Appears in the
Proceedings of the Eleventh ACM Symposium on Principles of Database Systems (PODS92),
San Diego, June 1992.

Learning Efficient Query Processing Strategies

Russell Greiner*
Siemens Corporate Research
Princeton, NJ 08540-6632
greiner@learning.siemens.com (609) 734-3627

Abstract

A query processor QP uses the rules in a rule base to reduce a given query to a series of attempted retrievals from a database of facts. The QP's *expected cost* is the average time it requires to find an answer, averaged over its anticipated set of queries. This cost depends on the QP's *strategy*, which specifies the order in which it considers the possible rules and retrievals. This paper provides two related learning algorithms, PIB and PAO, for improving the QP's strategy, *i.e.*, for producing new strategies with lower expected costs. Each algorithm first monitors the QP's operations over a set of queries, observing how often each path of rules leads to a sufficient set of successful retrievals, and then uses these statistics to suggest a new strategy. PIB hill-climbs to strategies that are, with high probability, successively better; and PAO produces a new strategy that probably is approximately optimal. We describe how to implement both learning systems unobtrusively, discuss their inherent time and space complexities, and use methods from mathematical statistics to prove their correctness. We also discuss additional applications of these approaches to several other database tasks.

1 Introduction

A query processor QP uses the rules in a rule base to reduce a given query to a series of attempted retrievals from a database of literals. As this can be a very slow process, there has been a great deal of work on query optimization for knowledge bases [BR86, Ull89, Smi89]; this flurry of research activity has produced a great deal of theoretical work and a number of prototype systems (*e.g.*, NAIL! [Ull89], LDL [Zan88], DEDGIN* [LV89]).

In general, these systems attempt to produce a QP with a low *expected cost*, where expected cost measures the average time QP requires to find an answer, averaged over its anticipated set of queries. This cost depends on the QP's *strategy* — *i.e.*, on the order in which it considers performing database retrievals and following rules to produce new subgoals. Our work focuses on improvements that modify a QP's strategy,[1] producing a new strategy with a lower expected cost.

This paper offers a new approach to query optimization for knowledge bases: in particular, it presents two methods, PIB and PAO,[2] for *learning* better strategies. These learning algorithms each monitor a QP as it solves a set of queries, collecting certain simple statistics. Each then uses these statistics to modify the QP's strategy, producing a new strategy whose performance will, with provably high probably, be better on subsequent queries. PIB is an *anytime algorithm* [BD88, DB88] that uses these statistics to hill-climb (with high probability) to successively better strategies; in particular, given any $\delta > 0$ and set of possible modifications, it selects and applies a sequence of modifications to a given initial strategy to produce successive strategies that are, with probability greater than $1-\delta$, successive improvements. PAO produces a new strategy that probably is approximately optimal; *i.e.*, given any ϵ, $\delta > 0$, it uses the statistics from a specified number of samples to identify, with probability greater than $1-\delta$, a strategy whose cost is within ϵ of the cost of the optimal strategy. We use

*Much of this work was performed at the Department of Computer Science in the University of Toronto, where it was supported by the Institute for Robotics and Intelligent Systems and by an operating grant from Canada's Natural Science and Engineering Research Council. I am grateful to Manolis Koubarakis for his help in writing a preliminary version of this article, and to Vinay Chaudhri, Igor Jurišica, Alberto Mendelzon and John Mylopoulos, for their general comments on this research.

[1] We can also view this as finding a new QP′ that differs from the original QP only by using this alternative strategy. For reasons discussed below, we will freely interchange these two descriptions.

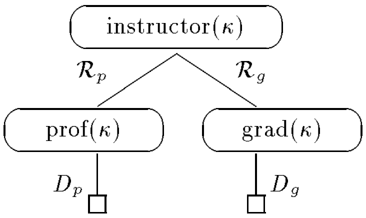
[2] for "Probably Incrementally Better" and "Probably Approximately Optimal". This systems are related to the work on "Probably Approximately Correct" learning; *cf.*, [Val84].

Rule Base

$\mathcal{R}_p$:	`instructor(X):-prof(X).`
$\mathcal{R}_g$:	`instructor(X):-grad(X).`

Database DB_1

```
prof(russ)
grad(manolis)
```



(Each $\mathcal{R}_j$ represents a rule; Each D_i represents a retrieval.)

Figure 1: A Rule Base and its associated inference graph G_A + Database DB_1

methods from mathematical statistics to prove the correctness of both learning systems. We also show that these algorithms are both time- and space- efficient — *i.e.*, run in polynomial time for many important classes of knowledge bases, and require very little additional storage.

Of course, this is not the first use of statistics in database systems. Existing relational systems (*e.g.*, system R [SAC+79]) keep statistics about the database and use them to determine the costs of different access methods. They do not, however, keep statistics about the queries, nor do they use such information to improve the query processing system. [HC76] presents a self-adaptive database management system that keeps query, update and domain selectivity statistics and utilizes them to automatically adjust the physical organization of its database. [LN90] uses a similar statistical method (adaptive sampling) to estimate the size of queries. However, none of these systems use statistics to optimize the query processing by modifying the QP's strategy.

Section 2 first uses a simple example to motivate and illustrate our approach. Sections 3 and 4 then present our main results: the PIB and PAO algorithms. Section 5 provides general comments about these algorithms, including a description of their various applications to database systems.

2 Motivation, Example and Definitions

This work deals with knowledge bases that contain a collection of ground atomic facts (the *database*) and a collection of Datalog rules (the *rule base*)[3]; *cf.*, Figure 1. While our results will generalize, we will assume a *query* is an atomic formula within this paper. A *query form* is an expression of the form q^α where q denotes a n-ary relation and α is an n-tuple from $\{\mathtt{b}, \mathtt{f}\}^n$, where the i^{th} element is `b` if the query's i^{th} argument is `b`ound, and is `f` if this i^{th} argument is `f`ree.

The objective of our query optimization is to find an efficient strategy for executing queries corresponding to a given query form. As an example, consider using the *inference graph*[4] in Figure 1 to answer to queries of the form $\mathtt{instructor}^{\langle \mathtt{b} \rangle}$ — *e.g.*, "`instructor(manolis)?`". There are two possible top-down strategies: The strategy $\Theta_1 = \langle \mathcal{R}_p\, D_p\, \mathcal{R}_g\, D_g \rangle$ follows the $\mathcal{R}_p$ rule down to D_p, representing the attempted `prof(manolis)` database retrieval. If this retrieval succeeds, the query processor returns "yes" and terminates. If not, Θ_1 then pursues the other edge of the graph (involving $\mathcal{R}_g$ and D_g) and returns either "yes" or "no", based on the success of the second retrieval, `grad(manolis)`. An alternative to this strategy, $\Theta_2 = \langle \mathcal{R}_g\, D_g\, \mathcal{R}_p\, D_p \rangle$, first follows $\mathcal{R}_g$ to D_g and then, if necessary, $\mathcal{R}_p$ to D_p. Which of the two strategies is better?

A good strategy is one that performs well in practice. As we only consider strategies that are guaranteed to find an answer to each query, if one exists, all strategies are equally accurate. We can therefore consider the efficiency of the different strategies. To quantify this, assume there is a distribution of queries that our QP will be asked to solve. We can then define a strategy's *expected cost* as the average cost required to find a solution to a query, averaged over this distribution of anticipated queries. This value depends on the computational cost of each rule reduction and database retrieval, and on the likelihoods that each particular retrieval will succeed — *i.e.*, on the conditional probability that a specific requested proposition will be in the knowledge base, given a query drawn from this distribution. For the knowledge base in Figure 1, assume that each reduction (*i.e.*, following a rule from goal to a subgoal) and each atomic retrieval costs 1 unit, and imagine we knew that 60% of the queries are `instructor(russ)`, 15% are `instructor(manolis)`, and the remaining 25% are `instructor(fred)`. This means the D_p retrieval will succeed 60% of the time and D_g, 15%. We can now compute Θ_1's expected cost, $C[\Theta_1]$: it is the cost of the initial $\langle \mathcal{R}_p\, D_p \rangle$ subpath, plus the probability that this

[3] Hence, we can represent all of the information as a conjunction of function-free clauses, where each clause containing exactly one positive literal.

[4] We use inference graphs to illustrate top-down query processing strategies. Inference graphs are similar in functionality to other types of graphs used in the literature to describe evaluation of queries; *e.g.*, rule/goal graphs [Ull89].

path will fail $(1-0.15 = 0.85)$ times the cost of $\langle \mathcal{R}_g\ D_g \rangle$; hence, $C[\Theta_1] = (1+1)+(1-0.15)(1+1) = 3.7$. In a similar way, we see that $C[\Theta_2] = (1+1)+(1-0.6)(1+1) = 2.8$. As $C[\Theta_2] < C[\Theta_1]$, we conclude that Θ_2 is the preferred strategy.

Notice that this analysis depends on the success probabilities; information that is usually not known initially (*e.g.*, we usually do *not* know that 60% of the queries will be `instructor(russ)`, etc.). [Smi89] presents one way of approximating their values, based on the (questionable) assumption that these probabilities are correlated with the distribution of facts in the database. For example, assume that the DB_2 database includes 2,000 facts of the form $\texttt{prof}^{\langle b \rangle}$ and 500 facts of the form $\texttt{grad}^{\langle b \rangle}$. Given any query of the form $\texttt{instructor}^{\langle b \rangle}$ — *e.g.*, `instructor(mark)` — that approach assumes that we are $\frac{2000}{500} = 4$ times more likely to find the corresponding `prof` fact (here "`prof(mark)`") than the `grad` fact ("`grad(mark)`"). This means [Smi89]'s algorithm would claim that Θ_1 is the optimal strategy, as it appears better than Θ_2, the only other legal candidate.

This, of course, need not be true; *i.e.*, there is no reason to assume the distribution of the user's queries will be correlated with the distribution of facts in the knowledge base. The user may, for example, only ask questions that deal with minors — here, none of the κ_is appearing in $\texttt{instructor}(\kappa_i)$ queries will be `prof`essors, meaning Θ_2 is clearly the superior strategy.

What we really need is the *conditional probability that a given retrieval will succeed, given that a query of a particular form is asked* — *i.e.*,

$$p_{prof} \quad = \quad Pr[\ \texttt{prof}(\kappa_i) \text{ retrieval succeeds } | \text{ query } \texttt{instructor}(\kappa_i) \text{ is asked }]$$

While there is no empirical way of finding such probabilities exactly, there is a meaningful way of estimating their values, based on the observation that the past can be a good predictor of the future. This basic observation is the key insight underlying "explanation-based learning" systems [DeJ88, MCK+89]. Each of these systems analyzes the solutions found to certain previous problems, and many use this information to suggest new strategies. The following sections presents two learning systems that use a similar approach for improving a query processing system; each uses statistics from the execution of a set of queries to estimate the distribution of queries, and then uses this information to produce a new QP′ with improved performance. Section 3 presents the PIB system that produces a series of query processors, $\text{QP}_1, \ldots \text{QP}_m, \ldots$, where each QP_{i+1} is, with provable high probability, more efficient than the prior QP_i. Section 4 presents the PAO system that will, with provable high probability, identify a query processing strategy that is approximately optimal over the space of all possible strategies.

While these sections use the above example to illustrate their basic ideas, *n.b.*, these ideas and algorithms apply in much greater generality — *i.e.*, each applies to a very broad space of possible inference graphs, etc. We first conclude this section by defining the necessary terms.

2.1 Definitions

An **inference graph** $G = \langle N, A, S, f \rangle$ is based on the rules (*i.e.*, the non-atomic definite clauses) in the knowledge base: N is the set of G's nodes, each corresponding to an atomic literal; $A \subset N \times N$ is the set of directed arcs, each corresponding either to an invocation of a rule or a database retrieval; and $S \subset N$ correspond to the "success nodes", reaching any such node $n \in S$ means the derivation has succeeded. (Each success node appears as a box in Figure 1.) The cost function $f : A \mapsto \Re^+$ maps each of G's arc to its positive real valued cost. (Note 4 below describes how to extend these definitions to deal with more complex graph structures.)

A query processing **strategy** Θ specifies how to search the inference graph to find an answer to a query, where that query corresponds to a node in the graph. We will write each strategy as a sequence of the elements of A, with the understanding that the remaining subsequence will be ignored after reaching a solution; *e.g.*, Θ_1 ignores $\langle \mathcal{R}_g\ D_g \rangle$ if the D_p retrieval was successful. That is, our QP is seeking only a single solution; this type of search is called a "satisficing search" [SK75].

(Subsection 5.2 underscores the importance of satisficing search to database systems by listing a variety of database situations that require this type of process — including the evaluation of ground and functional queries, using negation-as-failure, dealing with horizontally segmented distributed databases, etc. That subsection also discusses obvious variants of both PIB and PAO that can be used in yet other, non-satisficing contexts.)

Each **query processor** $\text{QP} = \langle G, \Theta \rangle$ can be viewed as an inference graph and a strategy. As we are considering way of changing the strategy, but not the graph, we will often identify each $\text{QP} = \langle G, \Theta \rangle$ with its strategy, Θ.

A query processing **context** is a pair $I = \langle q, DB \rangle$, where q is a query corresponding to a node of G and DB is a database of atomic literals. We let $\mathcal{I} = \{I_i\}$ refer to the set of all possible contexts; and assume that, in practice, these contexts are drawn at random from a stationary distribution, which we express using a probability function $Pr : \mathcal{I} \mapsto [0, 1]$ that maps each context to its probability of occurring.

On each execution, a query processor $\texttt{QP} = \langle G, \Theta \rangle$ is given a particular context as input; on successive runs, of course, QP will deal with different contexts. (This means the database of atomic literals can vary from one query processing context to another; the rule base, encoded as the inference graph G, is static.) On each run, dealing with a context $I = \langle q, DB \rangle$, QP traverses the inference graph G, beginning at the node corresponding to q, in the order described by Θ, searching for a success node $s \in S$. The search is complicated by the fact that QP cannot always traverse an arc, as some arcs can be "blocked" in some contexts; *e.g.*, if a required literal is not in the database. For example the D_p arc is blocked in the context $I_1 = \langle \texttt{instructor(manolis)}, DB_1 \rangle$ as the required literal `prof(manolis)` is not in DB_1. By contrast, neither rule-based reductions, $\mathcal{R}_p$ and $\mathcal{R}_g$, is blocked, nor is the D_g arc as `grad(manolis)` is in DB_1.

We can use the f function to compute $c(\Theta, I)$, the cost required for the strategy Θ to process the context $I \in \mathcal{I}$. For example, assuming each arc costs 1, then $c(\Theta_1, I_1) = 4$ and $c(\Theta_2, I_1) = 2$. Using $I_2 = \langle \texttt{instructor(russ)}, DB_1 \rangle$, $c(\Theta_1, I_2) = 2$ and $c(\Theta_2, I_2) = 4$. The *expected cost* of a strategy Θ with respect to the distribution of contexts, Pr — written $C_{Pr}[\Theta]$ — is the average time required over this distribution; *i.e.*,

$$C_{Pr}[\Theta] \quad \stackrel{def}{=} \quad E[\, c(\Theta, I)\,]$$
$$= \quad \underset{Pr,\ I \in \mathcal{I}}{average}\ c(\Theta, I) \quad = \quad \sum_{I \in \mathcal{I}} Pr(I) \times c(\Theta, I)\ .$$

We will omit the distribution subscript, Pr, when it is clear from context.

We close this subsection with several comments on this framework, and a few extensions.

Note 1. The above notation assumes only a finite number of contexts; there are obvious ways of extending this analysis to deal with distributions of an infinite number of contexts, see [GJ92].

Note 2. While there can be an infinite number of contexts, they can be partitioned into a large but 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 address a context depends only on which arcs are blocked, meaning we can identify each context $I = \langle q, DB \rangle$ with the subset of arcs that can are not blocked for the query q, based on the database DB. For example, we can identify the context $I_1 = \langle \texttt{instructor(manolis)}, DB_1 \rangle$ with the arc-set $\{\mathcal{R}_p, \mathcal{R}_g, D_g\}$.

Note 3. 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.*, $\Theta_1 \approx \langle \langle \mathcal{R}_p\, D_p \rangle, \langle \mathcal{R}_g\, D_g \rangle \rangle$. As a less trivial illustration, the Θ_{ABCD} strategy shown in Equation 4 (associated with the G_B inference graph shown in Figure 2) can also be expressed as

$$\langle \langle R_{ga}\, D_a \rangle, \langle R_{gs}\, R_{sb}\, D_b \rangle, \langle R_{st}\, R_{tc}\, D_c \rangle, \langle R_{td}\, D_d \rangle \rangle$$

The expected cost of a strategy is the weighted sum of the costs of its paths, 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.)

Note 4. The above definitions are sufficient for the class of simple "disjunctive inference 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.*, "`A :- B, C.`"), we must use directed hypergraphs, 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 [GO91, Appendix A]. For pedagogical reasons, however, this paper uses only the simple (not hyper) graph notation, in which a simple arc connects a node to but a single child, etc.

Also, 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. See [OG90].

Note 5. We will later need a few other definitions. The $f^* : A \mapsto \Re^+$ function maps each arc $a \in A$ to the sum of the costs of the arcs including and below a — hence, $f^*(\mathcal{R}_p) = f(\mathcal{R}_p) + f(D_p)$ and $f^*(\mathcal{R}_g) = f(\mathcal{R}_g) + f(D_g)$. (This definition of $f^*(\cdot)$ applies when the reduction subgraph under $\mathcal{R}_i$ is tree; [GJ92] deals with the complexities that arise in more general graphs.)

Second, the $F_\neg : A \mapsto \Re^+$ function maps each arc $a \in A$ to the the total cost of the arcs on the paths *other than* the path on which this a appears. As examples, for the inference graph G_A, $F_\neg[D_g] = f(\mathcal{R}_p) + f(D_p)$ and $F_\neg[D_p] = f(\mathcal{R}_g) + f(D_g)$.

Finally, we say that an inference graph is "tree shaped" iff there is a unique path of arcs from the root node down to each retrieval. Hence, Figure 1's G_A and Figure 2's G_B are each tree shaped, but the graph associated with the rule set

```
{     A :- B.    B :- C.    A :- C.  }
```

is not, as there are two distinct paths from `C` to `A`. We define $\mathcal{AOT}$ to be the set of all tree shaped inference graphs.

3 The PIB Algorithm

For pedagogical reasons, Subsection 3.1 first presents the stripped down PIB_1 algorithm; Subsection 3.2 then describes the more comprehensive PIB system.

3.1 Simple PIB_1 System

We view the PIB_1 system as a module that can incorporated within a general query optimizer, such as DEDGIN*, and used as a smart filter. In general, the overall optimizer will propose some particular transformation, that would produce a new query processor; for example, it may consider modifying the $\text{QP}_1 = \langle G_A, \Theta_1 \rangle$ query processor by interchanging $\langle \mathcal{R}_p \; D_p \rangle$ and $\langle \mathcal{R}_g \; D_g \rangle$, forming a new $\text{QP}_2 = \langle G_A, \Theta_2 \rangle$. PIB_1 will then permit this transformation *only if the resulting query processor is statistically likely to be an improvement over the initial query processor*; that is, if QP_2's expected cost is less than the cost of the original QP_1; *i.e.*, if $C[\Theta_2] < C[\Theta_1]$.

Stated more precisely and more generally, PIB_1 takes as input a specific $\text{QP} = \langle G, \Theta \rangle$ and confidence parameter $\delta > 0$, together with two of G's arcs, r_1, r_2 that descend from a common node (*e.g.*, $\mathcal{R}_p$ and $\mathcal{R}_g$ descending from G_A's root), and uses an oracle that produces contexts drawn randomly from the distribution. (This oracle could simply be the system's user, who is posing queries to the query processor that are relevant to one of his applications.) Let Θ' be the strategy that differs from Θ only by interchanging r_1 (and its descendents) with r_2 (and its descendents). (*E.g.*, Θ_2 differs from Θ_1 by interchanging $\mathcal{R}_p$ and its descendent D_p with $\mathcal{R}_g$ and its descendent D_g.) PIB_1 allows the overall optimizer to switch to the query process based on Θ' if, with confidence at least $1-\delta$, this Θ' is better than the original Θ; otherwise we continue to use original query processor, based on Θ.

This task would be trivial if we knew the distribution of contexts; we could then simply compute and compare the exact values of $C[\Theta]$ and $C[\Theta']$. Unfortunately, this distribution is not known *a priori*, and the only empirical way of determining whether $C[\Theta']$ is less than $C[\Theta]$ is to time how long each system requires to solve the *entire distribution of contexts*; a most impractical approach.

Fortunately, however, we can approximate this difference by observing the performance of these two query processors over some small sampling of the contexts, and using this to derive estimates of expected cost. We can then use statistical techniques to bound our confidence in these estimates.

In particular, we will want to approximate the value of $D[\Theta, \Theta'] \stackrel{def}{=} C[\Theta] - C[\Theta']$ based on a particular (fixed but unknown) distribution of contexts. Notice that Θ' is strictly better than Θ iff $D[\Theta, \Theta'] > 0$. We define the variables

$$\Delta_i \;=\; \Delta[\Theta, \Theta', I_i] \;=\; c(\Theta, I_i) - c(\Theta', I_i)$$

to be the difference of the costs of using Θ to find a solution to the context $I_i \in \mathcal{I}$, and the corresponding cost of using Θ'. As each context is selected randomly according to a fixed distribution, these Δ_is are independent identically distributed random variables whose common mean is $\mu = C[\Theta] - C[\Theta'] = D[\Theta, \Theta']$. We extend the Δ function by

$$\Delta[\Theta, \Theta', \{I_i\}_{i=1}^n] \;\stackrel{def}{=}\; \sum_{i=1}^{n} \Delta[\Theta, \Theta', I_i] \;=\; \sum_{i=1}^{n} \Delta_i$$

and use this to define $Y_n \stackrel{def}{=} \frac{1}{n}\Delta[\Theta, \Theta', \{I_i\}_{i=1}^n]$ to be the sample mean of n samples.

We know, of course, that this average will tend to the true population mean, $\mu = D[\Theta, \Theta']$, as $n \to \infty$; *i.e.*, $\mu = \lim_{n \leadsto \infty} Y_n$. Chernoff bounds [Che52] describe the probable rate of convergence: the probability that "Y_n is more than $\mu + \beta$" goes to 0 exponentially fast as n increases; and, for a fixed n, exponentially as β increases. Formally,[5]

$$\begin{aligned} Pr[Y_n > \mu + \beta] &\le e^{-2n\left(\frac{\beta}{\Lambda}\right)^2} \\ Pr[Y_n < \mu - \beta] &\le e^{-2n\left(\frac{\beta}{\Lambda}\right)^2} \end{aligned} \qquad (1)$$

where Λ is the range of possible Δ_i values; e.g., for Θ_1 and Θ_2, $\Lambda = f^*(\mathcal{R}_p) + f^*(\mathcal{R}_g)$, as $-f^*(\mathcal{R}_g) \le \Delta_i \le f^*(\mathcal{R}_p)$.[6] Hence, the probability that $\frac{1}{n}\Delta[\Theta, \Theta', \{I_i\}_{i=1}^n] \le D[\Theta, \Theta'] + \beta$ is over $1 - e^{-2n\left(\frac{\beta}{\Lambda}\right)^2}$.

By simple algebra, we see that we can be at least $1-\delta$ confident that $D[\Theta, \Theta'] > 0$, and hence that Θ' is better than Θ, whenever we observe that

$$\Delta[\Theta, \Theta', \{I_i\}_{i=1}^n] \quad > \quad \Lambda\sqrt{\frac{n}{2}\ln\frac{1}{\delta}} \qquad (2)$$

This equation gives us an *a posteriori* way of comparing two strategies: first construct the new Θ' and

[5] 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.

[6] The $f^*(\cdot)$ function is defined in Note 5 above.

then time both it, and the original Θ, solving a particular set of queries. We can then determine whether the observed difference is large enough, as a function of the number of samples. (In the simulation vernacular, this corresponds to the "paired-t confidence" [LK82].)

We are looking, however, for an efficient *a priori* way of deciding whether Θ' is better than Θ, without first building this Θ'. Once again, consider answering queries of form $\texttt{instructor}^{\langle \texttt{b} \rangle}$. We show below how to under-estimate the value of each Δ_i, call it $\tilde{\Delta}_i$, by examining the results of Θ_1 alone; *n.b.*, without building Θ_2. For our purposes, it suffices to show that this under-estimate is sufficiently greater than 0.

Suppose first that there is no solution to $I_i = \langle \texttt{instructor}(\kappa_i),\, DB_i \rangle$ in the subgraph under $\mathcal{R}_p$ but there is a solution under $\mathcal{R}_g$ (*i.e.*, $\texttt{grad}(\kappa_i)$ is in DB_i but $\texttt{prof}(\kappa_i)$ is not). Θ_2 is clearly better than Θ_1 in this context, as Θ_1 will first follow $\mathcal{R}_p$ to the unsuccessful database retrieval D_p, before following $\mathcal{R}_g$ to the successful D_g, meaning its cost is $f^*(\mathcal{R}_p)$ over the cost of using Θ_2; hence, here $\Delta_i = f^*(\mathcal{R}_p)$. Alternatively, if there is no solution under either $\mathcal{R}_p$ or $\mathcal{R}_g$, then both Θ_1 and Θ_2 will search the entire graph, meaning their costs are the same, and so $\Delta_i = 0$. Finally, if there is a solution under $\mathcal{R}_p$, then $\Delta_i \geq \tilde{\Delta}_i = -f^*(\mathcal{R}_g)$. (Proof: The real value of Δ_i for $I_i = \langle \texttt{instructor}(\kappa_i),\, DB_i \rangle$ depends on whether $\texttt{grad}(\kappa_i)$ is in DB_i. If not, then the (unbuilt) Θ_2 would have to search through the subgraph under $\mathcal{R}_g$, unsuccessfully, before trying the successful $\texttt{prof}(\kappa_i)$. By contrast, Θ_1 would only have to search through the latter subgraph, meaning $\Delta_i = -f^*(\mathcal{R}_g)$ in this case. Otherwise, if $\texttt{grad}(\kappa_i)$ is in DB_i, then Θ_2's cost $c(\Theta_2, I)$ would be less, meaning the value of $\Delta_i = c(\Theta_1,\, I_i) - c(\Theta_2,\, I_i)$ would be larger. Hence, in either case, $\Delta_i \geq -f^*(\mathcal{R}_g)$, as claimed.) *N.b.*, in all cases, we can determine an appropriate estimate $\tilde{\Delta}_i$ for Δ_i's value by simply observing Θ_1.

Now consider running Θ_1 for the m contexts, $S = \{\langle \texttt{instructor}(\kappa_i),\, DB_i \rangle\}_{i=1}^m$, and imagine it finds a solution under $\mathcal{R}_p$ (*i.e.*, $\texttt{prof}(\kappa_i) \in DB_i$) k_p times, and a solution under $\mathcal{R}_g$ but not under $\mathcal{R}_p$ k_g times (*i.e.*, $\texttt{prof}(\kappa_i) \notin DB_i$ and $\texttt{grad}(\kappa_i) \in DB_i$). We then have

$$\begin{aligned}\Delta[\Theta,\, \Theta', S] \quad &\geq \\ \tilde{\Delta}[\Theta,\, \Theta', S] \quad &= \quad [k_g \times f^*(\mathcal{R}_p) \;-\; k_p \times f^*(\mathcal{R}_g)]\end{aligned}$$

From Equation 2, we can be at least $1-\delta$ confident that $D(\Theta_1,\, \Theta_2)$ is above zero, and so Θ_2 is better than Θ_1, if

$$\begin{aligned}[k_g \times f^*(\mathcal{R}_p) \;-\; k_p \times f^*(\mathcal{R}_g)] \quad &\geq \\ [f^*(\mathcal{R}_p) + f^*(\mathcal{R}_g)]\sqrt{\tfrac{m}{2}\ln\tfrac{1}{\delta}}\end{aligned} \tag{3}$$

We can now describe how the overall 1-shot learning system will operate: As QP_1 is answering queries, PIB_1 will maintain the three counters that record the total number of samples considered (m), and of these, how often a success node was found under $\mathcal{R}_p$ (k_p), and how often under $\mathcal{R}_g$ but not under $\mathcal{R}_p$ (k_g). At some point, perhaps after observing that D_g has succeeded and D_p has failed, the general optimizer may consider transforming Θ_1 into Θ_2. PIB_1 would allow this transformation only if it is confident that Θ_2 is better than Θ_1; *i.e.*, if Equation 3 holds.

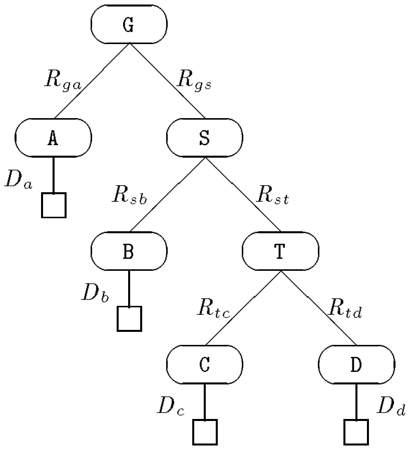


Figure 2: More Complicated Inference Graph, G_B

Notice PIB_1 has very modest space and computational requirements: only maintaining the three counters and computing Equation 3.

3.2 Anytime PIB Algorithm

The PIB_1 algorithm discussed above considers only a single modification — perhaps switching from Θ_1 to Θ_2 — and is only allowed to consider this modification once. The PIB algorithm is more general, in that it can consider several modifications at a time, and can also consider these various modifications several times, not just once.

To motivate why this is useful, consider the inference graph shown in Figure 2, and assume the initial $\text{QP}_0 = \langle G_B, \Theta_{ABCD} \rangle$ is using the strategy that traverses these arcs in depth-first, left-to-right order

$$\Theta_{ABCD} = \langle R_{ga}\, D_a\, R_{gs}\, R_{sb}\, D_b\, R_{st}\, R_{tc}\, D_c\, R_{td}\, D_d \rangle \ . \tag{4}$$

If we observe that the retrievals D_a, D_b and D_c all fail, but D_d succeeds, we might then consider various modifications: perhaps moving R_{td} and D_d to be before R_{tc} and D_c, leading to

$$\Theta_{ABDC} = \langle R_{ga}\, D_a\, R_{gs}\, R_{sb}\, D_b\, R_{st} \boxed{R_{td}\, D_d} R_{tc}\, D_c \rangle;$$

alternatively, we could move everything below R_{st} to be before R_{sb}, leading to

$$\Theta_{ACDB} = \langle R_{ga}\, D_a\, R_{gs} \boxed{R_{st}\, R_{tc}\, D_c\, R_{td}\, D_d} R_{sb}\, D_b \rangle;$$

or swap R_{ga} with R_{gs}; etc. Clearly each of these alternative strategies, and others, will cost less for this particular context. There may be yet other strategies that are good in other contexts.

The general PIB system is parameterized by a set of transformations $\mathcal{T} = \{\tau_j\}$, where each τ_j maps one strategy to another, perhaps by re-ordering a particular pair of arcs that descend from a common node. (*E.g.*, $\tau_{d,c}$ would rearrange the order of the R_{td} and R_{tc} arcs, which each descend from the T node. Hence, $\tau_{d,c}(\Theta_{ABCD}) = \Theta_{ABDC}$.) The set

$$\mathcal{T}(\Theta) \quad = \quad \{\ \tau_j(\Theta) \mid \tau_j \in \mathcal{T}\ \}$$

defines the set of possible "neighbors" of a given Θ. The PIB algorithm will "climb" to one of these neighbors $\Theta' \in \mathcal{T}(\Theta)$ if it is statistically confident that that neighbor Θ' is an improvement over the given Θ.

PIB reaches this decision using essentially the same approach that PIB_1 used: it first runs Θ_{ABCD}, collecting the statistics needed to estimate of the values of $\Delta[\Theta_{ABCD}, \Theta_{ABDC}, S]$, $\Delta[\Theta_{ABCD}, \Theta_{ACDB}, S]$, etc. In particular, for each context I_j and each proposed $\Theta' \in \mathcal{T}(\Theta_{ABCD})$, PIB computes an under-estimate, $\tilde{\Delta}[\Theta_{ABCD}, \Theta', I_j]$, satisfying

$$\tilde{\Delta}[\Theta_{ABCD}, \Theta', I_j] \quad \leq \quad \Delta[\Theta_{ABCD}, \Theta', I_j]\ .$$

As before, we can accurately determine the cost $c(\Theta_{ABCD}, I_j)$ of running Θ_{ABCD} in context I_j; the difficulty is in estimating $c(\Theta', I_j)$, the cost of running Θ' in L_j. Here again we may not have the information needed to compute this value exactly, as $c(\Theta', I_j)$ may depend on the success of various retrievals which were not even attempted by the Θ_{ABCD} process. To illustrate this difficulty, consider running Θ_{ABCD} in the context I_c, in which the first solution found by the strategy Θ_{ABCD} is D_c. While we do know that D_a and D_b are blocked in I_c and D_c is not, we do not know whether D_d is blocked. Now consider estimating $\Delta[\Theta_{ABCD}, \Theta_{ABDC}, I_c]$ in this context. Depending on whether D_d is blocked, the value will be either

$$\Delta[\Theta_{ABCD}, \Theta_{ABDC}, I_c] \quad = \quad \begin{cases} f^*(R_{tc}) - f^*(R_{td}) & \text{if } D_d \text{ is } \textit{not} \text{ blocked} \\ -f^*(R_{td}) & \text{if } D_d \text{ is blocked} \end{cases}$$

As our goal is to *under-estimate* this value, we chose the smaller of this pair; hence $\tilde{\Delta}[\Theta_{ABCD}, \Theta_{ABDC}, I_c] = -f^*(R_{td})$, which corresponds to the value of $\Delta[\Theta_{ABCD}, \Theta_{ABDC}, I_c]$ under the assumption that D_d *is* blocked in I_c.

This idea holds in general: the value of the under-estimate $\tilde{\Delta}[\Theta, \Theta', I_j]$ corresponds to the value of $\Delta[\Theta, \Theta', I_j]$ under the assumption that all of the arcs in the unexplored part of the inference graph will be blocked.

If PIB is considering switching from Θ to one of k different new strategies, $\mathcal{T}(\Theta) = \{\Theta_i\}_{i=1}^{k}$, it would maintain the k values of $\tilde{\Delta}[\Theta, \Theta_i, S]$ for each Θ_i for the sample $S = \{I_i\}_{i=1}^{n}$; and would swap to some Θ_j only if

$$\tilde{\Delta}[\Theta, \Theta_j, S] \quad > \quad \Lambda[\Theta, \Theta_j]\sqrt{\frac{|S|}{2}\ln\frac{k}{\delta}} \qquad (5)$$

where $\Lambda[\Theta, \Theta_j]$ is the range of values of $\Delta[\Theta, \Theta_j, I_i]$ over all possible I_is, which is never more than the sum of the costs of the arcs under the node where Θ deviates from Θ_j. (Hence, $\Lambda[\Theta_{ABCD}, \Theta_{ABCD}] = f^*(R_{tc}) + f^*(R_{td})$, and $\Lambda[\Theta_{ABCD}, \Theta_{ACDB}] = f^*(R_{sb}) + f^*(R_{st})$, etc.) The extra $k = |\mathcal{T}(\Theta)|$ in the radical of Equation 5 is used to keep below δ the probability of a false positive — here switching to *any* Θ_i when $D(\Theta, \Theta_i) < 0$. (If we simply used Equation 2 for each Θ_j, we would only know that the probability of a false positive is below $k \times \delta$, which is unacceptably high.) Of course, we can express each such Equation 5 as a simple computation involving easily-obtained statistics, *à la* Equation 3.

The other way in which PIB extends PIB_1 is by allowing sequential tests. To explain this, consider the KB_1 shown in Figure 1: after running Θ_1 for m_1 samples, we can use Equation 3 to determine whether to switch to the new strategy, Θ_2. If Equation 3 does *not* hold, we might want to continue running Θ_1 for more m_2 samples, and then ask, again, whether Θ_2 seems to be superior to Θ_1. Notice we cannot simply use Equation 3, based on the total statistics gathered during all $m = m_1 + m_2$ samples, as we only know that the chance of a false positive is only below $\delta + \delta$, which is not acceptable.

To deal with multiple sequential tests, based on successive sets of samples, PIB uses a modified version of Equation 3. For the i^{th} test, it replaces the confidence parameter δ with the value of the form $\delta_i = \frac{\delta}{i^2}\frac{6}{\pi^2}$. Here the probability of a false positive is bounded above by $\sum_{i=1}^{\infty} \delta_i = \delta \frac{6}{\pi^2} \sum_{i=1}^{\infty} \frac{1}{i^2} = \delta$, as desired.

The PIB system embodies both of these extensions. As shown in Figure 3 (and illustrated by Figure 4), PIB will observe a query processing system $\text{QP}_0 = \langle G, \Theta_0 \rangle$ solve a set of queries. After QP_0 solves each query q_m, PIB will update its statistics of the success rate of each attempted database retrieval, and use these statistics to update the values of $\tilde{\Delta}[\Theta', \Theta_0, \{q\}]$, for each $\Theta' \in \mathcal{T}[\Theta_0]$. If none of these quantities satisfy Equation 6, then QP_0 will continue using Θ_0, and will process the next input, q_{m+1}. Otherwise, PIB instructs the query processor to switch to the associated new strategy, Θ'. This is one hill-climbing step, to $\Theta_1 \leftarrow \Theta'$. PIB then collects statistics as this Θ_1 system solves queries, and will hill-climb to some Θ_2 if appropriate, etc.

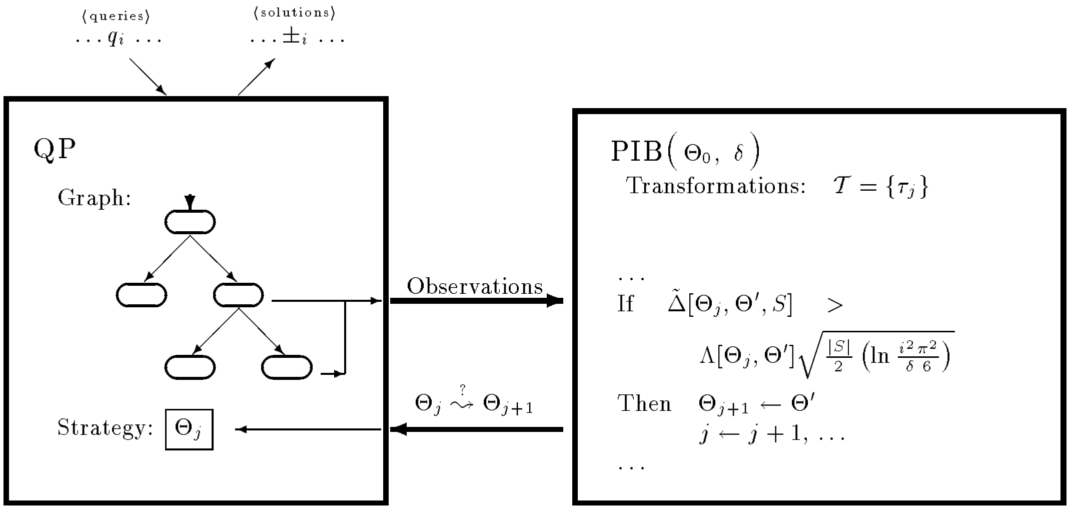


Figure 4: Basic Architecture of the Overall PIB System

Algorithm PIB(Θ_0, δ)

- Let $i \leftarrow 0$. *%% i is total number of trials*
  $j \leftarrow 0$. *%% Currently using j^{th} strategy*

L1: Let $S \leftarrow \{\}$. *%% S is set of samples for current Θ_j*

L2: Get context I
  Let $S \leftarrow S \cup \{I\}$
  Update $\tilde{\Delta}[\Theta_j, \Theta', S]$ for all $\Theta' \in \mathcal{T}(\Theta_j)$

- Let $i \leftarrow i + |\mathcal{T}(\Theta_j)|$.
  If there is some $\Theta' \in \mathcal{T}(\Theta_j)$ such that
  $$\tilde{\Delta}[\Theta_j, \Theta', S] \ \geq\ \Lambda[\Theta_j, \Theta']\sqrt{\frac{|S|}{2}\ln\left(\frac{i^2\,\pi^2}{6\,\delta}\right)} \qquad (6)$$
  then *%% Tell query processor to use strategy Θ_{j+1}*
  let $\Theta_{j+1} \leftarrow \Theta'$, $j \leftarrow j+1$.
  Return to **L1**.
- Otherwise, return to **L2**.

Figure 3: Code for PIB

It is trivial to see that this algorithm will seldom make a mistake:

Theorem 1 (Correctness of PIB Algorithm) *The PIB algorithm, shown in Figure 3, will successively consider the strategies $\Theta_0, \Theta_1, \dots, \Theta_j, \dots$. The probability that PIB will make a mistake — i.e., decide to move from Θ_j to Θ_{j+1} when in fact $C[\Theta_{j+1}] > C[\Theta_j]$ — is below δ:*
$$Pr[\ \exists j\ C[\Theta_{j+1}] > C[\Theta_j]\ ] \quad \leq \quad \delta \qquad \square$$

The above theorem justified our view that this PIB process is an *anytime algorithm* [BD88, DB88] as, at any time, it provides a strategy (here, the strategy produced at the i^{th} iteration) with the property that the longer we wait, the better the strategy; *i.e.*, $j > i$ means Θ_j is, with high probability, better than Θ_i.

We close this section with a few final comments about the PIB algorithm: First, the algorithm shown in Figure 3 performs the Equation 6 test after processing each individual query. Theorem 1 continues to hold if we modify PIB to perform this tests less frequently, perhaps after processing each successive sequence of $k > 1$ queries.

Second, as mentioned above, the PIB system can be viewed as "hill-climbing" using a specific set of possible transformations as operators [Nil80]; here each transformation exchanges the order of some particular pair of "sibling" arcs. The general PIB system can use (almost) arbitrary sets of transformations to hill-climb; *e.g.*, perhaps using a set of transformations that each add some macro-operator [MKKC86] or chunk [LNR87, LNR86] to the rule base. See [GJ92] for more details.

Third, [CG91] describes a related learning algorithm called PALO (for "Probably Approximately Locally Optimal"). Like PIB, PALO uses a set of possible transformations to hill-climb in a situation where the worth of each strategy can only be estimated by sampling. While PIB will continue collecting samples and potentially moving to new strategies indefinitely, PALO will stop when it reaches an "ϵ-local optimal" — *i.e.*, when it reaches a Θ_m with the property that

$$\forall \Theta \in \mathcal{T}(\Theta_m) \quad C[\Theta] \;\geq\; C[\Theta_m] - \epsilon$$

See [CG91, GJ92] for more details.

4 The PAO Algorithm

PAO's task is to identify a new strategy whose cost is, with high probability, very close to the cost of the optimal strategy. That is, let Θ_{opt} be a strategy whose expected cost is minimal — *i.e.*, $\forall\Theta\ C[\Theta_{opt}] \leq C[\Theta]$. Given any $\epsilon, \delta > 0$, PAO will return a strategy, Θ_{pao} whose cost is, with probability at least $1-\delta$, no more than ϵ over $C[\Theta_{opt}]$ — *i.e.*,

$$Pr[\ C[\Theta_{pao}] \leq C[\Theta_{opt}] + \epsilon\ ] \quad \geq \quad 1 - \delta.$$

Once again, this task would be relatively easy if we knew the distribution of contexts, as there are algorithms $\Upsilon_{\mathcal{G}}(G, p)$ that take a graph G in the class $\mathcal{G}$ (*e.g.*, the G_A graph shown in Figure 1, in the class $\mathcal{AOT}$ of tree shaped inference graphs[7]) and a vector of the success probabilities of the relevant retrievals p (*e.g.*, $p = \langle p_p, p_g\rangle = \langle 0.2, 0.6\rangle$, where $p_p = Pr[\,\texttt{prof} \mid \texttt{instructor}\,]$ and $p_g = Pr[\,\texttt{grad} \mid \texttt{instructor}\,]$) and produce the optimal strategy for that graph (here, Θ_2).[8] In particular, we will focus on $\Upsilon_{\mathcal{AOT}}$, which deals with arbitrary tree shaped inference graphs. Unfortunately, as we do not initially know this distribution, we must once again first use sampling techniques to approximate it, before we can hand this approximation to $\Upsilon_{\mathcal{AOT}}$.

The PAO algorithm takes, as input, a specific inference graph G, an error term $\epsilon > 0$ and a confidence parameter $\delta > 0$. It first computes a vector of values $M = \langle m_1, \ldots, m_n\rangle$ that specify the number of trials m_i required for each database retrieval d_i. PAO then collects statistics as an "adaptive query processor" QP^A (described in Subsection 4.1 below) solves a sufficient set of contexts to insure that each database retrieval d_i is sampled at least m_i times. This produces a list of frequency values $\hat{p} = \langle\hat{p}_1, \ldots \hat{p}_n\rangle$; PAO hands these values, together with the graph G, to the appropriate $\Upsilon_{\mathcal{G}}$ function, which returns a strategy $\Theta_{pao} = \Upsilon_{\mathcal{G}}(G, \hat{p})$.

To illustrate this process, consider again the G_A graph shown in Figure 1, and assume that the query for each context is of the form $\texttt{instructor}^{\langle b\rangle}$. Here, PAO first computes the vector $M = \langle m_p, m_g\rangle = \langle 30, 20\rangle$, based on some ϵ and δ values. PAO then watches QP^A solve problems until it has observed 30 trials of it trying the D_p retrieval (each of the form $\texttt{prof}(\kappa_i)$) and 20 involving the D_g retrieval (each of the form $\texttt{grad}(\kappa_i)$). Assume D_p succeeds 18 times of its 30 trials, and D_g, 10 times of its 20 trials. PAO then passes G_A and the vector $\hat{p} = \langle\frac{18}{30}, \frac{10}{20}\rangle$ to the $\Upsilon_{\mathcal{AOT}}$ function, which computes the strategy $\Upsilon_{\mathcal{AOT}}(G_A, \hat{p})$ — here, Θ_1. PAO then returns this strategy.

Notice this strategy would be the optimal one if those frequency values were really the true probability values; *i.e.*, if $p = \hat{p}$. In general, let $\Theta_{\hat{p}}$ be the strategy whose cost is minimal based on the estimate $\hat{p}$; *i.e.*, $\Theta_{\hat{p}} = \Upsilon_{\mathcal{AOT}}(G, \hat{p})$. Lemma 1 in Subsection 4.1[9] performs a sensitivity analysis of the $\Upsilon_{\mathcal{AOT}}(G, p)$ function to bound how much the cost of this strategy can differ from the cost of the optimal strategy $\Theta_p = \Upsilon_{\mathcal{AOT}}(G, p)$ when each $\hat{p}_i$ in $\hat{p}$ is close to the corresponding p_i in p — *i.e.*, showing that $|C[\,\Theta_{\hat{p}}\,] - C[\,\Theta_p\,]|$ is small whenever each $|\hat{p}_i - p_i|$ is small.

The only remaining challenge is to determine the number of samples required to be confident that $|\hat{p}_i - p_i|$ is small; this requires a few simple applications of Chernoff bounds (Equation 1). The resulting sample complexity appears below:

Theorem 2 (adapted from [GO91]) *Let $\epsilon, \delta > 0$ be given positive constants, and $G \in \mathcal{AOT}$ be any tree shaped inference graph with database retrievals $\{d_i\}_{i=1}^n$. Let $\hat{P} = \langle\hat{p}_1, \ldots, \hat{p}_n\rangle$ be the probability estimates obtained after sampling each retrieval d_i at least*

$$m(d_i) \;=\; \lceil 2\left(\frac{nF_\neg[d_i]}{\epsilon}\right)^2 \ln\frac{2n}{\delta}\rceil \tag{7}$$

times,[10] and let $\Theta_{pao} = \Upsilon_{\mathcal{AOT}}(G, \hat{P})$ be the strategy that is optimal, based on these estimates. Then, with probability at least $1-\delta$,

$$C[\Theta_{pao}] \quad\leq\quad C[\Theta_{opt}] + \epsilon$$

where $\Theta_{opt} = \Upsilon_{\mathcal{AOT}}(G, P)$ is the optimal strategy based on the correct probability values P. □

This theorem assumes that it is easy to obtain trials of each retrieval; the next subsection states why this can

[7] See Note 5 above

[8] These $\Upsilon_{\mathcal{G}}$ functions all assume that the success probabilities of the various retrievals are independent of one another. As there are $O(2^n)$ dependencies possible among n retrievals, it would require exponential time simply to express these dependencies.

[9] We delay presenting that lemma until we define various necessary terms.

[10] The $F_\neg[\cdot]$ function is defined in Note 5 above.

be problematic, and presents a more general solution to this task of obtaining the required samples.

Computational Efficiency: It is easy to see that the computational time of overall PAO algorithm is polynomial in the obvious parameters whenever the inference graph is non-recursive and its final step (involving the $\Upsilon_{\mathcal{G}}$ algorithm) can be performed efficiently. Unfortunately, this latter task is NP-hard for general graphs $\mathcal{G}$; see [Gre91].

Fortunately, however, $\Upsilon_{\mathcal{G}}$ is efficient for some subclasses $\mathcal{G}$. For example, [Smi89] presents an efficient algorithm $\Upsilon_{\mathcal{OT}}$ for the class of simple disjunctive tree shaped inference graphs (see Note 4 and Note 5). Moreover, there are polynomial time $\tilde{\Upsilon}_{\mathcal{G}}$ functions that can produce near optimal strategies for some classes $\mathcal{G}$ for which $\Upsilon_{\mathcal{G}}$ is intractable; *cf.*, [GO91, Appendix B]. That article also shows how to use these efficient approximation algorithms to build a variant of PAO that is efficient for such inference graphs.

4.1 Obtaining enough samples for PAO

The PAO algorithm requires estimates of the success probability of each retrieval; this subsection discusses how to obtain a sufficient number of samples of each retrieval. Notice we cannot always obtain the needed statistical information by simply watching a query processor that uses a *fixed* strategy solve a set of problems. Consider for example simply using Θ_1 for G_A. If the D_p retrieval always succeeds (*i.e.*, $\texttt{prof}(\kappa_i) \in DB_i$ always holds), then Θ_1 would never continue beyond that retrieval, and so PAO would not obtain any samples of the $\texttt{grad}(\kappa_i)$ retrieval.

This argument applies to any query processor that follows a *fixed* strategy. We must therefore use an "adaptive query processor" (designated QP^A above) that can follow different strategies over time. In particular, QP^A can adjust its strategy on each sample to make sure it will observe each retrieval a sufficient number of times. Hence, if $\langle m_p, m_g\rangle = \langle 30, 20\rangle$, QP^A may use Θ_1 for the first 30 contexts; this guarantees that we will have enough samples of D_p. QP^A will then use Θ_2 for the next (at most) 20 contexts, to obtain the 20 samples needed for D_g. (Of course, the data collection work can stop as soon as PAO has found the needed 20 samples. In the above example, as 18 of the 30 D_p retrievals succeeded, PAO would already have obtained $30-18 = 12$ samples of D_g, meaning it would only need an addition $20-12 = 8$ more samples, run using the Θ_2 strategy.)

This QP^A process could be implemented by maintaining a set of counters $\langle c_1, \ldots, c_n\rangle$, one per retrieval, which are each initialized to the proper $c_i \leftarrow m(d_i)$ value (from Equation 7). We would decrement the c_i counter each time the d_i retrieval is attempted. The QP^A strategy would always begin with the retrieval whose current counter value is largest; and the sampling phase would be over when all counters fall below 0. Hence, the total number of samples required is at most $\sum_i m(d_i)$, which is clearly polynomial in the various parameters.

There is another more subtle issue that complicates the task of finding enough samples of each retrieval: In a general inference graph, one cannot always reach some arcs in certain contexts. As an example, imagine our rule base also included the rule

```
grad(fred) :- admitted(fred, X).
```

Notice we can follow this rule only if the initial query is $\texttt{instructor(\underline{fred})}$. Otherwise, the arc from $\texttt{grad}(\kappa)$-to-$\texttt{admitted(fred, X)}$ will be blocked, which will prevent our PAO system from obtaining a sample of the $\texttt{admitted(fred, X)}$ retrieval. Of course, if we cannot obtain any samples of this retrieval, we will be unable to produce the needed estimates of its success probability.

Fortunately, the number of samples required for each retrieval d_i depends on the probability of reaching that retrieval, designated "$\rho(d_i)$" and defined in Definition 2 below. Lemma 1 below shows that the smaller the value of $\rho(d_i)$, the less sensitive $\Upsilon_{\mathcal{AOT}}$ is to the value of d_i's success probability $p_i = P(d_i)$, meaning (via Chernoff bounds) we need fewer samples of d_i. In the limit, if there is no chance of reaching d_i (*i.e.*, if $\rho(d_i) = 0$), then we will need *no* samples of d_i, as $\Upsilon_{\mathcal{AOT}}$ will produce arbitrarily good strategies using any estimate $\hat{p}_i$ of d_i's success probability, even as large as $|\hat{p}_i - p_i| = 1$.

Theorem 2 above used the fact that $\rho(d_i)$ is bounded by 1 to obtain a generous bound on the number of trials required for d_i. As explained above, we may not be able to obtain this many samples of d_i when $\rho(d_i) \ll 1$. Fortunately, fewer examples are required in this situation. Unfortunately, it is difficult to determine this smaller sample complexity as it is based on the value of $\rho(d_i)$, which depends on the unknown distribution.

Fortunately, we can approximate the value of $\rho(d_i)$ as we are obtaining the estimate of p_i. In essense, we need only "aim for d_i" a certain number of times; each time we reach d_i, we can obtain another sample towards estimating d_i's success probability, and each time our path to d_i is blocked, we can obtain a better estimate of the failure probability of $\rho(d_i)$.

Theorem 3 below quantifies this argument. Notice it deals with "probabilistic experiments" in general, which includes all blockable arcs, both database retrievals and rule-based reductions. In general, determining which strategy is optimal depends on the success probabilities of all of these experiments, not just retrievals. Of course, we need to extend the "inference graphs" defined above to accomodate these internal probabilistic experiments; the "search structures" defined in [OG90] are sufficient.

We first require the following definition:

Definition 1 (Aiming for an experiment) *Let e be any arc of $G \in \mathcal{AOT}$, which is a tree shaped inference graph; and let $\Pi(e)$ be the sequence of G's arcs that descend from G's root down to, but not including, e. We say that an adaptive query processor* QP^A *has "*attempted to reach e on a context I*" if it followed the path $\Pi(e)$ as far as it could, either reaching e or being blocked by an arc in $\Pi(e)$ before reaching e.*

Theorem 3 *Let $\epsilon, \delta > 0$ be given positive constants, and $G \in \mathcal{AOT}$ be any tree shaped inference graph with probabilistic experiments $\{e_i\}_{i=1}^n$. For each experiment e_i, assume that an adaptive query processor* QP^A *has attempted to reach e_i on at least*

$$m'(e_i) \;=\; \lceil 2\left(\sqrt{\frac{2\,\epsilon}{n\,F_\neg[e_i]}+1}-1\right)^{-2}\ln\frac{4n}{\delta}\rceil \qquad (8)$$

contexts.[11] *Assume* QP^A *has reached each e_i $k(e_i)$ times, and of these, found that e_i was not blocked $n(e_i)$ times. Let $\hat{P} = \langle \hat{p}_1, \ldots, \hat{p}_n\rangle$ be the success frequencies where $\hat{p}_i = \frac{n(e_i)}{k(e_i)}$ or $\hat{p}_i = 0.5$ if $k(e_i) = 0$; and let $\Theta_{pao} = \Upsilon_{\mathcal{AOT}}(G, \hat{P})$ be the optimal strategy, based on these estimates. Then, with probability at least $1-\delta$,*

$$C[\Theta_{pao}] \;\le\; C[\Theta_{opt}] + \epsilon$$

where $\Theta_{opt} = \Upsilon_{\mathcal{AOT}}(G, P)$ is the optimal strategy based on the correct probability values P.

Proof sketch: Assume that QP^A has attempted to reach e_i $m = m'(e_i)$ times, and has succeeded in reaching e_i a total of k of these attempts. Using Chernoff bounds, with probability at least $1-\frac{\delta}{2n}$,

$$\rho(e_i) \quad\le\quad \frac{k}{m} + \sqrt{\frac{1}{2m}\ln\frac{2n}{\delta}}$$

and also, with probability at least $1-\frac{\delta}{2n}$,

$$|\hat{p}_i - p_i| \quad\le\quad \sqrt{\frac{1}{k}\ln\frac{4n}{\delta}}$$

We can bound $\rho(e_i) \times |\hat{p}_i - p_i|$ by first defining $g(k) = \min\left\{1,\ \frac{k}{m} + \sqrt{\frac{1}{2m}\ln\frac{2n}{\delta}}\right\} \times \min\left\{1,\ \sqrt{\frac{1}{k}\ln\frac{4n}{\delta}}\right\}$ and then, using $\frac{\partial g(k)}{\partial k}$, observe that

$$g(k) \quad\le\quad \left(\frac{1}{2}+\sqrt{\frac{1}{2m}\ln\frac{4n}{\delta}}\right)^2 - \frac{1}{4}$$

for all k between 0 and m. Hence, plugging in the value of $m = m'(e_i)$ from Equation 8, with probability at least $(1-\frac{\delta}{2n})^2 \ge 1-\frac{\delta}{n}$,

$$\rho(e_i) \times |\hat{p}_i - p_i| \quad\le\quad \frac{\epsilon}{2\,n\,F_\neg[e_i]} \qquad (9)$$

With probability at least $(1-\frac{\delta}{n})^n \ge 1-\delta$, Equation 9 holds for each of the n experiments, which, using Lemma 1, means that $C[\Theta_{pao}] \le C[\Theta_{opt}]+\epsilon$, as desired. (This proof appears in full in [GO92].) □

Definition 2 (Reaching an Experiment) *Given any tree shaped inference graph G, any distribution P and any strategy Θ defined on G, define $\rho_p(e_i, \Theta)$ to be the probability that the strategy Θ will reach the experiment e_i; and define*

$$\rho_p(e_i) \quad\stackrel{def}{=}\quad \max_{\Theta}\rho_p(e_i, \Theta)$$

to the largest probability, over all strategies.
When the distribution p is clear from context, we will simply write "$\rho(e_i)$".

Hence, in any tree shaped inference graph, $\rho(e_i)$ is at most the product of the success probabilities of the arcs on the path $\Pi(e_i)$ from the root to e_i.[12]

Lemma 1 (Sensitivity Analysis of $\Upsilon_{\mathcal{AOT}}$) *Let $G \in \mathcal{AOT}$ be any tree shaped inference graph with experiments $\{e_i\}_{i=1}^n$; and let $P = \langle p_1, \ldots p_i, \ldots p_n\rangle \in [0,1]^n$ and $\hat{P} = \langle \hat{p}_1, \ldots \hat{p}_i, \ldots \hat{p}_n\rangle \in [0,1]^n$ be two probability vectors, where p_i (resp., $\hat{p}_i$) is the success probability of the i^{th} experiment, e_i. Let $\Theta_p = \Upsilon_{\mathcal{AOT}}(G, p)$ (resp., $\Theta_{\hat{p}} = \Upsilon_{\mathcal{AOT}}(G, \hat{p})$) be the strategy that is optimal, given the distribution p (resp., $\hat{p}$). Then*

$$C_P[\Theta_{\hat{P}}] - C_P[\Theta_P] \quad\le\quad 2\sum_{i=1}^{n} F_\neg[e_i] \times \rho(e_i) \times |p_i - \hat{p}_i|$$

where $\rho(e_i)$ is the probability of reaching e_i, based on the P distribution.

Proof Sketch: Given the vectors P and $\hat{P}$, let $P^{(i)} = \langle \hat{p}_1, \ldots, \hat{p}_i, p_{i+1}, \ldots, p_n\rangle$, and as special cases, $P^{(0)} = P$ and $P^{(n)} = \hat{P}$. [GO92] proves that $\frac{\partial C_p[\Theta]}{\partial p_i} \le \rho(e_i) \times F_\neg[e_i]$; using the mean-value theorem, this means that, for any strategy Θ,

$$|C_{P^{(i)}}[\Theta] - C_{P^{(i-1)}}[\Theta]| \quad\le\quad \rho(e_i) \times F_\neg(e_i) \times |\hat{p}_i - p_i|$$

and so

$$|C_P[\Theta] - C_{\hat{P}}[\Theta]| \quad\le\quad \sum_{i=1}^{n} \rho(e_i) \times F_\neg(e_i) \times |\hat{p}_i - p_i|.$$

[11] The leading term of an asymptotic expansion on $m'(e_i)$ by n is $2\left(\frac{nF_\neg[e_i]}{\epsilon}\right)^2 \ln\frac{4n}{\delta}$, which is very close to the formula given in Equation 7.

[12] We are continuing to insist that success probabilities of the experiments are independent of each other.

The lemma follows immediately from the observation that

$$\begin{aligned} & C_P[\Theta_{\hat{P}}] - C_P[\Theta_P] \\ & \quad = \quad C_P[\Theta_{\hat{P}}] - C_{\hat{P}}[\Theta_{\hat{P}}] \quad + \quad C_{\hat{P}}[\Theta_{\hat{P}}] - C_{\hat{P}}[\Theta_P] \\ & \qquad + \quad C_{\hat{P}}[\Theta_P] - C_P[\Theta_P] \\ & \quad \leq \quad |C_P[\Theta_{\hat{P}}] - C_{\hat{P}}[\Theta_{\hat{P}}]| \quad + \quad |C_{\hat{P}}[\Theta_P] - C_P[\Theta_P]| \end{aligned}$$

as $C_{\hat{P}}[\Theta_{\hat{P}}] \leq C_{\hat{P}}[\Theta_P]$. □

5 Conclusions

5.1 General comments about PIB and PAO

Both PIB and PAO were designed to work unobtrusively — each basically monitors a query processor as it deals with queries, and only occasionally suggests modifications. Each requires some initial information, including a description of the underlying inference graph and certain parameters: δ and possibly ϵ. In addition, each also requires certain statistical information. *N.b.*, this information can be obtained trivially, by simply recording (at most) the number of times a query processor attempts each database retrieval and how often that retrieval succeeds. Hence, the time and space requirements for the data-collection part of these algorithms is extremely minor: only maintaining one or two counters per retrieval.

The overall computational cost of the PIB processes is always minor: simply evaluating Equation 6 as often as requested. The overall computational cost of the PAO process depends on the efficiency of the $\Upsilon_{\mathcal{G}}$ function used: PAO is tractable iff the inference graph is acyclic and the $\Upsilon_{\mathcal{G}}$ function is tractable.

Finally, we can list the assumptions underlying the PIB and PAO algorithms: [1] each strategy Θ_j must be static and deterministic, meaning it will return the same answer each time it is run on a given context; [2] each Θ_j performs a "satisficing" search, meaning that it returns the first success node that it finds; and [3] the distribution of problem contexts is *stationary*, meaning that the success probabilities of the various retrievals do not change over time.

Each of these processes is defined for arbitrary classes of inference graphs, and can apply even if the success probabilities of different retrievals are dependent on one another. However, the PAO process is not always computationally efficient in such situations.

5.2 DataBase Applications of PIB and PAO

There are many database applications for the PIB and PAO systems. The previous sections discussed how they can be used to improve a system that is performing satisficing searches. As such, they can clearly help to reduce the cost of performing both ground queries and and existentially quantified queries (*i.e.*, queries asking for only one answer), which include "non-deterministic queries" [AV88], and "existential queries" [RBK88]. Such searches are also relevant to any system that uses negation as failure: Consider the rule

```
pauper(X) :- not( owns(X,Y) ).
```

and observe that we can determine whether some individual is, or is not, a pauper by finding a single item that he owns; *n.b.*, we do not have to find each of his multitude of possessions.

Our PIB and PAO techniques can apply to yet other tasks that fit into this "satisficing" framework. One obvious additional database application is deciding on the order in which to scan a set of horizontally segmented distributed databases [Des90, p. 673]. For example, imagine we have several physical files that each store the same types of facts about people. Given a query like `age(russ, X)`, we would like to scan these files in the appropriate order — hoping to find the file dealing with `russ` facts as early as possible.

Finally, there are obvious variants of these algorithms that can be used in related situations. For example, one set of variants seek the first k answers to a query, for some fixed $k > 1$. This can be useful in situations where we know that there can be only k answers to some query; *e.g.*, `parent(x,Y)` will only yield two bindings for `Y` for any fixed `x`, as will `senator(x,Y)`, etc.

5.3 Contributions

The view that query processing corresponds to searching through a graph [Vie89, Smi89] suggests one form of query optimization: finding an efficient graph search strategy. This paper describes two learning algorithms that can find good strategies; each first monitors a query processor as it solves a set of queries, collecting relevant statistics about the queries posed by the user, then uses this information to improve the system's performance on subsequent queries — that is, to produce a QP$'$ whose performance will, with high probability, be good.

The first improvement method, PIB, use these collected statistics to hill-climb to successive strategies that are, with high probability, successively better. This approach is quite general: it can be used efficiently with arbitrary inference graphs, and does not require that the success probabilities of the retrievals be independent. However, PIB can climb to a "local minimum", and so is not guaranteed to produce a strategy that is even close to a *globally* optimal strategy. The second method, PAO, avoids this problem by using the obtained statistical information to identify a strategy whose cost is, with high probability, close to the global optimum. Unfortunately, it is computationally efficient only for certain classes of inference graphs, and only if the success probabilities are independent. We close by commenting that both of these techniques apply in many other situations as well — in fact, in any situation that involves performing a set of probabilistic experiments until reaching a satisfying configuration of successes and failures where the cost of performing the experiments depends on the order chosen [OG90]; Subsection 5.2 lists many other applications that are relevant to database systems.

References

[AV88] Serge Abiteboul and Victor Vianu. Procedural and declarative database update languages. In *Proc. of 7th Symposium on Principles of Database Systems*, pages 240–50, Austin, TX, March 1988.

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

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

[BR86] F. Bancilhon and R. Ramakrishnan. An Amateur's Introduction to Recursive Query-Processing Strategies. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1–49, 1986.

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

[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 *AAAI-88*, pages 49–54, August 1988.

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

[Des90] Bipin C. Desai. *An Introduction to Database Systems*. West Publishing Company, NY, 1990.

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

[GO91] Russell Greiner and Pekka Orponen. Probably approximately optimal derivation strategies. In J.A. Allen, R. Fikes, and E. Sandewall, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference*, San Mateo, CA, April 1991. Morgan Kaufmann.

[GO92] Russell Greiner and Pekka Orponen. Probably approximately optimal satisficing strategies. Technical report, Siemens Corporate Research, 1992.

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

[HC76] Michael Hammer and Arvola Chan. Index Selection in a Self-Adaptive Data Base Management System. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1–8, 1976.

[LK82] Averill M. Law and W. David Kelton. *Simulation Modeling and Analysis*. McGraw-Hill Book Company, Toronto, 1982.

[LN90] R.J. Lipton and J.F. Naughton. Query size estimation by adaptive sampling. In *Proceedings of ACM SIGACT/SIGMOD Symposium on Principles of Database Systems*, pages 40–46, 1990.

[LNR86] John E. Laird, Allan Newell, and Paul S. Rosenbloom. Chunking in SOAR: The anatomy of a general learning mechanism. *Machine Learning*, 1(1):11–46, 1986.

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

[LV89] A. Lefebvre and L. Vieille. On Deductive Query Evaluation in the DedGin* System. In *Proceedings of the 1st International Conference on Deductive and Object-Oriented Databases*, pages 225–244, Kyoto, Japan, 1989.

[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.

[MKKC86] Thomas M. Mitchell, Richard M. Keller, and Smadar T. Kedar-Cabelli. Example-based generalization: A unifying view. *Machine Learning*, 1(1):47–80, 1986.

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

[OG90] Pekka Orponen and Russell Greiner. On the sample complexity of finding good search strategies. In *Proceedings of COLT-90*, pages 352–58, Rochester, August 1990.

[RBK88] R. Ramakrishman, C. Beeri, and R. Krishnamurthy. Optimizing existential datalog queries. In *Proc. of 7th Symposium on Principles of Database Systems*, pages 89–102, Austin, TX, March 1988.

[SAC+79] P.G. Selinger, M.M. Astrahan, D. D. Chamberlin, R.A. Lorie, and T.G. Price. Access Path Selection in a Relational Data Base Management System. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 23–34, 1979.

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

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

[Ull89] J. Ullman. *Principles of Data Base and Knowledge Base Systems*, volume 2. Addison Wesley, 1989.

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

[Vie89] L. Vieille. Recursive Query Processing: The Power of Logic. *Theoretical Computer Science*, 68(2), 1989.

[Zan88] Carlo Zaniolo. Design and Implementation of a Logic Based Language for Data Intensive Applications. In *Proceedings of the 5th International Conference on Logic Programming*, 1988.