Probabilistic Hill-Climbing

William W. Cohen
AT&T Bell Laboratories
Murray Hill, NJ 07974
wcohen@research.att.com

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

Dale Schuurmans
Department of Computer Science
University of Toronto
Toronto, Ontario M5S 1A4
dale@cs.toronto.edu

Abstract: Many learning tasks involve searching through a discrete space of performance elements, seeking an element whose future utility is expected to be high. As the task of finding the global optimum is often intractable, many practical learning systems use simple forms of hill-climbing to find a locally optimal element. However, hill-climbing can be complicated by the fact that the utility value of a performance element can depend on the distribution of problems, which typically is unknown. This paper formulates the problem of performing hill-climbing search in settings where the required utility values can only be estimated on the basis of their performance on random test cases. We present and prove correct an algorithm that returns a performance element that is arbitrarily close to a local optimum with arbitrarily high probability.

*Much of this work was performed at the University of Toronto, supported by the Institute for Robotics and Intelligent Systems and by an operating grant from the National Science and Engineering Research Council of Canada. All three author gratefully acknowledge receiving many helpful comments from David Mitchell and the anonymous reviewers.

1 Introduction

Many learning tasks can be viewed as a search through a space of possible performance elements [BMSJ78], seeking an element that is optimal, under some utility measure. For example, many inductive systems seek optimal classification functions that correctly label as many examples as possible [BFOS84]; and many speed-up learning systems try to produce optimally efficient problem solving systems [DeJ88, MCK+89, LNR87]. In each case, the utility of the candidate performance elements is defined as the value of some scoring function, averaged over the natural distribution of samples (queries, tests, problems, ...) that the system will encounter.

There are (at least) two potential problems with implementing such a learning system: First, the task of identifying the globally optimal element is intractable for many spaces; *cf.*, [Hau88], [Gre91]. A common solution to this problem is to use a hill-climbing approach to find a *locally* optimal solution. Two well-known inductive learning systems that use this approach are ID3 [Qui86], which uses a greedy technique to reduce the expected entropy of a decision tree, and BACKPROP [Hin89]. In addition, many speed-up learning methods can also be viewed as using hill-climbing to improve the expected performance of a problem solver: this view is clearly articulated in [GD91].

Unfortunately, even finding a locally optimal element can be problematic as it depends critically on the query distribution, which is often unknown. This paper thus addresses the following question: to what degree can hill-climbing search be approximated if the utility function is only estimated by random sampling? Our main result is a positive one, in the form of an algorithm PALO[1] that with high probability returns an element that is approximately locally optimal.

Section 2 motivates the use of "expected utility" as a quality metric for performance elements. Section 3 then defines the general PALO algorithm, which incrementally produces a series of performance elements $\mathrm{PE}_1, \ldots, \mathrm{PE}_m$ such that, with high confidence, each PE_{i+1}

[1]for "Probably Approximately Locally Optimal". As the name suggests, this is related to standard "Probably Approximately Correct", or PAC, learning [Val84].

is statistically likely to be an incremental improvement over PE_i and the performance of the final element PE_m is a local optimal in the space searched by the learner. This uses an analytic tool, based on mathematical statistics, for evaluating whether the result of a proposed modification is better than the original PE; this tool can be viewed as mathematically rigorous version form of [Min88]'s "utility analysis". The conclusion sketches various applications of this technique.

2 Framework

We assume as given a (possibly infinite) set of performance elements $\mathcal{PE}$, where each PE $\in \mathcal{PE}$ is a function that returns an answer to each given query (or problem or goal or ...). For example, in context of seeking a good classification function, each PE $\in \mathcal{PE}$ may be a particular decision tree [Qui86], or a specific boolean formula [Hau88], or a credulous prioritized default theory [Gre92]. Within the context of speed-up learning, [GJ92] views each PE $\in \mathcal{PE}$ as a particular PROLOG program, where all of the programs in $\mathcal{PE}$ include exactly the same clauses, but differ in the order of these clauses.

Let $\mathcal{Q} = \{q_1, q_2, \ldots\}$ be the (possibly countably infinite) set of possible queries, and $c\colon \mathcal{PE} \times \mathcal{Q} \mapsto \Re$ be the utility scoring function: $c(\mathrm{PE}, q)$ indicates how well the element PE does at solving q. For example, in a classification task, $c(\mathrm{PE}, q)$ may quantify the accuracy of PE's answer to the problem q; or in speed-up learning, the (negative of the) time PE requires to solve q. (Higher scores are better.) We require only that the value of $c(\mathrm{PE}, q)$ be in some bounded interval — *i.e.*,

$$\textit{for all } \mathrm{PE} \in \mathcal{PE},\ q \in \mathcal{Q}\colon\ c_\ell \ \leq\ c(\mathrm{PE}, q)\ \leq\ c_\ell + \lambda \tag{1}$$

for some constants $c_\ell \in \Re$ and $\lambda \in \Re^+$.

We can use this scoring function to determine which performance element is best for a single problem. Our performance elements, however, must be able to solve an entire ensemble of problems; and in general, no single element will be optimal for all possible problems. We therefore consider how well each element will perform over the distribution of problems that it will encounter, and prefer the element of $\mathcal{PE}$ whose average is best. We model the distribution using a probability function, $Pr\colon \mathcal{Q} \mapsto [0,1]$, where $Pr(q_i)$ denotes the probability that the problem q_i is selected. (That is, we assume that problems are selected at random, according to this arbitrary, but stationary, distribution.)

The utility measure used to evaluate an element PE is, accordingly, the expected value of $c(\mathrm{PE}, \cdot)$ with respect to this distribution, written $\mathrm{C}[\mathrm{PE}]$:

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

Our underlying challenge is to find the performance element whose expected utility is maximal. As mentioned above, there are two problems: First, the probability 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, that side-steps the above problems by using a set of sample queries to estimate the distribution, and by hill-climbing efficiently from a given initial PE_1 to one that is, with high probability, close to a local optimum. Subsection 3.1 first summarizes PALO's code, then states the fundamental theorem that specifies PALO's functionality, and presents the foundations for the proof. (The complete proof appears in the appendix.) Subsection 3.2 then discusses several extensions to this algorithm.

3.1 PALO's Behavior and Code

As shown in Figure 1, PALO takes as arguments an initial performance element $\mathrm{PE}_1 \in \mathcal{PE}$; error and confidence parameters $\epsilon, \delta > 0$; a scoring function $c(\cdot,\cdot)$ used to specify the expected utility; and a (possibly infinite) set of possible transformations $\mathcal{T} = \{\tau_j\}$, where

---

Algorithm PALO(PE_1, ϵ, δ, $c(\cdot,\cdot)$, $\mathcal{T}$)

For $j \leftarrow 1..\infty$ do

$\quad\mathcal{T}(\mathrm{PE}_j) \leftarrow \{\tau_k(\mathrm{PE}_j)\}_k$

$\quad$Take n_j samples, $S \leftarrow \{q_1, q_2, q_3, ..., q_{n_j}\}$, where

$$n_j \leftarrow \left\lceil \frac{8\lambda^2}{\epsilon^2} \ln \frac{j^2\,|\mathcal{T}(\mathrm{PE}_j)|\,\pi^2}{3\,\delta} \right\rceil \tag{2}$$

$\quad$For each $\mathrm{PE}' \in \mathcal{T}(\mathrm{PE}_j)$

$\quad\quad$let $\Delta[\mathrm{PE}', \mathrm{PE}_j, S] = \frac{1}{n_j}\sum_{q\in S} c(\mathrm{PE}', q) - c(\mathrm{PE}_j, q)$

$\quad$**If** $\exists\, \mathrm{PE}' \in \mathcal{T}(\mathrm{PE}_j)$ such that $\Delta[\mathrm{PE}', \mathrm{PE}_j, S] \geq \frac{\epsilon}{2}$

$\quad$**then** let $\mathrm{PE}_{j+1} \leftarrow \mathrm{PE}'$

$\quad$**else** [Here, *for all* $\mathrm{PE}' \in \mathcal{T}(\mathrm{PE}_j)$: $\Delta[\mathrm{PE}', \mathrm{PE}_j, S] < \frac{\epsilon}{2}$]

$\quad\quad$return PE_j

End For

End PALO

Figure 1: Code for PALO

each τ_j maps one performance element to another. Examples of such transformations include flipping the parity of a variable within a boolean formula, splitting a node in a decision tree [BFOS84], reordering the clauses in PROLOG program [GJ92] or adding a new macro rule to a problem solver [GD91].

PALO uses a set of sample queries drawn at random from the $Pr(\cdot)$ distribution to climb incrementally from the initial PE_1 to a new $\mathrm{PE}_2 = \tau_i(\mathrm{PE}_1)$ using one $\tau_i \in \mathcal{T}$, then onto a third $\mathrm{PE}_3 = \tau_j(\mathrm{PE}_2)$ using another $\tau_j \in \mathcal{T}$, and so on. PALO terminates on finding a locally-optimal PE_m: here, no single transformation can convert PE_m into a significantly better performance element. The theorem below formally specifies PALO's behavior; its proof appears in the appendix.

Theorem 1 *The* PALO(PE_1, ϵ, δ, $c(\cdot,\cdot)$, $\mathcal{T}$) *algorithm incrementally produces a series of performance elements* $\mathrm{PE}_1, \mathrm{PE}_2, \ldots, \mathrm{PE}_m$ *— requiring only a polynomial number of samples at each stage — such that, with probability at least* $1-\delta$*,*

1. *the expected utility of each performance element is strictly better than its predecessors i.e.,*

$$\text{for all } 1 \le i < j \le m: \ \mathrm{C}[\,\mathrm{PE}_j\,] \ > \ \mathrm{C}[\,\mathrm{PE}_i\,]; \quad \textit{and}$$

2. *(if* PALO *terminates) the final performance element returned by* PALO, PE_m*, is an "*ϵ*-local optimum" — i.e.,*

$$\text{for all } \tau_j \in \mathcal{T}: \quad \mathrm{C}[\,\mathrm{PE}_m\,] \ \ge \ \mathrm{C}[\,\tau_j(\mathrm{PE}_m)\,] - \epsilon \qquad \Box.$$

To give some intuitions regarding the statistical methods used in the proof: PALO climbs 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 $\mathrm{C}[\,\mathrm{PE}_{j+1}\,] > \mathrm{C}[\,\mathrm{PE}_j\,]$. Towards specifying this confidence, define

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

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

Let $Y_n = \Delta[\mathrm{PE}_\alpha,\, \mathrm{PE}_\beta,\, \{q_i\}_{i=1}^n\,] = \frac{1}{n}\sum_{i=1}^{n} c(\mathrm{PE}_\alpha,\, q_i) - c(\mathrm{PE}_\beta,\, q_i)$ be the sample mean over n sample queries. This average tends to the true population mean $\mu = \mathrm{C}[\,\mathrm{PE}_\alpha\,] - \mathrm{C}[\,\mathrm{PE}_\beta\,]$ 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 γ increases. Formally,[2]

$$Pr[\,Y_n > \mu + \gamma\,] \ \le \ e^{-\frac{n}{2}\left(\frac{\gamma}{\lambda}\right)^2} \qquad Pr[\,Y_n < \mu - \gamma\,] \ \le \ e^{-\frac{n}{2}\left(\frac{\gamma}{\lambda}\right)^2} \tag{3}$$

using the λ from Equation 1, which bounds the size of $c(\mathrm{PE},\, q)$'s range.

Based on these equations, PALO uses the values of $\Delta[\,\mathrm{PE}',\, \mathrm{PE}_j, S\,]$ to determine both how confident we should be that $\mathrm{C}[\,\mathrm{PE}'\,] > \mathrm{C}[\,\mathrm{PE}_j\,]$ 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; see the proof in Appendix A.

---

[2] See [Bol85, p. 12]. These are also called "Hoeffding's Inequalities". *N.b.*, these inequalities holds for essentially *arbitrary distributions*, not just normal distributions, subject only to the minor constraint that the random variables $\{d_i\}$ are bounded.

3.2 Notes and Extensions to PALO

Note#1. A "0-local optimum" corresponds exactly to the standard notion of local optimum; hence our "ϵ-local optimum" generalizes local optimality. Notice also that we can expect PALO's output PE_m to be a real local optimum if the difference in cost between every two distinct performance elements, PE and $\tau(\mathrm{PE})$, is always larger than ϵ; *i.e.*, if

$$\textit{for all } \mathrm{PE} \in \mathcal{PE},\ \tau \in \mathcal{T}:\ \tau(\mathrm{PE}) \neq \mathrm{PE} \quad \Rightarrow \quad |\mathrm{C}[\,\mathrm{PE}\,] - \mathrm{C}[\,\tau(\mathrm{PE})\,]| \ > \ \epsilon\ .$$

Thus, for sufficiently small values of ϵ, PALO will always produce a *bona fide* local optimum.

Note#2. 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 elements are better than earlier ones; *i.e.*, $i > j$ means $\mathrm{C}[\,\mathrm{PE}_i\,] > \mathrm{C}[\,\mathrm{PE}_j\,]$ with high probability. PALO differs from standard anytime algorithms by terminating on reaching a point of diminishing returns.

Note#3. Although we know the number of samples required 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 $\mathcal{T}$ transformations. However, it is easy to see that PALO will terminate with probability 1 if the space of systems is finite. (This is based on the observation that the only way PALO can fail to terminate is if it cycles infinitely often — thinking first that some PE_j is better than PE_i and so switching to it, and later, thinking that PE_i is better, switching back. From the proof in Appendix A, the probability that this will happen infinitely often goes to 0.)

Note#4. Equation 2 (from Figure 1) is meaningful only if the number of neighbors of each performance element PE is finite; that is, if $\mathcal{T}(\mathrm{PE}) = \{\tau_k(\mathrm{PE}) | \tau_k \in \mathcal{T}\}$ is finite.

This constraint is trivially satisfied if the total number of transformations $|\mathcal{T}|$ is finite. It also holds in certain important situations where $\mathcal{T}$ is infinite. Consider, for example, a typical EBL (Explanation-Based Learning) system that uses operator composition to transform performance elements [GD91]: Given an initial performance element PE_1 with n operators, PALO can consider $\binom{n}{2}$ distinct new performance elements, each formed by adding to PE_1 a new $n+1^{st}$ operator that is the result of composing two of PE_1's existing n operators. Ater it climbs to one of these elements, call it PE_2, PALO can then climb from this PE_2 to a yet newer PE_3 by adding to PE_2 a new operator formed by composing two of PE_2's $n+1$ operators, and so forth.

As each possible operator corresponds to a finite combination of some set of PE_1's operators, only a countable number of operators can ever be formed; call them $\mathcal{O} = \{o_i\}$. We can then define $\mathcal{T} = \{\tau_{o_i,o_j}\}_{i,j}$ to be the total set of possible transformations, where

$$\tau_{o_i,o_j}(\mathrm{PE}) \quad = \quad \begin{cases} \mathrm{PE} \ + \ `o_i \cdot o_j\text{'} & \text{if PE includes both } o_i \text{ and } o_j \\ \mathrm{PE} & \text{otherwise} \end{cases}$$

where $\mathrm{PE} \ + \ `o_i \cdot o_j\text{'}$ refers to the performance element that includes all of PE's operators plus the newly composed operator $o_i \cdot o_j$. Even though there are an infinite set of such transformations, notice only a finite number will map any particular element to a different element; hence $|\mathcal{T}(\mathrm{PE})| \ll \infty$ for every performance element PE.

Note#5. There are many other variants of the PALO algorithm that may be more efficient in some situations. First, the sample complexity of each step (which is the n_j from Equation 2) involves the constant λ defined in Equation 1. We can in general replace the λ in Equation 2 by

$$\Lambda_{max}(\mathrm{PE}_j) \quad \stackrel{def}{=} \quad \frac{1}{2}\max\{\ \Lambda(\tau_k(\mathrm{PE}_j), \mathrm{PE}_j) \mid \tau_k \in \mathcal{T}\ \},$$

where $\Lambda(\mathrm{PE}_i, \mathrm{PE}_j)$ is the actual range of the possible values of $\Delta[\,\mathrm{PE}_i,\ \mathrm{PE}_j,\ q\,] = c(\mathrm{PE}_i,\ q) - c(\mathrm{PE}_j,\ q)$ over the set of possible problems $q \in \mathcal{Q}$; *i.e.*, there is a $c_{ij} \in \Re$ such that

$$\textit{for all } q \in \mathcal{Q}\colon\ c_{ij} \ \le\ \Delta[\,\mathrm{PE}_i,\ \mathrm{PE}_j,\ q\,] \ \le\ c_{ij} + \Lambda(\mathrm{PE}_i, \mathrm{PE}_j)$$

By inspection, $\Lambda[\,\mathrm{PE}_i,\ \mathrm{PE}_j\,]$ is necessarily less than 2λ; hence $\Lambda_{max}(\mathrm{PE}_j)$ is always under λ. [GJ92] presents an example of a situation where $\Lambda_{max}(\mathrm{PE}_j) \ll \lambda$.

Second, [Gre92] presents a variant of our PALO algorithm, that differs by taking samples one at a time, rather than in batches of size n_j. As that algorithm can consider climbing to a (probabilistically) better element, or terminating, after seeing each individual sample, it can potentially require few samples than our algorithm.

Finally, each of these PALO systems must compute the values of $\Delta[\mathrm{PE}',\ \mathrm{PE}_j, S]$ for each $\mathrm{PE}' \in \mathcal{T}(\mathrm{PE}_j)$. The obvious way of obtaining these values involves constructing and then running each element in the set $\mathcal{T}(\mathrm{PE}_j)$ on each sample problem $q \in S$, and comparing this with the cost of running PE_j on S; this requires a total of $|S| \times (|\mathcal{T}(\mathrm{PE}_j)|+1)$ "calls" of some performance element on some sample. Alternatively, we could run only the single PE_j on the samples in S, and use an efficient analytic technique to estimate changes in performance; *i.e.*, to approximate the values of $\Delta[\tau(\mathrm{PE}_j),\ \mathrm{PE}_j, S]$ for each $\tau \in \mathcal{T}$. This reduces the total number of "PE on q" calls to $|S|$; see [GJ92] for an example of this approach.

Note#6. The samples that PALO uses may be supplied by a user of the performance system who is simply posing questions relevant to his current applications; in this case, PALO is unobtrusively gathering statistics as the user is solving his own problems [MMS85]. PALO must then compare the behavior of the current performance element with that of each alternative element; as discussed in Note 5 above, this can be done efficiently in some situations. Here, the total cost of the overall system, which both solves performance problems and learns by hill-climbing to successively better performance elements, will be only marginally more than the cost of only running the performance element to simply solve the performance problems.

Notice we are using these user-provided samples to estimate the average utility values of the performence elements, over the distribution of problems that the element will actually encounter. 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].

4 Conclusion

Applications: Several related papers present specific applications of this PALO system. [GJ92] illustrates how this approach fits into the framework of "explanation-based learning" systems, and in particular, that how this analysis extends and formalizes [Min88]'s "utility analysis". It also presents empirical evidence that this system does work effectively. Other papers, notably [Gre92], demonstrate the generality of this approach by presenting various other instantiations of the PALO system, each using its own set of transformations to find a near-optimal element within a particular set of performance elements, where optimality is defined in terms of efficiency, accuracy or categoricity.

Contributions: This report first poses two of the problems that can arise in learning systems that try to identify a performance element whose expected cost is optimal [Vap82, Hau90]: *viz.*, that the distribution is usually unknown and that finding a globally optimal performance element can be intractable. It then presents the PALO algorithm which addresses these shortcomings by using statistical techniques to approximate the distribution and by hill-climbing, efficiently, to produce a locally optimal element.

A Proof of Theorem 1

Theorem 1 *The* PALO(PE_1, ϵ, δ, $c(\cdot,\cdot)$, $\mathcal{T}$) *algorithm incrementally produces a series of performance elements* $\mathrm{PE}_1, \mathrm{PE}_2, \ldots, \mathrm{PE}_m$ *— requiring only a polynomial number of samples at each stage — such that, with probability at least* $1-\delta$,

1. *the expected utility of each performance element is strictly better than its predecessors i.e.,*

$$\text{for all } 1 \le i < j \le m: \; \mathrm{C}[\,\mathrm{PE}_j\,] \;>\; \mathrm{C}[\,\mathrm{PE}_i\,]; \quad \textit{and}$$

2. *(if* PALO *terminates) the final performance element returned by* PALO, PE_m, *is an "ϵ-local optimum" — i.e.,*

$$\text{for all } \tau_j \in \mathcal{T}: \quad \mathrm{C}[\,\mathrm{PE}_m\,] \;\geq\; \mathrm{C}[\;\tau_j(\mathrm{PE}_m)\,] \;-\; \epsilon \qquad \Box.$$

Proof: The n_j in Equation 2 is the number of samples required for each iteration; this quantity is clearly polynomial in $|\mathcal{T}(\mathrm{PE}_j)|$ and the other relevant parameters, including $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$.

Now to prove Parts 1 and 2: Consider first a single interation of the PALO algorithm. Notice there are two ways of making a mistake:

1. If some $\mathrm{PE}' \in \mathcal{T}(\mathrm{PE}_j)$ appears to be better than PE_j but is not; or

2. If some $\mathrm{PE}' \in \mathcal{T}(\mathrm{PE}_j)$ is more than ϵ better than PE_j, but appears not to be.

Let

$$\begin{aligned}
p_1^j &= Pr\left[\,\exists\,\mathrm{PE}' \in \mathcal{T}(\mathrm{PE}_j):\ \Delta[\mathrm{PE}',\,\mathrm{PE}_j, S] \geq \tfrac{\epsilon}{2} \ \text{ and } \mathrm{C}[\,\mathrm{PE}'\,] < \mathrm{C}[\,\mathrm{PE}_j\,]\,\right] \\
p_2^j &= Pr\left[\,\exists\,\mathrm{PE}' \in \mathcal{T}(\mathrm{PE}_j):\ \Delta[\mathrm{PE}',\,\mathrm{PE}_j, S] < \tfrac{\epsilon}{2} \ \text{ and } \mathrm{C}[\,\mathrm{PE}'\,] > \mathrm{C}[\,\mathrm{PE}_j\,] + \epsilon\,\right]
\end{aligned}$$

be the respective probabilities of these events. Now observe that

$$\begin{aligned}
p_1^j &\leq \sum_{\mathrm{PE}' \in \mathcal{T}(\mathrm{PE}_j)} Pr\left[\Delta[\mathrm{PE}',\,\mathrm{PE}_j, S] \geq \frac{\epsilon}{2} \ \text{ and } \mathrm{C}[\,\mathrm{PE}'\,] - \mathrm{C}[\,\mathrm{PE}_j\,] < 0\right] \\
&\leq \sum_{\mathrm{PE}' \in \mathcal{T}(\mathrm{PE}_j)} e^{-\frac{n_j}{2}\left(\frac{\frac{\epsilon}{2}}{\lambda}\right)^2} \qquad (4) \\
&\leq |\mathcal{T}(\mathrm{PE}_j)| e^{-\frac{1}{2}\left(\frac{8\lambda^2}{\epsilon^2}\ln\frac{j^2\,|\mathcal{T}(\mathrm{PE}_j)|\,\pi^2}{3\,\delta}\right)\left(\frac{\epsilon}{2\lambda}\right)^2} \\
&= |\mathcal{T}(\mathrm{PE}_j)| \frac{3\,\delta}{j^2\,|\mathcal{T}(\mathrm{PE}_j)|\,\pi^2} \quad = \quad \frac{1}{j^2}\frac{3\,\delta}{\pi^2}
\end{aligned}$$

Line 4 uses Chernoff bounds (Equation 3) and the observation that the expected value of each $\Delta[\mathrm{PE}', \mathrm{PE}_j, q]$ is $\mathrm{C}[\,\mathrm{PE}'\,] - \mathrm{C}[\,\mathrm{PE}_j\,]$. Similarly,

$$\begin{aligned} p_2^j &\leq \sum_{\mathrm{PE}' \in \mathcal{T}(\mathrm{PE}_j)} Pr\left[\Delta[\mathrm{PE}', \mathrm{PE}_j, S] \leq \frac{\epsilon}{2} \text{ and } \mathrm{C}[\,\mathrm{PE}'\,] - \mathrm{C}[\,\mathrm{PE}_j\,] > \epsilon\right] \\ &\leq \sum_{\mathrm{PE}' \in \mathcal{T}(\mathrm{PE}_j)} e^{-\frac{n_j}{2}\left(\frac{\frac{\epsilon}{2}}{\lambda}\right)^2} \leq \frac{1}{j^2}\frac{3\,\delta}{\pi^2} \end{aligned}$$

Hence, the probability of ever making either mistake at any iteration is under

$$\sum_{j=1}^{\infty} p_1^j + p_2^j \leq \sum_{j=1}^{\infty} 2\frac{1}{j^2}\frac{3\,\delta}{\pi^2} = \delta\frac{6}{\pi^2}\sum_{j=1}^{\infty}\frac{1}{j^2} = \delta\frac{6}{\pi^2}\frac{\pi^2}{6} = \delta$$

as desired. □.

References

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

[BFOS84] L. Breiman, J. Friedman, JR. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[BMSJ78] Bruce G. Buchanan, Thomas M. Mitchell, Reid G. Smith, and C. R. 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.

[Che52] Herman 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.

[DB88] Thomas Dean and Mark Boddy. An analysis of time-dependent planning. In *Proceedings of AAAI-88*, pages 49–54, August 1988.

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

[GD91] Jonathan Gratch and Gerald DeJong. A hybrid approach to guaranteed effective control strategies. In *Proceedings of IWML-91*, pages 509–13, Evanston, Illinois, June 1991.

[GJ92] Russell Greiner and Igor Jurišica. A statistical approach to solving the EBL utility problem. In *Proceedings of AAAI-92*, San Jose, 1992.

[Gol79] A. Goldberg. An average case complexity analysis of the satisfiability problem. In *Proceedings of the 4th Workshop on Automated Deduction*, pages 1–6, Austin, TX, 1979.

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

[Gre92] Russell Greiner. Probabilistic hill-climbing: Theory and applications. In *Proceedings of CSCSI-92*, Vancouver, June 1992.

[Hau88] David Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, pages 177–221, 1988.

[Hau90] David Haussler. Decision theoretic generalizations of the PAC model for neural net and other learning applications. Technical Report UCSC-CRL-91-02, Department of Computer Science, UC Santa Cruz, December 1990.

[Hin89] Geoff Hinton. Connectionist learning procedures. *Artificial Intelligence*, 40(1-3):185–234, September 1989.

[Kel87] Richard M. Keller. Defining operationality for explanation-based learning. In *Proceedings of AAAI-87*, pages 482–87, Seattle, July 1987.

[LNR87] John E. Laird, Allan Newell, and Paul S. Rosenbloom. SOAR: An architecture of general intelligence. *Artificial Intelligence*, 33(3), 1987.

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

[Min88] Steven Minton. *Learning Search Control Knowledge: An Explanation-Based Approach.* Kluwer Academic Publishers, Hingham, MA, 1988.

[MMS85] Thomas M. Mitchell, Sridhar Mahadevan, and Louis I. Steinberg. LEAP: A learning apprentice for VLSI design. In *Proceedings of IJCAI-85*, pages 573–80, Los Angeles, August 1985.

[Qui86] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[Val84] Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–42, 1984.

[Vap82] V.N. Vapnik. *Estimation of Dependences Based on Empirical Data.* Springer-Verlag, New York, 1982.