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.

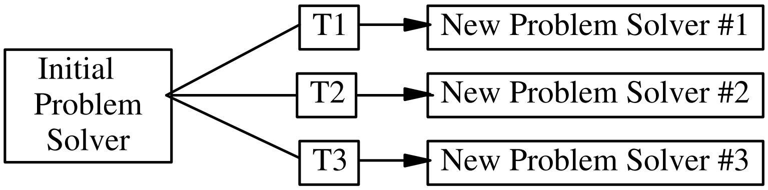

Figure 1: Learning serves to transform an initial problem solver into a transformed one. To accomplish this the learner must choose one of a set of possible transformations.

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:

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

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