Probabilistic Hill-Climbing:
Theory and Applications

Russell Greiner*
Siemens Corporate Research, Princeton, NJ 08540
greiner@learning.siemens.com (609) 734-3027

Abstract
Many learning systems search through a space of possible performance elements, seeking an element with high expected utility. As the task of finding the globally optimal element is usually intractable, many practical learning systems use hill-climbing to find a local optimum. Unfortunately, even this is difficult, as it depends on the distribution of problems, which is typically unknown. This paper addresses the task of approximating this hill-climbing search when the utility function can only be estimated by sampling. We present an algorithm that returns an element that is, with provably high probability, essentially a local optimum. We then demonstrate the generality of this algorithm by sketching three meaningful applications, that respectively find an element whose efficiency, accuracy or completeness is nearly optimal. These results suggest approaches to solving the utility problem from explanation-based learning, the multiple extension problem from nonmonotonic reasoning and the tractability/completeness tradeoff problem from knowledge representation.

1 Introduction
Many learning tasks can be viewed as a search through a space of possible performance elements seeking an element that is optimal, based on some utility measure. As examples, many inductive systems seek a function whose classification is optimal, i.e., which labels correctly as many examples as possible; and many explanation-based learning [DeJ88, MCK+89] and chunking [LNR87] systems seek a problem solving system that is optimally efficient [Min88, Gre89]. In each of these cases, the utility function used to compare the different elements is defined as the expected value of a particular scoring function, averaged over the distribution of samples (or goals, queries, problems, ... ) that will be seen [Hau88, OG90, GO91].

There are two problems with implementing such a learning system. First, we need to know the distribution of samples to determine which element is optimal; unfortunately, this information is usually unknown. There are, of course, standard statistical techniques that use a set of observed samples to estimate the needed information; and several classes of learning systems have incorporated these techniques. For example, many “PAC-learning” systems [Val84] use these estimates to identify an element that is approximately a global optimum.

This leads to the second problem: unfortunately, the task of identifying the globally optimal element, even given the correct distribution information, is intractable for many spaces of elements [Gre89, Hau88]. A common response is to build a system that hill-climbs towards a local optimum. Many well-known inductive learning systems, including BACKPROP [Hin89] and ID3 [Qui86], use this approach, as do many speedup learning methods; see especially [GD91]. Unfortunately, few existing systems guarantee that each hill-climbing step is even an improvement, meaning the final element is not always even superior to the initial one, much less an optimum in the space of elements. Moreover, fewer systems include a stopping criterion to determine when the learning has reached a point of diminishing returns.

The work presented here draws ideas from both of these themes: In particular, it describes a general learning algorithm, PALO, that hill-climbs to a local optimum, using a utility metric that is estimated by sampling. Given any parameters $\epsilon, \delta > 0$, PALO efficiently produces an element whose expected utility is, with probability greater than $1 - \delta$, an $\epsilon$-local optimal. Moreover, PALO can work unobtrusively [MMS85], passively gathering the statistics it needs by simply watching a performance element solve problems relevant to a user’s applications. Here, the incremental cost of PALO’s hill-climbing, over the cost of simply solving performance problems, can be very minor.

*Most of this work was performed at the University of Toronto, where it was supported by the Institute for Robotics and Intelligent Systems and by an operating grant from the National Science and Engineering Research Council of Canada. I also gratefully acknowledge receiving many helpful comments from William Cohen, Dale Schuurmans and the anonymous referees.

1Theorem 1 below defines both our sense of efficiency, and “$\epsilon$-local optimality”.
Section 2 motivates the use of “expected utility” as a quality metric for comparing performance elements. Section 3 then describes a statistical tool for evaluating whether the result of a proposed modification is better (with respect to this metric) than the original performance element PE; this tool can be viewed as a mathematically rigorous version of [Min88]’s “utility analysis”. We use this tool to define the general PALO algorithm, that incrementally produces a series of performance elements PE1, . . . , PEM such that each PEi+1 is statistically likely to be an incremental improvement over PEi and, with high confidence, the performance of the final PEM is a local optimal in the space searched by the learner. Section 4 demonstrates the generality of this approach by presenting three different instantiations of the PALO system, each using its own set of transformations to find a near-optimal element within various sets of performance elements, where optimality is defined in terms of efficiency, accuracy, or completeness, respectively.

2 Framework

We assume as given a (possibly infinite) set of performance elements PE = {PEi}, where each PEi ∈ PE is a system that returns an answer to each given problem (or query or goal, etc.) qi ∈ Q, where Q = {q1, q2, . . .} is the set of all possible queries. We also use the utility function c: PE × Q → R, where c(PE, q) measures how well the element PE does at solving the problem q. (Section 4 defines cε(PE, q), which quantifies the time PE requires to solve q; cε(PE, q), the accuracy of PE’s answer; and c(PE, q), PE’s categoricity.)

This utility function specifies which PEi is best for a single problem. Our performance elements, however, will have to solve an entire ensemble of problems. As we obviously prefer the element that is best overall, we must therefore consider the distribution of problems that our performance elements will encounter. We model this using a probability function, Pr: Q→[0, 1], where Pr(q) denotes the probability that the problem qi is selected.3

We then define the expected utility of a performance element:

\[ C(PE) = E[c(PE, q)] = \sum_{q \in Q} Pr(q) \times c(PE, q) \] (1)

Our underlying challenge is to find the performance element whose expected utility is maximal. As mentioned above, there are two problems: First, the problem distribution, needed to determine which element is optimal, is usually unknown. Second, even if we knew that distribution information, the task of identifying the optimal element is often intractable.

3 The PALO Algorithm

This section presents a learning system, PALO (for “Probably Approximately Locally Optimal”) that sidesteps the above problems by using a set of sample queries
to estimate the distribution, and by hill-climbing efficiently from a given initial PE0 to one that is, with high probability, essentially a local optimum. This section first states the fundamental theorem that specifies PALO’s functionality, then summarizes PALO’s code and sketches a proof of the theorem.

In more detail, PALO takes as arguments an initial PE0 and parameters ε, δ > 0. It uses a set of test problems drawn at random from the Pr[.] distribution to climb from the initial PE0 to a final PEM, using a particular set of possible transformations T = {τj}, where each τj maps one performance element to another; see Section 4. PALO then returns this final PEM. Theorem 1 states our main theoretical results.

**Theorem 1** The PALO(PE0, ε, δ) process incrementally produces a series of performance elements PE0, PE1, . . . , PEM, staying at a particular PEj for only a polynomial number of samples before either climbing to PEj+1 or terminating. With probability at least 1 − δ, PALO will terminate. It then returns an element PEM whose expected utility C(PEm) is, with probability at least 1 − δ, both

1. at least as good as the original PE0; i.e.,
   \[ C(PE_m) \geq C(PE_0); \] and

2. an ε-local optimum — i.e.,
   \[ \forall \tau_j \in T. C(PE_m) \geq C(\tau_j(PE_m)) - \epsilon \]

The basic code for PALO appears in Figure 1. In essence, PALO will climb from PEj to a new PEj+1 if PEj+1 is likely to be better than PEj; i.e., if we are highly confident that C[PEj+1] > C[PEj]. To determine this, define

\[ d_i = \Delta[PE_\alpha, PE_\beta, q_i] \overset{def}{=} c(PE_\alpha, q_i) - c(PE_\beta, q_i) \]

3This proof, and others, appear in the expanded version of this paper [Gre92].
to be the difference in cost between using $PE_\alpha$ to deal with the problem $q_i$, and using $PE_\beta$. As each query $q_i$ is selected randomly according to some fixed distribution, these $d_i$s are independent, identically distributed random variables whose common mean is $\mu = C[PE_\alpha] - C[PE_\beta]$. (Notice $PE_\alpha$ is better than $PE_\beta$ if $\mu > 0$.)

Let $Y_n \overset{\text{def}}{=} \frac{1}{n} \sum_{i=1}^{n} d_i$ be the sample mean over $n$ samples, where $\Delta[PE_\alpha, PE_\beta; S] \overset{\text{def}}{=} \sum_{q \in S} c(PE_\alpha, q) - c(PE_\beta, q)$ for any set of queries $S$. This average tends to the true population mean $\mu$ as $n \to \infty$; i.e., $\mu = \lim_{n \to \infty} Y_n$. Chernoff bounds [Che52] describe the probable rate of convergence: the probability that $Y_n$ is more than $\mu + \gamma$ goes to 0 exponentially fast as $n$ increases; and, for a fixed $n$, exponentially as $\gamma$ increases. Formally,

$$Pr[Y_n > \mu + \gamma] \leq e^{-2n(\mu + \gamma)^2}$$

$$Pr[Y_n < \mu - \gamma] \leq e^{-2n(\mu - \gamma)^2}$$

where $\Delta$ is the range of possible values of $c(PE_\alpha, q_i) - c(PE_\beta, q_i)$. This $\Delta = \Delta[PE_\alpha, PE_\beta]$ is also used in both the specification of $\Lambda_{\max}$ under Line L1 and in Equation 2.

The PALO algorithm uses these equations and the values of $\Delta[PE, PE_\beta; S]$ to determine both how confident we should be that $C[PE'] > C[PE_\beta]$ (Equation 2) and whether any “$T$-neighbor” of $PE_\beta$ (i.e., any $\tau_t(PE_\beta)$) is more than $\epsilon$ better than $PE_\beta$ (Equation 3).

We close this section with some general comments on the PALO framework and algorithm.

N-PALO1. The samples that PALO uses may be produced by a user of the performance system, who is simply asking questions relevant to his current applications; here, PALO is unobtrusively gathering statistics as the user is solving his own problems [MMS85]. This means that the total cost of the overall system, that both solves performance problems and “learns” by hill-climbing to successive performance elements, can be only marginally more than the cost of only running the performance element to simply solve the performance problems.

We are using these user-provided samples as our objective is to approximate the average utility values of the elements, over the distribution of problems that the performance element will actually address. This “average case analysis” differs from several other approaches as, for example, we do not assume that this distribution of problems will be uniform [Gol79], nor that it will necessarily correspond to any particular collection of “benchmark challenge problems” [Kel87].

N-PALO2. All three $c_i(PE, q)$ functions discussed in this paper are “bounded” i.e., satisfy

$$\forall PE \in \mathcal{P}E, q \in \mathcal{Q}, c_t \leq c_a(PE, q) \leq c_t + \lambda$$

for some constants $c_t \in \mathbb{R}$ and $\lambda \in \mathbb{R}^+$. Here, we can guarantee that $\Delta[PE, PE_\beta] \leq \lambda$. For certain transformations $\tau_t$, we can find yet smaller values for $\Delta[\tau_t(PE), PE]$; see [GJ92].

N-PALO3. Although Theorem 1 bounds the number of samples per iteration, it is impossible to bound the number of iterations of the overall PALO algorithm without making additional assumptions about the search space defined by the $T$ transformations. The theorem’s guarantee that PALO will terminate with probability at least $1 - \delta$ requires that the space of performance elements be finite; this is true in all three situations considered in this paper.

N-PALO4. Notice that a “0-local optimum” corresponds exactly to the standard notion of local optimum; hence our “$i$-local optimum” generalizes local optimality. Notice that PALO’s output, $PE_m$, will (probably) be a real local optimum if the difference in cost between every two distinct performance elements, $PE$ and $\tau_t(PE)$, is always larger than $\epsilon$. Thus, for sufficiently small values of $\epsilon$, PALO will always produce a bona fide local optimum.

N-PALO5. We can view PALO as a variant on anytime algorithms [BD88, DB88] as, at any time, PALO provides a usable result (here, the performance element produced at the $j^{th}$ iteration, $PE_j$), with the property that later systems are (probably) better than earlier ones; i.e., $i > j$ means $C[PE_i] > C[PE_j]$ with high probability. PALO differs from standard anytime algorithms by terminating on reaching a point of diminishing returns.

Notice finally that PALO will (probably) process more samples using later elements than using the earlier ones, as its tests (Equations 2 and 3) are increasingly more difficult to pass. This behaviour is desirable, as it means that the overall system is dealing with increasing numbers of samples using later, and therefore better, elements.

4 Instantiations of the PALO Algorithm

This section demonstrates the generality of the PALO algorithm by presenting three different instantiations of this framework. For each instantiation, we specify (1) the set of possible performance elements $\mathcal{P}E = \{PE_i\}$, (2) the set of transformations $T$ used in the hill-climbing process, and (3) the scoring function $c(\cdot, \cdot)$ used to specify the expected utility. We will also discuss how to obtain the values of $\Lambda[\tau_t(PE), PE]$. (The instantiations of these parameters are also summarized in Table 1.) For pedagogical reasons, each subsection begins with a quick simplistic description of the application, and then provides notes that describe how to build a more comprehensive system.

4.1 Improving Efficiency

Many derivation processes can be viewed as a satisficing search [SK75] through a given graph structure. As an example, notice that using the information shown in Figure 2 to find an answer to the hop($\kappa$) query, for some ground individual $\kappa$, corresponds naturally to a search
through the $G_A$ inference graph (formed from a given set of rules) seeking a successful database retrieval. A strategy specifies the order in which to perform the various rule-based reductions (e.g., the $a_1$ arc reduces the $N_0$: hep($\chi$) goal to the $N_1$: jaun($\chi$) subgoal, based on the rule $R_1$) and the database retrievals (e.g., the $a_2$ arc from $N_1$ to $N_2$ corresponds to the attempted database retrieval jaun($\chi$)). We can express each strategy as a sequence of $G_A$’s arcs; e.g., the strategy

$\Theta_0 = \langle a_1, a_2, a_3, a_4, a_5, a_6, a_7 \rangle$

corresponds to the obvious depth-first left-to-right traversal, with the understanding that the performance element using this strategy will stop whenever it reaches a “success node” (e.g., if the $a_2$ retrieval succeeds, then $\Theta_0$ reaches the success node $N_2$ and stops with success), or has exhausted all of its reductions. (Figure 2 doubly-boxes $G_A$’s success nodes, $N_2, N_5$ and $N_7$.) There are many other possible strategies, including

$\Theta_1 = \langle a_3, a_4, a_5, a_6, a_7, a_1, a_2 \rangle$,

as well as non-depth-first strategies, etc.

Each strategy will find an answer, if one exists. As this is a satisficing search, all answers are equally acceptable [SK75], which means that all strategies are equally accurate. We can therefore consider the costs of the strategies, preferring the one whose expected cost is minimal.

Letting $f_i \in \mathbb{R}^+$ be the positive cost of traversing the $a_i$, we can compute the $c_i(\Theta, q)$, the cost of using strategy $\Theta$ to find an answer to the query $q$. For example,

$c_i(\Theta_0, \text{hep}(b_2)) = f_1 + f_2$, \quad $c_i(\Theta_0, \text{hep}(b_1)) = f_1 + f_2 + f_3 + f_4 + f_5$, and $c_i(\Theta_1, \text{hep}(b_1)) = f_3 + f_4 + f_5$. (These different strategies have different costs for a given query as each stops as soon as it finds an answer.) The expected cost, of course, depends on the distribution of queries; i.e., on how often the query will be hep(b_1), hep(b_2), etc. Moreover, the task of finding the globally optimal strategy is NP-hard [Gre91].

5Here, hep($\chi$) means $\chi$ has hepatitis, jaun($\chi$) means $\chi$ is jaundiced, and badB($\chi$) means $\chi$ has “bad blood” hep(b_1) means $\chi$ tests positive for blood test #1.

This looks like a job for PALO.6 We first define the set of reordering transformations $T^{\text{RO}} = \{\tau_{a_1,a_2}\}$, where each $\tau_{a_1,a_2}$ maps one strategy to another by moving the subgraph under the $a_1$ arc before $a_2$ and its subgraph. For example, $\tau_{a_3,a_4}(\Theta_0) = \Theta_1$, and $\tau_{a_4,a_5}(\Theta_0) = \langle a_3, a_5, a_5, a_7, a_2, a_4 \rangle$. PALO also needs to compute $A[\tau(\Theta_0), \Theta_1]$; these values are bounded by $\mathcal{C}(G) = \sum_i f_i$, the sum of the costs of all of the arcs in the inference graph $G$; see Note 2 below.

N-Effect 1. This class of performance elements corresponds to many standard problem solvers, including PROLOG [CM81]; see also [GN87]. We can also use these inference graphs to describe operators working in state spaces; here each internal arc of the inference graph corresponds to an operator invocation and each leaf arc to a general “probabilistic experiment”. Using $G_A$, for example, $a_3$ could encode the “take some blood” operator, and $a_5$, the experiment that succeeds if the patient tests positive on $\text{bt#1}$, etc.

N-Effect 2. The companion paper [GJ92] provides more formal descriptions of inference graphs and strategies. That article also presents an efficient analytic way of computing upper and lower bounds of $\Delta[\tau_{a_1,a_2}(\Theta_j), \Theta_j, S]$ (which can be used in Equations 2 and 3, respectively), based only on running $\Theta_j$. This provides a way of obtaining good estimates of $\Delta[\tau_{a_1,a_2}(\Theta_j), \Theta_j, S]$ without first constructing and then executing each $\tau_{a_1,a_2}(\Theta_j)$ over all $S = \{q_1\}$ queries. It also presents empirical evidence that a system that uses those estimates can still work effectively.

That paper also discusses how this instantiation of the PALO algorithm fits into the framework of “explanation-based learning” systems, and in particular, argues that it provides a mathematical basis for [Min88]’s “utility analysis”.

6Of course, all of the signs in Figure 1 should be flipped, as we are here measuring cost rather than utility, and so prefer the element with minimal, rather than maximal, cost. Note also that we are viewing each strategy as a performance element.
4.2 Improving Accuracy

A nonmonotonic system can be ambiguous, in that it can produce many individually plausible but collectively incompatible solutions to certain queries [Reis87]. Unfortunately, only (at most) one of these solutions is correct; the challenge then is to determine which one. This is the essence of the “multiple extension problem” in knowledge representation [Reis87, HM86, Mor87], and corresponds to the “bias problem” in machine learning [Mit80, Utg84, RG87, Han88]. This subsection addresses this problem by selecting a credulous system, related to the given initial nonmonotonic system, that is “optimally correct”; i.e., which produces the correct answer most often.

In more detail, we assume there is a correct answer to each query $q$, denoted $O(q)$; hence $O[2 + 2 = ?x] = \text{Yes}$. Each correct answer is either “Yes” (possibly with a binding list, as shown here) or “No”. Using $PE(q)$ to represent the answer returned by the credulous performance element $PE$, we can define the utility function

$$c_q(PE, q) \triangleq \begin{cases} +1 & \text{if } PE(q) = O[q] \\ 0 & \text{if } PE(q) = \text{IDK} \\ -1 & \text{otherwise} \end{cases}$$

where $\text{IDK}$ represents “I don’t know”.

We focus on stratified THEORIST-style performance elements [PGA86] [Prz87, Bre89, VA90], where each element $PE = (F, \mathcal{H}, \Theta)$ corresponds to a set of factual information $F$, a set of allowed hypotheses $\mathcal{H}$ (each a simple type of default [Reis87]) and a specific ordering of the hypotheses. As a specific example, consider $PE_A = (F_0, \mathcal{H}_0, \Theta_A)$, where

$$F_0 = \{ \forall x, E(x) \& \Psi_E(x) \Rightarrow S(x, G) \\ \forall x, A(x) \& \Psi_A(x) \Rightarrow S(x, W) \\ \forall x, \neg S(x, G) \lor \neg S(x, W) \}$$

is the fact set;

$$\mathcal{H}_0 = \{ h_1 : \Psi_E(x) \}$$

is the hypothesis set, and $\Theta_A = \{ h_1, h_2 \}$ is the hypothesis ordering.

To explain how $PE_A$ would process a query, imagine we want to know the color of Zelda $-$ i.e., we want to find a binding for $?c$ such that $\sigma = S(Z, ?c)$ holds. $PE_A$ would first try to prove $\sigma$ from the factual information $F_0$ alone. This would fail, as we do not know if Zelda is a normal elephant or if she is a normal albino (i.e., whether $\Psi_E(Z)$ or $\Psi_A(Z)$ holds, respectively). $PE_A$ then considers using some hypothesis $-$ i.e., it may assert an instantiation of some element of $\mathcal{H}_0$ if that proposition is both consistent with the known facts $F_0$ and if it allows us to reach a conclusion to the query posed. Here, $PE_A$ could consider asserting either $\Psi_E(Z)$ (meaning that Zelda is a “normal” elephant and hence is colored Gray) or $\Psi_A(Z)$ (meaning that Zelda is a “normal” albino and hence is colored White). Notice that either of these options, individually, is consistent with everything we know, as encoded by $F_0$. Unfortunately, we cannot assume both options, as the resulting theory $F_0 \cup \{ \Psi_E(Z), \Psi_A(Z) \}$ is inconsistent.

We must, therefore, decide amongst these options. $PE_A$’s hypothesis ordering $\Theta_A$ specifies the priority of the hypotheses. Here $\Theta_A = \{ h_1, h_2 \}$ means that $h_1 : \Psi_E(x)$ takes priority over $h_2 : \Psi_A(x)$, which means that $PE_A$ will return the conclusion associated with $\Psi_E(Z) -$ i.e., Gray, encoded by $\text{Yes}[?c \mapsto G]$, as $F_0 \cup \{ \Psi_E(Z) \} \models S(Z, G)$.\(^1\)

Now consider the $PE_B = (F_0, \mathcal{H}_0, \Theta_B)$ element, which differs from $PE_A$ only in terms of its ordering: As $PE_B$’s $\Theta_B = \{ h_2, h_1 \}$ considers the hypotheses in the opposite order, it will return the answer $\text{Yes}[?c \mapsto W]$ to this query; i.e., it would claim that Zelda is white.

Which of these two elements is better? If we are only concerned with this single Zelda query, then the better (read “more accurate”) $PE_B$ is the one with the larger value for $c_q(PE, S(Z, ?c))$; i.e., the $PE_i$ for which $c_q(PE, S(Z, ?c)) = O[S(Z, ?c)]$. In general, however, we have to consider a less trivial distribution of queries. To illustrate this, imagine Equation 5’s “...” corresponds to $\{ A(Z_1), E(Z_1), ..., A(Z_{100}), E(Z_{100}) \}$, stating that each $Z_i$ is an albino elephant, and that the queries are of the form $S(Z_i, ?c)$, for various $Z_i$.

The best $PE_i$ now depends on the distribution of queries (i.e., how often each “$S(Z_i, ?c)$” query is posed) and also on the correct answers (i.e., for which $Z_i$ is $O[S(Z_i, ?c)] = \text{Yes}[?c \mapsto W]$ as opposed to $O[S(Z_i, ?c)] = \text{Yes}[?c \mapsto G]$, or some other answer). That is, it depends on the expected accuracy of each system $C_q[PE_i]$, which is defined by plugging Equation 4’s $c_q(\cdot, \cdot)$ function into Equation 1. We would then select the $PE_i$ system with the larger $C_q[\cdot]$ value.

In general, $PE = (F, \mathcal{H}, \Theta)$ can include a much larger set of hypotheses $\mathcal{H} = \{ h_1, ..., h_n \}$. As before, each ordering $\Theta = \{ h_{\eta(1)}, ..., h_{\eta(n)} \}$ is a sequence of $\mathcal{H}$’s elements. $PE$’s uses this information when answering queries: Let $i$ be the smallest index such that $F \cup \{ h_i \}$ is consistent and $F \cup \{ h_j \} \vdash q / \lambda_i$ for some answer $\lambda_i$; here $PE$ returns this $\lambda_i$. If there are no such $i$’s, then $PE$ returns IDK.

Our goal is identifying the ordering that is accurate most often. Unfortunately, the task of identifying this optimal ordering of the hypotheses is NP-complete even for the simplistic situation we have been considering (where every derivation involves exactly one hypothesis, etc.); see [Gre92].

Once again, PALO is designed to deal with this situation. We first define the set of transformations $\mathcal{A} = \{ \tau_{ij} \}_{i,j}$: where each $\tau_{ij}$ moves the $i^{th}$ term in the ordering to just before the $j^{th}$ term $-$ i.e., given any ordering $\Theta = \{ h_1, h_2, ..., h_n \}$.

\(^1\)This uses the instantiation $S(Z, G) = S(Z, ?c) / \text{Yes}[?c \mapsto G]$. We will also view “$q / \text{No}$” as “~$q$.”
to determine whether a query $\sigma$ follows from $\Sigma$, as shown in Figure 3: If $W \models \sigma$, PE terminates with “yes”; otherwise, if $S \not\models \sigma$, PE terminates with “no”. (Notice that these are the correct answers, in that $W \models \sigma$ guarantees that $\Sigma \models \sigma$, and $S \not\models \sigma$ guarantees that $\Sigma \not\models \sigma$). Moreover, these tests are linear in the sizes of $\Sigma$ and $S$ (respectively, $s$ and $w$) [DG84]. Otherwise, if $W \not\models \sigma$ and $S \models \sigma$, PE returns DDK. Notice this compiled system is usually tractable \[\ddagger\] yet can deal with an arbitrary propositional theory. It may, however, no longer be completely categoric; hence, we have (potentially) sacrificed complete accuracy for tractability [SK91].

We of course would like to find an approximation $(S, W)$ that minimizes the probability that the associated PE$(S, W)$ system will return DDK. To state this more precisely: Given any approximation $(S, W)$ and query $\sigma$, let $c_c((S, W), \sigma) \overset{\text{def}}{=} d(w, \sigma) + (1 - d(S, \sigma))$ where
\[
d(S, \sigma) \overset{\text{def}}{=} \begin{cases} 1 & \text{if } S \models \sigma \\ 0 & \text{otherwise} \end{cases}
\]

Hence, $c_c((S, W), \sigma) = 1$ if $\sigma$ is “covered” by $(S, W)$, in that either $W \models \sigma$ or $S \not\models \sigma$. Using Equation 1, we can then define $C_c(S, W)$ to be the expected value of $c_c((S, W), \cdot)$. Our goal is to determine the approximation $(S, W)$ with the largest $C_c(\cdot)$ value. As before, this task is NP-hard (see [Gre92]) and depends on the distribution, suggesting yet again that we use the PALO system.

Observe that the set of queries covered by a strengthening and a weakening are disjoint — i.e., for any approximation $(S, W)$, there is no query $\sigma$ such that both $W \models \sigma$ and $S \not\models \sigma$. This means an approximation $(S_t, W_t)$ is, with probability at least $1 - \varepsilon$, within $\varepsilon$ of a local optimum if $S_t$ (resp., $W_t$) is within $\varepsilon/2$ of a locally optimal strengthening (resp., weakening) with probability at least $1 - \varepsilon/2$. We can therefore decouple the task of finding a good strengthening from that of finding a good weakening and handle each separately. This paper discusses only how to finding a good strengthening: [Gre92] merges this with the algorithm that computes a good weakening.

We are seeking a strengthening $S_{\text{opt}}$ whose $D[S_{\text{opt}}]$ value is minimal, where $D[S_{\text{opt}}] = E[d(S, \cdot)]$ is the expected value of $d(S, \cdot)$. (Recall we want $S_{\text{opt}} \models \sigma$ to fail for as many queries as possible.) It is easy to see that this $S_{\text{opt}}$ should be a weakest strengthening; i.e., satisfy $\text{Opt}(S, S_{\text{opt}})$ where
\[
\text{Opt}(\Sigma, S_{\text{opt}}) \iff S \models \Sigma \land \text{Horn}(S) \land \exists T. [S \models T \land S = \Sigma \land \text{Horn}(S) \land S \not\models T]
\]

To compute these OptSs: Define a “horn-strengthening” of the clause $\gamma = \{a_1, \ldots, a_k, \neg b_1, \ldots, \neg b_l\}$ to be any maximal clause that is a subset of $\gamma$ and is Horn — i.e., each horn-strengthening is formed by simply discarding all but one of $\gamma$’ positive literals. Here, there are $k$ horn-strengthenings of $\gamma$, each of the form $\gamma' = \{a_j, \neg b_1, \ldots, \neg b_t\}$.  

\[\ddagger\] Note 2 below explains this caveat.

\[\ddagger\]
Table 1: Summary of Applications

<table>
<thead>
<tr>
<th>Efficiency</th>
<th>Accuracy</th>
<th>Categoricity</th>
</tr>
</thead>
<tbody>
<tr>
<td>sating strategies</td>
<td>hypothesis orderings</td>
<td>Horn-strengthenings</td>
</tr>
<tr>
<td>( PE(q) = \mathcal{C}[q] )</td>
<td>( S \models q )</td>
<td>( \leq 1 )</td>
</tr>
</tbody>
</table>

5 Conclusion

This paper first poses two of the problems that can arise in learning systems that seek a performance element whose expected utility is optimal [Hau90, Vap82], viz., that the distribution information (which is required to determine which element is optimal) is usually unknown, and that finding a globally optimal performance element can be intractable. It then presents the PALO algorithm that side-steps these shortcomings by using statistical techniques to approximate the distribution, and by hill-climbing to produce a near locally optimal element. After defining this algorithm and specifying its behaviour, we demonstrate its generality by showing how it can be used to find a near-optimal element in three very different settings, based on different spaces of performance elements and different criteria for optimality: efficiency, accuracy and categoricity. (See Table 1.) These results suggest approaches to solving the utility problem from explanation-based learning, the multiple extension problem from nonmonotonic reasoning and the tractability/completeness tradeoff problem from knowledge representation.

References


