In machine learning there is considerable interest in techniques which improve planning ability. Initial investigations have identified a wide variety of techniques to address this issue. Progress has been hampered by the utility problem, a basic tradeoff between the benefit of learned knowledge and the cost to locate and apply relevant knowledge. In this paper we describe the COMPOSER system. COMPOSER embodies a probabilistic solution to the utility problem. It is implemented in the PRODIGY architecture. We compare COMPOSER to four other approaches which appear in the literature: (1) PRODIGY/EBL's Utility Analysis; (2) STATIC's Nonrecursive Hypothesis; (3) DYNAMIC: A Composite System; and (4) PALO's Chernoff Bounds. (Contains 24 references.) (Author/ALF)
COMPOSER: A Probabilistic Solution to the Utility Problem in Speed-up Learning

by

Jonathan Gratch
Gerald DeJong

January 1992
COMPOSER: A Probabilistic Solution to the Utility Problem in Speed-up Learning

Jonathan Gratch and Gerald 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

Keywords
Function: learning
Knowledge: experience
Foundation: mathematical

Abstract
In machine learning there is considerable interest in techniques which improve planning ability. Initial investigations have identified a wide variety of techniques to address this issue. Progress has been hampered by the utility problem, a basic tradeoff between the benefit of learned knowledge and the cost to locate and apply relevant knowledge. In this paper we describe the COMPOSER system. COMPOSER embodies a probabilistic solution to the utility problem. It is implemented in the PRODIGY architecture. We compare COMPOSER to four other approaches which appear in the literature.

This research is supported by the National Science Foundation, grant NSF IRI 87-19766
1 INTRODUCTION
Increasingly, machine learning is entertained as a mechanism for improving the efficiency of planning systems. Investigation in this area has identified a wide array of techniques including macro-operators [DeJong86, Fikes72, Mitchell86], chunks [Laird86], and control rules [Minton88, Mitchell83]. With these techniques comes a growing battery of successful demonstrations in domains ranging from 8-puzzle to Space Shuttle payload processing. Unfortunately, in what is now called the utility problem, learned knowledge can hurt performance [Minton88]. This is underscored by a growing body of demonstrations where learning degrades planning performance [Etzioni90b, Gratch91a, Minton85, Subramanian90].

In an earlier paper we elaborated the limitations in a particular learning approach — PRODIGY/EBL [Gratch91b]. That paper also sketched the COMPOSER system which is one solution to these limitations. COMPOSER is intended as a general solution to the utility problem which provides probabilistic guarantees of improvement via learning. In this paper we detail our approach and report on an extensive series of empirical evaluations. These tests compare COMPOSER’s learning criterion against the approaches adopted by PRODIGY/EBL, STATIC [Etzioni90b], DYNAMIC [Etzioni90a], and PALO [Greiner92]. These results substantiate our earlier analyses. They also cast doubt on the efficacy of nonrecursive control knowledge. This is significant since the issue of nonrecursive control knowledge has received considerable attention in recent literature [Etzioni90b, Letovsky90, Subramanian90].

2 LEARNING AS SEARCH
Learning can be viewed as a transformational process in which the learning system applies a series of transformations to the original problem solver (see [Gratch90a, Greiner92]). The intended result is more effective planning behavior. Typically a planner is transformed with control knowledge. Different forms of control knowledge include macro-operators [Brayerman88, Laird86, Markovitch89], control rules [Drummond89, Etzioni90b, Minton88, Mitchell83], and static board evaluation functions [Utgoff91].

The transformations available to a learner define its vocabulary of transformations. These are essentially learning operators and collectively they define a transformation space. For instance, acquiring a macro-operator can be viewed as transforming the initial system (the original planner) into a new system (the planner operating with the macro-operator). A learning system must explore this space for a sequence of transformations which result in a better planner.

To evaluate different learning approaches we must clarify our intuitive notions of when one planner is more efficient than another. For this paper, we characterize planners through a numeric utility function which ranks the behavior of a planner over a fixed distribution of problems. In particular, we equate efficiency with minimizing planning time. Other measures are possible and our approach could apply to them as well. For any given problem,
utility increases as the time to solve the problem decreases. The utility of a planner is defined with respect to a particular problem distribution as the sum of problem utilities weighted by the probability of occurrence of each problem:

\[
U_{\text{LITY}}(\text{planner}) = -\sum_{\text{prob } \in \text{Distribution}} \text{Time}_{-}\text{Cost}(\text{planner}, \text{prob}) \times \text{Pr}(\text{prob})
\]

Note 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.

Utility is a preference function which ranks different planners. It is also useful to discuss the utility of individual transformations. The incremental utility of a transformation is defined as the change in utility that results from applying the transformation to a particular planner (e.g. adopting a control rule). This means the incremental utility of a transformation is conditional on the planner to which it is applied. We denote this as: \(\Delta U_{\text{LITY}}(\text{Transformation|Planner})\). Applying a transformation with positive incremental utility results in a more effective planner. A learning system need not explicitly compute utility values to identify preferred planners, but it must act (at least approximately) as if it does. In fact many learning systems do not explicitly evaluate utility.

3 COMPOSER

COMPOSER uses the previous definition of utility to evaluate and adopt control knowledge which, with high probability, improves planning performance. Its design was motivated by deficiencies in PRODIGY/EBL. Another paper illustrates how these deficiencies are shared by many other speed-up learning techniques [Gratch92]. In this paper we focus on the implementation of COMPOSER. In [Gratch91b] we note that PRODIGY/EBL adopts two heuristic simplifications to identify beneficial control rules. First, aspects of the problem distribution are learned from a single example. Secondly, control rules are treated as if they do not interact. These simplifications have the unfortunate consequence that PRODIGY/EBL can learn control strategies which yield planners which are up to an order of magnitude slower than the original planner. We replace this heuristic approach with a rigorous alternative.

3.1 Algorithm

The current implementation of COMPOSER learns control knowledge in the form of control rules. Other transformations could be adopted by suitably altering the statistics gathering procedure described in Section 3.2. COMPOSER is implemented within the PRODIGY 2.0 architecture. This system includes the PRODIGY planner which is a STRIPS-like system. It identifies plans through depth-first search. The learning component of PRODIGY/EBL analyzes solution traces and proposes control rules to correct any observed inefficiencies. These control rules are condition-action statements which inform the PRODIGY planner to
delete or prefer certain node, operator, or variable binding choices. COMPOSER primarily utilizes selection and rejection rules. This is discussed further in Section 5.

COMPOSER differs from PRODIGY/EBL in how statistics are gathered and how control rules are introduced into the PRODIGY planner. We implement a hill-climbing approach to the utility problem. The basic algorithm is sketched in Figure 1. We assume the user has provided a training set which is drawn according to the distribution of problems.

---

**Input:** TRAINING_EXAMPLES

**CONTROL_STRATEGY = Ø**

**CANDIDATE_SET = Ø**

While TRAINING_EXAMPLES

solve problem with PRODIGY+CONTROL_STRATEGY

learn new rules and add them to CANDIDATE_SET

gather statistics for all rules in CANDIDATE_SET

**POSITIVE_RULES = Ø**

For all rules $e$ CANDIDATE_SET

If $UTILITY$ (rule$|$PRODIGY+CONTROL_STRATEGY) significantly negative

remove rule from CANDIDATE_SET

If $UTILITY$ (rule$|$PRODIGY+CONTROL_STRATEGY) significantly positive

add rule to POSITIVE_RULES

If POSITIVE_RULES

append rule with highest utility to CONTROL_STRATEGY

remove this rule from CANDIDATE_SET

discard all statistics on rules in CANDIDATE_SET

**Output:** CONTROL_STRATEGY

---

Figure 1: The COMPOSER algorithm

Learning occurs with a single pass through the training examples. The algorithm incrementally adds control rules to a currently adopted control strategy. A rule is added only if it has demonstrated its benefit to a pre-specified confidence level. Once added, the rule changes how the planner behaves on subsequent training examples. New rules are proposed, and statistics gathered, with respect to the current control strategy. In this manner a control strategy is "grown" one rule at a time until the training set is exhausted.
3.2 Gathering Incremental Utility Statistics

Gathering incremental utility statistics is the one aspect of COMPOSER which ties it to a particular representation for control knowledge — namely control rules. Other transformations would require analogous data gathering procedures.

A control rule should only be adopted if it improves the efficiency of the problem solver on average. This average can be estimated by determining how the rule performs on individual problems and combining information from several problems. The next section discusses how to combine information. But first we will describe how COMPOSER extracts incremental utility values on individual problems.

How can we determine the incremental utility of a control rule on a particular problem? The obvious approach is to solve the problem twice — once using the current control strategy without the rule in question, and once using the strategy augmented with the candidate rule. The difference in problem solving cost between these two runs is the incremental utility of the control rule on that problem. This process must be repeated for every rule in the candidate set. Clearly this approach is too expensive in practice.

COMPOSER implements a more efficient approach for gathering incremental utility values. It can extract a utility value for each candidate rule simultaneously from a single solution trace. While PRODIGY/EBL also derives multiple estimates from a single example, its technique is rendered inaccurate by the interactions which occur among rules (see [Gratch91b]). COMPOSER solves the interaction problem by extracting estimates without allowing the candidate rules to change the search behavior of the planner. Control rules only effect search behavior if they are adopted into the control strategy.

In contrast to adopted rules, the actions of candidate rules are not acted upon. They are simply noted in the problem solving trace. After a problem is solved, COMPOSER analyzes the annotated trace, and identifies the search paths which would have been avoided by each 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. More details may be found in [Gratch90b].

It should be noted that this procedure is somewhat 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 benefit of candidate control rules. We are not aware of a reliable technique which avoids this additional cost.

3.3 Commitment Criterion

The incremental utility of a transformation across the problem distribution can be estimated by averaging utility values from several problems. COMPOSER uses average incremental
utility to decide a control rule's fate. The system must apply only transformations which have positive incremental utility. Additionally, as the data gathering process is expensive, the system must discard candidate transformations which have incremental negative utility. Both of the choices should be determined accurately but with as few examples as possible. In the field of statistics this is referred to as a \textit{sequential analysis problem}. Observations are gathered until some stopping criterion is satisfied. As this criterion will commit COMPOSER to adopting or discarding a transformation we refer to this as a \textit{commitment criterion}. In this case we are estimating the incremental utility of transformations to some specified confidence. We require the user to provide an error parameter, $\delta$, which specifies the acceptable probability of applying or discarding a transformation incorrectly.¹

Formally, COMPOSER must choose among two hypotheses for each candidate:

\begin{align*}
H_0: \ \Delta \text{UTILITY}(\text{rule}|\text{planner}+\text{control}\_\text{strategy}) &< 0, \\
H_1: \ \Delta \text{UTILITY}(\text{rule}|\text{planner}+\text{control}\_\text{strategy}) &> 0
\end{align*}

The average incremental utility is only an estimate of the true incremental utility. This estimate will differ from the true value, so the system must bound the discrepancy. In particular, if the rule is negative, the system must bound the probability that it will appear positive, and vice versa. This is equivalent to bounding the probability that the difference between the true utility and the estimate is larger than the magnitude of the true utility:

$$\Pr\left(|\text{ESTIMATE} - \text{UTILITY}| > |\text{UTILITY}| \right) = \delta$$

Nádas [Nádas69] describes a distribution-free commitment criterion which applies. After taking $M$ examples the probability of error is (approximately) $\delta$, where $M$ is defined as:

$$M = \min_{m \geq 1} \left\{ n: \left( \frac{V_{r,n} / \bar{X}_{r,n}}{n(1/a)^2} \right)^2 < n(1/a)^2 \right\}$$

where $\bar{X}_{r,n}$ is the average utility of the rule $r$ over $n$ problems, $X_{r,i}$ is the utility of $r$ on the $i$th problem, and $V_{r,n} = \sum (X_{r,i} - \bar{X}_{r,n})^2$ is an indicator of the variance in the sample. The parameter $a$ satisfies the constraint that $\Phi(a) = \delta/2$, where $\Phi$ is the cumulative distribution function of the standard normal distribution. In simpler words, COMPOSER take examples until the inequality, $(V_{r,n} / \bar{X}_{r,n})^2 < n(1/a)^2$, is satisfied.

The technique approximates the user specified level. If the user specifies an error level of $\delta$, the true error level will be close to but not necessarily equal to $\delta$ — it may be slightly more or less. The discrepancy is a function of the underlying distribution. Woodroofe provides a further analysis which indicates the approximation is very close in practice [Woodroofe82].

¹ Alternatively we could require that $\delta$ represent the cumulative error across all applied transformations. This requires determining a $\delta_i$ at each step such that the sum of all $\delta_i$'s equals $\delta$. 

5
In summary, the commitment criterion permits COMPOSER to identify when transformations are beneficial with some pre-specified probability. After each problem solving attempt, COMPOSER updates the statistics and evaluates the commitment criterion for each control rule in the candidate set. If no control rule has attained the confidence requirement, another problem is solved. If the commitment criterion identifies control rules with positive incremental utility (there may be more than one), COMPOSER adds the control rule with highest positive incremental utility to the current strategy, and removes it from the candidate set. Statistics for the remaining candidates are discarded as they are conditional on the previous control strategy, and meaningless in the context of the new strategy. If the commitment criterion identifies candidate rules with negative incremental utility, they are eliminated from the candidate set. Eliminating a candidate does not affect the current strategy, so the statistics associated with the remaining candidate control rules are not discarded. This cycle is repeated until the training set is exhausted. Each time a transformation is adopted the efficiency of the PRODIGY planner is increased, giving COMPOSER an anytime behavior [Dean88].

4 EVALUATION

We evaluated COMPOSER's commitment criterion against several other commitment criteria. Before discussing the experiments we will review these other criteria.

4.1 PRODIGY/EBL's Utility Analysis

PRODIGY/EBL adopts transformations with a heuristic utility analysis. As control rules are proposed they are added to the current control strategy. The savings afforded by each rule is estimated from a single example and this value is credited to the rule each time it applies. Match cost is measured directly. If the cumulative cost exceeds the cumulative savings, the rule is removed from the current control strategy. The issue of interactions among transformations is not addressed.

4.2 STATIC's Nonrecursive Hypothesis

STATIC's commitment criterion is based on Etzioni's structural theory of utility. The criterion is grounded in the nonrecursive hypothesis which states that transformations will have positive incremental utility, regardless of problem distribution, if they are generated from nonrecursive explanations of planning behavior. An explanation is defined as recursive if a predicate in a subgoal is derived using another instantiation of the same predicate. The issue of interactions between transformations is not addressed. STATIC applies this criterion to control rules but the idea has been applied to macro-operators as well [Letovsky90, Subramanian90].

STATIC has exceeded PRODIGY/EBL's performance on a number of domains, including one domain for which PRODIGY/EBL degrades planning performance. The nonrecursive hypothesis is cited as the principle reason for this success [Etzioni90b]. This claim is difficult
to evaluate as these systems can generate very different control rules. We clarify this issue by testing the nonrecursive hypothesis on PRODIGY/EBL's learning component.

We constructed the STATIC-RI system as a re-implementation of STATIC's nonrecursive hypothesis within the PRODIGY/EBL framework. STATIC-RI replace the commitment criterion of PRODIGY/EBL with the nonrecursive hypothesis. Instead of utility analysis, as rules are proposed, STATIC-RI adopts each rule which is based on a nonrecursive explanation and discards each rule which is based on a recursive explanation. In all other respects STATIC-RI is identical to PRODIGY/EBL.

4.3 DYNAMIC: A Composite System

Etzioni has suggested that the strengths of STATIC and PRODIGY/EBL could be combined into a single system [Etzioni90a]. The proposed DYNAMIC system incorporates a two layered utility criterion. The nonrecursive hypothesis acts as in initial filter, but the remaining nonrecursive control rules are subject to utility analysis and may be later discarded.

We implemented the DYNAMIC-RI system to test this learning criterion. As control rules are proposed by PRODIGY/EBL's learning module, they are first filtered on the basis of the nonrecursive hypothesis. The remain rules undergo utility analysis as in PRODIGY/EBL.

4.4 PALO's Chernoff Bounds

Greiner and Cohen have recently proposed an approach which is similar to COMPOSER. The Probably Approximately Locally Optimal (PALO) approach also adopts a hill-climbing technique and evaluates transformations by a statistical method. The primary difference between our two methods is the commitment criteria. COMPOSER uses the sequential analysis technique of Nádas, while PALO uses Chernoff bounds.

Both techniques require similar assumptions — namely that the problem distribution is fixed, training examples are randomly drawn from this distribution, and that the distribution of utility values over this problem distribution possesses a finite variance. The difference is that Chernoff bounds provide somewhat stronger guarantees at the cost of more examples. The Nádas technique implements approximate significance levels — the true error level will be close to but not necessarily equal to δ. PALO's technique provides worst case bounds. This means that if the user specifies an error level of δ, the true error level will never exceed δ, and may in fact be much lower.

The PALO-RI system evaluates this approach. PALO-RI is a modification of COMPOSER where the Nádas technique is replaced with PALO's commitment criterion. Examples are gathered until a control rule satisfies the following inequality:

\[
\sum_{i=1}^{n} X_{r,i} > \Lambda \sqrt{\frac{n}{2} \ln \frac{1}{\delta}}
\]
where $X_{ri}$ is the utility of rule $r$ on problem $i$, and $\Lambda$ is a parameter which is related to the maximal cost of solving problems. One disadvantage of this technique is it is difficult to assess the optimal value for $\Lambda$ and the chosen value can significantly impact the number of examples required by the method. Our resolution of this issue is discussed in the experiments.

4.4 Experiments

COMPOSER was tested on the STRIPS domain from [Minton88], the AB–WORLD domain from [Etzioni90a] for which PRODIGY/EBL produced harmful strategies, and the BIN–WORLD domain from [Gratch91a] which yielded detrimental results for both STATIC and PRODIGY/EBL's learning criteria. The results are summarized in Figure 2. In each domain the systems are trained on 100 training examples drawn randomly from a fixed distribution. A learning curve is generated by saving the current control strategy after every twenty training examples. The graphs illustrate learning curves where the independent measure is the number of training examples and the dependent measure is execution time for 100 test problems drawn from the same distribution. This process is repeated five times, using different but identically distributed training and test sets. Values in Figure 2 represent the average of these five trials. The header "# Rules" indicates the average number of rules learned by the system; "Train Time" is the number of seconds required to process the 100 training examples; "Test Time" refers to the number of seconds required to generate solutions for the 100 test problems.

As we stated earlier, COMPOSER does not implement a general approach to evaluating preference rules. In particular, it cannot properly evaluate the incremental utility of preference rules in the AB–WORLD and STRIPS domains. To ensure that differences reflect the commitment criteria and not the vocabulary of transformations, we disabled the learning of preference rules for every system in the STRIPS and AB–WORLD domains. We evaluated the ramifications of this change by comparing PRODIGY/EBL with and without preference rules and found that, in both domains, more efficient strategies resulted when preference rules were disabled. This is consistent with statements made by Minton concerning preference rules [Minton88 p. 129]. COMPOSER and PALOR–RI require a parameter which represents the confidence level for adding a transformation. This was set at 95%. For PALOR–RI's $\Lambda$ parameter, we tried to assign a value which is close to the maximal problem solving cost without going under. It was set as follows: AB–WORLD – 75 seconds, STRIPS – 150 seconds, BIN–WORLD – 150 seconds.

2. More precisely, this is the maximal cost of the problem solver with the current control strategy plus the maximum problem solving cost of the problem solver using the control rule which we are analyzing.

3. PRODIGY/EBL's utility analysis requires an additional settling phase after training. Each control strategy produced by PRODIGY/EBL and DYNAMIC–RI received a settling phase of 20 problems where learning was disabled by utility analysis continued to filter rules.
It was quickly apparent that PALO-RI would not adopt any transformations within the 100 training examples. We tried to give the system enough examples to reach quiescence but this proved too expensive. The problem is twofold — first, too many training examples were required; secondly, and as a consequence of the first problem, the candidate set grew large since harmful rules were not discarded as quickly as in COMPOSER. This increased the cost to solve each training example. To collect statistics on PALO-RI we only performed one instead of five learning trials. Furthermore, we terminated PALO-RI after the first transformation was adopted or 10,000 examples, whichever came first.

Figure 2: Summary of empirical results
4.5 Discussion

The results illustrate several interesting features. COMPOSER exceeded the performance of all other approaches in every domain. In AB–WORLD and STRIPS, COMPOSER identified beneficial control strategies. In BIN–WORLD the system did not adopt any transformations. It does not appear that any control rule improves performance in this domain. It should be stressed that all systems utilized the same learning module. Therefore the results represent differences in commitment strategies rather than in the vocabulary of transformations.

As expected, COMPOSER and PALO–RI had the highest learning times as they incur the precondition cost of candidate control rules without gaining the benefit of their recommendations. The one exception was BIN–WORLD where COMPOSER quickly discarded a very expensive control rule which PRODIGY/EBL, STATIC–RI, and DYNAMIC–RI retained. An encouraging result is that COMPOSER’s learning times were not substantially higher than the non–statistical systems. PALO–RI’s learning times were significantly higher.

The results cast doubts on the nonrecursive hypothesis. STATIC–RI yielded the worst performance on all domains. Even in conjunction with utility analysis the results are mixed—benefit on the AB–WORLD, slightly worse than utility analysis alone in STRIPS, and worse than no–learning in BIN–WORLD. A post–hoc analysis of control strategies did indicate that the best rules were nonrecursive, but many nonrecursive rules were also detrimental. The slow–down on BIN–WORLD primarily results from one nonrecursive control rule. Thus it appears that nonrecursiveness may be an important property but is insufficient to ensure performance improvements. These results are interesting since Etzioni reports that STATIC outperforms PRODIGY/EBL and No Learning in AB–WORLD. The nonrecursive hypothesis cannot account for this difference. We attribute the difference to the fact that STATIC and STATIC–RI entertain different sets of control rules. STATIC–RI was constrained to use the vocabulary which was available to PRODIGY/EBL while STATIC has its own rule generator.

Finally, although PALO–RI did not improve performance within the 100 training examples, we believe that if it were given sufficient examples it would out perform all other systems. With extended examples it did exceed COMPOSER’s performance in AB–WORLD. This is because the PALO approach commits to transformations with highest incremental utility while COMPOSER balances incremental utility against variance. Unfortunately the cost of PALO’s performance improvement is very high, both in terms of examples and learning time. Thus, while COMPOSER may identify somewhat less beneficial strategies, it achieves much faster convergence.

5 LIMITATIONS AND FUTURE RESEARCH

Our investigations have exposed two important issues for future research.
5.1 Preference Rules

There are difficulties in extending COMPOSER’s utility gathering approach to preference rules. It is easy to record the match cost for these rules. The problem stems from determining how much a rule would save if it were added to the control strategy. This is straightforward in the case of rules which delete alternatives. The search space explored by the planner using such a rule will always be a subset of the search space explored without the rule. This is not necessarily the case with preference rules. A candidate preference rule might suggest a search path which was not explored when the training example was solved (this can arise if there are multiple solutions to the training example). To determine the potential savings of the preference rule under these circumstances, the system must re-invoke the planner and explore this alternative path. This need may arise many times in one problem and other candidate preference rules might require even different paths to be expanded.

This discussion points to a general issue that some transformation vocabularies may be easier to implement within the COMPOSER framework than others. Perhaps the issue can be resolved by identifying alternative means to gather utility values. This problem disappears if we are willing to solve training problems twice — once with and once without the transformation — but this is unlikely to be feasible in practice.

5.2 Commitment Criteria

Both Greiner and Cohen’s approach and our own provide probabilistic guarantees of improvement through learning. The commitment criteria used by these systems exhibit different behaviors. Chernoff bounds produce better control strategies but at a higher learning cost. Neither technique directly accesses the tradeoff between the improvement due to learning and the cost to achieve that improvement. Currently we are investigating ways to apply decision theoretic methods to resolve this tradeoff in a principled way.

6 CONCLUSIONS

Learning shows great promise to extend the generality and effectiveness of planning techniques. Unfortunately, many learning approaches are based on poorly understood heuristics. In many circumstances a technique designed to improve planning performance can have the opposite effect.

In this paper we discussed one general approach to the utility problem which gives probabilistic guarantees of improvement through learning. Our implementation is restricted to control rules but could be extended to other representations of control knowledge. We contrasted COMPOSER with four other learning techniques — three which do not provide guarantees, and one which does. The utility analysis method of PRODIGY/EBL, the nonrecursive hypothesis of STATIC, and even a combination of both can produce substantial performance degradations. Greiner and Cohen’s PALO approach should yield somewhat better performance improvements than COMPOSER but at a substantially higher learning cost.
References


<table>
<thead>
<tr>
<th>Reference</th>
<th>Author(s)</th>
<th>Title</th>
<th>Venue</th>
<th>Pages</th>
</tr>
</thead>
</table>
In machine learning there is considerable interest in techniques which improve planning ability. Initial investigations have identified a wide variety of techniques to address this issue. Progress has been hampered by the utility problem, a basic tradeoff between the benefit of learned knowledge and the cost to locate and apply relevant knowledge. In this paper we describe the COMPOSER system. COMPOSER embodies a probabilistic solution to the utility problem. It is implemented in the PRODIGY architecture. We compare COMPOSER to four other approaches which appear in the literature.