A Statistical Approach to Solving the EBL Utility Problem

Russell Greiner*
Siemens Corporate Research
Princeton, NJ 08540
greiner@learning.siemens.com

Igor Jurišica†
Department of Computer Science
University of Toronto
Toronto, Ontario M5S 1A4, Canada
juris@cs.toronto.edu

Abstract

Many "learning from experience" systems use information extracted from problem solving experiences to modify a performance element PE, forming a new element PE′ that can solve these and similar problems more efficiently. However, as transformations that improve performance on one set of problems can degrade performance on other sets, the new PE′ is not always better than the original PE; this depends on the distribution of problems. We therefore seek the performance element whose *expected performance*, over this distribution, is optimal. Unfortunately, the actual distribution, which is needed to determine which element is optimal, is usually not known. Moreover, the task of finding the optimal element, even knowing the distribution, is intractable for most interesting spaces of elements. This paper presents a method, PALO, that side-steps these problems by using a set of samples to estimate the unknown distribution, and by using a set of transformations to hill-climb to a local optimum. This process is based on a mathematically rigorous form of *utility analysis*: in particular, it uses statistical techniques to determine whether the result of a proposed transformation will be better than the original system. We also present an efficient way of implementing this learning system in the context of a general class of performance elements, and include empirical evidence that this approach can work effectively.

1 Introduction

Problem solving is inherently combinatorially expensive [Nil80]. There are, of course, many methods designed to side-step this problem. One collection of techniques is based on the observation that many problems occur repeatedly; this has led to a number of "learning from experience" (or "LFE") systems [DeJ88, MCK+89, LNR87] that each use information gleaned from one set of problem solving experiences to modify the underlying problem solver, forming a new one capable of solving similar problems more efficiently.

Unfortunately, a modification that improves the problem solver's performance for one set of problems can degrade its performance for other problems [Min88b, Gre91]; hence, many of these modifications will in fact *lower* the system's overall performance. This paper addresses this problem (often called the "EBL[1] utility problem" [Min88b, SER91]) by using a statistical technique to determine whether the result of a proposed modification will, with provably high confidence, be better than the original system. We extend this technique to develop a LFE algorithm, PALO, that produces a system whose performance is, with arbitrarily high probability, arbitrarily close to a local optimum. We then focus on an instantiation of this general PALO algorithm that can solve a learning problem that provably cannot be algorithmically solved in a stronger sense, as well as empirical data that demonstrates PALO's effectiveness.

In more detail [BMSJ78]: A performance element PE is a program that attempts to solve the given problems. A learning element LFE uses information gleaned from these problem solving experience(s) to transform PE into a new performance element, PE′.[2] Just as concept learning can be characterized as a search through a space of possible concept descriptions [Mit82], so the LFE system can be viewed as searching through a space of possible PEs, seeking a new performance element PE′ whose performance is superior to that of the original PE. Typical LFEs traverse the space of possible PEs using transformations that modify a given PE by adding

*Much 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. We also gratefully acknowledge receiving many helpful comments from William Cohen, Dave Mitchell, Dale Schuurmans and the anonymous referees.

†Supported by a University of Toronto Open Fellowship and a Research Assistantship from the Department of Computer Science.

[1] "EBL" abbreviates "Explanation-Based Learning".

[2] These two components, PE and LFE, may correspond the same bundle of code; *cf.*, SOAR [LNR87]. We often view this PE′ as a modified version of PE.

macro-rules, re-ordering the rules, adding censors to existing rules, and so on.

Our previous papers have presented algorithms that find the PE′ whose performance is optimal [GL89, Gre91] or nearly-optimal [OG90, GO91], where performance is measured as the *expected cost* of a PE over some fixed distribution of problems. Unfortunately, the task of finding the globally optimal PE is intractable for most interesting cases [Gre91].

In contrast, most other previous LFE research [DeJ88, MCK+89, LNR87] has focused on experimental techniques for incrementally modifying a problem solver, producing a series of performance elements $\mathrm{PE}_0, \ldots, \mathrm{PE}_m$ where each PE_{i+1} is a modification of PE_i (*e.g.*, PE_{i+1} might contain one new macro-rule not in PE_i). Unfortunately, existing methods do not always guarantee that each PE_{i+1} is an *improvement* over PE_i; *a fortiori*, the overall m-step process may produce a final PE_m that is not even superior to the initial PE_0, much less one that is optimum in the space of PEs.[3]

This paper integrates ideas from both lines of research, by describing a tractable *incremental* algorithm that is (probabilistically) guaranteed to find a *locally optimal* performance element. In particular, Section 2 motivates the use of "expected cost" as a quality metric for 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 PE; this tool can be viewed as mathematically rigorous version of [Min88a]'s "utility analysis". It uses this tool to define the general PALO algorithm, that incrementally produces a series of performance elements $\mathrm{PE}_0, \ldots, \mathrm{PE}_m$ such that each PE_{i+1} is statistically likely to be an incremental improvement over PE_i and, with high confidence, the performance of the final element PE_m is essentially a local optimal. Finally, Section 4 presents an instantiation of this program that uses a specific set of transformations to "hill-climb" in a particular, very general, space of performance elements. It also presents an efficient way of obtaining approximations to the information PALO needs, and provides empirical evidence that this program does work effectively. Here, PALO is efficiently finding a locally optimal PE_{lo} in a space of PEs for which the globally optimal PE_{go} cannot be tractably found. The conclusion discusses how this work extends related research.

2 Framework

We view each performance element PE as a function that takes as input a problem (or query or goal, etc.) to solve, q, and returns an answer. In general, we can consider a large set of (implicitly defined) possible performance elements, $\{\mathrm{PE}_i\}$; Section 4 considers the naturally occurring set of problem solvers that use different control strategies.

Which of these elements should we use; *i.e.*, which is "best"? The answer is obvious: The best PE is the one that performs best in practice. To quantify this, we need to define some measure on these elements: We start by defining $c(\mathrm{PE}_j, q_i)$ to be the "cost" required for PE_j to solve the problem q_i. (The $c(\cdot, \cdot)$ function defined in Section 4 measures the time required to find a solution.)

This cost function specifies which PE_j is best for a single problem. Our PE_js, however, will have to solve an entire ensemble of problems $\mathcal{Q} = \{ q_i \}$; we clearly prefer the element that is best overall. We therefore consider the distribution of queries that our performance element will encounter, which is modelled by a probability function, $Pr : \mathcal{Q} \mapsto [0,1]$, where $\Pr[q_i]$ denotes the probability that the problem q_i is selected. This $\Pr[\cdot]$ reflects the distribution of problems our PE is actually addressing; *n.b.*, it is not likely to be a uniform distribution over all possible problems [Gol79], nor will it necessarily correspond to any particular collection of "benchmark challenge problems" [Kel87].

We can then define the *expected cost* of a performance element:

$$\mathrm{C}[\,\mathrm{PE}\,] \ \stackrel{def}{=}\ E[\,c(\mathrm{PE}, \mathbf{q})\,] \ = \ \sum_{q \in \mathcal{Q}} \Pr[\,q\,] \times c(\mathrm{PE}, q)$$

Our underlying challenge is to find the performance element whose expected cost is minimal. There are, however, two problems with this approach: First, we know to know the distribution of queries to determine the expected cost of any element and hence which element is optimal; unfortunately the distribution is usually not known. 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, that side-steps the above problems by using a set of sample queries to estimate the distribution, and by efficiently hill-climbing from a given initial PE_0 to one that is, with high probability, close to a local optimum. This section first states the theorem that specifies PALO's functionality, then summarizes PALO's code and sketches a proof of the theorem.

PALO takes as arguments an initial PE_0 and parameters $\epsilon, \delta > 0$. It uses a set of sample queries drawn at random from the $\Pr[\cdot]$ distribution[4] to climb from the initial PE_0 to a final PE_m, using a particular set of possible transformations $\mathcal{T} = \{\tau_j\}$, where each τ_j maps one given performance element into another; see Subsection 4.2. PALO then returns this final PE_m. Theorem 1 states our main theoretical results.[5]

Theorem 1 *The* PALO$(\mathrm{PE}_0, \epsilon, \delta)$ *process incrementally produces a series of performance elements* $\mathrm{PE}_0, \mathrm{PE}_1, \ldots, \mathrm{PE}_m$*, staying at a particular* PE_j *for only a polynomial number of samples before either climbing to* PE_{j+1} *or terminating. With probability at least* $1-\delta$*,* PALO *will terminate. It then returns an element*

[3] Section 5 provides a more comprehensive literature search, and includes a few exceptions to the above claims.

[4] These samples may be produced by the user, who is simply asking questions relevant to one of his tasks.

[5] This proof, and others, appear in the expanded version of this paper [GJ92].

Algorithm PALO(PE_0, ϵ, δ)

- $i \leftarrow 0 \qquad j \leftarrow 0$

L1: Let $S \leftarrow \{\}$ $\quad$ Neigh $\leftarrow \{\, \tau_k(\mathbf{PE}_j)\,\}_k$

$\Lambda_{max} \;=\; \max\{\, \Lambda[\mathrm{PE}', \mathrm{PE}_j] \mid \mathrm{PE}' \in \text{Neigh}\,\}$

L2: Get query q (from the user).

Let $\quad S \;\leftarrow\; S \cup \{q\} \qquad i \;\leftarrow\; i + |\text{Neigh}|$

- If there is some $\mathbf{PE}' \in$ Neigh such that

$$\Delta[\mathrm{PE}', \mathrm{PE}_j, S] \;\geq\; \Lambda[\mathrm{PE}', \mathrm{PE}_j]\sqrt{\frac{|S|}{2}\ln\left(\frac{i^2\,\pi^2}{3\,\delta}\right)} \qquad (1)$$

then let $\quad \mathbf{PE}_{j+1} \leftarrow \mathbf{PE}', \quad j \leftarrow j+1.$

Return to **L1**.

- If $\;|S| \;\geq\; \frac{2\Lambda^2_{max}}{\epsilon^2}\ln\left(\frac{i^2\pi^2}{3\;\delta}\right)$ and

$$\forall\, \mathbf{PE}' \in \text{Neigh}.\quad \Delta[\mathrm{PE}', \mathrm{PE}_j,\, S] \;\leq\; \frac{\epsilon\,|S|}{2}, \qquad (2)$$

then halt and return as output $\mathbf{PE}_j$.

- Otherwise, return to **L2**.

Figure 1: Code for PALO

PE_m *whose expected utility* $C[\,\mathrm{PE}_m\,]$ *is, with probability at least* $1-\delta$*, both*

1. *at least as good as the original* PE_0*; i.e.,* $C[\,\mathrm{PE}_m\,] \;\leq\; C[\,\mathrm{PE}_0\,]$*; and*
2. *an ϵ-local optimum*[6] *— i.e.,* $\forall \tau_j \in \mathcal{T}.\; C[\,\mathrm{PE}_m\,] \;\leq\; C[\,\tau_j(\mathrm{PE}_m)\,] + \epsilon \qquad \Box.$

The basic code for PALO appears in Figure 1. In essence, PALO will climb from PE_j to a new PE_{j+1} if PE_{j+1} is likely to be strictly better than PE_j; *i.e.*, if we are highly confident that $\mathrm{C}[\,\mathrm{PE}_{j+1}\,] < \mathrm{C}[\,\mathrm{PE}_j\,]$. To determine this, define

$$d_i \;=\; \Delta[\mathrm{PE}_\alpha,\, \mathrm{PE}_\beta, q_i] \;\stackrel{def}{=}\; c(\mathrm{PE}_\alpha,\, q_i) - c(\mathrm{PE}_\beta,\, q_i)$$

to be the difference in cost between using PE_α to deal with the problem q_i, and using PE_β. As each query q_i is selected randomly according to a fixed distribution, these d_is are independent, identically distributed random variables whose common mean is $\mu = \mathrm{C}[\,\mathrm{PE}_\alpha\,] - \mathrm{C}[\,\mathrm{PE}_\beta\,]$. (Notice PE_β is better than PE_α if $\mu > 0$.)

Let $Y_n \;\stackrel{def}{=}\; \frac{1}{n}\Delta[\mathrm{PE}_\alpha,\, \mathrm{PE}_\beta,\, \{q_i\}_{i=1}^n]$ be the sample mean over n samples, where $\Delta[\mathrm{PE}_\alpha,\, \mathrm{PE}_\beta, S] \;\stackrel{def}{=}\; \sum_{q\in S} c(\mathrm{PE}_\alpha,\, q) - c(\mathrm{PE}_\beta,\, q)$ for any set of queries S. This average tends to the population mean, μ as $n \rightarrow \infty$; *i.e.*, $\mu = \lim_{n\rightsquigarrow\infty} Y_n$. 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 γ increases. Formally,

$$Pr[\,Y_n > \mu + \gamma\,] \;\leq\; e^{-2n\left(\frac{\gamma}{\Lambda}\right)^2}$$
$$Pr[\,Y_n < \mu - \gamma\,] \;\leq\; e^{-2n\left(\frac{\gamma}{\Lambda}\right)^2}$$

[6] Notice a "0-local optimal" corresponds to the standard notion of "local optimal"; hence "ϵ-local optimal" generalizes local optimality.

where Λ is the range of possible values of $c(\mathrm{PE}_\alpha,\, q_i) - c(\mathrm{PE}_\beta,\, q_i)$.[7] This $\Lambda = \Lambda[\mathrm{PE}_\alpha, \mathrm{PE}_\beta]$ is also used in both the specification of Λ_{max} and in Equation 1. Section 4.2 below discusses how to compute this value for relevant $\mathrm{PE}_i/\tau_j(\mathrm{PE}_i)$ pairs.

The PALO algorithm uses these equations and the values of $\Delta[\,\mathrm{PE}',\, \mathrm{PE}_j, S\,]$ to determine both how confident we should be that $\mathrm{C}[\,\mathrm{PE}'\,] > \mathrm{C}[\,\mathrm{PE}_j\,]$ (Equation 1) and whether any "$\mathcal{T}$-neighbor" of PE_j (*i.e.*, any $\tau_k(\mathrm{PE}_j)$) is more than ϵ better than PE_j (Equation 2).

4 Instantiation: Learning Good Strategies

The algorithm shown above can deal with essentially arbitrary sets of performance elements, cost functions and sets of transformations. This section presents a particular instantiation of this framework: Subsection 4.1 presents a model for a general class of "graph-based performance elements" $\mathcal{PE}_G$ and the obvious cost function. Subsection 4.2 then describes the set of "re-ordering" transformations $\mathcal{T}^{RO}$, each of which re-arranges the order in which PE traverses the arcs of the graph. It also describes an efficient way of approximating the values of $\Delta[\tau_j(\mathrm{PE}),\, \mathrm{PE}, S]$. Subsection 4.3 presents some empirical results that demonstrate that a system that uses these approximations can be effective.

We choose the $\mathcal{PE}_G$ class of performance elements as it corresponds to many standard problem solvers (including PROLOG [CM81]; see also [GN87]); and the $\mathcal{T}^{RO}$ class of transformations on strategies, as it corresponds to many EBL systems and moreover, the task of finding the *global* optimality strategy is NP-hard [Gre91].

4.1 Graph Based PEs

This subsection uses a particularly simple performance element PE_0 to illustrate the class $\mathcal{PE}_G$, whose elements each correspond to a finite graph whose arcs have fixed costs. After describing a relatively simple model, it presents several extensions, leading to a more realistic, comprehensive model.

The PE_0 element is based on the rules shown in the upper left corner of Figure 2 (producing the corresponding "reduction graph" shown in that figure), operating with respect to the facts shown in the lower left corner. We focus on how this system deals with queries of the form `GoodCar(`κ`)`, for some ground κ — *e.g.*, returning `Yes` to the queries `GoodCar(D1)` and `GoodCar(D2)`, and `No` to the queries `GoodCar(D4)` and `GoodCar(Fido)`.

In general, we can identify each performance element $\mathrm{PE} = \langle G,\, \Theta\rangle \in \mathcal{PE}_G$ with a reduction graph G and a strategy Θ, where a reduction graph $G = \langle N, A, S, f\rangle$ is a structure formed by a set of rules: N is a set of nodes (each corresponding to a proposition; *e.g.*, the node N_0 corresponds to "`GoodCar(`κ`)`" and N_2 corresponds to the empty disjunction), and $A \subset N \times N$ is a set of arcs, each corresponding either to the use of a rule (*e.g.*, the a_1 arc

[7] See [Bol85, p. 12]. *N.b.*, these inequalities holds for essentially *arbitrary distributions*, not just normal distributions, subject only to the minor constraint that the sequence $\{\Delta_i\}$ has a finite second moment.

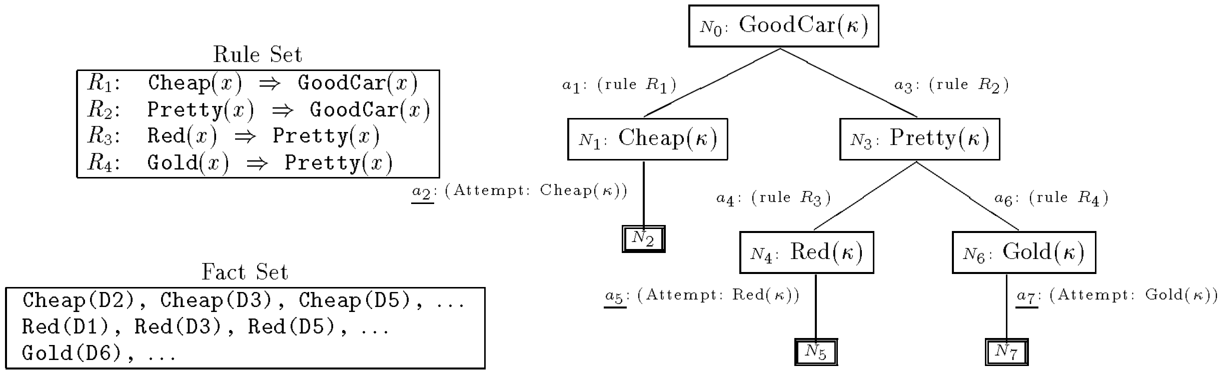

Figure 2: "Reduction Graph" G_A (used by PE_0 and PE_1)

from N_0 to N_1 is based on the rule R_1) or a database retrieval (*e.g.*, the a_2 arc from N_1 to N_2 corresponds to the attempted database retrieval `Cheap(`κ`)`). The set $S \subset N$ is the subset of N's "success nodes" (here, each is an empty disjunction such as N_2 or N_5, shown in doubled boxes); reaching any of these nodes means the proof is successful. The cost function $f\colon A \mapsto \mathcal{R}_0^+$ maps each arc to a non-negative value that is the cost required to perform this reduction. We will let f_i refer to the value of $f(a_i)$.

The strategy Θ specifies how the PE will traverses its graph G. Here, it corresponds to a simple sequence of arcs, *e.g.*,

$$\Theta_{\langle crg\rangle} = \langle a_1, a_2, a_3, a_4, a_5, a_6, a_7\rangle \tag{3}$$

is the obvious left-to-right depth-first strategy, with the understanding that $\mathrm{PE} = \langle G_A, \Theta_{\langle crg\rangle}\rangle$ stops whenever it reaches a success node (*e.g.*, if a_2 succeeds, then PE_0 reaches N_2 and so stops with success), or has exhausted all of its reductions.[8] There are other possible strategies, including other non–left-to-right depth-first strategies, *e.g.*,

$$\Theta_{\langle rgc\rangle} = \langle a_3, a_4, a_5, a_6, a_7, a_1, a_2\rangle \tag{4}$$

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

We focus on two members of $\mathcal{PE}_G$: $\mathrm{PE}_0 = \langle G_A, \Theta_{\langle crg\rangle}\rangle$ and $\mathrm{PE}_1 = \langle G_A, \Theta_{\langle rgc\rangle}\rangle$.

Cost of Solving Problems: We can compute the cost for PE_j to solve q_i, $c(\mathrm{PE}_j, q_i)$, from the above specification. For example, $c(\mathrm{PE}_0,$ `GoodCar(D2)`$) = f_1 + f_2$, and $c(\mathrm{PE}_0,$ `GoodCar(D1)`$) = f_1 + f_2 + f_3 + f_4 + f_5$, as the $\langle a_1, a_2\rangle$ path failed as `Cheap(D1)` is not in the fact set. As each strategy stops as soon as it finds an answer, different strategies can different costs for a given query; *e.g.*, $c(\mathrm{PE}_1,$ `GoodCar(D1)`$) = f_3 + f_4 + f_5$ differs from $c(\mathrm{PE}_0,$ `GoodCar(D1)`$)$, etc.

We can view each strategy as a sequence of paths, where each path is a sequence of arcs that descend from some already-visited node down to a retrieval; *e.g.*, $\Theta_1 \approx \langle\, \langle a_1\, a_2\rangle, \langle a_3\, a_4\, a_5\rangle, \langle a_6\, a_7\rangle\, \rangle$. We define the *expected cost* of a strategy as the weighted sum of the costs of its paths, each weighted by the probability that we will need to pursue this path, *i.e.*, that none of the prior paths succeeded [Smi89, GO91]. (Of course, the cost of a path is the sum of the cost of its arcs.)

While the models of performance elements and cost presented above are sufficient for the rest of this article, they appear quite limited. We close this subsection by presenting some of the extensions that lead to a more comprehensive framework. *N.b.*, the model presented in [GJ92] incorporates all of these extensions.

Extend1. (General Graph) The above definitions are sufficient for the class of simple "disjunctive reduction graphs", which consist only of rules whose antecedents each include a single literal. To deal with more general rules, whose antecedents are conjunctions of more than one literal (*e.g.*, "`B`(x)&`C`$(x) \Rightarrow$ `A`(x)"), we must use directed hyper-graphs, where each "hyper-arc" descends from one node to a *set* of children nodes, where the conjunction of these nodes logically imply their common parent. We would also define S to be a set of subsets of N, where the query processor would have to reach each member of some $s \in S$ for the derivation to succeed. This extension leads to additional complications in specifying strategies; see also [GO91, Appendix A].

Extend2. (Probabilistic Experiments) We say that "the arc a_i is blocked in the context of the query q" if no strategy can traverse a_i when answering the query q; *e.g.*, the retrieval arc a_2 is blocked in the context of `GoodCar(D1)` as the associated literal `Cheap(D1)` is not in the fact set. So far, we have implicitly assumed that retrieval arcs can be blocked, but rule-based arcs cannot. If we permit the literals in the rules to include constants, however, rule-based arcs can also be blockable. Consider, for example, adding the rule "$\forall x$ `Owner(Fcar,` x`)` $\Rightarrow$ `GoodCar(Fcar)`", which states that the particular car `Fcar` is good if it is owned by anybody. Notice a performance element will be able to traverse the rule-based reduction arc from `GoodCar(`x`)` to `Owner(Fcar,` x`)` only if the query is `GoodCar(Fcar)`; notice this arc is blocked for every other query. Our

[8] Notice that strategies, including PE_0, accept the first solution found, meaning they are performing "satisficing searches" [SK75].Hence, we are only considering the cost required to produce an answer, and *not* the quality of the answer itself. There are obvious extensions to this cost model that can incorporate different utility values for answers of different "qualities"; see [GE91].

model can handle these situations by allowing any arc (not just retrieval arcs) to be blockable.

Extend3. (General Cost Function) The algorithms presented in this paper can accommodate more complicated $f(\cdot)$ cost functions, which can allow the cost of traversing an arc to depend on other factors — *e.g.*, the success or failure of that traversal, which other arcs have already been traversed, etc.

Extend4. (Infinite Set of Queries) Our analysis can accommodate even an infinite number of queries, as we can partition them into a finite set of equivalence classes, where all members of an equivalence classes have the same cost for each strategy. This follows from the observation that the cost of using a strategy to solve a query depends only on which arcs are blocked, meaning we can identify each query with the subset of arcs that can are blocked for that query. For example, we can identify the query `GoodCar(B1)` with the arc-set $\{a_2, a_7\}$ and `GoodCar(B2)` with $\{a_5, a_7\}$, etc.

4.2 Re-Ordering Transformations

This subsection considers a way of modifying a performance element PE $= \langle G, \Theta\rangle$ by reordering the strategy (*i.e.*, changing from Θ to Θ') while preserving the underlying reduction graph G. For example, after finding that $\langle a_1, a_2\rangle$ failed but the $\langle a_3, a_4, a_5\rangle$ path succeeded, one might transform $\mathrm{PE}_0 = \langle G_A, \Theta_{\langle crg\rangle}\rangle$ into $\mathrm{PE}_1 = \langle G_A, \Theta_{\langle rgc\rangle}\rangle$, by moving a_3 (and its children) before a_1 (and its children). In general, given any reduction graph $G = \langle N, A, S, f\rangle$, define $\mathcal{T}^{RO} = \{\tau_{r1,r2}\}_{r1,r2}$ to be the set of all possible "simple strategy transformations", as follows: Let $r1, r2 \in A$ be two arcs that each descend from a single node (*e.g.*, a_1 and a_3 each descend from the node N_0); and consider any strategy

$$\Theta_A = \pi_1 \circ \pi_2 \circ \pi_3 \circ \pi_4, \tag{5}$$

where the $\circ$ operator is concatenation and each π_j is a (possibly empty) sequence of arcs, and in particular, $\pi_2 = \langle r1, \ldots\rangle$ corresponds to $r1$ and its children, and $\pi_3 = \langle r2, \ldots\rangle$, to $r2$ and its children.[9] Then $\Theta_B = \tau_{r1,r2}(\Theta_A)$ will be a strategy that differs from Θ_B only in that $r1$ and all of its descendents are moved earlier in the strategy, to before $r2$; *i.e.*,

$$\Theta_B = \tau_{r1,r2}(\Theta_A) = \pi_1 \circ \underline{\pi_3} \circ \underline{\pi_2} \circ \pi_4\ . \tag{6}$$

(To understand the transformation from $\Theta_A = \Theta_{\langle crg\rangle}$ to $\Theta_B = \tau_{a_3,a_1}(\Theta_{\langle crg\rangle}) = \Theta_{\langle rgc\rangle}$: let $\pi_2 = \langle a_1, a_2\rangle$, $\pi_3 = \langle a_3, a_4, a_5, a_6, a_7\rangle$ and $\pi_1 = \pi_4 = \langle\rangle$.) Notice that the $\tau_{r1,r2}$ transformation will map a strategy Θ to itself if $r1$ already comes before $r2$ in Θ. $\mathcal{T}^{RO}$ is the set of all such $\tau_{r1,r2}$s.

Approximating $\Delta[\mathrm{PE}_i, \mathrm{PE}', S]$: The PALO algorithm requires values of $\Delta[\mathrm{PE}_i, \tau_j(\mathrm{PE}_i), S]$ for each $\tau_j \in \mathcal{T}$. One obvious (though expensive) way of obtaining these values is to construct each $\tau_j(\mathrm{PE}_i)$ performance element, and run this element on each $q \in S$, recording the total cost each requires. This can be expensive, especially when there are many different $\tau_j(\mathrm{PE}_i)$s. Fortunately, there is an alternative that involves running only the PE_i element and using the statistics obtained to find both under-estimates $L(\mathrm{PE}_i, \tau_j(\mathrm{PE}_i), S) \le \Delta[\mathrm{PE}_i, \tau_j(\mathrm{PE}_i), S]$ and over-estimates $U(\mathrm{PE}_i, \tau_j(\mathrm{PE}_i), S) \ge \Delta[\mathrm{PE}_i, \tau_j(\mathrm{PE}_i), S]$, that can be used in Equations 1 and 2, respectively.

In general, the $\mathrm{PE}_A = \langle G, \Theta_A\rangle$ element terminates as soon as it finds an answer; based on the decomposition shown in Equation 5, there are four cases to consider, depending on the path (one of $\{\pi_1, \pi_2, \pi_3, \pi_4\}$) in which this first answer appears. (For our purposes here, we view "finding no answer at all" as "finding the first answer in the final π_4".) If this first answer appears in either or π_1 or π_4, then $\Delta[\mathrm{PE}_A, \mathrm{PE}_B, q] = 0$, where $\mathrm{PE}_B = \langle G, \Theta_B\rangle$ from Equation 6. If the first answer appears in π_3, then we know that $\Delta[\mathrm{PE}_A, \mathrm{PE}_B, q] = f(\pi_2)$, where $f(\pi_i)$ in general is the sum of the costs of the arcs in π_i. For example, consider using PE_0 to deal with the `Goodcar(D1)` query: As the a_2 arc fails and a_5 succeeds, the first answer appears within $\pi_3 = \langle a_3, a_4, a_5, a_6, a_7\rangle$. PE_0 will find that answer, after it has first examined the path $\pi_2 = \langle a_1, a_2\rangle$ at a cost of $f(\pi_2) = f(\langle a_1, a_2\rangle) = f_1 + f_2$. Notice PE_1 would also find this same solution in π_3. However, PE_1 would *not* have first examined π_2, meaning its cost is $f(\pi_2)$ *less* than the cost of PE_0. Hence, $\Delta(\mathrm{PE}_0, \mathrm{PE}_1, \texttt{GoodCar(D1)}) = f(\pi_2)$.

The situation is more complicated if the first answer appears in π_2, as the value of $\Delta[\mathrm{PE}_A, \mathrm{PE}_B, q]$ depends on information that we cannot observe by watching PE_A alone. (*E.g.*, consider using PE_0 to deal with `Goodcar(D2)`. As a_2 succeeds, PE_0's first answer appears in π_2. As PE_0 then terminates, we do not know whether an answer would have appeared within π_3.) While we cannot determine the exact value of $\Delta[\mathrm{PE}_A, \mathrm{PE}_B, q]$ in these situations, we can obtain upper and lower bounds, based on whether a solution would have appeared in those unexplored paths: here $-f(\pi_3) \le \Delta[\mathrm{PE}_A, \mathrm{PE}_B, q] \le f(\pi_2) - f^+(\pi_3)$, where $f^+(\pi_3)$ is defined to be the cost of finding the first possible solution in the path π_3. (*E.g.*, $f^+(\langle a_3, a_4, a_5, a_6, a_7\rangle) = f_3 + f_4 + f_5$, as this is the first solution that could be found. Notice this under-estimates the cost of this path in every situation, and is the actual cost if the retrieval associated with a_5 succeeds.)

The table below gives the lower and upper bounds $L(q) \le \Delta[\mathrm{PE}_A, \mathrm{PE}_B, q] \le U(q)$, for all four cases (one for each π_i):[10]

if first answer is in	$L(q)$	$U(q)$
π_1	0	0
π_2	$-f(\pi_3)$	$f(\pi_2) - f^+(\pi_3)$
π_3	$f(\pi_2)$	$f(\pi_2)$
π_4	0	0

This table only bounds the value of $\Delta[\mathrm{PE}_A, \mathrm{PE}_B, q]$ for a single sample. The value of $\Delta[\mathrm{PE}_A, \mathrm{PE}_B, S]$ will be between $L(\mathrm{PE}_A, \mathrm{PE}_B, S) \stackrel{def}{=} \sum_{q\in S} L(q)$ and

[9] To simplify the presentation, this article will only consider depth-first strategies; [GJ92] extends this to deal with arbitrary strategies.

[10] [GJ92] shows how to obtain slightly tighter bounds, based on information that is available from PE_A's computation.

$U(\mathrm{PE}_A, \mathrm{PE}_B, S) \stackrel{def}{=} \sum_{q \in S} U(q)$. To compute these bounds, we need only *maintain a small number of counters*, to record the number of times a solution is found within each subpath: let k_2 (resp., k_3) be the number of times the first solution appears within π_2 (resp., π_3); then

$$\begin{array}{rcl} L(\mathrm{PE}_A, \mathrm{PE}_B, S) & = & k_3 \cdot [f(\pi_2)] \;-\; k_2 \cdot [-f(\pi_3)] \\ U(\mathrm{PE}_A, \mathrm{PE}_B, S) & = & k_3 \cdot [f(\pi_2)] \;+\; k_2 \cdot [f(\pi_2) - f(\pi_3)] \end{array}$$

PALO$'$ **Process:** Now define PALO$'$ to be the variant of PALO that differs only by using these $L(\mathrm{PE}', \mathrm{PE}_j, S)$ values (resp., $U(\mathrm{PE}', \mathrm{PE}_j, S)$ values) in place of $\Delta[\mathrm{PE}', \mathrm{PE}_j, S]$ in Equation 1 (resp., Equation 2). PALO$'$ can compute upper and lower bounds of $\Delta[\mathrm{PE}, \tau_j(\mathrm{PE}), S]$ *for each* $\tau_{r1,r2} \in \mathcal{T}^{RO}$ using only the values of a small number of counters: in general, it needs to maintain only one counter per retrieval.

To complete our description: PALO$'$ also needs one more counter to record the total number of sample queries seen, corresponding to $|S|$. Equation 1 needs the (static) values of $\Lambda[\tau_{r1,r2}(\mathrm{PE}_A), \mathrm{PE}_A)]$ for each $\tau_k \in \mathcal{T}^{RO}$. Each $\tau_{r1,r2}$ induces a particular segmentation of each strategy into the subsequences $\Theta_A = \pi_1 \circ \pi_2 \circ \pi_3 \circ \pi_4$. Here, $\Lambda[\tau_{r1,r2}(\mathrm{PE}_A), \mathrm{PE}_A] = f(\pi_2) + f(\pi_3)$, as each value of $\Delta[\,\tau_{r1,r2}(PE_A),\, \mathrm{PE}_A,\, q\,]$ is in the range $[-f(\pi_3),\; f(\pi_2)]$.

4.3 Empirical Results

The PALO algorithm only works "statistically", in that its results are guaranteed only if the samples it sees are truly representative of the distribution, and moreover, if the distribution from which these samples is drawn is stationary. The PALO$'$ algorithm is even more problematic, as it only uses approximations of the needed statistics.

Given these hedges, it is not obvious that the PALO$'$ algorithm should *really* work in a real domain. We are beginning to experiment with it in various real domains, including expert systems and natural language processors.

Here, we report on its performance in various artificial settings, where we can insure that the distribution is stationary.[11] Consider again the reduction graph shown in Figure 2, and assume unit cost for each arc, whether it represents a rule-based reduction or a database retrieval. We will define the distribution of queries in terms of the (independent) probabilities of the various database retrievals; here,

$$\begin{array}{lcl} P(\,\texttt{Cheap}(\kappa)\ \text{in Fact Set} \mid \texttt{GoodCar}(\kappa)\ \text{query asked}\,) & = & 0.01 \\ P(\,\texttt{Red}(\kappa)\ \text{in Fact Set} \mid \texttt{GoodCar}(\kappa)\ \text{query asked}\,) & = & 0.2 \\ P(\,\texttt{Gold}(\kappa)\ \text{in Fact Set} \mid \texttt{GoodCar}(\kappa)\ \text{query asked}\,) & = & 0.8 \end{array}$$

Given these values, it is easy to compute the expected costs of the various strategies [Smi89]: $\mathrm{C}[\,\Theta_{\langle crg\rangle}\,] = 3.772$, $\mathrm{C}[\,\Theta_{\langle cgr\rangle}\,] = 3.178$, $\mathrm{C}[\,\Theta_{\langle rgc\rangle}\,] = 2.96$ and $\mathrm{C}[\,\Theta_{\langle grc\rangle}\,] = 2.36$; hence, the optimal strategy is $\Theta_{\langle grc\rangle}$.[12] Of course, we do not initially know these probability values, and so do not know which strategy is optimal.

We ran a set of experiments to determine whether PALO$'$, starting with $\Theta_{\langle crg\rangle}$, would be able to find a good strategy. We set $\delta = 0.05$ (*i.e.*, a 95% confidence bound), and considered $\epsilon \in \{1.0, 0.5, 0.2, 0.1, 0.05\}$, trying 10 trials for each value.

Using $\epsilon = 1.0$, PALO$'$ quickly found the strategy $\Theta_{\langle rgc\rangle}$, which is a 1.0-local optimum (even though it is not the global optimum). As $\Theta_{\langle rgc\rangle}$ is "$\mathcal{T}^{RO}$-adjacent" to the initial $\Theta_{\langle crg\rangle}$, this meant PALO$'$ performed only one hill-climbing step. PALO$'$ used an average of $|S| \approx 5.3$ sample queries to justify climbing to $\Theta_{\langle rgc\rangle}$, and another on average ≈ 44 queries to realize this strategy was good enough; hence, this total learning process required on average ≈ 49 total queries. For the smaller values of ϵ, PALO$'$ always went from $\Theta_{\langle crg\rangle}$ to $\Theta_{\langle rgc\rangle}$ as before, but then used a second hill-climbing step, to reach the globally-optimal $\Theta_{\langle grc\rangle}$. As would be expected, the number of steps required for each transition were about the same for all values of ϵ (notice that Equation 1 does not involve ϵ): for $\epsilon = 0.5,\ 0.2,\ 0.1,\ 0.05$, PALO$'$ required about 6.3, 6.6, 5.0, 5.4 samples to reach $\Theta_{\langle rgc\rangle}$, and then an additional 31.5, 36.6, 39.0, 29.8 samples to reach $\Theta_{\langle grc\rangle}$.

The major expense was in deciding that this $\Theta_{\langle grc\rangle}$ was in fact an ϵ-local optimum; here, this required an additional 204, 1275, 5101, 20427 samples, respectively. Notice this is not time wasted: the overall "$\Theta_{\langle grc\rangle}$-performance-element–&–PALO$'$-learning-element" system is still solving relevant, user-supplied, problems, and doing so at a cost that is only slightly more expensive than simply running the $\Theta_{\langle grc\rangle}$-performance-element alone, which we now know is an optimal element. In fact, if we ignore the Equation 2 part of PALO's code, we have, in effect, an *anytime algorithm* [BD88, DB88], that simply returns better and better elements over time.

Of course, there are advantages to knowing when we have reached a local optimum: First, we can then switch off the learning part and thereafter simply run this (probably locally) optimal performance element. Second, if we are not happy with the performance of that element, a PALO-variant can then jump to different performance element in another part of the space, and begin hill-climbing from there, possibly using a form of simulated annealing approach [RMt86].

The extended paper [GJ92] presents other experimental data, based on other probability distributions, reduction graphs, parameter values, and so forth. In general, PALO$'$'s performance is similar to the above description: For each setting, PALO$'$ climbs appropriately, requiring successively more samples for each step. Our one surprise was in how conservative our approximations were: using the $\delta = 0.05$ setting, we had anticipated that PALO$'$ would miss (*i.e.*, not reach an ϵ-local optimal) approx-

[11] We decided against using a blocks-world example as it would be more complicated to describe, but would be no more meaningful as we would still have to make up a distribution of problems, specifying how often a problem involves stacking blocks, versus forming arches, versus ...

[12] We continue to identify each strategy with the sequence of database retrievals that it will attempt. Hence, $\Theta_{\langle grc\rangle} = \langle a_5, a_6, a_7, a_3, a_4, a_1, a_2\rangle$.

imately 1 time in 20. However, after several hundred runs, with various settings and graphs, we have found that PALO$'$'s error rate is considerably under this rate. We are now experimenting with variants of PALO$'$ that are less conservative in their estimates, in the hope that they will be correspondingly less sample-hungry. (See also [GD92].)

Finally, while this paper has focused on but a single set of proposed transformations $\mathcal{T}^{RO}$, there are many other transformation sets $\mathcal{T}^{X}$ that can also be used to find an efficient satisficing system; *e.g.*, [Gre92a] discusses a set of transformations that correspond to operator compositions.[13] The "PALO-style" approach is not restricted to speed-up learning; it can also be used to build learning systems that can find performance elements that are nearly optimal in terms of other measures, including *accuracy* [Gre92d] or *categoricity* [Gre92b]; see also [GE91, Gre92c].

5 Conclusion

Comparison with other relevant research: There are many other research projects — both theoretical and empirical — that also address the task of using a set of examples to produce a more efficient performance element. Most of the formal models, however, either deal with learning problems that are far harder than the problems actually attempted by real learning systems (*e.g.*, [GL89, Gre91]) or model only relatively narrow classes of learning algorithms (*e.g.*, [NT88, Coh90]). By contrast, our model is very general and directly relevant to many systems.

There are also a great number of existing LFE systems, and considerable experimental work on the utility problem. Our research is not simply a retrospective analysis of these systems; it also augments that experimental work in the following specific ways. First, we show analytically that one subproblem of the utility problem — the problem of determining if a proposed modification is in fact an improvement — can be (probabilistically) solved *a priori* (*i.e.*, before building that proposed modified system), based on only a polynomial number of test cases. This result analytically confirms previous experimental results. Second, we show that utility analysis can be used to probabilistically guide an incremental learner to a performance element that is essentially a locally optimal PE. (While existing systems have used utility analysis when climbing to elements with superior performance, none have used it to produce elements that are guaranteed to be optimal, in even our weak sense.) Finally, we can use our utility analysis to determine when *not* to learn — *i.e.*, to determine when none of the possible transformations is (likely to be) an improvement. While this aspect of utility analysis has not yet been investigated empirically, it is likely to be important in practice, as it can prevent a learning algorithm from modifying, and therefore possibly degrading, an initial element that happens to already be optimal. The correct action for the learner to take for such initial PEs is simply to leave them unmodified — *i.e.*, not to learn.

The work reported in [GD91, GD92] is perhaps the most similar to ours, in that their system also uses a statistical technique to guarantee that the learned control strategy will be an improvement, based on a utility analysis. Our work differs, as we formally prove specific bounds on the sample complexity, and provide a learning system whose resulting PE$'$ is (with high probability) both superior to the initial PE and a local optimal.

Contributions: Learning from experience (LFE) research is motivated by the assumption that problems are likely to reoccur, meaning it may be worth transforming an initial performance element into a new one that performs well on these problems. Most existing LFE systems actually perform a series of such transformations; in essence searching through a space of possible PEs, seeking an efficient performance element PE$'$. This underlying efficiency measure depends on the overall distribution, which unfortunately is typically unknown. We therefore define an algorithm PALO that can use samples to reliably navigate through this space of possible performance elements, to reach a PE$'$ that is essentially a local optimal. These transformations require certain statistical information; we also describe how to obtain such information efficiently — at a cost that is only minimally more expensive than running a single performance element. Finally, we present a specific application of this algorithm for a particular relevant space of PEs, one for which the task of finding a globally optimal PE is NP-complete, and include empirical data that confirms that the PALO system can work effectively.

[13] This requires a slight variant of the basic PALO algorithm shown in Figure 1: That algorithm assumes that there is a fixed set of neighbors to a given performance element. By contrast, the number of possible macros depends on the number of rules in the system, which grows as more rules are added. This involves certain changes to the PALO algorithm; see [CG91].

References

[BD88] M. Boddy and T. Dean. Solving time dependent planning problems. Technical report, Brown University, 1988.

[BMSJ78] B. Buchanan, T. Mitchell, R. Smith, and C. Johnson, Jr. Models of learning systems. In *Encyclopedia of Computer Science and Technology*, volume 11. Dekker, 1978.

[Bol85] B. Bollobás. *Random Graphs*. Academic Press, 1985.

[CG91] W. Cohen and R. Greiner. Probabilistic hill climbing. In *Proceedings of CLNL-91*, Berkeley, September 1991.

[Che52] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sums of observations. *Annals of Mathematical Statistics*, 23:493–507, 1952.

[CM81] W. Clocksin and C. Mellish. *Programming in Prolog*. Springer-Verlag, New York, 1981.

[Coh90] W. Cohen. Using distribution-free learning theory to analyze chunking. In *Proceeding of CSCSI-90*, 1990.

[DB88] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceedings of AAAI-88*, 1988.

[DeJ88] G. DeJong. AAAI workshop on Explanation-Based Learning. Sponsored by AAAI, 1988.

[GD91] J. Gratch and G. DeJong. A hybrid approach to guaranteed effective control strategies. In *Proceedings of IWML-91*, 1991.

[GD92] J. Gratch and G. DeJong. COMPOSER: A probabilistic solution to the utility problem in speed-up learning. In *Proceedings of AAAI-92*, 1992.

[GE91] R. Greiner and C. Elkan. Measuring and improving the effectiveness of representations. In *Proceedings of IJCAI-91*, 1991.

[GJ92] R. Greiner and I. Jurišica. EBL systems that (almost) always improve performance. Technical report, Siemens Corporate Research, 1992.

[GL89] R. Greiner and J. Likuski. Incorporating redundant learned rules: A preliminary formal analysis of EBL. In *Proceedings of IJCAI-89*, 1989.

[GN87] M. Genesereth and N. Nilsson. *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1987.

[GO91] R. Greiner and P. Orponen. Probably approximately optimal derivation strategies. In *Proceeding of KR-89*, 1991.

[Gol79] A. Goldberg. An average case complexity analysis of the satisfiability problem. In *Proceedings of CADE-79*, 1979.

[Gre91] R. Greiner. Finding the optimal derivation strategy in a redundant knowledge base. *Artificial Intelligence*, 50(1):95–116, 1991.

[Gre92a] R. Greiner. Effective operator composition. Technical report, Siemens Corporate Research, 1992.

[Gre92b] R. Greiner. Learning near optimal horn approximations. In *Proceedings of Knowledge Assimilation Symposium*, Stanford, 1992.

[Gre92c] R. Greiner. Probabilistic hill-climbing: Theory and applications. In *Proceedings of CSCSI-92*, 1992.

[Gre92d] R. Greiner. Producing more accurate representational systems. Technical report, Siemens Corporate Research, 1992.

[Kel87] Richard M. Keller. Defining operationality for explanation-based learning. In *Proceedings of AAAI-87*, 1987.

[LNR87] J. Laird, A. Newell, and P. Rosenbloom. SOAR: An architecture of general intelligence. *Artificial Intelligence*, 33(3), 1987.

[MCK+89] S. Minton, J. Carbonell, C. Knoblock, D. Kuokka, O. Etzioni, and Y. Gil. Explanation-based learning: A problem solving perspective. *Artificial Intelligence*, 40(1-3):63–119, September 1989.

[Min88a] S. Minton. *Learning Search Control Knowledge: An Explanation-Based Approach.* Kluwer Academic Publishers, Hingham, MA, 1988.

[Min88b] S. Minton. Quantitative results concerning the utility of explanation-based learning. In *Proceedings of AAAI-88*, 1988.

[Mit82] T. Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203–26, March 1982.

[Nil80] N. Nilsson. *Principles of Artifical Intelligence.* Tioga Press, Palo Alto, 1980.

[NT88] B. Natarajan and P. Tadepalli. Two frameworks for learning. In *Proceedings of IML-88*, 1988.

[OG90] P. Orponen and R. Greiner. On the sample complexity of finding good search strategies. In *Proceedings of COLT-90*, 1990.

[RMt86] D. Rumelhart, J. McClelland, and the PDP Research Group, editors. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1: Foundations. The MIT Press, Cambridge, 1986.

[SER91] A. Segre, C. Elkan, and A. Russell. A critical look at experimental evaluations of EBL. *Machine Learning Journal*, 6(2), 1991.

[SK75] H. Simon and J. Kadane. Optimal problem-solving search: All-or-none solutions. *Artificial Intelligence*, 6:235–247, 1975.

[Smi89] D .Smith. Controlling backward inference. *Artificial Intelligence*, 39(2):145–208, 1989.