the problem solver to zero. Instead, the rule generator constrains control rules to be "truth preserving" in the sense that they only eliminate provably irrelevant portions of the search space. Thus, presumably, if a problem is solvable, it cannot become unsolvable with learning.

Unfortunately, there is a further complication. Problem solving is combinatorially expensive. Planners, like PRODIGY, impose resource limitations on their problem solving. This makes problem solving tractable at the expense of accuracy. The planner simply aborts problem solving when it reaches the resource limit. It might appear that PRODIGY/EBL can enhance enhance accuracy by simply minimizing problem solving cost. This happens when a problem which is to expensive to solve becomes solvable with the learned strategy. But a learned control strategy can also reduce accuracy. This is a legacy of the average cost model of utility. By reducing average cost, a strategy can increase the cost of certain problems. The technique reduces accuracy if the solutions to these problems require more resources that the limit allows.

Finally, there is an issue of how to account for the resources expended during training. The most popular approach is to assume learning cost can be amortized over a large body to test problems. Minton adopts this approach and learning cost does not participate in his performance data. There are some alternative approaches. The training phase can be shown to be tractable (i.e., polynomial) [Natarajan89, Tadepalli91]. Another possibility is to include training time in the cost models [Yamada91].

### 2.3 Assumptions

In this paper we will preserve several of the assumptions embodied in PRODIGY/EBL. Therefore, we assume our goal is to increase the efficiency of *satisficing search* [Simon75]. In this situation the problem solver may search for any valid solution. Therefore, as in PRODIGY/EBL, solution quality is not an issue. We will also discount training cost, assuming it can be amortized over future problem solving. We make one additional assumption to avoid tradeoffs between efficiency and accuracy. For the remainder of this paper we assume that all problems are solvable within the resource bounds of the PRODIGY planner. Together with the assumption the PRODIGY/EBL generates truth preserving rules, this insures that reducing CPU cost does not affect problem solving accuracy.

## 3 CRITIQUE OF PRODIGY/EBL: Single-rule strategies

For the moment we will ignore how rules combine and consider the reduced problem of finding an effective single-rule control strategy. The system may learn a control rule if the planner finds no solution in a large subtree of the problem space. PRODIGY/EBL analyzes such instances of wasted effort and proposes control rules to avoid the situation. What we will show is that is that PRODIGY/EBL cannot accurately determine rule utility. As will become apparent, we refer to this as the *utility generalization problem.*

### 3.1 Savings Variance

When PRODIGY/EBL learns a control rule, it applies to a particular planning context. This context is defined by a world state and a set of unsatisfied goals. PRODIGY/EBL uses analytic techniques to generalize the rule, ignoring aspects of the context which did not participate in the failure. The resulting rule can then apply to a large set of planning situations and we are guaranteed that in each instance, the rule avoids fruitless alternatives[1]. While the generalization preserves correctness, it does not guarantee that the savings observed in the training instance will reflect the savings everywhere the rule applies. If savings varies too much, it is unlikely that the initial observation of rule savings will reflect the average. This violates the assumption that perceived utility approximates true utility.

In practice, the savings induced by a rule is highly dependent on information dropped by generalization. To illustrate this, consider Figure 3 which displays a portion of a search space for the blocks–world domain. Boxes contain the current goals at a node. Operators connect boxes. The goal is to clear block B. There are three operators which achieve this effect: UNSTACK, PUTDOWN, and STACK. Each operator may apply to multiple blocks. This results in ten alternative paths for satisfying the goal. Assume that we explore the space from left to right and from top to bottom. If the rule in Figure 2 is available to the planner, it eliminates the two alternatives using the PUTDOWN operator and the four alternatives using the STACK operator (six of the ten choices).

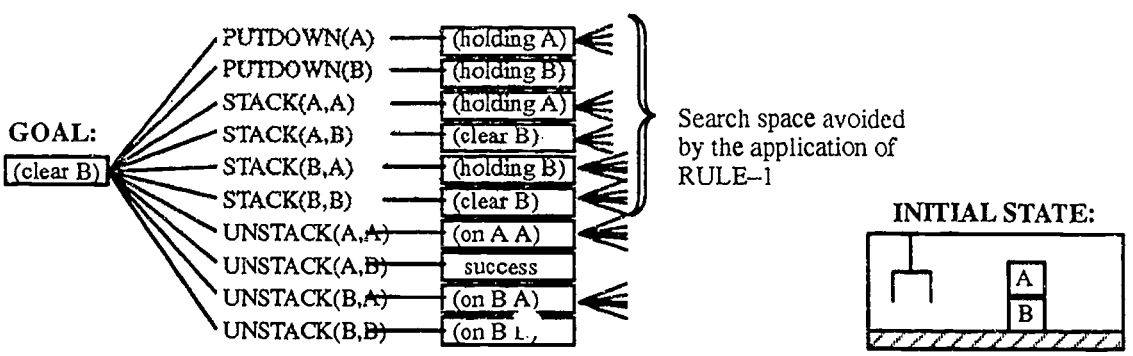



Figure 3: An example search space for the blocksworld domain

Next, consider adding an irrelevant block to the table. This creates more ways to instantiate PUTDOWN and UNSTACK (three for PUTDOWN and eight for UNSTACK), each of which is trimmed by RULE–1. In general, this rule saves $n + n(n-1)$ alternatives where $n$ is the number of blocks in the current state. The lesson is that savings provided by a rule on its generalized set of instances may vary greatly [Gratch91].

### 3.1 Quantifying the Effects of Variance

We can borrow notions from statistics to understand how the variance in savings effects utility analysis. We can view the savings that a control rule provides as a random variable (**SAV**). Rule savings can then be described by its average ($\overline{\mathbf{SAV}}$), and a probability density function (p.d.f.). The bell–shaped curves in Figure 4 are examples of p.d.f.'s. The horizontal axis describes legal values for the random variable. The vertical axis represents probability. The probability that an instance of the variable will lie within a specified range is the integral of the p.d.f over that range. Average savings must be attenuated by success rate before it can be compared with the match cost. To simplify

1. This guarantee only holds for control rules which eliminate alternatives. PRODIGY/EBL can also learn rules which re–order alternatives. These "preference rules" are only heuristics.

the discussion, we define another random variable **S** = **SAV**×Success_rate(r). $\overline{\mathbf{S}}$ = $\overline{\mathbf{SAV}}$×Success_rate(r).

We can use these properties to discuss the likelihood that the utility of a single–rule strategy will be mis–represented. There are two ways in which the utility analysis can err. Either the system can keep a harmful rule (false positive) or the system can discard a helpful rule (false negative). Each case is the dual of the other so we only discuss the case of false positives.

Two conditions must hold to retain a control rule with negative utility. First, the learning module must generate a rule with negative utility (a failure of the operationality criteria). Second the rule must have positive perceived utility (a failure of utility analysis). In terms of Equations 1a and 1b, $\overline{\mathbf{S}}$ < Match_cost(r) and Initial_Savings(r)×Success_rate(r) > Match_cost(r). The likelihood of the former depends on the effectiveness of the generation bias. The likelihood of the latter depends on the p.d.f for **SAV**. Problems with operationality criterion will be discussed in Section 7. Here we will consider the probability of a false–positive given that the generator produced a rule with negative utility.

If a generated rule has negative true utility, in can be mistakenly retained. Figure 4 illustrates the probability of a false–positive for two such control rules. The control rules have identical p.d.f.'s but different match costs. $\overline{\mathbf{S}}_i$ is the average savings times success rate for rule *i*. $\overline{\mathbf{C}}_i$ is the match cost for the rule. |Utility| is difference between $\overline{\mathbf{S}}_i$ and $\overline{\mathbf{C}}_1$. Control rule 2 has greater match cost and therefore its utility is more negative than the utility of control rule 1.

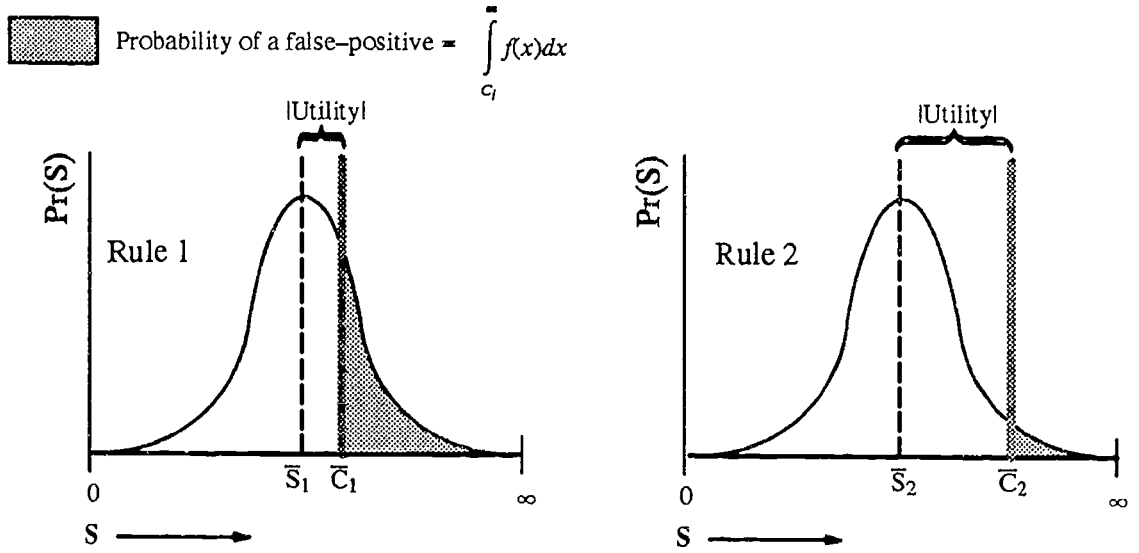



Figure 4: Probability of a false–positive for two control rules.

To mistakenly retain a rule, the system must overestimate the average savings such that savings appears greater than cost. As this estimate is based on a single observation drawn from the p.d.f, the chance of obtaining a false positive is simply the probability mass to the right of the average cost (the shaded region of the p.d.f.). Notice that the probability of mis–classifying control rule 2 is much less than the probability of mis–classifying control rule 1. This p.d.f. has the desirable property that as the difference between savings and cost grows, the probability of mis–classification diminishes In other words, when mistakes are made, they are likely to be small. A very different situation is

illustrated in Figure 5. In this case the p.d.f has a bi-modal distribution. This is an example of one class of p.d.f's which can allow large mistakes to occur with high probability.

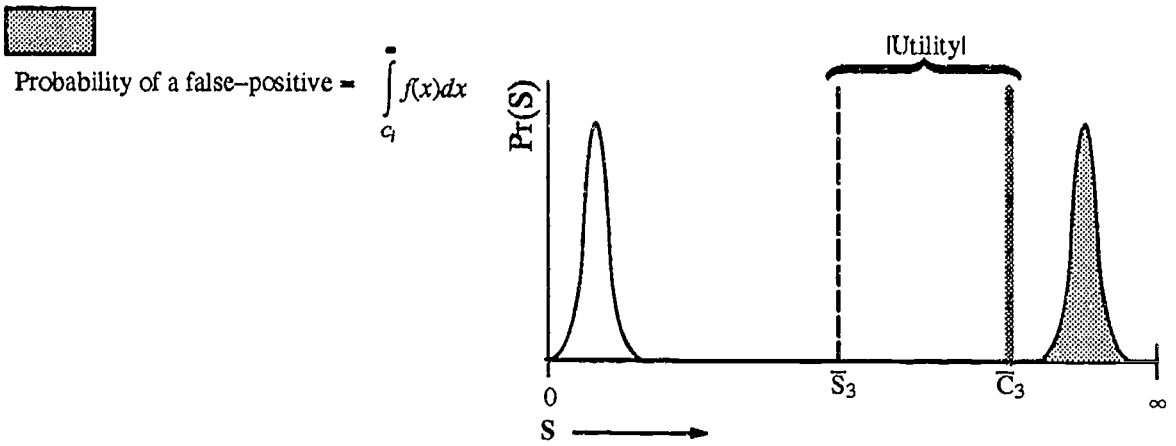



Figure 5: Probability of a false–positive given a bi–modal p.d.f.

The previous discussion highlights the importance of *bounded error*. False positives may be acceptable if we ensure that the mistakes are small. We can ensure Equation 1 exhibits bounded error if a combination of the following properties hold:

1) *small average match cost* — in the worst case a rule will save nothing. Utility is then 0 – Match–cost(r). By bounding match cost we can guarantee small negative utility.

2) *savings has small variance* — this reduces the likelihood of large discrepancies between estimated and actual savings.

3) *savings is normally distributed* — this ensures that the likelihood of a false positive diminishes with the harmfulness of the rule.

Some of these properties hold in the domains PRODIGY/EBL is tested on. For example, problems are generated by a procedure which randomly varies several problem parameters. These parameters exhibit little variance. In the STRIPS domain the number of blocks present in the world vary from two to five. In the scheduling domain the number of objects vary from two to four. The savings for many control rules learned in these domains vary with these parameters. Because the parameters do not vary, these control rules exhibit the small savings variance property.

## 4 DOCUMENTING SINGLE–RULE MISTAKES

The preceding section illustrates the shortcommings of the utility analysis described by Equation 1b. In this section we illustrate a simple domain (Figure 6) which exhibits this problem. The domain theory will also be utilized in the experiment in Section 6. The domain is for a robot assembly task where the goal is to construct a component from its parts. All parts for a component are contained in a parts bin. If all the parts in the bin are free of defects, the component may be assembled. Otherwise another bin must be found.

When PRODIGY/EBL is given a problem in this domain, it considers multiple instantiations of the INSPECT–BIN operator — one for each bin in the initial state. If the first bin contains a defect, it produces the control rule in Figure 7. As future problems are solved, this rule avoids instantiations of INSPECT–BIN which lead to failure. As in RULE–1 above, the savings provided by this rule depends on information not mentioned in the rule. The savings increases as we increase the number

| ASSEMBLE–COMPONENTS | INSPECT BIN |
|---|---|
| PRECONDITIONS:<br>∃ ?BIN : parts–bin(?BIN)<br>defect–free–components(?BIN)<br>ADD:<br>assembly–complete() | PRECONDITIONS:<br>∀ ?PART : in–bin(?PART ?BIN)<br>good(?BIN ?PART)<br>ADD:<br>defect–free–components(?BIN) |

Figure 6: A simple assembly domain

of parts per bin. If bin size exhibits a large variance then the small savings variance property will be violated. We exploit this property to demonstrate single–rule failures of Equation 1b.

RULE–2: IF *current–node* is ?node
*current–goal* at ?node is assembly–complete()
*current–operator* at ?node is INSPECT–BIN
*candidate–bindings* at ?node is (?bin)
∀ ?part : in–bin(?part ?bin)
good(?bin ?part)
*candidate–bindings* at ?node is (?other–bin)
THEN prefer (?bin) to (?other–bin)

Figure 7: A control rule from the assembly domain

It is not immediately apparent why RULE–2 would be conjectured. The control rule examines all contents of a bin to decide if the planner should examine all contents of a bin. In fact, this rule reduces planning time in many cases. The rule avoids the overhead of generating a problem space (generating intermediate nodes, searching the domain theory for relevant operators, etc.). However, the potential effectiveness of the control rule is irrelevant to this discussion. The importance of utility analysis is that permits harmful rules to be generated. What we demonstrate in this section is that PRODIGY/EBL fails in this task. The reasons for this failure are independent of the actual form of the control rule.

### 4.1 Methodology

We violate the small savings variance property by creating a problem distribution which varies bin size bi–modally. We accomplish this with two classes of problems. In each problem class the first bin contains defects. This forces the planner to backtrack and, consequently, to produce RULE–2. Equation 1b credits the rule with an average savings commensurate with the number of parts in this bin. The first class contains problems with fifty bins of two parts each. The second class contains problems with two bins of two hundred parts each. If PRODIGY/EBL learns the rule on a problem from the first class, it should have little perceived savings. If it learns the rule on a problem from the second class, it should have high perceived savings.

Problems are randomly generated, half from the first class and half from the second. Using this distribution, we train PRODIGY/EBL following the methodology outlined in Minton's thesis [Minton88 pp. 117–118]. We present the system with 100 training problems followed by a "settling phase" of 25 problems. The settling phase is required so that control rules learned at the end of the training phase can undergo utility analysis. This regimen is repeated for ten independent trials with different problem sets (from the same distribution) on each trial.

All trials were executed on an IBM RT 125 with 16MB of memory, using LUCID Common LISP and PRODIGY 2.0.[2]

### 4.2 Results

Results are summarized in Table 1. This reports the mean problem solving time across the ten trials. We computed a 95% confidence interval for each mean using a t–test. We also generated a learning curve, illustrated in Figure 1. This is constructed with the same regimen but varying the size of the training set.

| System Type | Execution Time (100 problems) |
|---|---|
| without learning | 346±9 CPU sec. |
| with learning | 5839±98 CPU sec. |

Table 1: Empirical results from single–rule experiment

RULE–2 produces a large performance degradation for problems from the first class (50 bins of 200 parts each). and a moderate performance enhancement for problems of the second class (2 bins of 200 parts each). The overall effect is a large performance degradation. If the rule is learned on a problem from the first class, PRODIGY/EBL uniformly perceives the rule to produce little savings. In this case utility analysis correctly discards the rule. If learned from the second class, the system uniformly perceives the rule to have high savings. Thus, the rule is mistakenly retained. Discarded rules may be relearned, so eventually the rule is learned on a problem from the second class.

The results indicate that learning substantially degrades problem solving performance. From this we can conclude that perceived utility can substantially diverge from true utility. Thus the utility analysis embodied by Equation 1b can retain rules with high negative utility.

The experiment also illustrates the potential to discard a good rule (false–negative). PRODIGY/EBL produces a small savings estimate for RULE–2 if it is learned on a problem with small bin size. This results in an underestimate of savings when the system solves problems with large bin size. In this the underestimate did not effect the performance of utility analysis because the rule has negative utility. However false negatives could result if the rule has positive utility. This could be achieved, for example, by increasing the likelihood of problems with high bin size.

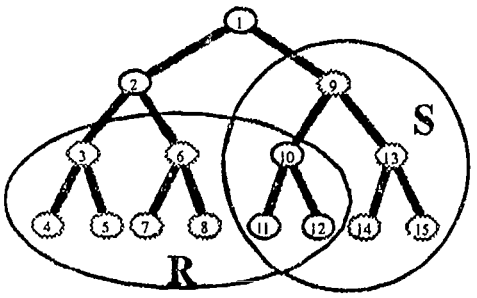
## 5 CRITIQUE OF PRODIGY/EBL: Multi–rule strategies

As we have seen, Equation 1b may misrepresent the utility of a control rule. This section illustrates that even with accurate savings estimates, this utility analysis can still produce undesirable results. The problem is that control rules may interact such that the utility of multiple control rules cannot be predicted by simply knowing their utilities in isolation. This dependency is noted in Markovitch's definition for the value of knowledge [Markovitch89 pp. 6–7] We call this property the *composability problem.*

2. PRODIGY is available through Carnegie Mellon University. Contact prodigy@cs.cmu.edu. The domain theory and problem generators used in these experiments are available upon request from the authors. Contact gratch@cs.uiuc.edu..

There are many ways that the presence of one control rule can influence the utility of another. Two rules may avoid the same areas of the problem space. As there is no added benefit in ignoring an area twice, the utility of the rules together is not equivalent to the sum of their individual utilities. A subtle example occurs when a control rule has different match costs in different portions of the problem space. A second control rule which removes portions of this problem space may substantially change the average match cost of the first rule.

A particular interaction between two control rules is illustrated in Figure 8. This shows a hypothetical problem space of fifteen nodes. Supposed $r$ and $s$ are two control rules which prune the nodes in sets R and S respectively when compared to problem–solving with no control rules. |R| is the number of nodes trimmed by rule $r$. |S| is similarly defined. When used in isolation, rule $r$ is checked six times (i.e. 15 – |R|). It successfully applies twice: at node 2 saving nodes 3–8 and at node 9 saving nodes 10–12. Rule $s$ is checked eight times (i.e. 15 – |S|) and succeeds at node 1, saving nodes 9–15. Assume the average match cost of $r$ is $M_r$, the average match cost of $s$ is $M_s$, and the average cost to expand a node is $g$.




R–S = nodes saved only by rule $r$
S–R = nodes saved only by rule $s$
R∩S = nodes saved by both rules $r$ and $s$

$M_r$ = Average match cost of rule $r$
$M_s$ = Average match cost of rule $s$
$g$ = Average cost to expand a node

Figure 8: example of interacting rules

Utility(X) is the utility of a set of control rules. The interaction between two rules is the amount to which their utilities are not additive:

$$\text{Residue} = \text{Utility}(\{r, s\}) - [\text{Utility}(\{r\}) + \text{Utility}(\{s\})] = |R-S| \times M_s + |S-R| \times M_r - |R \cap S| \times g \quad (2)$$

The residue in Equation 2 is the amount by which the utilities of $r$ and $s$ are not composable. The rules combine synergistically if this value is positive. If negative, they engage in a harmful interaction. Two rules with positive utility can potentially combine to yield a strategy worse than neither.

Interactions force us to discard the notion of rule utility as defined in Equation 1. Instead, we propose *conditional utility* to capture the benefit of a control rule. The conditional utility of a rule is the change in performance a rule provides when added to an existing set of rules. Thus, for the example in Figure 8, Utility($s|r$) is the utility of adding rule $s$ to an existing strategy of rule $r$ alone. More generally, for any two sets of rules X and Y:

$$\text{Utility}(X \cup Y|\varnothing) = \text{Utility}(X|\varnothing) + \text{Utility}(Y|X) \quad (3)$$

where $\varnothing$ is the empty strategy. For a rule $r$, the utility of $r$ in Equation 1 is equivalent to its conditional utility with respect to the empty set of rules: Utility($r$) = Utility($r|\varnothing$).

Ignoring these interactions can lead to degraded performance. As the savings estimate is fixed at learning time, the perceived savings can diverge from the true average. The directly measured parameters(Success–rate and Match–cost) are also impacted by this property. PRODIGY/EBL bases these parameters on the average of many observations. For an average to be meaningful the observa-

tions must be drawn from the same distribution. However, as control rules are acquired or discarded, the distribution can change. As a result, these parameters have questionable semantics.

## 6 DOCUMENTING MULTI–RULE MISTAKES

The preceding section suggests another way that the utility analysis described by Equation 1b can fail. In this section we illustrate a simple control rule interaction which degrades performance in our assembly domain. Again we use two problem classes, each of which has equal representation. The first class has forty bins of 20 parts each, most of which have defects. The second class has forty bins of 20 parts each, all of which have defects.

The first problem class results in the generation of RULE–2 from Figure 7. In combination with the second class of problems, this rule has high negative utility and, after a few subsequent problems, utility analysis correctly discards it (notice that bin size does not vary in these problem classes). The second problem class results in the generation of RULE–3 in Figure 9. This rule checks every part in every bin, searching for a defect free bin. If it does not find such a bin, it terminates problem solving. This rule has high negative utility and is quickly discarded by utility analysis.

```
RULE–3:  IF candidate–node is ?node
            is–top–level–goal assembly–complete()
            ∀ ?bin : is–bin(?bin)
              ¬defect–free–components(?bin)
               ∃ ?part : is–part(?part ?bin)
                   ¬good(?bin ?part)
         THEN  reject ?node
```

Figure 9: A control rule from the assembly domain

A different situation arises if the system learns RULE–3 before it discards RULE–2. RULE–2 is expensive to match on problems from the second class, and it provides no savings (there is no defect–free bin to prefer). As a result, the problem takes much longer to solve. This greater problem solving time is reflected in a greater savings estimate for RULE–3. With this estimate, RULE–3 is retained. If RULE–2 remained in the system, this estimate would accurately reflect the savings for RULE–3. However, as RULE–2 has negative utility, utility analysis eventually discards it. When it is discarded, the estimate is not updated and RULE–3 is mistakenly retained.

We tested this domain using the same methodology as in section 4. It is possible that the degradation could arise through factors other than the composability problem. We control for this situation by introducing another test condition. Our analysis indicates that RULE–3 is retained through an interaction with RULE–2. If this analysis is correct, RULE–3 should not be learned if RULE–2 is never learned. The new test condition prevents the learning of RULE–2.

Table 2 summarizes the results. PRODIGY/EBL learned the control strategy containing RULE–3. This degraded performance by a factor of three. When RULE–2 is suppressed, no rule is acquired, yielding results equivalent to the condition without learning[3]. This confirms that PRODIGY/EBL retains RULE–3 through a control rule interaction.

## 7 OPERATIONALITY CRITERION

In this section we argue that the limitations in PRODIGY/EBL's utility analysis translate into general problems for speed–up learning. We illustrate this by considering the alternative argument. PRODI-

3. As the control condition acquired no control rules, the same timing data is reported for the no learning and the control conditions.

| System Type | Execution Time (100 problems) |
|---|---|
| without learning | 2292 ±4 CPU sec. |
| with learning | 7436± 81 CPU sec. |
| without RULE–2 | 2292± 4 CPU sec. |

Table 2: Empirical results from multi–rule experiment

GY/EBL can acquire harmful knowledge. But this is a reflection of two failures. First the knowledge must be mistakenly generated and then it must be mistakenly retained. We have only demonstrated the latter failure. A better rule generator would avoid the former. In fact, a perfect rule generator would obviate the need for utility analysis. Much of the research in speed–up learning investigates alternate criteria for generating knowledge.

The composability problem raises a serious obstacle to this argument. An operationality criteria is designed to prevent the generation of harmful rules. However, the existence of rule interactions calls into questions the the notion of a harmful rule. A control rule which is harmful in one context may result in improved performance in a different context. Most criteria ignore these interactions (e.g., [Etzioni90a, Letovsky90, Mitchell86, Segre87, Subramanian90, Tambe89, Yamada89]). Furthermore, reasoning about interactions can be costly. A set of $i$ control rules yields $2^i$ distinct control strategies (the power set of the $i$ rules). In the worst case we must consider all these alternatives.

The variance in savings also raises difficulties. Most current criteria ignore distribution information. For example, the *nonrecursive hypothesis* [Etzioni90b] states that explanation–based learning "is effective when it is able to curtail search via nonrecursive explanations." A recursive explanation contains assertions which depend on instances of the same assertion. An example is where a sorted list is explained by explaining how sublists are sorted. This hypothesis claims that beneficial rules can be identified by their syntactic structure alone. A similar claim is stated in [Letovsky90, Subramanian90] in the context of macro–operators.

The harmful control rules learned in our experiments are nonrecursive by the definitions in [Etzioni90a, Letovsky90, Subramanian90] which directly contradicts the nonrecursive hypothesis. These experiments solidly demonstrate that utility varies across problems. From this we must conclude that utility for a rule depends on the problem distribution. For example, RULE–2 enhances performance if we limit the distribution to problems with large bin size. Criteria which ignore distribution information are insufficient. Furthermore, it is difficult to obtain this information. A system must know more that the distribution of problems. It must know the distribution of rule applications both within and across these problems. It must also know how features of these rule application impact utility. This is especially difficult as these features may not appear in the body of the control rule (e.g., the utility for RULE–2 varies with the size of the bin).

Researches have not addressed these limitations, in part, because of the historical development of the field. Speed–up learning techniques evolved from earlier work in concept learning. A concept learning system must learn classification rules to identify some target concept accurately. Thus, the focus was on techniques which produced accurate generalizations of examples. In the context of speed–up learning these techniques can accurately generalize the conditions for applying a control decision. We have inherited this focus on accuracy. However, in speed–up learning, accuracy is no longer the primary issue. Instead a system must balance accuracy with efficiency [Keller87]. Accuracy is only tenuously related to efficiency. For example, RULE–2 accurately predicts when a bin of arbitrary size will succeed. It benefits the system if generalized to problems with large bin size. However, its effects are disastrous when applied to the full range of sizes. But in each case the rule is accurate.

The weak link between accuracy and efficiency is observed in other systems as well. For example, Carlson, Weinberg, and Fisher [Carlson90] learn strategies with a probabilistic concept hierarchy. This approach accurately eliminates fruitless alternatives, but it produces strategies with worse execution time (negative utility). A similar effect is observed in DÆDALUS, a case–based planner which incorporates macro–operators into a probabilistic concept hierarchy [Allen90].

## 8 PERFORMANCE ELEMENT

Different problem solvers implement different search mechanisms. One way to view this is to say that a problem solver implements a body of default control knowledge. From this perspective, the composability problem suggests that the same learned control knowledge should have different utility when used with different problem solvers. Indeed, Mooney demonstrated this in several experiments [Mooney89]. He shows that macro–operators have very different effects when used with a depth–first planner or a breadth–first planner.

## 9 COMPOSER

Utility is a complex function of the problem solver, the structure of the domain theory, a possibly unknown problem distribution, and other learned knowledge. In this section we introduce a statistical approach to utility analysis, called COMPOSER, which addresses these issues. The technique is implemented in conjunction with PRODIGY/EBL and takes the place of the utility analysis of Equation 1b.

Equation 3 suggests a simple hill climbing approach for avoiding interactions. If a control rule has positive conditional utility with respect to a control strategy X, adding the control rule to X must result in a more effective strategy. The greedy technique begins with X initialized to the empty set and incrementally adds to X a control rule with the highest estimated conditional utility with respect to X. This cycle continues until no rule remains with positive conditional utility. In this way the problem of finding an effective control strategy is reduced to the problem of finding a control rule with positive conditional utility.

PRODIGY/EBL misrepresents the utility of a single control rule because it is restricted to a single observation of the rule's savings. We avoid this limitation by using many observations. These observations are combined to derive a mean utility and a confidence interval on that mean. Observations are made with respect to the current control strategy, and the method allows multiple rules to be evaluated simultaneously.

The COMPOSER approach works in conjunction with an existing planner and control rule generating system. Our implementation is built on top of the PRODIGY/EBL system but it can be readily

adapted to work with alternative rule conjecturing schemes. PRODIGY/EBL is provided with several control rule classes. Our implementation currently implements only a subset of these classes. We implement rejection rules and selection rules which unequivocally remove alternatives. Preference rules are not implemented but we anticipate little difficulty in extending the approach to this class.

### 9.1 Gathering Observations

Learning proceeds much as in PRODIGY/EBL. The planner generates solutions and a problem solving trace. As in PRODIGY/EBL, the trace includes the resources spent at each node, including time spent evaluating control rules. The PRODIGY/EBL learning module analyzes this trace and conjectures control rules. However, instead of directly adding these rules to the current control strategy, they are placed on a list of pending rules. Pending rules are allowed to match against the current planner state and the match cost recorded. However the actions of pending rules are not performed. Rather the system annotates the problem trace with the choices it would have eliminated. After a problem run is complete, the cost of each subtree which would have been pruned can be determined. If the control rule is checked but does not apply, it is credited with zero savings.

We can illustrate this with an example. Recall RULE–1 in Figure 2 and the blocksworld search space, reproduced in Figure 10. If RULE–1 is on the pending list, it is consulted as the planner explores (generates) the problem space. In this case example the rule applies at node N1. If the rule was allowed to apply it would eliminate the first six alternatives. As the rule is on the pending list, these alternatives are not eliminated. Instead a marker is placed on each link. After problem solving is complete, these markers are identified and the resources expended in the subtree below the marker are recorded. This total is the potential savings for the particular rule application associated with that marker.

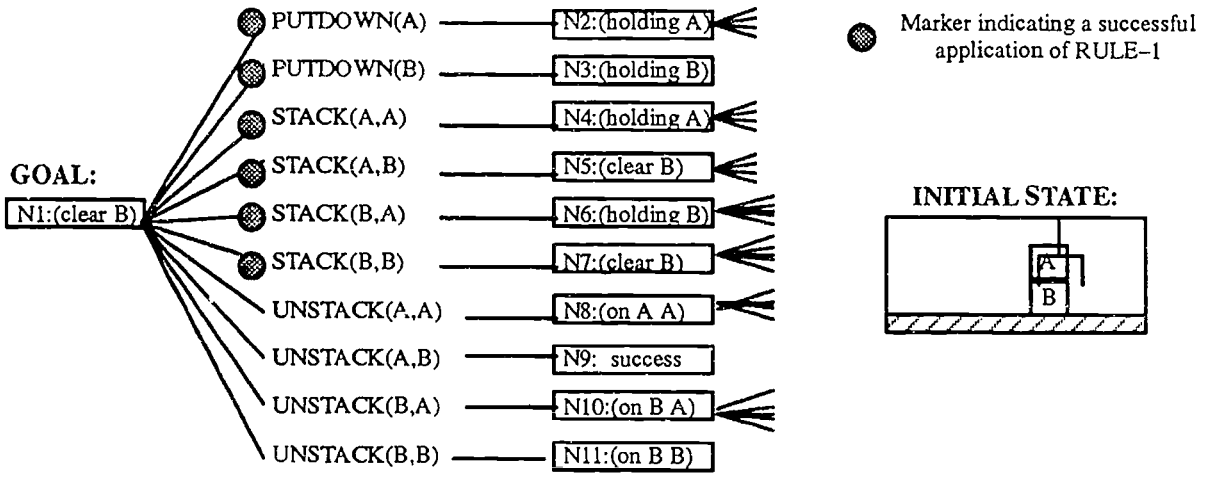



Figure 10: COMPOSER analyzing a search space for the blocksworld domain

There are three additional points. First, the system must not attempt a pending control rule in a portion of the space which would have been trimmed by that rule. For example, RULE–1, if activated, trims the first six alternatives of N1. Since the rule is pending, these alternatives are explored. It is possible that the rule applies in other nodes within these alternatives. For example the rule potentially applies at nodes N5 and N7. These must not be considered as valid applications because they would never have been reached if the control rule was activated.

Second, we require that pending rules do not effect the behavior of the planner. However, the planner expends resources to evaluate the preconditions of pending rules. If the planner has resource bounds (as in PRODIGY), these bounds must be insensitive to this additional cost. The match cost of pending rules must also be discounted when summing the resources expended below an application marker.

Finally, all of the observations are contingent on a particular control strategy. All observations must be discarded each time a new control rule is added to the current strategy. To see why contexts effects are important, imagine that a rule which eliminates the STACK alternatives is selected as the next active rule. Before this addition, RULE–1 saved all six unsuccessful alternatives in the example. With this new rule, RULE–1 only avoids the two PUTDOWN alternatives. If we did not discard the old observations, the estimate would be skewed by these higher savings observations.

### 9.2 Estimating Utility

We now have a mechanism for gathering observations of conditional utility. To identify the pending rule with the highest conditional utility we must compute an average conditional utility for a control rule. We must place a confidence bound on this mean. Only control rules which have positive utility with high confidence will be added to the current strategy. Deriving a bound is difficult because utility varies within any given problem. Furthermore, different problems will have different patterns of variance. For example, one problem may be dominated by control rule applications which have positive utility, while another problem may be dominated with application of negative utility. The final mean must reflect the composite of these individual distributions. Standard statistical approaches require sampling utility randomly from any place within any problem. Unfortunately the constraints of problem solving force us to sample at the level of complete problems. This means that our observations will consist of all the rule applications in one randomly selected problem, followed by all the rule applications of the next randomly selected problem, etc. We describe a statistical technique known as *cluster sampling* which is designed for this task. Our description of this technique is derived from the presentation in [Kish65 pp. 148–216].

The basic problem is that both the total utility (i.e., the numerator of the sample mean) and the number of rule applications (i.e., the denominator of the sample mean) are random variables. Because we are sampling problems randomly from a population of problems and the number of control rule applications within problems in not constant across problems, the sample size is random. The sampling plan can be thought of as a cluster sample with problems representing the clusters and rule applications representing observations within the clusters. Rule applications appear in the sample because their problem was selected (i.e., problems are the primary sampling unit).

We first introduce some notation. These definitions are with respect to a particular rule, R.

$u_{ij}$ is the utility of the *ith* rule application of R within the *jth* problem. Utility is the savings resulting from that application minus the match cost for that application

$x_j$ is the number of applications of rule R in problem (cluster) $j$

$u_j = \sum_i u_{ij}$, $(i = 1, x_j)$, is the sum of the utilities for each application of rule R in problem $j$. This is also called the *sample total* for cluster $j$.

$u = \sum_j u_j$, $(j = 1, a)$, is the sum of the sample totals for each problem where there are $a$ problems selected from a population of size $A$

$x = \sum_j x_j$, $(j = 1, a)$, is the total number of applications of rule R in the selected problems.

Then the average utility of R over its applications is

$$r = u/x = (1/x)\sum_j u_j = (\sum_j u_j) / (\sum_j x_j)$$

Note that $r$ is also a weighted mean of the problem means:

$$r = u/x$$
$$= (1/a)\Sigma_j [x_j/(x/a)]y'_j \text{ where } y'_j = y_j/x_j$$

Thus, the average utility of rule R for each problem is weighted by the specific number of rule applications in that problem relative to the average number of rule applications across all problems.

Given these relations and definitions, the variance of the average utility for rule R is found to be:

$$\text{Var}(r) = (1/x^2)[\text{Var}(u) + r^2 \text{Var}(x) - 2 r \text{Cov}(u,x)]$$

where Var( ) indicates variance of the variable in ( ) and Cov( ) indicates the covariance between the two variables listed in ( ). The variables are as previously defined.

This formula is approximate and is reliable only if $se(x)/x < .20$ where $se(x)$ is the standard error of x.

There are many equivalent expressions for the variance. We will utilize the following expression. A derivation can be found in [Kish65 p. 189, Equation 6.3.6]:[4]

$$\text{Var}(r) = 1/ [a(a-1)] \{ \Sigma_j [(x_j\, a/x)(y'_j - r)]^2 \}$$

That is, the squared deviations of the problem means from the grand mean $(y'_j - r)^2$ are weighted by the relative sample sizes in the clusters $(x_j/(x/a))$.

### 9.3 Putting it Together

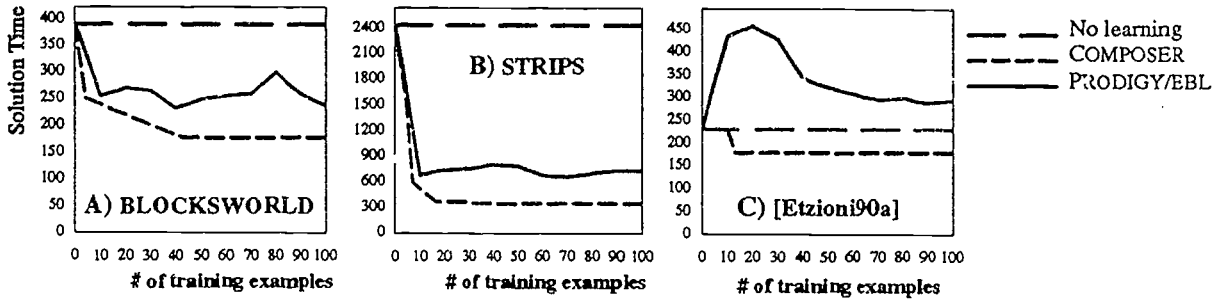
With cluster sampling we can combine the observations of conditional utility for a particular control rule into a meaningful average and a bound on that average. An average and bound is maintained for each rule on the pending list. After each problem solving attempt, COMPOSER updates the statistics for pending rules and then considers incorporating a control rule into the current strategy. A control rule is only considered for inclusion if it has positive utility within a confidence interval of $\delta$. For our current implementation $\delta$ is set arbitrarily at 95%. After each problem is executed, the system checks if any rules satisfy the confidence requirement. If so, COMPOSER adds the rule with highest positive to the current strategy, and removes this rule from the pending list. Statistics for the remaining pending rules are discarded as they are meaningless in the context of the resulting control strategy. The same method identifies rules with negative utility. If a control rule has negative utility with confidence $\delta$, it is eliminated from the pending list. This operation does not affect the current strategy, so the statistics associated with the remaining pending rules are left unchanged. This cycle is repeated until the training set is exhausted.

### 9.4 Evaluating COMPOSER

Necessarily, adding a rule of positive conditional utility will increase the efficacy of the composite strategy. The COMPOSER technique could fail, however, if conditional utility is not properly estimated. To test this possibility, we instigated a series of experiments which are summarized in Figure 11. These graphs illustrate learning curves where the independent measure is the number of random training examples and the dependent measure is execution time for 100 test problems. The methodology is identical to that described in Section 4. As COMPOSER does not implement preference

4. Kish multiplies this equation by a faction $(1-f)$ where $f$ is the probability of selecting a particular rule application, which is the same for all applications. Specifically, $f = f_a f_b$ where $f_a$ is the ratio of $a$/A (the number of problems selected relative to the size of the population of available problems), and $f_b$ is the fixed probability of selecting a rule application within a problem. In our case we will use all rule applications associated with a problem. Consequently, $f_b = 1.0$. In addition we will assume the general case where there are an infinite number of problems. In this case $a$/A approaches zero. Tighter bounds can be achieved if A is finite.

rules, differences in performance could be attributed to this difference, rather than the improved utility analysis. To control for this effect we tested two versions of PRODIGY/EBL: the default version and a version which cannot learn preference rules. Results are presented for the most effective of these two systems, which in each case was the system without preference rules. All problem generators were supplied with the PRODIGY 2.0 system. More effective strategies have *lower* solution times. Ideally COMPOSER should be compared against the optimum control strategy but it is computationally infeasible to do this. Instead we provide PRODIGY without learning and PRODIGY/EBL as benchmarks. The systems are tested on two domains from [Minton88] and the domain in [Etzioni90a] for which PRODIGY/EBL produced harmful strategies. The system could not be tested on the two domains reported here as these involve preference rules which have not been implemented in COMPOSER. However, in similar domains which did not involve the learning of preference rules, COMPOSER accurately avoided learning harmful control rules.




| DOMAIN | COMPOSER | | PRODIGY/EBL | | No Learning |
|---|---|---|---|---|---|
| | Rules Learned | Solution Time | Rules Learned | Solution Time | Solution Time |
| A | 2 | 177 sec. | 14 | 238 sec. | 390 sec. |
| B | 4 | 344 sec. | 23 | 724 sec. | 2436 sec. |
| C | 1 | 178 sec. | 9 | 293 sec. | 229 sec. |

Figure 5. Summary of empirical results

The results illustrates several interesting features. On all domains COMPOSER exceeded the performance of PRODIGY/EBL. An important result is that the execution times associated with COMPOSER are monotonically decreasing. This suggests that conditional utility is accurately estimated. A surprising fact is that in the domains where COMPOSER acquired a strategy, only one or two control rules account for most, if not all, of the savings. This indicates that most of the rules acquired by PRODIGY/EBL are, at best, superfluous.

### 9.5 Limitations and Extensions

The COMPOSER technique corrects limitations of previous utility estimation techniques, but this guarantee may come at a considerable cost. Rules of high utility may require many examples before reaching an acceptable level of confidence. More importantly, the estimation of utility requires that rule preconditions be evaluated many times within a training problem, and each precondition match requires an expense of resources. If the number or size of rules inactive rules grows too large, training problems may take prohibitively long to solve. Resolution of this issue requires making conservative choices for which rules to consider, quickly discarding bad choices, and relaxing some guarantees. A number of these approaches are discussed below.

One interesting extension involves exploring the use of an approximate domain theory for judging utility to drive the EBL component. Currently, the EBL component does not use a theory of utility at all. Instead it exploits only a theory of rule correctness. Since the hybrid method relaxes the need for a complete and correct domain theory, an approximate theory may be possible.

Alternatively, or in addition, a correct but incomplete utility domain theory might be entertained. It may be possible, for example, to fashion a theory that recognize sub–cases in which two rules, R and S, have identical effects but R has more general preconditions. From this information we can conclude that that R subsumes the savings of S and that R and S should never appear together in the same strategy. If we can further state that the match cost of R is less than the match cost of S then R *dominates* S. Any strategy containing R is guaranteed to have higher utility than a strategy containing S.

The empirical component estimates conditional utility for a rule across all problems in the distribution. In practice, rule utility varies systematically across different problems. For example the savings of the rule in Figure 2 is a function of the number of blocks in the initial state. If problems can be classified based on features which effect rule utility, tighter utility bounds may be achieved. This extension would allow a flexible control strategy which utilize different rules for different problem classes.

Another important consideration is that the greedy reductionist algorithm is a hill–climbing technique and thus, while guaranteeing improvement, may terminate with a non–optimal strategy. It is also conceivable that no strategy will be found when beneficial strategies do, in fact, exist. It can happen that all rules have individual negative conditional utilities but combine synergistically to produce a good strategy, confounding the greedy approach. The method for combining rules can be viewed as a strong bias on the space of possible control strategies. The appropriateness of this bias needs further investigation.

Finally it is useful to consider when simplifications of this technique are sufficient to produce positive strategies. This is an important consideration because the guarantees provided by COMPOSER may come a considerable cost in increased learning time. For example, the PRODIGY/EBL system [Minton88] does not address the composability problem and yet has demonstrated success on a number of domains. Equation 2 indicates that in the case where control rules are nearly independent, conditional utility can be approximated by a measure which is independent of the current strategy.

## 10 RELATED WORK

COMPOSER is one approach to the utility generalization and composability problems. In this section we describe other work which addresses these issues. We have organized the presentation into four basic trends.

### 9.1 Elaboration

The problems with PRODIGY/EBL arise from its simplified cost model. A natural approach is to elaborate the model. COMPOSER is one such elaboration. Leckie and Zukerman describe another approach [Leckie91]. They present an inductive system which reasons about some control rule interactions. They define a global cost model which is a function of a finite set of possible control rules. The problem of which rules to keep is reduced to the problem of minimizing this function. The model makes several assumptions. For example, all rules are considered to have the same constant match cost. Even so, the model must entertain all $2^i$ alternative combinations of $i$ control rules.

### 9.2 Simplification

Complete models of utility appear intractable. An alternative is to introduce deliberate simplifications into a complete model. Any simplification will partition domains into two sets: those for which the simplification is appropriate, and those for which it is not. For example, Equation 1b is a simplified model which operates correctly in the domains reported in [Minton88] and incorrectly in the domains reported here. Given a simplification, we must formalize the qualities of a domain which affect its classification. This allows potential users to decide if the tool fits their problem. This paper can be viewed as a preliminary attempt to formalize the properties of domains with respect to Equation 1b. Formal treatments of the macro–operator approach appear in [Greiner89, Korf87, Tadepalli91].

The work of Oren Etzioni is similar in spirit [Etzioni90a]. Etzioni looked extensively into to control rules which were rejected by PRODIGY/EBL's utility analysis. He then identified a property of these rules which seemed to hold across multiple domains. He captured this in a syntactic criteria — the nonrecursive hypothesis. This transfers an aspect of utility analysis into the rule generator. We have demonstrated in this paper that the nonrecursive criteria is a simplification. The next step is then to identify the domain constraints which influence the accuracy of this method.

Admittedly there are many possible simplifications, many of which create useless distinctions. An alternative approach is to identify a set of "natural" domains and design simplifications appropriate to them. Unfortunately there is little consensus on the extent of this set. Leaving these problems aside, regularities in these domains can suggest simplifications to a complete mode of utility. We are not aware of any research in this area.

### 9.3 Specificity

One of the primary reasons for the utility generalization and composability problems is that a control strategy is required to improve performance over an entire set of problems. Thus, knowledge acquired during one problem can affect performance on every other problem solved. A beneficial rule may well decrease performance on some problems as long as it makes up for this in other enhancements. The resulting tradeoffs can be quite complex.

This need for a global performance improvement exacerbates the utility generalization problem. A control rule may have to make recommendations about vastly different problems. Thus its savings can be expected to have wide variance as well. This global property also insures that many irrelevant rules will be entertained while solving a particular problem, increasing the opportunity for interactions.

A natural alternative is to be conservative about rule use; do not generalizing a control rule to apply at every legal opportunity. For example, a system could only consider a control rule if it was learned on a problem which is "similar" to the current problem being solved. This approach is taken by Fisher and Yoo [Fisher91] where problem classification rules to control search. There is also psychological evidence that humans perform limited generalization in the context of problem solving [Medin89]. They explain this effect in terms of a case–based reasoning model.

### 9.4 Theories of Utility

As we mentioned, speed–up learning techniques have focused on generalization techniques which preserve the accuracy of control decisions. Utility considerations have been patched on as a filter to traditional generalization techniques. An alternative is to identify utility preserving generalizations. For example, PRODIGY/EBL constructs control rules based on a theory of rule accuracy. An

alternative would be to generate rules from a theory of rule utility. Hirsh suggested such an approach in [Hirsh87].

## 7 CONCLUSION

The PRODIGY/EBL system [Minton88] was one of the first works to directly attack the problem of strategy utility. The problem of finding effective strategies was reduced to the problem of finding effective rules. However, this paper illustrates limitations of the approach. There are two basic difficulties. The first arises from the fact that the utility of a control rule cannot be accurately determined from a single instance of the rule. This is a manifestation of a more basic problem which we term the utility generalization problem. The difficulty is that the generalization techniques employed by speed–up learning systems are accuracy preserving but not utility preserving.

The second difficulty is that control rules interact such that the utility of one control rule is a function of the other control rules in the system. This composability problem means that systems cannot reduce the problem of learning effective strategies to the problem of identifying rule utility in isolation.

We documented the seriousness of these problems with an example domain theory. With this theory, PRODIGY/EBL generated control strategies which were up to seventeen times slower than the original planner. While this raises serious questions about the effectiveness of PRODIGY/EBL, we also claim the the utility generalization and composability problems are basic issues which are not adequately addressed by current speed–up learning techniques.

Finally, we introduced an alternative technique called COMPOSER. This system is based on a sound statistical model which is validated with a series of experiments. COMPOSER successfully avoids the utility generalization and composability problems. However, the technique may result in substantially higher learning cost. Our future research seeks to reduce this learning cost by identifying acceptable simplifications to the complete model.

**Acknowledgements**

We would like to thank Michael Barbehenn, Scott Bennett, David Francis, Pat Langley, Steve Minton, and Brian Ross for their useful suggestions. This research is supported by the National Science Foundation, grant NSF IRI 87–19766.

### References

[Allen90] J. Allen and P. Langley, "Integrating Memory and Search in Planning," *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, San Diego, CA, November 1990, pp. 301–312.

[Carlson90] B. Carlson, J. Weinberb and D. Fisher, "Search Control, Utility, and Concept Induction," *Proceedings of the Seventh International Conference on Machine Learning*, Austin, TX, June 1990, pp. 85–91.

[DeJong86] G. F. DeJong and R. J. Mooney, "Explanation–Based Learning: An Alternative View," *Machine Learning 1*, 2 (April 1986), pp. 145–176. (Also appears as Technical Report UILU–ENG–86–2208, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana–Champaign.)

[Eskey90] M. Eskey and M. Zweben, "Learning Search COntrol for Constraint–Based Scheduling," *Proceedings of the National Conference on Artificial Intelligence*, Boston, MA, August 1990, pp. 908–915.

[Etzioni90a] O. Etzioni, "Why Prodigy/EBL Works," *Proceedings of the National Conference on Artificial Intelligence*, Boston, MA, August 1990, pp. 916–922.

[Etzioni90b] O. Etzioni, "A Structural Theory of Search Control," Ph.D. Thesis, Department of Computer Science, Carnegie–Mellon University, Pittsburgh, PA, In preparation, 1990.

[Fisher91] D. Fisher and J. Yoo, "Combining Evidence of Deep and Surface Similarity," *Proceedings of the Eighth International Workshop on Machine Learning*, Evanston, IL, June 1991, pp. 46–50.

[Gratch90] J. Gratch and G. DeJong, "Utility Generalization and Composability Problems in Explanation–Based Learning," Technical Report UIUUCDCS–R–91–1681, Department of Computer Science, University of Illinois at Urbana–Champaign, 1990.

[Gratch91] J. Gratch and G. DeJong, "A Hybrid Approach to Guaranteed Effective Control Strategies," *Proceedings of the Eighth International Workshop on Machine Learning*, Evanston, IL, June 1991, pp. 509–513.

[Greiner89] R. Greiner and J. Likuski, "Incorporating Redundant Rules: A Preliminary Formal Analysis of EBL," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989, pp. 744–749.

[Hirsh87] H. Hirsh, "Explanation–Based Generalization in a Logic–Programming Environment," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987, pp. 221–227.

[Keller87] R. M. Keller, "Concept Learning in Context," *Proceedings of the Fourth International Workshop on Machine Learning*, University of California, Irvine, June 1987, pp. 91–102.

[Kish65] L. Kish, *Survey Sampling*, John Wiley & Sons, New York, NY, 1965.

[Korf87] R. E. Korf, "Planning as Search: A Quantitative Approach," *Artificial Intelligence 33*, (1987), pp. 65–88.

[Leckie91] C. Leckie and I. Zukerman, "Learning Search Control Rules for Planning: An Inductive Approach," *Proceedings of the Eighth International Workshop on Machine Learning*, Evanston, IL, June 1991, pp. 422–426.

[Letovsky90] S. Letovsky, "Operationality Criteria for Recursive Predicates," *Proceedings of the National Conference on Artificial Intelligence*, Boston, MA, August 1990, pp. 936–941.

[Markovitch89] S. Markovitch, "Information Filtering: Selection Mechanisms in Learning Systems," Ph.D. Thesis, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor, MI, 1989.

[Medin89] D. L. Medin and B. H. Ross, "The Specific Character of Abstract Thought: Categorization, Problem Solving, and Induction," in *Advances in the Psychology of Human Intelligence*, R. Sternberg (ed.), Lawrence Erlbaum and Associates, Hillsdale, NJ, 1989.

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

[Minton88] S. N. Minton, "Learning Effective Search Control Knowledge: An Explanation–Based Approach," Ph.D. Thesis, Department of Computer Science, Carnegie–Mellon University, Pittsburgh, PA, March 1988. (Also appears as CMU–CS–88–133)

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

[Mooney89] R. J. Mooney, "The Effect of Rule Use on the Utility of Explanation–based Learning," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989, pp. 725–730.

[Mostow89] J. Mostow and A. Prieditis, "Discovering Admissible Heuristics by Abstracting and Optimizing: A Transformational Approach," *The Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, 1989, pp. 701–707.

[Natarajan89] B. K. Natarajan, "On Learning from Exercises," *Proceedings of the Second Annual Workshop on Computational Learning Theory*, Santa Cruz, CA, JULY 1989, pp. 72–87.

[Segre87] A. M. Segre, "On the Operationality/Generality Trade–off in Explanation–Based Learning," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987, pp. 242–248.

[Segre91] A. M. Segre, C. Elkan and A. Russell, "A Critical Look at Experimental Evaluations of EBL," *To appear in: Machine Learning 6*, 2 (1991),.

[Simon75] H. A. Simon and J. B. Kadane, "Optimal problem–solving search: all–or–none solutions," *Artificial Intelligence 6*, (1975), pp. 235–247.

[Subramanian90] D. Subramanian and R. Feldman, "The Utility of EBL in Recursive Domain Theories," *Proceedings of the National Conference on Artificial Intelligence*, Boston, MA, August 1990, pp. 942–949.

[Tadepalli91] P. Tadepalli, "Learning with Inscrutable Theories," *Proceedings of the Eighth International Workshop on Machine Learning*, Evanston, IL, June 1991, pp. 544–548.

[Tambe89] M. Tambe and P. Rosenbloom, "Eliminating Expensive Chunks by Restricting Expressiveness," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989, pp. 731–737.

[Yamada89] S. Yamada and S. Tsuji, "Selective Learning of Macro–operators with Perfect Causality," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989, pp. 603–608.

[Yamada91] S. Yamada, "Computing the utility of EBL in a logic programming environment.," *Proceedings of the Workshop on Emperical Machine Learning*, Keio University, JAPAN, 1991, pp. 107–124.

[Yoo91] J. Yoo and D. Fisher, "Identifying Cost Effective Boundaries of Operationality," *Proceedings of the Eighth International Workshop on Machine Learning*, Evanston, IL, June 1991, pp. 569–573.

| BIBLIOGRAPHIC DATA SHEET | 1. Report No. UIUCDCS-R-91-1681 | 2. | 3. Recipient's Accession No. |
|---|---|---|---|

4. Title and Subtitle

UTILITY GENERALIZATION AND COMPOSABILITY PROBLEMS IN EXPLANATION-BASED LEARNING

5. Report Date: August 1991

6.


7. Author(s): Jonathan M. Gratch and Gerald F. DeJong

8. Performing Organization Rept. No. R-91-1681

9. Performing Organization Name and Address

Department of Computer Science
1304 W. Springfield Avenue
Urbana, IL 61801


10. Project/Task/Work Unit No.

11. Contract/Grant No.

NSF IRI 87 19766

12. Sponsoring Organization Name and Address

NSF

13. Type of Report & Period Covered

Technical

14.

15. Supplementary Notes

16. Abstracts


The PRODIGY/EBL system [Minton88] was one of the first works to directly attack the problem of strategy utility. The problem of finding effective strategies was reduced to the problem of finding effective rules. However, this paper illustrates limitations of the approach. There are two basic difficulties. The first arises from the fact that the utility of a control rule cannot be accurately determined from a single instance of the rule. This is a manifestation of a more basic problem which we term the utility generalization problem. The difficulty is that the generalization techniques employed by speed-up learning systems are accuracy preserving but not utility preserving. The second difficulty is that control rules interact such that the utility of one control rule is a function of the other control rules in the system. This composability problem means that systems cannot reduce the problem of learning effective strategies to the problem of identifying rule utility in isolation. We document the seriousness of these problems with an example domain theory. With this theory, PRODIGY/EBL generates control strategies which are up to seventeen times slower than the original planner. While this raises serious questions about the effectiveness of PRODIGY/EBL, we also claim the the utility generalization and composability problems are basic issues which are not adequately addressed by current speed-up learning techniques. We introduce an alternative technique called COMPOSER. This system is based on a sound statistical model which is validated with a series of experiments. COMPOSER successfully avoids the utility generalization and composability problems.


17. Key Words and Document Analysis. 17a. Descriptors

Explanation-based Learning
Utility Problem
Control Rules

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

| 18. Availability Statement | 19. Security Class (This Report) | 21. No. of Pages |
|---|---|---|
| Unlimited | UNCLASSIFIED | 22 |
| | 20. Security Class (This Page) UNCLASSIFIED | 22. Price |

FORM NTIS-35 (10-70)

USCOMM-DC 40329-P71