Technical Report UIUC-BI-AI-94-04, 1994 (submitted to the Journal of Artificial Intelligence)

A Decision-theoretic Approach to Adaptive Problem Solving

Jonathan Gratch and Gerald DeJong

Beckman Institute
University of Illinois
405 N. Mathews, Urbana, IL 61801

May 1994

Abstract

This article argues that it is desirable, and possible, to construct general problem solving techniques that automatically adapt to the characteristics of a specific application. Adaptive problem solving is a means of reconciling two seemingly contradictory needs. On the one hand, general purpose techniques can ease much of the burden of developing a application and satisfy the oft argued need for declarative and modular knowledge representation. On the other hand, general purpose approaches are ill-suited to the specialized demands of individual applications. General approaches have proven successful, only after a tedious cycle of manual experimentation and modification. Adaptive techniques promise to reduce the burden of this modification process and, thereby, take a long step toward reconciling the conflicting needs of generality and efficiency. A principal impediment to adaptive techniques is the utility problem – the realization that learning strategies can degrade performance under difficult to predict circumstances. We develop a formal characterization of the utility problem and introduce COMPOSER, a statistically rigorous approach to this problem. COMPOSER has been successfully applied to learning heuristics for planning and scheduling systems. This article includes theoretical results and an extensive empirical evaluation of the approach on learning adaptations to a domain-independent planner. The approach significantly outperforms several other proposed solutions to the utility problem.

1 Introduction

There is a wide gulf between *general* approaches and *effective* approaches to problem solving. Practical success has come from custom techniques like expert systems, reactive planners [Miller92, Schoppers92], or other application specific techniques that require extensive human investment to complete. AI researchers have also developed domain-independent algorithms such as non-linear planning and constraint satisfaction algorithms. Unfortunately, when general approaches show success, it is usually only after extensive domain-specific adjustments. The resulting systems, while derived from a general technique, bear more resemblance to the custom approaches.

Adaptive problem solving is a potential means for circumventing this generality/performance tradeoff in repetitive problem solving situations. A general algorithm might learn the idiosyncrasies of a domain through its problem solving experiences and automatically transform itself into an effective problem solver. In fact, machine learning techniques have successfully demonstrated this capacity in limited contexts [Fikes71, Laird86, Minton88]. Nonetheless, adaptive problem solving is still far from realization in any general sense.

The principal impediment to adaptive problem solving is characterizing when an automatically hypothesized transformation actually results in improved problem solving performance. Steve Minton introduced to machine learning the term *utility problem* to refer to this difficulty in insuring performance improvements [Minton88]. Minton originally discussed the problem in the context of improving the average problem solving speed via search control heuristics called control rules. While there has been considerable progress on this issue [Etzioni90b, Holder92, Lewins93, Minton88], the proposed methods are often ad hoc, poorly understood, and can fail to improve performance, or worse, actually degrade problem solving performance under certain circumstances.

In this article we introduce a more general characterization of the utility problem, formalized in the terminology of decision theory. This characterization eliminates some deficiencies in the original definition and generalizes the problem. We then present a general statistical technique that rigorously addresses this more general version of the utility problem. We discuss the approach's formal properties and empirically demonstrate its effectiveness over several alternative approaches.

1.1 LEARNING TO EXPLOIT PROBLEM DISTRIBUTION

Learning algorithms work by identifying and exploiting the constraints implicit in given application. Researchers have identified two basic sources of information that learning algorithms can exploit: constraints on the problem distribution and constraints on the domain structure. Distributional constraints relate to the pattern of problems as they arise in the intended application. Structural constraints refer to syntactic restrictions on the representation of the domain. Arguably it is the distributional constraints that are essential in producing an effective problem solver, as structural constraints are rarely justified in principle but frequently sufficient with respect to the likely problem distribution.

Learning algorithms are especially apt at acquiring problem distribution information and exploiting such information can lead to signification performance improvements. Even worst-case intractable algorithms perform well under certain distributions. For example, Goldberg suggests that naturally occurring satisfiability problems are frequently solved in $O(n^2)$ time [Goldberg82]. Recent work has focused on characterizing these easy distributions [Cheeseman89, Mitchell92] or devising techniques that can exploit specific distribution information when it is available [Borgida89]. Research into self-organizing systems [Melhorn84 pp. 252–285] and dynamic optimization [Laird92] exploit the fact that learning the expected distribution of tasks can allow the construction of a problem solver with substantially better expected performance. For example, when linearly searching a list of records for one with a particular name, ordering records by their likelihood of being queried (distribution information) leads to substantial performance improvement.

Many structural constraints have been identified that a learning system could exploit for more effective inference. For example, general purpose problem solvers provide flexible representations that can encode a variety of domains of discourse, but these cannot be reasoned with tractably [Bylander92, Chapman87, Eorl92, Freuder82]. However, this full generality is often unnecessary given the structure of a particular application. In contrast, there are a variety of effective problem solving techniques that apply if a domain obeys certain structural restrictions. For example, Eorl, Nau and Subrahmanian summarize complexity results for several specializations of STRIPS operators, some of which support worst-case *polynomial* planning [Eorl92]. Constraint satisfaction problems can be solved efficiently if the constraint graph is expressible as a tree [Dechter87, Freuder82]. Korf shows a domain supports efficient problem solving if it exhibits the property of *serial decomposability* [Korf87]. Many integer programming programming problems can be efficiently solved in practice by exploiting their structural properties [Fisher81].

In a sense, structural constraints arise from constraints on the distribution of tasks, thus highlighting the importance of acquiring distributional information. Structural constraints are rarely justifiable in principle as for almost any realistic problem solving task one can identify circumstances that require fully general reasoning mechanisms. However, often a domain engineer can identify a restricted problem solving formalisms that work well for the tasks that arise in practice. In other cases the domain engineer might be forced to restrict the scope of the task distribution in order to ensure reasonable performance. For example, in control theory, fundamentally non-linear problems are solved with simple linear systems by ensuring that the process stays in almost-linear regions [Dean91 p. 40]. While outside the scope of this article, this later option raises an interesting issue that a truly general learning system might require the ability suggest restrictions on the scope of the problem solver.

1.2 OVERVIEW

The structure of this article is as follows. In the next section we provide a formal characterization of the utility problem as a problem of decision-theoretic inference. This introduces the notion of expected utility as a metric of problem solving performance and casts the utility problem as a search through a space of problem solving transformations for a problem solver with high expected utility. Section 3 describes the COMPOSER algorithm, a probabilistic solution to the utility problem. COMPOSER performs a probabilistic search through the transformation space and incorporates several techniques to ensure the efficiency of this process. Section 4 describes an extensive evaluation of the approach in the context of learning control rules for a domain-independent problem solver. The experiments indicate that COMPOSER compares favorably with existing approaches to the utility problem. Section 5 provides an average case analysis of the algorithm's complexity. COMPOSER is shown to be polynomial in the number of transformations considered and in the statistical confidence required. Finally we discuss some limitations and conclusions.

2 A Formal Characterization of the Utility Problem

Before it is possible to construct a rigorous learning algorithm, one must explicitly characterize what the learning system is attempting to achieve. In this section we introduce a formal characterization of a general class of learning tasks. This formalism serves two purposes. First it makes precise and explicit our intuitive and often unstated notions of what learning systems should do. Second, it provides a structure for formal analysis, enabling us to make definitive and precise statements.

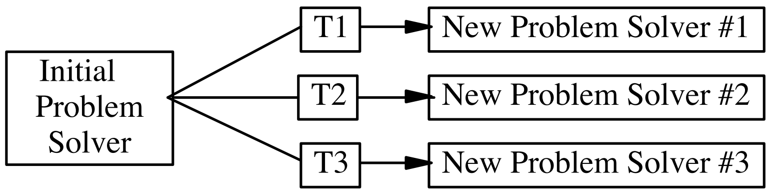


Figure 1: Learning serves to transform an initial problem solver into a transformed one. To accomplish this the learner must choose one of a set of possible transformations.

Abstractly, a learning algorithm operates on an initial problem solver, transforming it into a better problem solver, where the improvement is assessed relative to an evaluation criterion and a pattern of tasks associated with the intended application. This is illustrated in Figure 1 where a set of hypothesized transformations defines a set of potential problem solvers. The learning system must decide which among many hypothesized transformations to actually adopt. To make this statement concrete we use *decision theory* as a common framework for characterizing this decision problem. Decision theory provides a widely accepted framework for characterizing decision making under uncertainty (see [Berger80]). Jon Doyle has persuasively argued for the merits of decision-theory as a standard for evaluating artificial intelligence systems [Doyle90] and it has seen increasing acceptance, both in artificial intelligence at large [Horvitz89, Russell89, Schwuttke92, Wellman92] and machine learning in particular [Gratch92a, Greiner92a, Laird92, Subramanian92]. We will use decision theory as a common framework for characterizing the utility problem.

2.1 EXPECTED UTILITY

Different learning decisions result in different transformed problem solvers. Decision theory relies on the observation that preferences over these different outcomes can, under some natural assumptions, be characterized by a real-valued *utility function*. An outcome *A* is preferred to outcome *B* iff the utility of *A* is greater than the utility of B.

Under uncertainty a decision may not always results in the same outcome. In these circumstances a decision is characterized by a set of outcomes and a probability distribution across this set. Decision theory characterizes preferences over this set of outcomes by combining their individual utilities into an *expected utility* of the set. This is the utility of all possible outcomes weighted by their probability of occurrence. Under uncertainty, the correct decision maximizes expected utility.

For learning, the characteristics of the intended application determine what transformed problem solvers are preferred. With decision theory we represent these preferences with a utility function, allowing learning to be cast as the decision problem of choosing a transformation that increases expected utility. To accomplish mapping, however, we must assume the application obeys the following two restrictions.

Fixed distribution Assumption: The pattern of tasks in the problem solving environment must be characterizable as a random selection of tasks according to an unknown but fixed probability distribution over the domain of discourse. This restriction states that there is some probability of occurrence associated with each task, that this probability is independent of what tasks we have already seen, and that this probability does not change with time. This is a common simplification that applies to a great many applications. However, it does impose limitations that we discuss in Section 6.

Expected-utility Assumption: The evaluation criterion must be expressed by a *utility function*. This is a function from a problem solver and problem to a numeric value. The function must be chosen such that problem solver A is preferred over problem solver B if and only if the expected utility of A is greater than the expected utility of B. Decision theory posits the *expected-utility* hypothesis that there exists such a utility function for any consistent set of preferences (see [DeGroot70] Chapter 7). The utility function must also be computable by the learning system. For example, if the application requires an efficient problem solver, the utility function could be the CPU cost to solve a problem. This can be computed by actually solving the problem (assuming they are all solvable) and measuring the time. The expected utility of a problem solver would then be its average problem solving time for the given fixed problem distribution.

It may not be immediately clear how to represent preferences in terms of a single utility measure. For example in many real world situations, comparisons between actions are made on the basis of several performance attributes. In planning problems we may care not only about how fast a plan is created, but also about other attributes such as plan intelligibility, execution cost, and robustness.

A large literature in the decision sciences is devoted to how to translate these "multi-attribute" problems into a single utility function (see, for example, [Roy71]).

With the fixed distribution and expected-utility assumptions we can characterize the value of a problem solver by its expected utility. Formally, let PS denote a problem solver, D denote the set of possible problems expressible in the domain, $Pr_D(x)$ be the probability of occurrence for problem $x \in D$, and $U(PS, x)$ be the utility of PS on problem x. The expected utility of PS with respect to the distribution D, written $E_D[U(PS)]$, is defined as:

$$E_D[U(PS)] = \begin{cases} \int_D U(PS,x)\,\mathrm{Pr}_D(x)dx & (D\ \ continuous) \\ \sum_{x \in D} U(PS,x)\,\mathrm{Pr}_D(x) & (D\ \ discrete) \end{cases} \tag{1}$$

2.2 COMPOSITE TRANSFORMATIONS

Next we must formalize the effects of learning. A learning algorithm maps some initial problem solver PS_{old} into a new problem solver PS_{new}. We introduce the abstract notion of a *composite transformation* to denote the structural changes performed to a problem solver in the course of learning. A composite transformation is whatever is required to transform PS_{old} into PS_{new} and it may be built from several individual structural changes. For example, in the adaptive problem solving system PRODIGY/EBL [Minton88] a composite transformation is built from a sequence of learned control rules. A given learning algorithm has the potential to produce a variety of composite transformations depending on the initial problem solver and the distribution of problems. This corresponds to the notion of a hypothesis space in classification learning and we characterize it as a set of possible composite transformation. Alternatively, this set can be thought of as the set of all problem solvers reachable by the learning algorithm. Note that this set will be quite large (possibly infinite) for a non-trivial learning approach.

A composite transformation maps PS_{old} into some PS_{new}. The value of this composite transformation can be measured by the difference in expected utility between PS_{new} and PS_{old}. This provides a metric for assessing the performance of a learning algorithm. Given the set of composite transformations that a learning algorithm could produce, it should choose one that increases expected utility.

2.3 OPTIMALITY VS. IMPROVEMENT

Expected utility defines a total preference ordering over a set of composite transformations associated with a learning approach. The ideal learning algorithm would choose the composite transformation in this set with the highest expected utility, thus producing the most preferred problem solver. Such an algorithm can be considered optimal with respect to the set of composite transformations. One might consider optimality as part of the requirement for solving the utility problem.

Unfortunately, optimality is an extremely expensive requirement for the type of learning problem we would like to consider. For many machine learning algorithms it can be shown that it is computationally intractable to identify the optimal transformation. For example Greiner shows the inherent difficulties when transformations are constructed from macro-operators [Greiner89]. Therefore we have chosen not to insist on optimality and instead we adopt a weaker requirement for solving the utility problem. In our analysis, when a learning algorithm adopts a composite transformation it must increase expected utility, but it need not be the optimal choice.

2.4 UNKNOWN INFORMATION

For a given learning problem there is some true but unknown expected utility associated with each possible transformed problem solver: both the utility of a transformed problem solver on any given problem, and the probability distribution over the space of possible problems are typically unknown in advance. A learning system can only estimate this information through training examples. For example, a learning system might estimate the value of a transformation by solving some randomly selected problems with the original and transformed problem solvers. Increasing the number of examples would result in a better estimate, but it will still differ from the true expected utility.

Unknown information means a learning algorithm cannot guarantee that a composite transformation increases expected utility. There is always the possibility that a transformation is estimated to increase expected utility when in fact it does not. Nevertheless, in defining the utility problem we would like to explicitly quantify the chance that learning does not improve performance. Therefore we adopt a probabilistic requirement. A solution to the utility problem is required to adopt a composite transformation only if has a high probability of improving expected utility, where this probability is pre-specified and may be arbitrarily close to one.

2.5 THE UTILITY PROBLEM

We now can define the utility problem using the constructs just introduced.

DEFINITION 1: THE UTILITY PROBLEM.

Given: *(1) an application described by a utility function, a set of problems, and access to problems drawn according to a fixed probability distribution,*
(2) an initial problem solver for this application, $PS_{old,}$
(e) a set of candidate composite transformations, and (4) a confidence level.
Identify: *a composite transformation such that, with the specified confidence, the expected utility of the resulting problem solver,* PS_{new}, *is greater than or equal to expected utility of* PS_{old}.

This specification provides a clear criteria by which to judge a learning algorithm. We say a learning algorithm *solves* the utility problem if it satisfies the requirements of this definition.

3 A Solution to the Utility Problem

The preceding framework not only defines the goal of a learning technique, it suggests a natural solution to the utility problem. The effectiveness of a transformation can be judged in terms of its expected utility, which can be estimated to an arbitrary level of confidence using statistical procedures. However there are many difficulties that must be resolved before this theoretical idea can be realized in an efficient and practical algorithm. This section outlines our strategies for addressing these difficulties. These strategies are realized in our COMPOSER algorithm which implements an efficient yet general solution to the utility problem.

3.1 INCREMENTAL LEARNING

The most significant difficulty arises in how to efficiently investigate the vast set of possible composite transformations. Frequently there is some internal structure to transformations that can be exploited. In most learning techniques a composite transformation consists of many individual *atomic transformations* (later we refer to atomic modifications simply as *transformations*). For example, SOAR constructs a new problem solver from individual chunks [Laird86]. PRODIGY/EBL builds a learned control strategy from individual control rules [Minton88]. Two different composite transformations may share many of the same individual components.

Instead of making a global decision among all possible composite transformations, an *incremental* learning system efficiently builds up a composite transformation by making many local decisions. The *composability problem* is the name we give to the problem of identifying an effective composite transformation given that it must be constructed from multiple atomic transformations. This is analogous to planning problem. A planner does not solve a goal by searching through the set of all complete plans. Rather it uses operators to make incremental progress. In COMPOSER we view atomic transformations as *learning* operators. Just as individual planning operators can be flexibly combined to solve a variety of goals, individual atomic transformations can be combined to address the particular combination of problem solving inefficiencies. In fact, most learning for planning techniques can be viewed from this perspective, although they are not generally described in such terms.

3.1.1 Operationality Criteria: Independent Measures

The composability problem constrains the acceptable local measures of atomic transformation quality. A measure must ensure that locally beneficial transformations combine into a beneficial composite. One solution is to develop *independent measures*. These are local measures that assess the quality of an atomic transformation independently of whatever other transformations appear, or will appear, in the final composite transformation. Mitchell *et.al.*'s operationality criteria [Mitchell86] and Etzioni's non-recursive hypothesis [Etzioni90a] can be seen as attempts to provide such a measure.

Unfortunately, independent measures are in general not possible. Atomic transformations potentially interact with each other in difficult to predict ways. Such interactions have long been recognized as a source of difficulty in planning, but have been overlooked in learning, often with unfortunate consequences. Simple syntactic independent measures like operational and the non-recursive hypothesis, while useful heuristics, cannot provide even weak guarantees against detrimental results.

The ways in which interactions can arise are as varied as the space of learning techniques. We illustrate the difficulty with interactions as they arise in one common learning for planning formalism. Consider the problem of building a composite transformation where the atomic transformations are search control rules, as in PRODIGY/EBL [Minton88]. Let utility be the time to solve a problem. Control rules can reduce the time to solve a problem by eliminating search, but they introduce an evaluation overhead: the control rule's preconditions must be continually evaluated to see if a portion of the search tree can be pruned. In isolation, a control rule increases utility if the search saved exceeds this evaluation cost. Thus search time saved minus evaluation cost is a candidate for an independent evaluation criterion. Unfortunately, control rules interact so that the improvement of multiple control rules is not the sum of the improvements of the rules in isolation. For example, two control rules may avoid the same search nodes. As there is no added benefit in ignoring an area twice, the utility of the two together is not equivalent to the sum of their improvements in isolation. A more subtle example involves a control rule that has different evaluation costs in different portions of search space. A second control rule that removes portions of this search space may substantially change the average evaluation cost of the first.

To make this quantitative, consider the interaction between the control rules illustrated in Figure 2. This shows a hypothetical search space of fifteen nodes. Suppose *r* and *s* are two heuristics that prune the nodes in sets R and S respectively. |R| is the number of nodes trimmed by *r*. |S| is similarly defined. When used in isolation, *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. Heuristic *s* is checked eight times (i.e. 15 – |S|) and succeeds at node 1, saving nodes 9–15. Assume the average evaluation cost of *r* is M_r, the average cost of *s* is M_s, and the average time to expand a node is *g*.

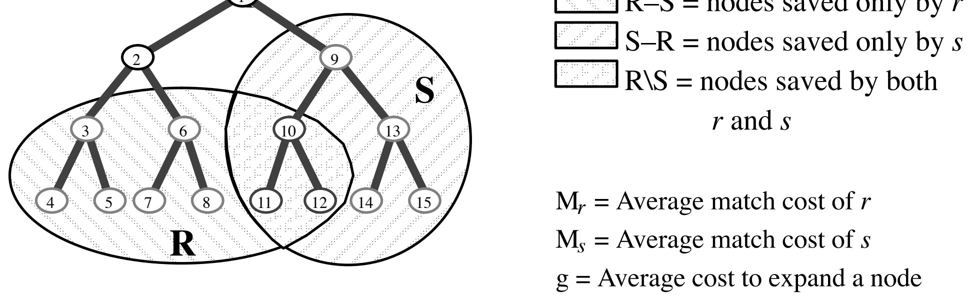

1
2
9
S
3
6
10
13
4
5
7
8
11
12
14
15
R
▨ R–S = nodes saved only by r
▨ S–R = nodes saved only by s
▨ R\S = nodes saved by both r and s
Mr = Average match cost of r
Ms = Average match cost of s
g = Average cost to expand a node

Figure 2: Example of interacting heuristics

Let $U(X, p)$ be the utility of a problem solver using the set of rules X on problem p. The interaction between two rules on a problem is the amount to which their utilities are not additive:

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

This residue is measures the interaction between atomic transformations. The transformations combine synergistically if this value is positive, for example when one control rule reduces the match cost of another. If negative, they engage in a harmful interaction, for example when there is large overlap in the search they avoid. The key point is that two control rules with positive utility in isolation can potentially combine to yield a strategy worse than neither. Such interactions would seem to preclude effective independent criteria for determining the benefit of atomic transformations.

3.1.2 Incremental Utility: A Context-Sensitive Measure

While we cannot develop a general independent measure of atomic transformation quality, we can develop a *context-sensitive* measure that accounts for the context of other atomic transformations participating in the composite transformation. In particular, we can view a composite transformation as a sequence of problem solvers: $PS_{old}, PS_1, PS_2, \ldots, PS_{new}$, each with some expected utility. We adopt a context-sensitive measure of utility that states how much a given atomic transformation improves expected utility if appended to an existing sequence of transformations.

Let PS denote a problem solver and let $PS^{\tau} = Apply(\tau, PS)$ denote the problem solver that results from applying an atomic transformation τ to PS. We define the *incremental utility* of τ to be the difference between the expected utility of PS^{τ} and the expected utility of PS.[1] We denote incremental utility as $\Delta U_D(\tau|PS)$, meaning the conditional change in expected utility provided by transformation τ over distribution D given problem solver PS. We can state this formally as:

$$\Delta U_D(\tau|PS) = E_D[U(PS^{\tau})] - E_D[U(PS)] \quad \text{or equivalently:}$$

$$\Delta U_D(\tau|PS) = \begin{cases} \displaystyle\int_D \left[U(PS^{\tau}, x) - U(PS, x)\right] \Pr_D(x)dx \;\; (D \;\; continuous) \\ \displaystyle\sum_{x \in D} \left[U(PS^{\tau}, x) - U(PS, x)\right] \Pr_D(x) \;\; (D \;\; discrete) \end{cases} \tag{2}$$

The change in expected utility provided by a composite transformation is equivalent to the sum of the incremental utilities of the each transformation:

$$\Delta U_D(\tau_0|PS_{old}) + \Delta U_D(\tau_1|PS_{old}^{\tau_0}) + \Delta U_D(\tau 2|PS_{old}^{\tau_0, \tau_1}) + \ldots$$

The definition of incremental utility clarifies two important properties of transformations. First the effect on utility that results from a transformation is dependent on the distribution D. Second, the effect is conditional on the problem solver to which it is applied. In general the incremental utility

1. In other learning algorithms incremental utility is simply referred to as utility. We feel the additional terminology helps to highlight the difference between the utility of a problem solver, which is of interested to the user, and the utility of the a transformation which is only of interest to the learning algorithm.

of a transformation will vary unpredictably as we change either the distribution or the problem solver to which it is applied. The conditional nature of incremental utility ensures that in general we cannot identify a globally maximal composite transformation without considering a potentially explosive number of conditional utility values.

3.2 COMPOSER

COMPOSER is a statistical approach that provably solves the utility problem as stated in Definition 1. Given a learning element that provides atomic transformations, COMPOSER incrementally constructs a composite transformation, guaranteeing (with as high confidence as required) an improvement in expected utility. COMPOSER requires as input a transformable initial problem solver, a learning element with certain, to be specified, characteristics, a utility function, and a source of training problems drawn randomly from the problem distribution. We first describe COMPOSER's method for searching the set of composite transformations. We next define the constraints on the method for proposing atomic transformations. Finally we describe how the system achieves its statistical guarantee.

3.2.1 Overview

Because of the composability problem, it is typically intractable to determine the best sequence of transformations. However, we want as great an improvement as possible. COMPOSER uses incremental utility in conjunction with a greedy hill-climbing procedure to explore the set of possible composite transformations. COMPOSER begins its search with the original problem solver and incrementally adopts transformations that are estimated to possess positive incremental utility. Each new transformation is assessed with respect to the problem solver that results from applying the previous transformation. COMPOSER uses statistical methods to estimate incremental utility from training examples drawn randomly according to the distribution of problems. This hill-climbing approach successfully avoids the difficulty of negative interactions. One shortcoming is it cannot exploit positive interactions. Therefore solutions may be local optima.

3.2.2 Transformation Generator

COMPOSER requires a source of transformations for each step in the search. This is abstractly formalized as a function we call a *transformation generator*. This is a function $TG : PS \times X \rightarrow \{\tau_1, \ldots, \tau_k\}$ that maps a problem solver and an optional set of training examples into a set of candidate transformations. Each transformation in the set should map the problem solver into some new, possibly improved, problem solver. For example, SOAR can be viewed as a transformation generator takes the current problem solver and the single training problem that produced an impasse, and generates a set of chunks. Oren Etzioni's STATIC system [Etzioni90a] can be seen as a transformation generator that takes the original problem solver and no training examples, and generates a set of control rules.

3.2.3 Statistical Inference

COMPOSER must estimate incremental utility from training data and assess the accuracy of these estimates. There are several philosophical stances for reasoning about these statistical issues. The computational learning community has favored worst-case statistical models (also called *non-parametric* techniques). These are quite useful to make theoretical statements but are too inefficient for most practical uses. *Parametric* techniques are a more practical alternative. In these the distribution of utility values is assumed to a function of some unknown parameters which must be estimated from the data. Bayesian models are a popular parametric alternative to worst-case models but they require the specification of prior knowledge about the probable performance of the alternative transformations. We prefer to avoid dependence on prior information, therefore we have adopted so-called frequentist statistical models which have the efficiency of bayesian approaches without the need for the specification of prior information.[2] When we must balance between absolutely insuring error does not exceed a bound and achieving reasonable efficiency, we err to the side of efficiency, as long there are good arguments that the error bounds are not exceeded in practice. In a later section we describe how to adjust this tradeoff when necessary.

COMPOSER must ensure that the *overall* error remains below some threshold δ. Formally:

$$\Pr\Big(E_D[U(PS_{new})] < E_D[U(PS_{old)})]\Big) \leq \delta \tag{3}$$

To achieve this, COMPOSER must account for three sources of error. Each time COMPOSER identifies a new transformation to adopt, it compares a set of estimates, one for each transformation considered at that step. First, there is some probability of error associated with each of these estimates. Second, these individual errors combine into a somewhat larger probability of error for the overall decision of what to adopt at a given step. Third, the final problem solver, PS_{new}, is produced as a result of several steps, and the error across all of these steps must be accounted for as well.

We satisfy Definition 1 by bounding the first sense of error (the error of each individual incremental utility estimate) in such a way that as they combine into the second and third sense, the overall error remains below the total acceptable bound of δ. The bound for each estimate is specified by the function $Bound(\delta, |T|)$. This defines the acceptable error for an estimate as a function of the overall error, the current step in the hill-climbing search, and the size of the set of transformations T at that step:[3]

$$Bound(\delta, |T|) = \frac{\delta}{|T|}$$

Given this definition it remains to construct a statistical procedure that estimates incremental utility of a transformation to the specified error bound. This involves two issues. First we must determine

2. There is controversy between bayesians and frequentists as to which approach is more appropriate; in many cases it can be shown that bayesian approaches with non-informative priors are equivalent to frequentist approaches. We do not take a strong stand on this issue. One could easily replace our frequentist model with a bayesian one without changing the principal theoretical contributions of this work.

how estimates are generated from training problems. Second we must decide how many training problems are needed to attain sufficiently accurate estimates.

We can estimate incremental utility by randomly drawing problems according to the distribution D and, for each transformation under consideration, averaging several observations of this value. Call $\overline{\Delta U_n}(\tau|PS)$ the estimated incremental utility from n data points. To gather the necessary data points, COMPOSER must determine, given a current problem solver PS, a set of transformations, and a problem x, the difference in utility between the current and each of the transformed problem solvers: $\forall\tau\in T$, $U(PS^{\tau}, x) - U(PS, x)$. Recall that, by definition, the utility of a problem solver on a problem is measurable by observing the behavior of the problem solver on the problem. Thus, given a set of m transformations, we can compute the necessary incremental utility values by solving the problem with PS and then with $PS^{\tau 1}, PS^{\tau 2}, ..., PS^{\tau m}$. Processing each training example involves m+1 problem solving attempts. The complexity of processing an example is therefore tied to the complexity of each of the m+1 problem solvers. We call this *brute-force* processing and is the default method used by COMPOSER. We show in Appendix A that for many applications there are other more efficient methods for obtaining these data points.

We can determine a suitable sample size by considering how the accuracy of estimates improves as we increase the number of examples used in the computation. Specifically, COMPOSER must determine a sufficiently large sample size such that it can bound the probability that incremental utility is estimated to be positive when in fact it is negative and vice versa. The estimates must be accurate but we would like them to be based on as few examples as possible. COMPOSER relies on a *sequential* statistical technique to determine how many examples are sufficient to make this inference [Govindarajulu81]. Sequential procedures differ from the more common fixed-sample techniques in that the number of examples is not determined in advance, but is a function of the observations. Sequential procedures provide a test called a *stopping rule* that determines when sufficient examples have been taken. An important advantage of sequential procedures is that the average number of examples required is smaller than the number required by a fixed–sample technique.

We determine the sample size using a stopping rule proposed by Nádas [Nádas69]. The intuition behind the stopping rule is quite simple. Given a sample of values and a confidence level α, we can construct confidence intervals that contain, with probability $1 - \alpha$, the true incremental utility lies. In other words, there is only probability α that the true incremental utility lies outside the confidence interval. With more examples, the width of the interval tends to shrinks.[4] The stopping rule determines how many examples are needed for the confidence interval to shrink to the point of being entire-

3. This definition embodies a compromise between bounding statistical error and example efficiency which works well in practice. The rational behind the compromise is discussed in Appendix A, where we illustrate some other possible definitions.

4. The interval size is actually a random function of the data. It will tend to shrink with more examples, but not monotonically. It will grow, temporarily, if an outlying data point is encountered.

ly above or below zero incremental utility. This is illustrated in Figure 3. When this occurs, we can state, with probability 1 – α, that the transformation has positive (negative) incremental utility if the estimate is positive (negative).

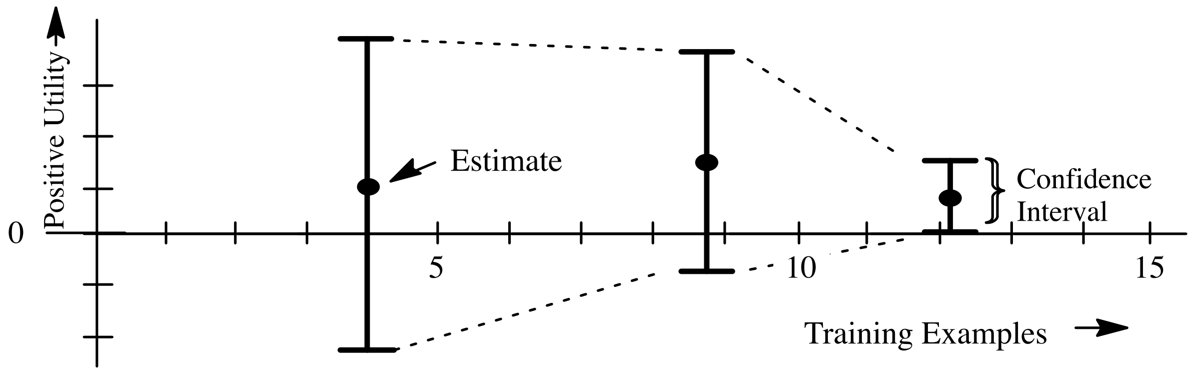


Figure 3: Sizing the confidence interval. We would like to take enough examples such that the confidence interval lies entirely above or below the axis.

After each example is processed the Nádas stopping rule is evaluated. Sampling should terminate when the rule evaluates to true. When this occurs we can state that the given transformation will speed-up (slow down) *PS* if its estimated incremental utility is positive (negative) with confidence 1–α. Examples are taken until the following equation holds:

$$n \geq n_0 \ AND \ \frac{S_n^2(\tau|PS)}{\left[\overline{\Delta U_n}(\tau|PS)\right]^2} \leq \frac{n}{Q(\alpha)^2} \tag{4}$$

where *n* is the number of examples taken so far, $\overline{\Delta U_n}(\tau|PS)$ is the transformation's average improvement, $S_n^2(t|PS)$ is the observed variance in the transformation's improvement, α is the acceptable error in the estimate, n_0 is a small finite integer indicating an initial sample size, and *Q*(α) is a parametric function that models the discrepancy between the true and estimated incremental utility:

$$Q(\alpha) := x \ \ where \ \int_x^\infty \left(1/\sqrt{2\pi}\right) e^{-0.5y^2} dy = \frac{\alpha}{2}$$

The function *Q*(*n*,*a*) makes the parametric statistical assumption that estimated incremental utility is normally distributed about the true incremental utility. The reasonableness of this parametric model is justified by an important theorem in statistics, the Central Limit Theorem [Hogg78 p. 192]. This theorem states that regardless of the distribution of a random variable, the average of these values will tend to be normally distributed about the true mean of the distribution.

The choice of initial sample size, n_0, relates to the definition of *Q*(α) and is discussed in Appendix A. By default we use a sample size of fifteen which has worked well in our empirical investigations.

3.2.4 The COMPOSER Algorithm

COMPOSER illustrated in Figure 4. After each problem solving attempt, COMPOSER updates its statistics and evaluates the stopping rule for each candidate transformation. If stopping rules are satisfied for transformations with *positive* incremental utility (there may be more than one) COMPOSER adopts the one with highest incremental utility, removes it from the candidate set, and discards the statistics for the remaining candidates as these apply to the previous problem solver. If instead, stopping rules are satisfied for transformations with *negative* incremental utility, these are eliminated from further consideration (note that eliminating a candidate does not affect the current problem solver, so the statistics associated with the remaining candidates are unaffected). This cycle repeats until the training set is exhausted. Each time a transformation is adopted the expected utility of the resulting problem solver is higher than its predecessor, giving COMPOSER an anytime behavior [Dean88].

```
Given: $PS_{old},\ TG(\cdot),\ \delta$
[1] $PS := PS_{old};\ T := TG(PS);\ i := 0;\ n := 0;\ \alpha := Bound(\delta, |T|);$
[2] While more examples and $T \neq \emptyset$ do
[3]     $n := n + 1$
[4]     $\forall \tau \in T$: Get $\Delta U_n(\tau/PS)$     /* Gather statistics and find transformations that have reached significance */
[5]     $\alpha\text{–}estimates := \left\{ \tau \in T : n \geq n_0 \text{ and } \frac{S_n^2(\tau|PS)}{\left[\overline{\Delta U_n}(\tau|PS)\right]^2} < \frac{n}{\left[Q(\alpha)\right]^2} \right\}$     /* Collect all trans. that */ /* have reached statistical */ /* significance. */
[6]     $T := T - \left\{ \tau \in \alpha\text{–}estimates : \overline{\Delta U_n}(\tau|PS) < 0 \right\}$     /* Discard trans. that decrease expeced utility */
[7]     If $\exists \tau \in \alpha\text{–}estimates : \overline{\Delta U_n}(\tau|PS) > 0$ Then     /* Adopt τ that most increases expected utility */
[8]         $PS = Apply(x \in \alpha\text{–}estimates : \forall y \in \alpha\text{–}estimates \left[\overline{\Delta U_n}(x|PS) > \overline{\Delta U_n}(y|PS)\right], PS)$
[9]         $T := TG(PS);\ i := i + 1;\ n := 0;\ \alpha := Bound(\delta, |T|)$
Return: $PS$
Defaults: $Bound(\delta, |T|) := \frac{\delta}{|T|}$ ,   $Q(\alpha) := x$ where $\int_x^\infty \left(1/\sqrt{2\pi}\right) e^{-0.5y^2} dy = \frac{\alpha}{2}$
```

Figure 4: The COMPOSER algorithm

4 Utility Problem in the PRODIGY Problem Solver

This section describes one of two extensive evaluations we have performed with the COMPOSER system. We describe an application of COMPOSER to learning search control strategies for the PRODIGY planning system [Minton88]. PRODIGY is a well studied planning system that has served as the basis for several learning investigations [Etzioni90b, Knoblock89, Minton88] and has attained the status as a benchmark for learning systems. COMPOSER has also been successfully applied to the problem of learning heuristic control strategies for a NASA scheduling domain, which is described elsewhere [Gratch93].

A main goal of this evaluation is to contrast COMPOSER's approach with several other methods for addressing the utility problem, including methods developed explicitly for the PRODIGY system as well as a more general statistical approach similar to COMPOSER. We would like to compare the theoretical basis for these alternatives rather than implementation details. Therefore we take pains to provide a fair comparison by minimizing the differences between the systems. Each method is re-implemented within the context of the PRODIGY problem solving system and each method is constrained to use the same source of learned transformations.

4.1 THE APPLICATION

This implementation is constructed within the PRODIGY 2.0 architecture which is available from Carnegie Mellon University. PRODIGY is a general purpose means-ends problem solver based on the STRIPS planner [Fikes71] with a few enhancements. Plans are identified by depth-first search. Search proceeds by recursively applying four control decisions: (1) choosing a node to expand in the current search space, (2) choosing an unachieved goal at that node, (3) choosing an operator that possibly achieve with the goal, and (4) choosing a binding list for the operator. PRODIGY implements a default control method for each of these decisions and these methods may be modified by the introduction of heuristic knowledge called *control rules*. Control rules are described in Section 6.2.

4.1.1 Problem Distributions

We evaluate COMPOSER's ability to identify effective modifications to PRODIGY on three different domain theories. The STRIPS domain was reported in [Minton88]. It is a problem of a robot moving boxes through interconnected rooms with lockable doors. The AB-WORLD domain was reported in [Etzioni91]. It is a variant of the standard blocksworld domain, designed to highlight deficiencies in Minton's PRODIGY/EBL approach. The BIN-WORLD domain was introduced in [Gratch91a] and was designed to highlight deficiencies in both PRODIGY/EBL and Etzioni's STATIC approach. This domain is a simple construction domain.

Problem distributions for the STRIPS domain and AB-WORLD domain are constructed with the problem generators provided with PRODIGY 2.0. Following the methodology in [Minton88], the set of problems were biased by filtering out problems that were judged too difficult or too easy; problems were excluded if the default PRODIGY control strategy required less than 1 CPU second or more than 80 CPU seconds. Problems for the BIN-WORLD were generated in a distribution designed to highlight deficiencies in PRODIGY/EBL and STATIC. This is described in Appendix B. A more detailed description of the experiments appears in [Gratch93d].

4.1.2 Expected Utility

We follow the established PRODIGY methodology of measuring problem solving performance by the cumulative time in CPU seconds to solve a set of problems (e.g. [Etzioni91, Minton88]). Under the decision-theoretic interpretation of the evaluation criterion, this is captured as a utility function based on the time required to solve a problem. In particular, we let the utility of the problem solver over a problem be the negative of the CPU time required to solve it. Maximizing expected utility therefore translates into decreasing the average time to required to solve a problem.

4.2 TRANSFORMATION GENERATOR

Minton introduced a technique for generating atomic modifications to the PRODIGY control strategy. The approach uses explanation-based learning (EBL) [DeJong86, Mitchell86] to construct atomic control rules based on traces of problem solving behavior and a theory of the problem solver. Sets of control rules can be associated with any of the four control points. The control rules are condition–action statements which alter the way PRODIGY explores the space of possible plans. By default, the planner entertains 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 5 illustrates a control rule learned by the EBL method in the blocksworld domain. The blocksworld has several operators for clearing a block. RULE–1 asserts that in situations where the block is not held, 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 5: An example control rule

Not all control rules increase the efficiency of planning. Control rules avoid search in the plan space, however they introduce the cost of matching their preconditions. A rule is harmful when the precondition evaluation cost exceeds the savings. Furthermore, control rules interact in subtle ways. Without a criterion for choosing among possible rule sets, the learning algorithm quickly degrades performance. Minton introduced a heuristic empirical procedure for addressing the utility problem in this context. This procedure attempts to account for the distributional nature of the incremental utility of individual control rules. Minton calls the overall approach of EBL learning and heuristic utility analysis PRODIGY/EBL. Unfortunately, while it performs quite well on some domains, PRODIGY/EBL has since been shown to have undesirable properties. Oren Etzioni illustrated how seemingly innocuous changes to a domain theory result in degraded problem solving performance [Etzioni90b]. We showed that this behavior is due to the utility procedure's inability to correctly estimate distribution information and to handle the composability problem (see [Gratch91]). COMPOSER's statistically sound utility estimation procedure corrects these problems and should result in a more effective learning algorithm.

For this evaluation the EBL technique serves as the transformation generator. The EBL component requires training problems to be solved with PRODIGY using some control strategy (in this case a control strategy is a set of control rules). The component analyzes a trace of each solution attempt, and conjectures new control rules. Each of these control rules is an atomic transformation to the current search control strategy. In Minton's system, all control points are given equal importance so our implementation contains no independence or dominance relations between control points.

4.3 COMPOSER IMPLEMENTATION DETAILS

We used COMPOSER to construct a adaptive problem solver for the three applications. We call the resulting implementation COMPOSER/PRODIGY. PRODIGY with no control rules acts as the initial problem solver. Minton's EBL learning element acts as the transformation generator.[5] Thus, COMPOSER/PRODIGY takes the problem solver with the empty set of control rules as PS_{old} and produces a PS_{new} by incrementally adding control rules with positive incremental utility.

Several properties of this application allowed us to tailor COMPOSER to achieve greater statistical efficiency. We exploited a property of control rules to more efficiently gather incremental utility data points. The implementation extracts incremental utility values for all candidate control rules with an unobtrusive procedure of using the trace of a single solution attempt. To accomplish this, we modified the PRODIGY planner to distinguish between adopted and candidate control rules. Adopted control rules are those which have been shown to have positive incremental utility, added into the control strategy of the PRODIGY planner, and therefore effect PRODIGY's behavior. Candidate rules are those that have been proposed by the EBL technique, but not yet validated. When solving a problem, candidate rules are checked, their precondition cost and the search paths they would eliminate are recorded, but the search paths are not actually eliminated. After a problem is solved, the annotated trace can be analyzed to identify those search paths which would have been eliminated by candidate control rules. 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 control rule for that problem.[6]

5. The EBL unit produces two control rule types: rejection rules and preference rules. Minton has noted that preference rules seem to be less effective. We have verified that PRODIGY/EBL actually produces strategies with higher utility if it is prevented from producing preference rules [Gratch91]. We disabled the learning of preference rules in this implementation because it enabled a more efficient means of gathering incremental utility data points.

6. Appendix A describes how for some applications the *Bound* function can be modified to improve the efficiency of utility analysis. For this implementation we used Equations 6 and 7 of the appendix to define a variant of the default *Bound.,* As stated in Section A.1.1, this allows transformations to be added to T at any time in the utility analysis. Due to the relatively low variable of control rules we used an initial sample size of three.

4.4 EVALUATION

We evaluated COMPOSER/PRODIGY's performance against four other proposed criteria for addressing the utility problem: the heuristic utility analysis of PRODIGY/EBL [Minton88], the non-recursive hypothesis of STATIC [Etzioni90a], a hybrid of PRODIGY/EBL and STATIC suggested by Etzioni [Etzioni90a] to overcome limitations of the two systems, and PALO [Greiner92a], a statistical approach similar to COMPOSER but based on a more conservative statistical model. Before discussing the experiments we review these techniques. For the evaluations we tried to minimize differences between the systems that are not theoretically motivated. All systems are implemented within the PRODIGY problem solving framework and are constrained to use the same transformation generator.

4.4.1 PRODIGY/EBL'S Utility Analysis

This is the approach developed by Minton for use in PRODIGY/EBL. The technique 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 (the example problem from which the rule was learned) and this value is credited to the rule each time it applies. Match cost is measured directly from problem traces and averaged across multiple training examples. 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 as estimates are gathered as if there were no interactions.

4.4.2 STATIC's Nonrecursive Hypothesis

STATIC utilizes a control rule selection criterion based on Etzioni's structural theory of utility. The criterion is grounded in the *nonrecursive hypothesis*. This states that "EBL is effective when it is able to curtail search via nonrecursive explanations" [Etzioni90a p. 6]. The hypothesis admits several interpretations. The strongest interpretation is that transformations have positive incremental utility, regardless of problem distribution, if they are generated from nonrecursive explanations of planning behavior (i.e. no predicate in a subgoal is derived using another instantiation of the same predicate). A weaker reading is that a composite strategy will improve expected utility if it is constructed from non-recursive elements (admitting that some transformations will have negative incremental utility, but a set of non-recursive transformations will improve performance). The issue of interactions between transformations is also not addressed. STATIC applies this criterion to control rules but the issue is important in macro–operators as well [Letovsky90, Subramanian90].

STATIC out-performs PRODIGY/EBL's on several domains. The nonrecursive hypothesis is cited as a principal reason for this success [Etzioni90b][7]. This claim is difficult to evaluate as these algorithms use different vocabularies to construct their control rules. We wish to focus on the effectiveness of the non-recursive hypothesis, and therefore must remove the complicating factor of a different

7. Etzioni and Minton have subsequently suggested that some of the success of STATIC is due to the fact that its global analysis allows for more concise rules [Etzioni92].

rule generator. To achieve this goal we constructed the NONREC algorithm, a re-implementation of STATIC's nonrecursive hypothesis within the PRODIGY/EBL framework. NONREC replaces PRODIGY/EBL's empirical utility analysis with a syntactic criterion which only adopts nonrecursive control rules. This acts as a filter, only allowing PRODIGY/EBL to generate non-recursive rules.

4.4.3 A Composite Algorithm

Etzioni suggests that the strengths of STATIC and PRODIGY/EBL can be combined into a single approach [Etzioni90a]. He proposed a hybrid algorithm which embodies several advancements including a two layered utility criterion. The nonrecursive hypothesis acts as an initial filter, but the remaining nonrecursive control rules are subject to utility analysis and may be later discarded.

We implemented the NONREC–UA algorithm to test this hybrid criterion. As control rules are proposed by PRODIGY/EBL's learning module, they are first filtered on the basis of the nonrecursive hypothesis. The remaining rules undergo utility analysis as in PRODIGY/EBL.

4.4.4 PALO'S Chernoff Bounds

Greiner and Cohen have proposed an approach similar to COMPOSER's [Greiner92b]. The Probably Approximately Locally Optimal (PALO) approach also adopts a hill–climbing technique and evaluates transformations by a statistical method. PALO differs in its stopping rule and and that it incorporates a criterion for when to stop learning. PALO terminates learning when it has (with high probability) identified a near–local maximum in the transformation space. Our evaluation focuses on the different stopping rule which is based on Chernoff bounds.

Chernoff bounds provide a much more conservative model of the discrepancy between a the sample mean and true mean of a distribution. As a result, PALO provides stronger bounds on statistical error at the cost of more examples. 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.[8] Our PALO–RI algorithm evaluates this approach. Like COMPOSER, PALO–RI uses a candidate set of rules. In this case the size of the set is fixed before learning begins. A candidate is adopted when the following stopping rule holds:

$$\sum_{i=1}^{n} X_{r,i} \; > \; \Lambda_r \sqrt{2n \ln\left(Cs^2\pi^2/(3\delta)\right)}$$

where $X_{r,i}$ is the incremental utility of rule r on problem i, C is the maximum size of the candidate set, s is one plus the number of rules in the current control strategy, and Λ_r is a is the maximal per problem ΔUTILITY of rule r. The parameter C is the size of the candidate set. PALO-RI uses the same method as COMPOSER/PRODIGY to obtain incremental utility statistics. We discuss the setting of the system's various parameters in the next section.

8. In addition to the conservative stopping rule, PALO adopts the conservative definition of *Bound* that follows from adopting Equations 5 and 9.

4.4.5 Experimental Procedure

We investigated the STRIPS domain from [Minton88], the AB-WORLD domain from [Etzioni90a] for which PRODIGY/EBL produced harmful strategies, and the BIN-WORLD domain from [Gratch91b] which yielded detrimental results for both STATIC's and PRODIGY/EBL's learning criteria. Results are summarized in Figure 6. In each domain the algorithms were trained on 100 training examples drawn randomly from a fixed distribution. The problem distributions were constructed using the random problem generator provided with the PRODIGY architecture. The current control strategy was saved after every twenty training examples.[9] The independent measure for the experiments is the number of training examples and the dependent measure is the execution time in CPU seconds over 100 test problems drawn from the same distribution. This process was repeated eight times, using different but identically distributed training and test sets and all results are averaged over these trials. Figure 6 illustrates the results along with the average number of rules learned by the algorithm, the number of seconds required to process the 100 training examples, and the number of seconds required to generate solutions for the 100 test problems.

COMPOSER and PALO–RI require an error parameter δ which is set at 10% for the experimental runs. PALO–RI's behavior is strongly influenced by parameters whose optimal values are difficult to assess. We tried to assign values close to optimal given the information available to us.[10]

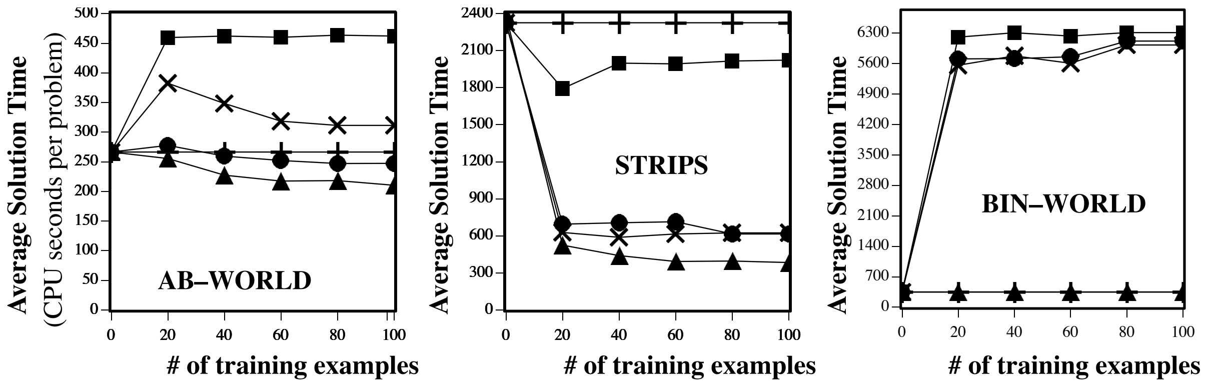
During the evaluation, it was apparent that PALO–RI would not adopt any transformations within the 100 training examples. We tried to give the algorithm enough examples to reach quiescence but this proved too expensive. The problem is twofold — first, too many training examples were required; secondly, and as a consequence of the first problem, the candidate set grew large since harmful rules were not discarded as quickly as in COMPOSER/PRODIGY. This increased the cost to solve each training example. To collect statistics on PALO–RI we only performed one instead of eight learning trials. Furthermore, we terminated PALO–RI after the first transformation was adopted or 10,000 examples, whichever came first.

4.5 ANALYSIS

The results illustrate several interesting features. The implementation developed with COMPOSER exceeded the performance of all other approaches in every domain. The learned strategies yielded higher expected utility and were more succinct (contained fewer control rules). In AB-WORLD and STRIPS, COMPOSER/PRODIGY identified beneficial control strategies. In BIN-WORLD the algorithm did not adopt any transformations. In fact, it does not appear that any control rule improves

9. PRODIGY/EBL's utility analysis requires an additional settling phase after training. Each control strategy produced by PRODIGY/EBL and NONREC+UA received a settling phase of 20 problems following the methodology outlined in [Minton88].

10. *C* was fixed based on the size of the candidate list observed in practice. In the best case a rule can save the entire cost of solving a problem, so for each domain, lambda for each rule was set at the maximum problem solving cost observed in practice. AB–WORLD – C=30 lambda=15; STRIPS – C=20, lambda=100; BIN–WORLD – C=5, lambda=150.



	AB–WORLD			STRIPS			BIN–WORLD		
ALGORITHM	Rules Added	Train Time	Test Time	Rules Added	Train Time	Test Time	Rules Added	Train Time	Test Time
+ No Learning	—	—	266	—	—	2323	—	—	346
▲ COMPOSER/PRODIGY	1	1667	210	4	4133	382	0	3425	346
× PRODIGY/EBL	11	1253	311	20	3775	622	2	6383	6020
■ NONREC	17	1253	462	25	4012	2021	4	6710	6305
● NONREC+UA	9	1259	247	10	3821	614	2	6359	6110

Performance after first rule adopted	PALO–RI		
	Train Exmpls	Train Time	Test Time
AB–WORLD	6069	104,387	182
STRIPS	1182	41,370	1223
BIN–WORLD	10,000+	—	346

Figure 6: Results for the PRODIGY application. Learning curves show performance as a function of the number of training examples.

performance in this domain. It should be stressed that all algorithms use the same transformation generator. Therefore the results represent differences in approaches to the utility problem rather than in the vocabulary of transformations.

We expected COMPOSER/PRODIGY to have higher learning times than PRODIGY/EBL or NONREC due to its more rigorous in their assessment of incremental utility. Surprisingly, the learning times were not much higher than the non-statistical approaches and COMPOSER/PRODIGY actually learned more quickly on BIN-WORLD where it quickly discarded a control rule with high match cost which PRODIGY/EBL, NONREC, and NONREC+UA retained. As expected, PALO–RI's learning times were significantly higher than the other systems.

The results cast doubts on the efficacy of the nonrecursive hypothesis. NONREC yielded the worst performance on all domains. Even in conjunction with utility analysis the results were 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 indicated that the best rules were nonrecursive, but many nonrecursive rules were also detrimental. In BIN-WORLD, non-recursive rules produced substantial performance degradation. 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 completely account for this difference. We attribute the remaining difference to the fact that STATIC and NONREC entertain somewhat different sets of control rules. In our experiments we constrained NONREC to use the rule vocabulary which was available to PRODIGY/EBL while in Etzioni's experiments STATIC entertained a somewhat different space of rules. This conjecture was recently supported by Minton and Etzioni [Etzioni92].

Finally, although PALO–RI did not improve performance within the 100 training examples, if given sufficient examples it would likely out-perform all other approaches. With extended examples it did exceed COMPOSER/PRODIGY's performance in AB-WORLD. This is because under PALO's more conservative criteria, the transformation with the highest incremental utility will tend to have the smallest stopping time. Instead, with COMPOSER both incremental utility and the variance of utility values determine when a transformation has reached significance – COMPOSER tends to recognize low variance transformations more quickly. 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 Complexity Results

We now turn to a computational analysis of the COMPOSER algorithm. Many of the algorithm's characteristics will be, of course, domain specific. However there are some general results we can state that elucidate properties of the algorithm. Given our emphasis on the practical aspects of the algorithm, our analysis will not consider worst-case behavior. Rather, we provide *average-case* complexity results for the algorithm for both *sample complexity*, the number of examples required to make a statistical inference, and run-time complexity of the algorithm.

We focus on the number of examples and the amount of work required to perform a single step in the hill-climbing search as the total number of examples used by the algorithm, the number of hill-climbing steps, and therefore the run-time complexity are determined by how many examples the user provides and properties of the transformation generator. This number depends on domain specific factors, but these can be related to complexity in meaningful ways. For example, the amount of work required at a hill-climbing step depends on the number of candidate transformations at that step, but we can specify the exact function relationship between work and the number of transformations. With such knowledge, a user of COMPOSER can assess how best to organize the transformation generator for a specific learning problem.

5.1 PROPERTIES OF THE NÁDAS STOPPING RULE

The properties of COMPOSER follow from the properties of its method for statistical inference. Therefore, we first consider the characteristics of the Nádas stopping rule. Given some transformation, τ, and an error level α, this stopping rule determines sufficient examples to show that the incremental utility of τ is positive (negative) with probability $1-\alpha$. The characteristics of this stopping rule have been proven by Author Nádas in [Nádas69]. The proofs are technical but we will restate the results and give an intuitive explanation of why they hold.

COMPOSER take examples until the following inequality holds (Equation 4):

$$n \geq n_0 \ AND \ \frac{S_n^2}{\left[\overline{\Delta U_n}\right]^2} \leq \frac{n}{Q(\alpha)^2}$$

where n is the number of examples taken so far, $\overline{\Delta U_n}$ is the transformation's average improvement, S_n^2 is the observed variance in the transformation's improvement, α is the acceptable error in the estimate, n_0 is a small finite integer indicating an initial sample size, and $Q(\alpha)$ is the function that models the discrepancy between the true and estimated incremental utility:

$$Q_\Phi(\alpha) = x \text{ such that } \int_x^\infty \left(1/\sqrt{2\pi}\right) e^{-0.5y^2} dy = \frac{\alpha}{2}$$

For a given sequence of training examples the stopping rule will be satisfied after some number of examples, called the stopping time. For this sequence of examples, call the stopping time ST_1. A second, different sequence of training would yield a different stopping time, ST_2. The complexity of the stopping rule is characterized by the expected value of the stopping time: $E[ST]$. This is the average sample complexity of the stopping rule. From the results of Nádas we can derive the following relationship for the expected stopping time.

Theorem 1: Let *ST* be the stopping time associated with the Nádas stopping rule for a given transformation. Let α be the requested error level, σ^2 be the actual variance of the distribution associated with the transformation and μ be the actual incremental utility. Then:

1) For small $1/\alpha$ the expected stopping time is governed by the following relationship:

$$E[ST] \leq \frac{\sigma^2}{\alpha^2\mu^2}\sqrt{2/\pi}$$

2) For large $1/\alpha$ the expected stopping time is governed by the following relationship:

$$E[ST] \approx \ln(1/\alpha)\frac{\sigma^2}{\mu^2}$$

This result states that the stopping time is determined by the error level parameter,which is under control of the user, and two fixed but unknown constants, σ^2 and μ, which are properties of the inference problem. The average stopping time associated with a particular transformation is bounded by a quadratic in $1/\alpha$ (or to log of $1/\alpha$ for large $1/\alpha$), linearly with the variance of the transformation and quadratically with the inverse of its mean. This makes intuitive sense: the greater the required confidence, the more difficult it is to bound the mean, the greater the variance in incremental utility values, the more difficult it is to bound its mean, and the closer the incremental utility is to zero, the more difficult it is to show that the transformation is better (worse) than the default strategy.

Proof: While the theorem is too technical to describe here, it is easy to show why a result like Theorem 1 should hold. The Central Limit Theorem states that the normalized difference between the true incremental utility and the sample incremental utility should be normally distributed. Using this observation it is easy to compute a confidence interval around the mean, given a sample of *n* observations. Any introductory statistics book shows (with a suitable mapping of notations) that:

$$\Pr\left[\overline{\Delta U_n} - Q(\alpha)\sqrt{\frac{S_n^2}{n}} \leq \mu \leq \overline{\Delta U_n} + Q(\alpha)\sqrt{\frac{S_n^2}{n}}\right] = 1 - \alpha$$

or in other words, with probability $1-\alpha$, the true incremental utility of a transformation lies within the interval $\overline{\Delta U_n} \pm Q(\alpha)\sqrt{S_n^2/n}$. The Nádas stopping rule is designed to take examples until the size of this confidence interval is twice the size of the unknown mean, μ. Formally:

$$Q(\alpha)\sqrt{\frac{S_n^2}{n}} = \mu$$

Solving this relationship for n we get the following relationship:

$$n = Q(\alpha)^2 \frac{S_n^2}{\mu^2}$$

This shows that n is the number of examples that will produce a confidence interval of the appropriate size. Or stated differently, n should be the stopping time, which is the result of Theorem 1.

It then remains to show how $Q(\alpha)$ grows as a function of $1/\alpha$. The property that $Q(\alpha)$ is bounded by a linear function is easily proven using a result known as Markov's inequality. More precisely, this shows $Q(\alpha) \leq \sqrt{2/\pi} \times 1/\alpha$. For large $1/\alpha$, $Q(\alpha)$ converges to about $\sqrt{\ln(1/\alpha)}$. This can be obtained using the asymptotic expansion of the standard normal distribution. Both derivations are given in [Gratch93d].

5.2 SAMPLE COMPLEXITY

The sample complexity is the number of examples required to perform statistical inference. COMPOSER takes some number of examples at each step of the hill-climbing search. As stated, there is no way to bound the number of steps COMPOSER will take as this is a function of the particular transformation space associated with an application. However we can characterize the expected number of examples taken at a each step in the hill-climbing search in terms of several parameters.

Within a particular hill-climbing step there is some set of transformations T. Recall that α is the allowable statistical error associated with an incremental utility estimate for each transformation in T. The value α is a bound on the probability that a transformation with negative incremental utility is perceived positive or *vice versa*. Let the value β be the allowable error at a step. This is a bound on the probability that a transformation with negative incremental is adopted at this step. The error α is related to β by the *Bound* function and by default, $\alpha = \beta/|T|$.

As validation proceeds within the step, COMPOSER consumes training examples, dynamically computing estimates for transformations in T. Eventually, either 1) all transformations are shown to have negative incremental utility and are discarded or 2) some transformation with positive incremental utility is identified and adopted. Sample complexity is the number of examples required before one of these events occurs. The following theorem describes the relationship that governs the sample complexity.

Theorem 2: Let ST^* be the stopping time associated with a step in COMPOSER's hill-climbing search under the default settings (Equations 5 and 7), where β is the acceptable error level and T is the set of transformations at that step. Then:

1) For small |T|/β the expected sample complexity is bounded by a polynomial in *T* and 1/β:

$$E[ST^*] \leq c\left(\frac{|T|}{\beta}\right)^2$$

2) For large |T|/β the expected stopping time is governed by:

$$E[ST^*] \approx c\ln\left(\frac{|T|}{\beta}\right)$$

where *c* is a constant whose value depends on the expected incremental utility and variance in incremental utility values for the transformations in *T*.

Proof: This follows directly from Theorem 1 and the default definition of $\alpha = \beta/|T|$. The constant *c* is the expected value of σ_i^2/μ_i^2 where *i* is last transformation adopted/rejected at a step.

Thus, the expected number of examples required at a step grows at most quadratically in the number of transformations at that step and grows at most quadratically in 1/δ. This means, all other things being equal, there will be an increase in the expected number of examples required at a step as we increase the number of candidates. Similarly, the sample complexity will increase polynomially as we require greater statistical confidence.

5.3 RUN-TIME COMPLEXITY

The expected run time of the algorithm depends on the number of examples used by the algorithm and the cost to process each example which may not be possible to bound in advance. Therefore, we provide results for the complexity of performing a single step. Under the brute-force method for gathering incremental utility statistics, the algorithm actually tries out each transformed problem solver in *T* over each example, so the cost of processing is tied to the complexity of the problem solver.

Theorem 3: Let R be a bound on the cost of solving a problem. Using the default settings for COMPOSER:

1) For small $|T|/\beta$ the expected run time complexity is:

$$O\left(R\frac{|T|^3}{\beta^2}\right)$$

2) For large $|T|/\beta$ the expected run time complexity is:

$$O\left[R \cdot |T| \cdot \ln\left(\frac{|T|}{\beta}\right)\right]$$

Proof: Where $|T|/\beta$ is small, from Theorem 2 the number of samples required at a step is:

$$O\left(\frac{|T|^2}{\beta^2}\right)$$

Using the default means for gathering incremental utility statistics requires solving each sample $|T|$ + 1 times. Each solution attempt can cost at most R leading to a maximum cost of $R|T|$ to process each example. The analogous argument holds for large $|T|/\beta$.

Therefore, expected cost of a hill-climbing step grows linearly in R, at most quadratically in the required confidence, $1/\delta$, and at most cubically in the the number of transformations at that step, T.

5.4 DISCUSSION

Theorem 1 has some interesting consequence for the COMPOSER's performance. A step will terminate when COMPOSER identifies a transformation with positive incremental utility or when it exhausts the set of possible transformations. If there are many transformations with positive incremental utility, COMPOSER will adopt the one that required the fewest examples. Theorem 1 states that the transformation requiring the fewest examples is not necessarily the one with the highest incremental utility, but rather the one with the highest ratio between its variance and the square of its incremental utility. Thus, COMPOSER does not necessarily perform steepest-ascent hill-climbing.

The case where every transformation in T has negative incremental utility points to a potential problem. When this occurs the step will not end until all transformations have been rejected. The last transformation to be rejected is expected to be the one with the maximum ratio of its variance to the square of its incremental utility. As the incremental utility of this transformation tends to zero, its sample complexity increases dramatically. A problem occurs if the transformation has zero incremental utility. While this may be unlikely, if such a case arises, it is not clear if the step will even terminate. The algorithm will simply exhaust all of its training examples with out making any improvements in expected utility. Depending on the transformation generator associated with the application, under such circumstances it might make sense to terminate the step early and proceed to a different step in the hill-climbing search. We discuss this possibility in Chapter 6.

6 Limitations and Future Work

COMPOSER provides a probabilistic solution to the utility problem and has demonstrated its practicality in the PRODIGY application and in two other implementations reported elsewhere [Gratch93b, Gratch93c]. These successes are encouraging, however it is important to realize that COMPOSER embodies many design commitments that restrict its generality. In this chapter we discuss these limitation and possible extensions to the approach. We first characterize some conditions on when COMPOSER will lead to successful results. We then consider limitations and extensions to three key aspects of the utility problem: organizing the search through the space of modifications, estimating utility of transformations, and gathering statistics. COMPOSER embodies a particular set of commitments for each of these aspects and thus can be seen as one point in a large space of possible commitments (see also [Gratch92b]).

6.1 APPLICABILITY CONDITIONS

COMPOSER embodies numerous restrictions in achieving the goal of efficient learning. As a result, the technique will not apply well to every situation. We can summarize four basic conditions that, if satisfied by an application, should lead to successful results.

1. Well Structured Transformation Space

A modified problem solver is constructed by composing some body of atomic modifications. In general, the space of possible composite modifications will be so large as to make exhaustive search intractable. COMPOSER requires a transformation generator that structures this space into a sequence search steps, with relatively few transformations at each step.

Clearly, COMPOSER's performance is tied to the transformations it is given. If COMPOSER is to be effective, there must exist good methods for the control points that make up a strategy. Because of the nature of hill-climbing, even if a good strategy exists, there is no guarantee that COMPOSER will find it. One may have to consider carefully how to organize the transformation space.

2. Availability of Representative Training Problems

COMPOSER's statistical approach assumes that the pattern of tasks can be represented by a fixed problem distribution. To estimate this distribution, the algorithm must be provided with a relatively large body of training problems that are representative of this distribution.

A problem arises when the distribution cycles through a set of different patterns, or if the pattern shifts slowly over time. Shifts in the distribution, cyclic or otherwise, violate the fixed distribution assumption. However there are approaches for partially overcoming these difficulties. With cyclic patterns, it may suffice to average over the cycle. One can draw problems throughout the extent of the cycle and then randomly shuffle them. Training on this shuffled distribution will result in a problem solver that does not take advantage of the shifts, but it will improve average performance. With slowly shifting distributions one can take windows of problems, train on then, and periodically re-train the problem solver as the distribution shifts. Tracking and taking advantage of predicted shifts is an important area of future research.

3. Low Problem Solving Cost

By default, COMPOSER extracts incremental utility statistics by solving each training example using various transformations. This is only feasible if problems can be solved with a sufficiently low cost in resources. This is perhaps the strongest limitation of the technique. It is not feasible to use COMPOSER to improve, for example, an average-case exponential-time problem solver. The technique is quite appropriate, however, if the problem solver is initially tractable, but highly inefficient. Potential ways of relaxing this reliance on problem solving are discussed below.

4. Domain Regularity

The motivation behind learning is that there is application-specific information to exploit. There must be restrictions in the domain of discourse or regularity in the pattern of tasks, such that a specialized problem solver can outperform a general purpose technique.

6.2 ORGANIZING SEARCH

A key complication is how to successfully navigate through the vast space of possible composite modifications. COMPOSER's hill-climbing restriction attempts to ensure efficient search. In many ways this restrictions is too strong. When there are strong interactions between transformations, COMPOSER may find poor local-maxima, or not find any solutions at all. In other ways the restriction is too weak. It says nothing about how many transformations will be considered at each step.

It is vitally important to focus the search on the most promising alternatives first. COMPOSER follows a generate and test paradigm. Performance can be improved by making generation as intelligent as possible. PRODIGY's learning component achieves some measure of intelligent generation through its EBL component. This carefully analyzes each problem solving trace for search inefficiencies and only proposes transformations that address observed deficiencies. Where applicable, heuristics like Etzioni's non-recursive hypothesis can help filter out unpromising transformations.

An interesting issue arises when transformations are continuous rather than discrete, for instance, when the problem solver has a real-valued parameter the influences its behavior. As an example, consider A* search where heuristic distance is a linear combination of two attributes. Changing the coefficients in the combination will change the heuristic measure, and presumably the behavior of the search. Such a learning problem is not well suited to COMPOSER's hill-climbing. To encode it, one would have to discretize the combination into several distinct values, and treat each as a distinct transformation. A better approach would be to attempt to approximate the functional relationship, and set the parameter accordingly. This suggest a much different control structure than simple hill-climbing search. Future work will consider such alternatives.

Finally, we are investigating how to apply notions from work on bounded rationality to help control the search [Doyle90, Horvitz89]. COMPOSER is directed towards identifying transformations with high incremental utility. The utility function determines the value of transformations, and thus the direction of this search. The efficiency of search is ensured by imposing some restrictive bias on how exploration proceeds. The actual search behavior of the algorithm arises from the interaction of the utility function and these biases. We would like to develop a more principled method for characterizing the value of investigating transformations which directly relates their incremental utility and the cost to achieve this improvement. For some results in this area see [Gratch93a, Gratch94].

6.3 ESTIMATING INCREMENTAL UTILITY

The search through the transformation space relies on the ability to accurately estimate the incremental utility of alternative transformations. This is made difficult by the distributional nature of incremental utility. Typically the exact shape of the distribution, or even its general form are unavailable and must be estimated by applying training data to some statistical model. While the number of examples required to form these estimates grows only as the log of the required confidence, reducing this cost is a significant practical concern. Another chief limitation is that the accuracy of estimates is ultimately tied to the appropriateness of the statistical model. In the case of COMPOSER, appropriateness is tied to an initial sample size parameter which may have to be adjusted for a particular domain. Reliance on such parameters is unfortunate, but the only obvious statistical alternative seems to be to use weak-method statistical models, like chernoff bounds, which result in substantially higher sample complexities. An important area of future work is the investigation of alternative stopping rules which lie between the Nádas technique and chernoff bounds.

One inefficiency in how COMPOSER gathers statistics is that it treats each transformation as an independent entity, even though there is often a relationship between transformations that would allow a more efficient use of information. As an extreme example, if two identical transformations are provided to COMPOSER, the algorithm will maintain twice the statistics necessary. Sometimes it may be possible to develop a single statistical model from which one can derive the incremental utility of multiple transformations. This is the approach taken by [Subramanian92] and [Laird92]. While not always possible, this is an important area of future research.

Several statistical methods can be applied to further improve the efficiency of the estimation process. For example, currently a transformation is eliminated only if significantly worse than the default control strategy. However, given that other transformations may be better than the default, transformations could be more quickly eliminated if they are compared with the most promising transformation rather than the default control strategy. We investigate this and other extensions to COMPOSER in [Chien94]. Similar strategies for improving the statistical inference appear in [Maron94, Moore94].

Heuristics and prior information can replace or augment statistical estimates. Syntactic measures like the *operationality criteria* of Mitchell, Keller, and Kedar-Cabelli [Mitchell86] can be seen as approximate binary measures of incremental utility. Unfortunately, syntactic measures have difficulty capturing the distributional nature of incremental utility. Instead, we are investigating the use of such measures as a bias on the estimation process. For example, COMPOSER could be modified to require less statistical confidence when transformations satisfy certain syntactic measures of utility, thereby allowing estimates to be based on fewer examples. In a related issue, we observed in the PRODIGY application that many of the same control rules considered in one hill-climbing step are also considered in subsequent hill-climbing steps. Currently COMPOSER discards all information across steps as each step is conditional on a different control strategy. However, information gained from a previous step may be useful as a bias on future estimation.

6.4 GATHERING STATISTICS

The need for efficient search and estimation arises from the fact that it can be quite expensive to gather incremental utility statistics. The default method requires solving problems to obtain incremental utility values. COMPOSER is limited to cases where it is feasible to solve problems with the original and intermediate problem solvers. While this dependance on problem solving is a serious limitation, in some applications it can be overcome or mitigated. For example, in the PRODIGY domain we were able to take advantage of properties of the transformations to process examples more efficiently. This example is just one instance of a general observations that that some transformation vocabularies may be easier to implement within the COMPOSER framework than others. Perhaps the issue can be resolved by identifying hybrid statistical/analytic means to estimate utility values.

An important area of future work is the possibility of basing incremental utility estimates on weaker and cheaper-to-obtain information. For example, Greiner and Jurisica [Greiner92a] propose one method for evaluating several transformations from a single solution attempt by maintaining upper and lower bounds on the utility of the novel search paths. Other authors have suggested that it may be possible to gain useful information about currently intractable problems by first learning from simpler problems (e.g. [Natarajan89]) or by observing a teacher (e.g. [Tadepalli91]).

7 Conclusion

This article has argued that it is desirable, and possible, to construct general problem solving techniques that automatically adapt to the characteristics of a specific application. Adaptive problem solving is a means of reconciling two seemingly contradictory needs. On the one hand, general purpose techniques can ease much of the burden of developing a application and satisfy the oft argued need for declarative and modular knowledge representation. On the other hand, general purpose approaches are ill-suited to the specialized demands of individual applications. General approaches have proven successful, only after a tedious cycle of manual experimentation and modification. Adaptive techniques promise to reduce the burden of this modification process and, thereby, take a long step toward reconciling the conflicting needs of generality and efficiency.

In this article we have developed a formal characterization of the utility problem which connects work on adaptive problem solving to the rich field of decision theory. This has been a fertile connection, giving rise to COMPOSER. COMPOSER is a statistically rigorous algorithm built upon the decision-theoretic foundation. It transforms a general problem solving technique into one specialized to a specific application. COMPOSER is still a heuristic approach, but by casting it within a statistically sound framework we are able to clearly articulate the assumptions which underly the technique and predict their consequences. Most importantly, since these assumptions are stated explicitly, they can be subjected to empirical investigations.

A larger theme that runs through this work is that any learning algorithm must strike a balance between the potential for performance improvement and pragmatic considerations such as computational efficiency. COMPOSER embodies numerous commitments to achieve efficient learning performance. We have argued that the utility problem is composed of essentially three basic and roughly independent problems. First, there is a problem of searching the space of possible composite modifications. Second, there is the issue of of obtaining estimates of local properties of transformations across the pattern of task, in our case estimating incremental utility. Finally, there is an issue of efficiently gathering the information to produce these estimates. By decomposing the problem in this way, it is possible to consider approaches like COMPOSER as not just a single algorithm, but as a collection of methods, each of which can be tested individually. It is our hope then that future research in this area will not proceed simply by the development of large techniques like COMPOSER, or PRODIGY, etc., but by the development of smaller, well understood, methods that may be combined in a variety of ways to produce a complete algorithm.

Appendix A Implementation Tradeoffs

Our motivation in designing COMPOSER was not simply to provide a statistically sound learning technique, but to provide a practical tool. A chief drawback of COMPOSER's statistical approach is that it can be expensive. This section discusses pragmatic issues and techniques for improving COMPOSER's performance by tailoring it to the specific characteristics of an application.

The principal impediment to COMPOSER's approach to the utility problem is managing the computational expense of identifying good transformations. To maximize COMPOSER's performance we would like it to efficiently process each example and to use as few examples as possible to perform its statistical inferences. More efficient use of examples allows more hill-climbing steps with the same number of examples, and a greater potential increase in expected utility. When applying COMPOSER to a particular application there are several ways in which this expense can be mitigated. This can be addressed in three ways:

1. Tailoring of the *Bound* function
2. Tailoring of the discrepancy modeling function $Q(\alpha)$
3. Tailoring of methods for gathering incremental utility statistics

This section describes the rationale behind the standard configuration of COMPOSER and describes alternative approaches. The tailorable aspects allow the application designer to take advantage of domain specific knowledge to improve learning efficiency.

A.1 TAILORING *BOUND*(δ, $|T|$)

The Bound function defines the error level for each incremental utility estimate in such a way that the overall probability of learning a worse problem solver is less than δ. Here we consider a more general definition of *Bound* that includes the current step in the hill-climbing search: *Bound*(δ, i, $|T|$). To do this, one must account for two sources of error, the error at each step in the hill-climbing search given that we are choosing a step from a set T of estimates, and the cumulative error over each step in the hill-climbing search. We will look at these sources individually. We use α to denote the error of each estimate, β_i to denote the acceptable error in step i.

A.1.1 Error in a Step

On step i we are investigating a set T of possible transformations. Given that the error of each of the estimates is α, the expected error for the step, β_i, will tend to be greater than α, This is most clearly seen in the worst case where every member of T has negative incremental utility. The correct decision for this step is to not adopt any transformation. However, there is probability α that a given transformation will be estimated to have positive incremental utility and be adopted. Typically, the larger the number of transformations, the greater the probability that at least one of the estimates will be in error. Thus, β_i is a function of the size of T. The actual relationship depends on the covariance between the distribution of incremental utility data points associated with each transformation. Unfortunately, this information is generally unavailable.

It is difficult, if not impossible, to characterize the precise relationship between the size of T and β_i. We have considered two methods for setting α to achieve a step error of β_i:

$$WORST-CASE : \alpha := \beta_i/|T| \quad \text{(default)} \tag{5}$$

$$BEST-CASE : \alpha := \beta_i \tag{6}$$

In the worst case the error can grow linearly with the size of the number of transformations, for example when every transformation has negative incremental utility and the covariances between the distributions are the worst possible. This situation will probably never arise in practice, however, it does provide a strong guarantee that the observed statistical error will not be higher than expected. The best-case model, Equation 6, assumes that the error does not grow appreciably as the size of *T* grows. We have performed some empirical evaluations that show that this assumption can be reasonable if *T* is relatively small (e.g., 30). The advantage of this assumption is that the size of *T* does not have to be known in advance so we can allow the transformation generator to add new transformations into T as we are evaluating the existing members.

A.1.2 Error Across Steps

Let β_i denote the chance of adopting a transformation with negative incremental utility on the *i*th step. As the number of steps grows, so does the chance that at least one step will result in a decrease in expected utility. However, even if some steps reduce utility, the final problem solver may still be a significant improvement. By default, we implement a liberal policy. COMPOSER allows an error of δ at each step, the rational being that it is worth one step backwards to quickly take several steps forward. The alternative, worst-case, approach is to limit the probability that COMPOSER will adopt *any* incorrect step to less than or equal to δ (i.e., $\Sigma\beta_i \le \delta$). This guarantees that COMPOSER satisfies the error requirement, but may require many more examples than the former approach. There are different ways to implement the worst-case approach depending on if we know the number of possible steps in advance. We have considered three methods for setting β_i in terms of the overall error parameter δ:

$$\mathit{LIBERAL\text{–}BOUND} : \beta_i := \delta \quad \text{(default)} \tag{7}$$

$$\mathit{WORST\text{–}CASE\text{–}BOUND/LIMITED\text{–}STEPS} : \beta_i := \delta / k \tag{8}$$

$$\mathit{WORST\text{–}CASE\text{–}BOUND/UNLIMITED\text{–}STEPS} : \beta_i := \frac{6\delta}{i^2\pi^2} \tag{9}$$

The default policy, Equation 7, relies on the assumption that the magnitude of the incremental utility of incorrect steps is comparable to the magnitude of the incremental utility of correct steps so that even if some steps reduce utility, the final result will tend to improve on the initial problem solver. This assumption has held across several simulation experiments. The later two equations are useful when efficiency must be sacrificed for rigor. When the number of steps can be limited in advance, one can simply divide the error evenly over each of the *k* steps (Equation 8). When the number of steps is unbounded in advance, the error at each step must be such that no matter what the final number of steps, the total error sums to less than δ. Equation 9 satisfies this requirement.[11]

11. This equation was suggested to us by Russell Greiner. and is the basis for his PALO algorithm (see [Greiner92a]).

Once the application implementor chooses a model for the error within a step and a model for the error across steps, these should be unified into a overall function *Bound*(δ, i, $|T|$) which combines the two choices into a error level for each estimate for the *i*th step in the hill-climbing search.

A.2 TAILORING $Q(\alpha)$ AND n_0

The function $Q(\alpha)$ models the normalized expected discrepancy between the estimated incremental utility and the true incremental utility. Here we consider a more general definition, $Q(\alpha, n)$. The estimate is the average of a finite sample of n data points. This sample will be roughly representative of the actual distribution of data points as it is drawn randomly from the fixed distribution. However this sample will be more or less representative depending on random chance, and so the mean of the sample will be more or less close to the true mean of the distribution. The Nádas stopping rule bounds the *normalized* difference, d, between the sample mean and the true mean. The normalized difference is the difference divided by the observed variance in the sample mean. The expected normalized difference can be modeled by a probability distribution function which shows the likelihood that a particular normalized difference arises from a random sample. Such a distribution function is illustrated in Figure 7. The bell shaped curve shows the likelihood of observing different normalized differences. The function $Q(n, \alpha)$, also called the $\alpha/2$th *quantile* of this distribution, is the positive difference d such that the probability of observing a difference greater than this is less than or equal to $\alpha/2$.

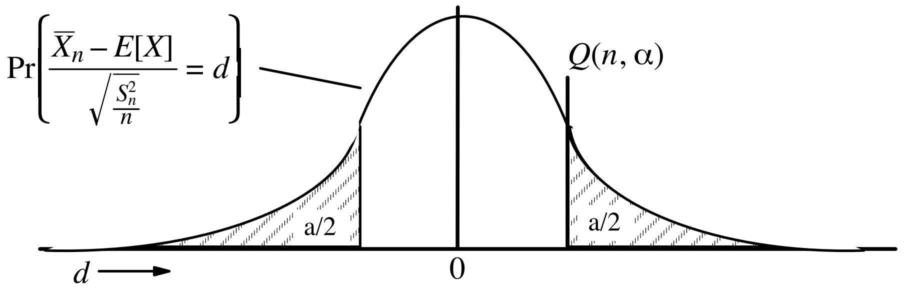

$\Pr\left\{\frac{\overline{X}_n - E[X]}{\sqrt{\frac{S_n^2}{n}}} = d\right\}$
$Q(n, \alpha)$
a/2
a/2
d
0

Figure 7: Probability distribution of the normalized difference between the sample mean and true mean of the original distribution. $Q(n, \alpha)$ is the value such that the probability of achieving a distance greater than this is less than or equal to $\alpha/2$.

It is rarely possible to know the exact distribution of differences for a given learning situation. Fortunately, no matter what the underlying distribution, as n increases, the distribution of differences converges to a normal distribution with zero mean and unit variance (also called a *standard normal distribution*). This property is asserted by the *central limit theorem* in statistics. This fact implies, under some weak conditions,[12] that function $Q(n, \alpha)$ can be approximated satisfactorily using the quantile of a standard normal distribution. The approximation improves as the sample size increases. This

12. The distribution must have positive variance and hence finite mean.

is the motivation behind the n_0 parameter. Taking a sufficiently large initial sample of data points ensures an accurate approximation. We have investigated two approximation methods for $Q(n, \alpha)$:

$$STANDARD-NORMAL : Q_\Phi(n,\alpha) = x \text{ such that } \int_x^\infty \left(1/\sqrt{2\pi}\right) e^{-0.5y^2} dy = \frac{\alpha}{2} \quad \text{(default)} \tag{10}$$

$$T : Q_t(n,\alpha) = x \text{ such that } \int_x^\infty \frac{\Gamma[(n+1)/2]}{\sqrt{\pi n}\,\Gamma(n/2)(1+y^2/r)^{(n+1)/2}} dy = \frac{\alpha}{2} \tag{11}$$

The first, COMPOSER's default, is based on the standard normal distribution model. The second is based on a model called the *student t* distribution. This second model is accurate when there is high variance in the sample, but it is more expensive to compute. For a given learning situation, the function $Q(n, \alpha)$ and n_0 should be chosen to best model the expected discrepancy in the given learning situation. If an exact model can be determined then an initial sample size is unnecessary. In general, higher variance in incremental utility values requires a greater n_0 ensure the approximation model is a close approximation. Smaller values for δ require more precise modeling of the error and therefore a better approximation. In general, the smaller the requested error level, the greater the n_0 should be to ensure a close approximation to δ. If n_0 is set too small, the likely result is a higher-than-requested statistical error. If n_0 is too large, an excessive number of examples is required to perform statistical inference. The optimal setting for n_o is difficult to determine, however the general experience in the statistical community is that the normal approximation becomes quite good after only a few initial samples. We recommend a value around fifteen.

A.3 GATHERING STATISTICS

Given a current problem solver *PS* and a set of transformations *T*, COMPOSER must infer from examples the likely difference in utility between the current problem solver and each of the transformed problem solvers. For many application there are natural ways to both reduce the number of utility values necessary to make inferences, and to reduce to cost of obtaining each utility value.

As was shown in the theoretical analysis, the number of examples needed to make statistical inferences grows with the variance in the incremental utility values. Using a sampling technique called *blocking* it is often possible to minimize this variance [Büringer80]. To understand blocking, consider the problem of finding the highest yielding variety of wheat. Wheat yield is effected by several factors including the factor of interest, the variety of wheat, and other *nuisance factors* (e.g., the weather conditions in the year the crop was grown). Often these nuisance factors have the greatest influence, washing out the contribution of the factor of interest, and thus increasing the variance in the data. A standard solution, called *randomized block design* is to combine all data with identical values on their nuisance factors into a single block, and only consider the differences in the observations within the block when computing utility values.

In COMPOSER, the nuisance factors are the specific characteristics of each problem drawn from the task distribution – some problems are easy, others hard, and these differences are likely to overwhelm the differences due to the choice of transformation. The solution we have by default adopted is to block transformations by problem. We take each problem (the block) and observe the behavior of each possible transformation on that problem. Incremental utility values are then derived by subtracting the utility of the default strategy from the utility of each transformation in turn, for that block. When the problem influences are dominant this procedure can lead to a significant reduction in the number of examples needed for statistical inference. These problem influences tend to dominate in many of the intended applications as transformations generally make relatively small incremental changes to the current problem solver, and therefore each transformed problem solver will perform similarly on similar problems. The alternative to blocking is to compute the incremental utility where each utility value is derived from a different problem. In some situations it may be necessary to perform this strategy as it may not be possible to repeatedly solve the identical problem. Blocking will also not help much if problem differences are small relative to the effect of the transformations.

In some applications it may also be possible to reduce the cost of obtaining utility data. The simple strategy we recommend is to actually solve a given problem with each of the candidate problem solvers. The complexity of this brute-force processing an example is therefore tied to the complexity of each of the $|T|$ problem solvers. In some learning situations brute-force processing may prove too expensive. For example, one or more of the candidate problem solvers may be intractable. Furthermore, the brute-force force method is *intrusive* – it requires explicit experimentation with alternative problem solvers. In some learning situations is is desirable to learn passively, through the normal operations of the problem solver. As gathering statistics is COMPOSER's principal expense, it is important to take into account any information that could reduce this cost.

Learning cost can be dramatically reduced if there is a detailed cost model of the problem solver that efficiently derives the ramification of proposed transformations without actually solving the problem (e.g. [Greiner89, Subramanian90]). With such a model we could simply draw a random training example and then use the model to determine the effect of different transformations. Such models are rarely available. Short of this, it may be possible to extract the necessary statistics to determine the effectiveness of different transformations by solely by observing the normal operations of the current problem solver. In Section 4 we describe one such *unobtrusive* implementation. Sometimes it is only possible to extract partial information in this way. Greiner and Jurisica [Greiner92a] propose one method for using such partial information that does not conflict with COMPOSER's assumptions and could incorporated.

Appendix B BIN-WORLD Domain

This domain, introduced in [Gratch91b], highlights deficiencies in PRODIGY/EBL's utility analysis and Etzioni's non-recursive hypothesis. The domain is a robot assembly task where the goal is to construct a composite part from a set of components. All the components for a particular part are stored in a bin. If all the components in the bin are free of defects, the part may be assembled. Otherwise another bin must be examined for acceptability. The INSPECT–BIN operator determines if a given bin is suitable for assembly. The ASSEMBLE–COMPONENTS operator constructs the part.

B.1 DOMAIN THEORY

```
(ASSEMBLE–COMPONENTS (<BIN>)
  (preconds (exists (<BIN>) (parts–bin <BIN>) (defect–free–components <BIN>)))
  (effects ((add (TOP)))))

(INSPECT–BIN (<BIN>)
  (preconds (forall (<MEMBER>) (in–bin <MEMBER> <BIN>)
                              (GOOD <BIN> <MEMBER>)))
  (effects ((add (defect–free–components <BIN>)))))
```

B.2 PROBLEM DISTRIBUTION

A problem distribution is defined by enumerating a set of problem classes and assigning probabilities to each class. A set of problems is created by randomly constructing problems according to the distribution. The experiment is based on a uniform distribution over two problem classes. This means that each class has an equal chance of participating in a problem solving attempt. The first class contains problems with fifty bins of two components each. Forty-nine bins contain a defective component. One bin contains no defects. The bins are ordered with the defect-free bin last. The second class contains problems with two bins of two hundred components each. One bin is defect–free. The other bin contains a defective component. The components are ordered with the defective component last. The bins are also ordered with the defect–free bin last.

The rational behind this problem distribution is to construct a distribution with high variance. PRODIGY/EBL bases its utility estimates on a single example. Estimating utility of a bi-modal distribution (or any distribution with high variance) from a single example results in an inaccurate representation of the true incremental utility of any learned control rule.

References

[Berger80] J. O. Berger, *Statistical Decision Theory and Bayesian Analysis*, Springer Verlag, 1980.

[Borgida89] A. Borgida and D. W. Etherington, "Hierarchical Knowledge Bases and Efficient Disjunctive Reasoning," *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, Toronto, Ontario, Canada, May 1989, pp. 33–43.

[Büringer80] H. Büringer, H. Martin, and K. –H. Scriever, *Nonparametric Sequential Selection Procedures*, Birkhäuser Publishers, Boston, 1980.

[Bylander92] T. Bylander, "Complexity Results for Extended Planning," *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, College Park, Maryland, June 1992, pp. 20–27.

[Chapman87] D. Chapman, "Planning for Conjunctive Goals," *Artificial Intelligence 32*, 3 (1987), pp. 333–378.

[Cheeseman89] P. Cheeseman, B. Kanefsky and W. M. Taylor, "Where the Really Hard Problems Are," *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, Sidney, Australia, August 1989, pp. 163–169.

[Chien94] S. A. Chien, J. M. Gratch, and M. C. Burl, "On the Efficient Allocation of Resources for Hypothesis Evaluation in Machine Learning: A Statistical Approach," Technical Report UIUC–BI–AI–94–01, Beckman Institute, University of Illinois at Urbana-Champaign, January 1994.

[Dean91] T. L. Dean and M. P. Wellman, *Planning and Control*, Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[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.)

[Dean88] T. Dean and M. Boddy, "An Analysis of Time–Dependent Planning," *Proceedings of The Seventh National Conference on Artificial Intelligence*, Saint Paul, MN, August 1988, pp. 49–54.

[Dechter87] R. Dechter and J. Pearl, "Network–Based Heuristics for Constraint–Satisfaction Problems," *Artificial Intelligence 34*, 1 (December 1987), pp. 1–38.

[Dechter92] R. Dechter, "Constraint Networks,"in *Encyclopedia of Artificial Intelligence*, Stuart C Shapiro (ed.), John Wiley and Sons, Inc., 1992.

[DeGroot70] M. H. DeGroot, *Optimal Statistical Decisions*, McGrawHill Book Co., New York, 1970.

[Doyle90] J. Doyle, "Rationality and its Roles in Reasoning (extended version)," *Proceedings of the National Conference on Artificial Intelligence*, Boston, MA, 1990, pp. 1093–1100.

[Eorl92] K. Eorl, D. S. Nau and V. S. Subrahmanian, "On the Complexity of Domain–Independent Planning," *Proceedings of the National Conference on Artificial Intelligence*, San Jose, CA, July 1992, pp. 381–386.

[Etzioni90a] O. Etzioni, "A Structural Theory of Search Control," Ph.D. Thesis, Department of Computer Science, Carnegie–Mellon University, Pittsburgh, PA, 1990.

[Etzioni90b] O. Etzioni, "Why Prodigy/EBL Works," *Proceedings of the National Conference on Artificial Intelligence*, Boston, MA, August 1990, pp. 916–922.

[Etzioni91] O. Etzioni, "STATIC a Problem–Space Compiler for PRODIGY," *Proceedings of the National Conference on Artificial Intelligence*, Anaheim, CA, July 1991, pp. 533–540.

[Etzioni92] O. Etzioni and S. Minton, "Why EBL Produces Overly–Specific Knowledge: a Critique of the PRODIGY Approaches," *Proceedings of the Ninth International Conference on Machine Learning*, Aberdeen, Scotland, July 1992, pp. 137–143.

[Fikes71] R. E. Fikes and N. J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence 2*, 3/4 (1971), pp. 189–208.

[Fisher81] M. Fisher, "The Lagrangian Relaxation Method for Solving Integer Programming Problems," *Management Science 27*, 1 (1981), pp. 1–18.

[Freuder82] E. C. Freuder, "A Sufficient Condition for Backtrack–Free Search," *J. Association for Computing Machinery 29*, 1 (January 1982), pp. 24–32.

[Goldberg82] A. Goldberg, P. W. Purdom and C. A. Brown, "Average time analysis of simplified Davis–Putnam procedures," *Information Process. Lett. 15*, (1982), pp. 72–75.

[Govindarajulu81] Z. Govindarajulu, *The Sequential Statistical Analysis*, American Sciences Press, INC., Columbus, OH, 1981.

[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.

[Gratch92a] J. Gratch and G. DeJong, "COMPOSER: A Probabilistic Solution to the Utility Problem in Speed–up Learning," *Proceedings of the National Conference on Artificial Intelligence*, San Jose, CA, July 1992, pp. 235–240.

[Gratch92b] J. Gratch and G. DeJong, "A Framework of Simplifications in Learning to Plan," *First International Conference on Artificial Intelligence Planning Systems*, College Park, MD, 1992, pp. 78–87.

[Gratch93a] J. Gratch and G. DeJong, "Rational Learning: A Principled Approach to Balancing Learning and Action," Technical Report UIUCDCS–R–93–1801, Department of Computer Science, University of Illinois, Urbana, IL, April 1993.

[Gratch93b] J. Gratch and S. Chien, "Learning Search Control Knowledge for the Deep Space Network Scheduling Problem," *Proceedings of the Tenth International Conference onf Machine Learning,* Amherst, MA, July 1993.

[Gratch93c] J. Gratch, S, Chien, and G, DeJong, "Learning Search Control Knowledge to Improve Schedule Quality," IJCAI93 scheduling workshop.

[Gratch93d] J. M. Gratch, "COMPOSER: A Decision-Theoretic Approach to Adaptive Problem Solving," Technical Report UIUCDCS–R–93–1806, Department of Computer Science, University of Illinois, Urbana, IL, 1993.

[Gratch94] J. Gratch, S. Chien, and G. DeJong, "Improving Learning Performance Through Rational Resource Allocation," *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, WA, August 1994.

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

[Greiner92a] R. Greiner and I. Jurisica, "A Statistical Approach to Solving the EBL Utility Problem," *Proceedings of the National Conference on Artificial Intelligence*, San Jose, CA, July 1992, pp. 241–248.

[Greiner92b] R. Greiner and W. W. Cohen, "Probabilistic Hill–Climbing," *Proceedings of Computational Learning Theory and 'Natural' Learning Systems*, 1992 (to appear).

[Hogg78] R. V. Hogg and A. T. Craig, *Introduction to Mathematical Statistics*, Macmillan Publishing Co., Inc., London, 1978.

[Holder92] L. B. Holder, "Empirical Analysis of the General Utility Problem in Machine Learning," *Proceedings of the National Conference on Artificial Intelligence,* San Jose, CA, July 1992, pp. 249–254.

[Horvitz89] E. J. Horvitz, G. F. Cooper and D. E. Heckerman, "Reflection and Action Under Scarce Resources: Theoretical Principles and Empirical Study," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989, pp. 1121–1127.

[Knoblock89] C. Knoblock, "Learning Hierarchies of Abstraction Spaces," *Proceedings of the Sixth International Workshop on Machine Learning*, Ithaca, NY, 1989, pp. 241–245.

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

[Laird86] J. E. Laird, P. S. Rosenbloom and A. Newell, *Universal Subgoaling and Chunking: The Automatic Generation and Learning of Goal Hierarchies*, Kluwer Academic Publishers, Hingham, MA, 1986.

[Laird92] P. Laird, "Dynamic Optimization," *Proceedings of the Ninth International Conference on Machine Learning*, Aberdeen, Scotland, July 1992, pp. 263–272.

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

[Lewins93] N. J. Lewins, "Practical Solution-caching for PROLOG: An Explanation-based Learning Approach, Ph.D., Thesis, Department of Computer Science, University of Western Australia, July 1993.

[Maron94] O. Maron and A W. Moore, "Hoeffding Races: Accelerating Model Selection Search for Classification and Function Approximation," In *Advances in Neural Information Processing Systems 6,* Morgan Kaufmann, April 1994.

[Melhorn84] K. Melhorn, *Data Structures and Algorithms 1: Sorting and Searching*, Springer Verlag, 1984.

[Miller92] D. P. Miller, R. S. Desai, E. Gat, R. Ivlev and J. Loch, "Reactive Navigation through Rough Terrain: Experimental Results," *Proceedings of the National Conference on Artificial Intelligence*, San Jose, CA, July 1992, pp. 823–828.

[Minton88] S. Minton, in *Learning Search Control Knowledge: An Explanation–Based Approach*, Kluwer Academic Publishers, Norwell, MA, 1988.

[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.

[Mitchell92] D. Mitchell, B. Selman and H. Levesque, "Hard and Easy Distributions of SAT Problems," *Proceedings of the National Conference on Artificial Intelligence*, San Jose, CA, July 1992, pp. 459–465.

[Moore94] A. W. Moore and M. S. Lee, "Efficient Algorithms for Minimizing Cross Validation Error," *Proceedings of the Eleventh International Conference on Machine Learning,* July 1994.

[Mostow81] J. Mostow, "Mechanical Transformation of Task Heuristics into Operational Procedures," Ph.D. Thesis, Department of Computer Science, CMU, Pittsburgh, PA, 1981.

[Nádas69] A. Nádas, "An extension of a theorem of Chow and Robbins on sequential confidence intervals for the mean," *The Annals of Mathematical Statistics 40*, 2 (1969), pp. 667–671.

[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.

[Pérez92] M. A. Pérez and O. Etzioni, "DYNAMIC: a new rule for trining problems in EBL," *Proceedings of the Ninth International Conference on Machine Learning*, Aberdeen, Scotland, July 1992, pp. 367–372.

[Roy71] B. Roy, "Problems and Methods with Multiple Objective Functions," *Mathematical Programming*, Vol. 1, No. 2, 1971.

[Russell89] S. Russell and E. Wefald, "Principles of Metareasoning," *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, Toronto, Ontario, Canada, May 1989, pp. 400–411.

[Schoppers92] M. Schoppers, "Building Plans to Monitor and Expoint Open–Loop and Closed–Loop Dynamics," *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, College Park, Maryland, June 1992, pp. 204–213.

[Schwuttke92] U. M. Schwuttke and L. Gasser, "Real–time Metareasoning with Dynamic Trade–off Evaluation," *Proceedings of the National Conference on Artificial Intelligence*, San Jose, CA, July 1992, pp. 500–506.

[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.

[Subramanian92] D. Subramanian and S. Hunter, "Measuring Utility and the Design of Provably Good EBL Algorithms," *Proceedings of the Ninth International Conference on Machine Learning*, Aberdeen, Scotland, July 1992, pp. 426–435.

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

[Wellman92] M. P. Wellman and J. Doyle, "Modular Utility Representation for Decision–Theoretic Planning," *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, College Park, Maryland, June 1992, pp. 236–242.