Online Distributed Predicate Evaluation

Tomás Feder Rajeev Motwani Liadan O'Callaghan Rina Panigrahy
Dilys Thomas *

December 10, 2003

Abstract

An increasing number of databases store not only alphanumeric data, but also images, histograms, and other, more complicated data objects. Querying such databases can involve the evaluation of expensive predicates. Furthermore, since the data from a single database may be stored on a distributed system, certain predicate evaluations may involve remote procedure calls or transfers over a network. Various algorithms exist for efficiently answering queries under such conditions [3, 4, 6, 10]. Here we consider the following formalization: given relations A and B, and predicates P and Q, find all $(a, b) \in A \times B$ with $P(a)$ and $Q(b)$. In the "max" version of the problem, the goal is to minimize the number of times either P or Q is evaluated, where both may be evaluated simultaneously; in the "sum" version it is to minimize the sum of these quantities for P and Q. We provide a linear-time 2-approximation for the "sum" case and show that no deterministic algorithm can do better. We show a lower bound of 1.5 for the approximation ratio achievable by randomized algorithms and give algorithms achieving this bound for many classes of graphs. For the "max" problem, the best known upper bounds for polynomial-time algorithms is 3 for deterministic and 2.67 for randomized algorithms, while 2 is the lower bound for deterministic algorithms. We consider a "partial max" problem, where the evaluation of a predicate may be suspended and continued later, and provide a linear-time, deterministic 2-approximation algorithm. For a generalization to k-partite graphs we obtain for both "partial max" and "sum" problems a tight k-approximation deterministic algorithm, and give better approximations in many special cases using randomization. We also generalize the deterministic 3 upper bound for the "max" problem to a bound of $O(k \log^2 k)$ on k-partite graphs.

Classification: Online Algorithms, Distributed Databases

1 Introduction

As more and more information is stored in distributed databases that increasingly contain images, histograms, documents, and other large objects in addition to traditional alphanumeric values, it is essential to develop efficient algorithms for mining data stored in a distributed fashion, and integrating data from autonomous and heterogeneous sources [11]. Deciding which tuples from a cross-product of relations will appear in a join may entail the evaluation of expensive predicates that require the retrieval of information from a remote source, external function calls, table manipulation, or image analysis, for example [7]. We consider an abstraction of this problem, which we call the Dynamic Bipartite Ordering Problem, or DBOP.

The DBOP is, given a bipartite graph with vertices that have known query costs and unknown binary values ("true" or "false"), to query vertices for their binary values until the set of "true"

*tomas@theory.stanford.edu, rajeev@cs.stanford.edu, loc@cs.stanford.edu, rinap@cisco.com, dilys@cs.stanford.edu Supported in part by NSF Grant EIA-0137761

edges has been identified. (Here, a "true" edge is simply an edge whose endpoints are both "true.") The DBOP is important for database query optimization, as noted by Bouganim *et al.* [2] and Laber *et al.* [7, 8]. This problem arises when one wishes to identify all members (a, b) of a given subset of $A \times B$ (for database relations A and B) that satisfy $P(a)$ and $Q(b)$, where P and Q are predicates that are expensive to evaluate. For example, the relations could be $X = \{x_1, x_2, ...x_m\}$, a table of satellite images, and $Y = \{y_1, y_2, ..y_n\}$, a table of cities, and the predicates could be P, defined on X, indicating that the image shows cloudy weather, and Q, defined on Y, indicating that the city has high average yearly pollution. The elements of X and Y would each have an associated weight indicating the cost of evaluating the corresponding predicate on that element. Given a query such as, "Show all pairs of a city y_i and an image x_j such that y_i represents a photo taken in the city x_j and showing cloudy weather, and the city x_j has at least some minimum level l of pollution," the query engine would aim to identify the correct set of pairs while minimizing either the sum of all query costs, or the maximum of the amount spent evaluating P and the amount spent evaluating Q. Chaudhuri and Shim [3], Chimenti *et al.* [4], Hellerstein and Stonebraker [6], and Mayr and Seshadri [10] describe practical database systems in which the DBOP arises.

1.1 Problem Statement

An instance of the DBOP consists of a bipartite graph $G = (U, V, E)$ with for each $v \in U \cup V$ a nonnegative cost $w(v)$ and an unknown binary value ("true" or "false"). The binary value of a vertex v can be obtained at a cost of $w(v)$. An edge of G is true if both its endpoints are true, and false otherwise. The aim is to query a subset of the vertices, so as to identify the set of true edges, i.e., to guarantee that, for every true edge e, both endpoints of e will have been queried. We study two objective functions: $\sum_{v \in U' \cup V'} w(v)$ is the first one, while for the second one we only count a cost incurred as $w(v)$ for $v \in U'$ and as $w(v')$ for $v' \in V'$ only once if the two predicates are evaluated simultaneously, where U' is the subset of U that is ultimately queried, and V' is the set of queried members of V. We will refer to the minimization of the first function as the "sum" problem, and to the minimization of the second function as the "max" problem. The model for max is thus a time model, where a side of the bipartition queries v between times t and $t + w(v)$, and the total time for both sides is measured. We also consider the "partial max" problem, where the evaluation of a predicate may be partially carried out at some point in time, and then possibly continued later, so that possibly only a fraction w' of $w(v)$ or $w(v')$ is incurred, between times t and $t + w'$.

A generalization of the sum problem is the Dynamic k-partite Ordering Problem, an instance of which consists of a hypergraph $G = (V, E)$, with $V = \bigcup_{i=1}^{k} V_i$, such that each hyperedges in E contains at most 1 vertex from each V_i. Here, a hyperedge is true just if all its member vertices are true. The aim is to query a subset of the vertices so as to guarantee that the set of true hyperedges is identified (i.e., that the vertices of all true hyperedges have all been queried). This problem is a simple extension to the case that we are identifying which tuples from a join satisfy k predicates simultaneously. We also study "sum", "max", and "partial max" variants of Dynamic k-partite ordering.

1.2 Summary of Results

First we study the bipartite ordering problem. For the max problem, the previous best known algorithms were a deterministic 3-approximation algorithm and a randomized 2.67-approximation algorithm by Laber *et al.* [7]. In Section 3 we provide a deterministic 2-approximation algorithm for the partial max problem which a corresponding lower bound.. In Section 4 we give a 2-

approximation to the sum problem and show it is tight. In Section 5 we show that no randomized algorithm for the sum problem can have an expected approximation ratio lower than 1.5, and we present algorithms achieving this bound for many classes of graphs. The above results are then superseded by the generalization to k-partite graphs; we give tight k-approximation algorithms for both the single-processor (sum) and the multiple-processor (partial max) variants of the problem, as well as a bound for the multi-processor (max) of the form $O(k \log^2 k)$, where W is the largest cost, Sections 6 and 7. Randomization allows better approximations in many special cases, as shown in Section 8. Finally in Section 9 we indicate some open problems.

2 Preliminaries

We first provide a characterization of the vertices that need to be queried regardless of the strategy, which forms the basis of the results that follow. A subset of the following result appears in Laber *et al.* [7].

Lemma 1 *A set S of queried vertices terminates the querying process if and only if (1) S contains a vertex cover for G, and (2) S contains all neighbors of true vertices in G.*

PROOF: If S does not contain a vertex cover for G, then G has an edge $e = (u, v)$ such that neither u nor v have been queried, and it is possible that both u and v are true so that e is true. If S does not contain all neighbors of true vertices, then G has an edge $e = (u, v)$ for which u is true and v has not been queried, and it is possible that v is true so that e is true.

Conversely, if both conditions (1) and (2) are met, then for every edge $e = (u, v)$:

- If u and v are both true, they have both been queried as neighbors of true vertices, and so e is known to be true.
- If u is true and v is false, then v has been queried as a neighbor of a true vertex, and so e is known to be false.
- If u and v are both false then either u or v must belong to the vertex cover and will have been queried, and so e is known to be false.

$\blacksquare$

To facilitate analysis, we first define an "offline" version of the problem, which differs from the "online" formulation given above. Suppose that it is known in advance exactly which vertices are true. Then the offline problem is to identify a minimum-cost set of vertices containing a vertex cover and including every vertex with at least one true neighbor. The online problem is to identify such a minimum-cost set of vertices, without prior knowledge of the vertex values, but with the advantage that each query is answered before the next must be posed. The approximation factor of an online algorithm will be computed as the supremum, over all problem instances, of the ratio of the cost incurred by the online algorithm to the cost incurred by the optimal offline algorithm.

3 Deterministic Algorithm for the Partial Max Problem

Laber *et al.* [7, 8] and Bouganim *et al.* [2] study the DBOP in the model where the two predicates, one on each side of the bipartite graph, are evaluated by two processors in parallel. In this model the cost of a set of vertices queried equals the maximum of the total costs of each side rather than their sum. For this model, the offline unit cost problem is NP-complete [8] even in the case where all vertices are false, since it corresponds to finding a vertex cover minimizing the maximum of the number of vertices chosen from one side and the number of vertices from the other side. In fact, it is not known how to approximate this version of vertex cover to a factor better than 2. For the online problem, the lower bounds of 2 for deterministic algorithms (shown by Laber *et al.* [8]) and 1.5 for randomized algorithms (shown by Laber *et al.* [7]) were similarly obtained, and the best known upper bounds for a polynomial-time online algorithm are 3 for deterministic algorithms [8] and 2.67 for randomized algorithms [7]. We achieve a deterministic, linear-time, online 2-approximation for the related partial max problem. Our new linear-time techniques are also more practical for database systems having millions of tuples, as compared to previous polynomial-time algorithms which rely on LP rounding and randomization. This Section resolves the open problems that are posed by Laber *et al.* [7].

Theorem 1 *The maximum cost of the two sides with arbitrary given costs can be approximated by a deterministic online algorithm within a factor of* 2.

PROOF: We consider a flow problem on $G = (U, V, E)$ where each vertex u in U has supply equal to the cost of u, each vertex v in V has demand equal to the cost of v, and the edges in E are oriented from U to V with infinite capacity.

The algorithm computes a maximal flow for G, and queries the set S_1 of vertices in U whose supply was fully used and also partially queries vertices in U whose supply was partially used, queries the set S_2 of vertices in V whose demand was fully satisfied and also partially queries vertices in V whose demand was partially satisfied. Then the algorithm queries the set S_3 of neighbors of true vertices in S_1, and the set S_4 of neighbors of true vertices in S_2. The algorithm returns the set $S = S_1 \cup S_2 \cup S_3 \cup S_4$.

Let P be the optimal solution. Divide the cost of P into two parts Q and R, where a queried vertex v in P with cost w_v and flow $f_v \leq w_v$ contributes f_v to Q and $w_v - f_v$ to R. Divide Q into Q_1 and Q_2 for vertices in U and V respectively, and divide R into R_1 and R_2 for vertices in U and V respectively. The cost of the optimal solution is then $\max(Q_1 + R_1, Q_2 + R_2)$.

Let P' be the solution obtained by the algorithm. Define similarly Q', R', Q'_1, Q'_2, R'_1, R'_2, so that the cost of the solution obtained by the algorithm is then $\max(Q'_1, Q'_2) + \max(R'_1, R'_2)$. Note that $Q'_1, Q'_2 \leq F$, where F is the value of the flow, while $Q_1 + Q_2 \geq F$ since the optimal solution must form a vertex cover. Note also $R'_1 \leq R_1$ and $R'_2 \leq R_2$ since R'_1, R'_2 consist only of neighbors of true which must also be queried for R_1, R_2.

The bound then follows from

$$\begin{aligned}\max(Q'_1, Q'_2) + \max(R'_1, R'_2) &\leq F + \max(R_1, R_2) \leq Q_1 + Q_2 + \max(R_1, R_2) \\ &\leq Q_1 + Q_2 + R_1 + R_2 \leq 2\max(Q_1 + R_1, Q_2 + R_2).\end{aligned}$$

■

The approximation-ratio lower bounds of 2 for deterministic and 1.5 for randomized algorithms, hold for the max online problem even on the family of matchings $G = (V_1, V_2, E)$ with n disjoint edges and vertices of unit cost. A very simple randomized online algorithm achieves ratio 1.5 on matchings. The algorithm randomly selects n vertices to be queried, one from each edge, in such a way that $\lfloor \frac{n}{2} \rfloor$ of the queried vertices are in one side and $\lceil \frac{n}{2} \rceil$ of the queried vertices are in the other side. After querying these n vertices, the algorithm finishes by querying the neighbors of true vertices queried.

Theorem 2 *The max problem on a matching of even size with unit costs has a randomized online* 1.5 *approximation algorithm.*

PROOF: See Appendix. ■

4 Deterministic Algorithm for the Sum Problem

We first show a polynomial-time algorithm that solves the offline problem exactly. Next, we will show that no deterministic online algorithm can guarantee an approximation ratio lower than 2, and then we will present a deterministic algorithm that achieves this bound.

Theorem 3 *The offline sum problem can be solved exactly in polynomial time.*

PROOF: The offline algorithm first queries the set S_1 of neighbors of true vertices, and removes S_1 from the graph to obtain a graph G' on the remaining vertices. The algorithm then queries a minimum-cost vertex cover S_2 for the bipartite graph G'. Note that $S_1 \cup S_2$ satisfies the conditions of Lemma 1. Furthermore for any solution S, by Lemma 1, S_1 must be contained in S, and S must also contain a vertex cover for G' which has cost at least the cost of S_2, so the solution obtained is indeed of minimum possible cost. ■

Theorem 4 *No deterministic online algorithm can solve the unit cost sum problem with an approximation factor better than* 2.

PROOF: Suppose $G = (X, Y, E)$, with $X=\{x_1, x_2, ...x_n\}$, $Y=\{y_1, y_2, ..., y_n\}$, and $E = \{(x_i, y_i)|1 \leq i \leq n\}$. That is, G consists of a perfect matching of the vertices. Suppose that for every i, exactly one of x_i and y_i is true, and let all vertices be of equal cost. Then the optimal algorithm verifies just the n false vertices. By contrast, the queries of a deterministic algorithm can be answered in such a way that, for each pair (x_i, y_i), the first vertex queried is true and the second false. In this case, the deterministic algorithm must query all $2n$ nodes, incurring twice the optimum cost. ■

We now provide a deterministic algorithm that guarantees a 2-approximation, which, by the previous result, is tight.

Theorem 5 *There is a deterministic, linear-time, online algorithm that guarantees a 2-approximation to the sum problem, when the vertices have arbitrary costs.*

PROOF: The algorithm for the max problem suffices as

$$Q'_1 + R'_1 + Q'_2 + R'_2 \leq 2F + R_1 + R_2 \leq 2(Q_1 + R_1 + Q_2 + R_2).$$

■

This linear-time algorithm for the sum problem scales well for database applications with millions of tuples unlike algorithms that can be obtained from results in Laber *et al.* [8] which are based on computing the minimum vertex cover (and neighbours of true vertices), like the offline algorithm.

In fact, we can similarly obtain a 2-approximation for the sum problem in graphs that are not necessarily bipartite, as a special case of the generalization to hypergraphs in the Section 6.

5 Randomized Algorithm for the Sum Problem

In this section we first show that a randomized algorithm cannot guarantee an approximation ratio better than 1.5. We then provide randomized algorithms that guarantee 1.5-approximations for many special classes of graphs.

Theorem 6 *No randomized online algorithm can guarantee an expected approximation ratio better than* 1.5 *for the unit-cost problem.*

PROOF: Suppose $G = (X, Y, E)$, with $X = \{x_1, x_2, ...x_n\}$, $Y = \{y_1, y_2, ..., y_n\}$, and $E = \{(x_i, y_i) | 1 \leq i \leq n\}$. If for every i, exactly one of x_i and y_i is true, then the optimal algorithm verifies just the n false vertices. However, if for each (x_i, y_i), each vertex has probability 0.5 of being the true member of the pair, the randomized algorithm will pay an expected cost of 1.5 per pair x_i, y_i, since it will pay 1 if it first queries the false vertex, and 2 if it first queries the true vertex. The expected cost for the randomized algorithm is thus $1.5n$, 1.5 times the best possible. ■

We are able to achieve the factor of 1.5 via randomization in several special cases.

Theorem 7 *In the unit cost case, if G has a perfect matching, then a simple randomized online algorithm achieves an approximation factor of* 1.5. *In particular this factor* 1.5 *is achieved if G is regular of degree d.*

PROOF: See Appendix. ■

To generalize this result to the weighted case, we consider a flow problem on $G = (U, V, E)$ where each vertex u in U has supply equal to the cost of u, each vertex v in V has demand equal to the cost of v, and the edges in E are of infinite capacity and oriented from U to V. A perfect flow in G is a flow that meets all supplies and demands.

Theorem 8 *With arbitrary costs, if G has a perfect flow, then a randomized online algorithm achieves an approximation ratio of* 1.5. *In particular this ratio is achieved if the cost of each vertex equals its degree.*

PROOF: See Appendix. ■

Theorem 9 *If G is a forest with unit costs, then a randomized online algorithm achieves an approximation ratio of* 1.5.

PROOF: See Appendix. ■

Theorem 10 *If $G = (U, V, E)$ is a complete bipartite graph, and $\Sigma_u w(u) = a, \Sigma_v w(v) = b$, then a randomized online algorithm achieves factor $r = 1 + \frac{ab}{a^2+b^2} \leq 1.5$.*

PROOF: See Appendix. ■

6 Deterministic Algorithm for Sum on k-sets

We now consider the following generalization of the problem on a hyper-graph $G = (V, E)$ We are given a set of vertices V with a cost $w(v)$ for each $v \in V$. We are also given a set E of hyper-edges e, which are subsets $e \subseteq V$ of size at most k. Every vertex in V is either true or false. The goal is to identify all hyper-edges e such that all elements of e are true while querying a subset of vertices of minimum possible total cost (sum problem).

Sections 6 and 7 extend the results in Laber *et al.* [8] and resolve the open problems posed therein. We further provide tight online algorithms that achieve provable lower bounds. Results of previous Sections can be extended to non-bipartite graphs using these algorithms.

Define a k-matching $G = (V, E)$ to have vertices v_{ij} for $1 \leq i \leq n$, $1 \leq j \leq k$, and hyper-edges $e_i = \{v_{ij} : 1 \leq j \leq k\}$, with $|e| = k$.

Theorem 11 *The unit cost sum problem on a k-matching cannot be approximated by a deterministic online algorithm within a factor better than k.*

PROOF: Consider an instance where exactly one v_{ij} from each e_i is false. The optimal solution queries the false v_{ij} from each e_i, at a total cost of n. Given a deterministic algorithm, we can set each v_{ij} queried to true, unless it is the last v_{ij} to be queried among the elements of some e_i, in which case we set v_{ij} to false. This forces the deterministic algorithm to query all kn vertices, costing a factor of k. ■

We now consider the following algorithm for the sum problem on a hyper-graph $G = (V, E)$. Repeatedly select a hyper edge $e_i = \{v_{ij}\}$. Let w be the minimum of $w(v_{ij})$ over $v_{ij} \in e_i$. Query all $v_{ij} \in e_i$ having $w(v_{ij}) = w$. For each v_{ij} queried, if v_{ij} is false, then remove from E all hyper-edges e' containing v_{ij}; if v_{ij} is true, then remove from each hyper-edge $e' \in E$ with $v_{ij} \in e'$ the element v_{ij}, that is, replace e' with $e' \setminus \{v_{ij}\}$, and if e' becomes empty then remove e' from E. Finally replace the weight $w(v_{ij})$ by $w(v_{ij}) - w$ for each $v_{ij} \in e$, and repeat the process by selecting the next hyper-edge e. The algorithm terminates when E becomes empty.

Theorem 12 *The above deterministic online algorithm approximates the sum problem within a factor of k.*

PROOF: The algorithm correctly solves the sum problem, since each hyper-edge removed has either all elements true or at least one element false.

Consider a vertex v of weight $w(v)$. When a hyper-edge e with $v \in e$ is selected and weight w is subtracted from $w(v)$, set $w(v, e) = w$. Thus $w(v) \geq \sum w(v, e)$ for each v, e with $v \in e$. The optimal solution must query at least one vertex v from each e selected, and pay cost $w = w(v, e)$. Let C be the sum of these costs w that must be paid by the optimal solution for each edge e. Then $kC \geq \sum w(v, e)$, since each hyper-edge e selected contains at most k vertices v with $w(v, e) = w$. Furthermore, if v is queried, then its final value $w(v)$ is zero, so $w(v) = \sum w(v, e)$. Therefore the sum of the costs of elements queried is at most kC, proving a factor of k approximation. ■

This algorithm can be made to run in k querying phases, where in each phase we select a set of vertices to be queried, and submit these queries simultaneously. The reason is that in each phase we can select a set of hyper-edges e and subtract the corresponding w from vertices in e, such that for each hyper-edge e' of maximal cardinality, at least one vertex v in e' has its weight reduced to zero and is therefore queried; the size of the maximum cardinality hyper-edge is thus reduced by one in each phase, so that the number of querying phases is bounded by k.

7 Deterministic Algorithm for Max on k-sets

Suppose now that the hypergraph $G = (V, E)$ has its vertex set V given by the union of disjoint sets V_j with $1 \leq j \leq k$. Suppose further that each $e \in E$ contains at most one element from each V_j. Let now the cost of an algorithm be the maximum over all V_j of the cost incurred within V_j.

Theorem 13 *The unit cost max and partial max problems on a k-matching cannot be approximated by a deterministic online algorithm within a factor better than k.*

PROOF: Set $n = rk$, and the vertices of the hyper-graph partitioned into $V_j = \{v_{ij} : 1 \leq i \leq n\}$ for $1 \leq j \leq k$. Consider an instance where exactly one v_{ij} from each e_i is false, and exactly r v_{ij} from each V_j are false. The optimal solution queries the false v_{ij} and has cost r for each V_j, hence the maximum cost is $r = n/k$. We claim that any deterministic algorithm queries all n vertices in some V_j, and therefore has cost $n = rk$, off by a factor of k from the optimum. For each vertex queried, return true, until for some s of the k sets V_j, $1 \leq s \leq k-1$, we have $n\frac{k-s}{k} = r(k-s)$ hyper-edges e_i for which all $v_{ij} \in e_i$ with V_j being one of the s selected sets V_j have v_{ij} queried and true. The remaining hyper-edges e_i must have $v_{ij} \in e_i$ false for some V_j out of the s selected V_j. The argument follows by induction on these rs hyper-edges e_i and the selected s sets V_j, so that one of these s sets V_j will have all rs hyper-edges e_i queried for a total of $rs + r(k-s) = n$ elements queried in V_j. The base case $s = 1$ requires querying all the r elements in the corresponding V_j since they must be false. This proves factor $n/r = k$. ■

Theorem 14 *The above deterministic online algorithm approximates the partial max problem within a factor of k.*

PROOF: Define $w(v, e)$ as for the sum problem, and let C be the sum of the costs w that must be paid by the optimal solution to the sum problem for each edge e. For each set V_j, at most one element $v \in V_j$ gets charged $w(v, e) = w$ for the hyper-edge e, since e contains at most one element from V_j. Then $\sum_{v \in V_j} w(v, e) \leq C$, and since each element queried has $w(v) = \sum w(v, e)$, the cost paid by the algorithm for each set V_j is at most C. Since the optimal solution for the sum problem pays cost C, the optimal solution for the max problem pays cost at least C/k for some V_j, proving a factor of k approximation. ■

More generally, we may assume that $k = rs$, the hypergraph $G = (V, E)$ has its vertex set V given by the union of disjoint sets V_j with $1 \leq j \leq r$, and furthermore each $e \in E$ contains at most s elements from each V_j. Let now the cost of an algorithm be the maximum over all V_j of the cost incurred within V_j. Again, if C is the cost of the optimum solution to the sum problem, then the optimal solution to this partial max problem pays cost at least C/r on some V_j, while the solution given by the algorithm has cost at most Cs, giving again an approximation bound of $rs = k$.

Theorem 15 *The max problem can be approximated by a deterministic algorithm within a factor of $k^2 - k + 1$. This bound can be improved to $k(2\log_5(k-1) + c_k)^2 + 1$ for $k \geq 13$, with $0.2229 < c_k < 0.5177$.*

PROOF: To obtain a factor of $k^2 - k + 1$, we apply the algorithm as for partial max, but we delay the querying until we have selected enough hyperedges so that at least one element in each hyperedge has its cost reduced to 0. We then query all the elements that have their cost reduced to 0, with total cost that does not exceed the cost for sum, and thus at most k times the cost for max. After this first querying phase, the size k of the hyperedges has been reduced to $k - 1$, then the next querying phase reduces the size of the hyperedges to $k - 2$, and so on. After $k - 1$ phases, the size of the hyperedges is reduced to 1, and the cost incurred so far is at most $(k-1)k$ times the cost for max. The last phase has hyperedges of size 1, and thus incurs at most the cost for max, giving at total cost of at most $(k-1)k + 1$ times the cost for max.

For the improved bound, consider the k phases just described for partial max. The last phase k with hyperedges of size 1 queries elements that must be queried, so if these elements are queried we incur at most the cost for max. Consider the first $k - 1$ phases. Suppose we are considering $\lceil x \rceil \leq k - 1$ consecutive phases, and an element v with cost $w(v)$ whose querying by partial max is done within these $\lceil x \rceil$ phases. Partition the $\lceil x \rceil$ phases into 5 consecutive subsets of at most $\lceil x/5 \rceil$ phases. Set $\alpha_x = \frac{1}{2}(2\log_5 x + c)$ for some constant c, and consider β_x such that $1/\alpha_x + 1/\beta_x = 1$. If a fraction at least $w(v)/(4\alpha_x)$ of $w(v)$ is queried during one of these 5 subsets without completing the querying of v, then v may be queried fully at the end of this subset of phases. Such queries cost at most $4\alpha_x$ times the cost for sum. Otherwise a fraction at least $w(v)/\beta_x$ is queried in a subset of phases that completes the querying of v. This gives the recurrence $f(x) = 4\alpha_x + f(x/5)\beta_x$, with the total cost for the algorithm given by $kf(k-1)+1$. The recurrence resolves to $f(x) = (2\alpha_x)^2$, since $(2\alpha_x)^2 = 4\alpha_x + (2\alpha_x - 2)^2(\alpha_x/(\alpha_x - 1))$. For the base case with $4 < x \leq 20$, if $x \leq rs$ for integers r and s, then we may decompose the $\lceil x \rceil$ phases into r groups of at most s phases, so the recurrence gives $f(x) = (r-1)\alpha + s\beta = (\sqrt{r-1} + \sqrt{s})^2$ for an appropriate choice of α, β with $1/\alpha + 1/\beta$, and one may verify that $f(x) = (2\alpha_x)^2$ for an appropriate choice of c for each $4 < x \leq 20$ satisfying $0.2229 < \sqrt{6} - 2\log_5 6 \leq c \leq \sqrt{13} - 2\log_5 12 < 0.5177$. ■

8 Randomized Algorithm for Sum on k-sets

We do not know if the approximations achieved can be improved with randomization, since the lower bounds are weaker.

Theorem 16 *The unit cost sum and max problems on a k-matching cannot be approximated by a randomized online algorithm within a factor better than $\frac{k+1}{2}$.*

PROOF: Consider the solution that selects a single false vertex at random for each of the n hyper-edges e_i. The optimal solution for the sum problem has cost n since it queries the n false vertices. For the sum problem, the probability that we will query at most t of the k vertices in a given e_i is $\frac{t}{k}$, so the probability that we will query exactly t of the k vertices is $\frac{1}{k}$. The expected number of vertices queried for a given e_i is thus $\sum_{1 \leq t \leq k} \frac{t}{k} = \frac{k+1}{2}$, so the expected number of vertices queried is $\frac{k+1}{2}n$, a factor of $\frac{k+1}{2}$ from the optimal solution. For the max problem, the expected maximum number of vertices queried over all V_i is thus at least $\frac{k+1}{2k}n$, while with very high probability no V_i will end up with more that $\frac{n}{k}(1 + o_n(1))$ vertices being false, proving the lower bound of $\frac{k+1}{2}$ for randomized algorithms. ■

Consider the offline problem. We must query all true vertices that belong to hyper-edges that only contain true vertices. We may then remove all true vertices from the remaining hyper-edges. The resulting problem on false vertices is to find a vertex cover of minimum cost. The offline problem is thus as easy and as hard to approximate as hyper-graph vertex cover. It is not known how to approximate the sum vertex cover on general hyper-graphs within a factor better than k; our deterministic online algorithm achieves this bound for the sum problem. It is also not known how to approximate the max vertex cover on k-partite hyper-graphs within a factor better than k; our deterministic online algorithm also achieves this bound for the max problem. The sum vertex cover on k-partite hyper-graphs, on the other hand, can be approximated within a factor of $\frac{k}{2}$ by an algorithm due to Lovász [1, 9, 5]. It is thus conceivable that a randomized online algorithm for the sum problem on k-partite hyper-graphs could match the lower bound $\frac{k+1}{2}$, just like we have shown in many cases for $k = 2$ to obtain a factor of 1.5.

More generally, the result of Lovász has been generalized as follows by Aharoni *et al.* [1]. Consider a partition $p_1 + p_2 + \cdots + p_r$ of k such that $p_i \leq \frac{k}{2}$, and suppose that the hyper-graph $G = (V, E)$ whose hyper-edges have at most k elements has its vertex set partitioned into disjoint sets $V_1, V_2, \ldots, V_r$ such that no V_i contains more than p_i vertices in any given hyper-edge e. Then the sum vertex cover can be approximated within a factor of $\frac{k}{2}$. This motivates the following result.

Given a hyper-graph $G = (V, E)$, a perfect flow on G is an assignment of nonnegative flow values $f(e)$ to each hyper-edge in E, such that for each vertex v in G, the sum of the flows $f(e)$ over hyper-edges e containing v equals $w(v)$.

Theorem 17 *Suppose the hypergraph $G = (V, E)$ has a perfect flow. Suppose also that the vertex set V is partitioned into two disjoint sets V_1 and V_2 such that for some $0 < \alpha < 1$, each hyper-edge*

e contains at most αk vertices in V_1 and at most $(1-\alpha)k$ vertices in V_2. Then the sum problem on G can be approximated by a randomized online algorithm within a factor $(1-\alpha(1-\alpha))k$. In particular, if $\alpha = \frac{1}{2}$, so that each of V_1, V_2 has at most $\frac{k}{2}$ vertices from any hyper-edge e, then we obtain an approximation factor of $\frac{3}{4}k$ when a perfect flow exists.

PROOF: Let F be the value of the perfect flow, and let C be the cost of an optimal solution to the sum problem. Since a solution must cover every hyper-edge, we may assign $f(e)$ to the vertex v that covers e in the optimal solution, so that no such vertex v is assigned more than $w(v)$. Therefore $C \geq F$.

The algorithm is as follows. Query all of V_1 with probability α, or all of V_2 with probability $1-\alpha$. If V_1 is first queried, then solve the problem in V_2 by the deterministic algorithm achieving a factor of $(1-\alpha)k$, since hyper-edges in V_2 have at most $(1-\alpha)k$ vertices. Similarly, if V_2 is first queried, then solve the problem in V_1 by the deterministic algorithm achieving a factor of αk, since hyper-edges in V_1 have at most αk vertices.

If V_1 is first queried, then this costs at most $\alpha k F$, since the perfect flow $f(e)$ assigned to the at most αk vertices in V_1 containing e accounts for all $w(v)$ for v in V_1. Similarly, if V_2 is first queried, then this costs at most $(1-\alpha k)F$, since the perfect flow $f(e)$ assigned to the at most $(1-\alpha)k$ vertices in V_2 containing e accounts for all $w(v)$ for v in V_2.

Decompose the optimal solution as $C = C_1 + C_2$, where C_i is the cost for vertices in V_i. If V_1 is first queried, then the total cost is at most $\alpha k F + (1-\alpha)kC_2 \leq \alpha k C + (1-\alpha)kC_2$. If V_2 is first queried, then the total cost is at most $(1-\alpha)kF + \alpha k C_1 \leq (1-\alpha)kC + \alpha k C_1$. The expected total cost is thus at most

$$\alpha(\alpha k C + (1-\alpha)kC_2) + (1-\alpha)((1-\alpha)kC + \alpha k C_1)$$
$$= (\alpha^2 + (1-\alpha)^2 + \alpha(1-\alpha))C = (1-\alpha(1-\alpha))kC.$$

This proves the $(1-\alpha(1-\alpha))k$ approximation factor. ■

Say that a hypergraph $G = (V, E)$ is an l-forest for $l < k$ if either E is empty or E contains a hyper-edge e that meets the remaining hyper-edges in at most l vertices, and the hyper-graph resulting from removing e from E is an l-forest. A hyper-forest is an l-tree for $l = 1$.

Theorem 18 *If G is an l-forest with unit costs, then a randomized online algorithm for the sum problem achieves factor $\frac{k+l+1}{2}$. In particular, if G is a hyper-forest with unit costs, then the factor is $\frac{k}{2}+1$.*

PROOF: See Appendix. ■

A complete k-partite hypergraph $G = (V, E)$ has V given by the union of disjoint sets V_i for $1 \leq i \leq k$ and E consisting of all subsets of V having exactly one element in each V_i.

Theorem 19 *If G is a complete k-partite hypergraph, then a randomized online algorithm for the sum problem achieves factor $\frac{k+1}{2}$.*

PROOF: See Appendix. ■

9 Open Problems

The following problems remain open. For the max and sum problems for bipartite graphs, are there randomized a-approximation algorithms for $1.5 \leq a < 2$? Also, more generally, are there randomized b-approximation algorithms for the max or sum problems in hypergraphs that have at most k nodes per edge, where $\frac{k+1}{2} \leq b < k$?

References

[1] R. Aharoni, R. Holzman, and M. Krivelevich, "On a theorem of Lovász on covers in r-partite hypergraphs", Combinatorica 16 (1996) 149–174.

[2] L. Bouganim, F. Fabret, F. Porto, and P. Valduriez, "Processing queries with expensive functions and large objects in distributed mediator systems", IEEE Intl. Conf. Data Engineering 2001.

[3] S. Chaudhuri and K. Shim, "Query optimization in the presence of foreign functions", Intl. Conf. Very Large Databases 1993.

[4] D. Chimenti, R. Gamboa, and R. Krishnamurthy, "Towards an Open Architecture for LDL", Intl. Conf. Very Large Databases 1989.

[5] Z. Füredi, "Matchings and covers in hypergraphs," Graphs and Combinatorica 4 (1988) 115–206.

[6] J. M. Hellerstein and M. Stonebraker, "Predicate migration: optimizing queries with expensive predicates", ACM SIGMOD Intl. Conf. Management of Data 1993, 267–276.

[7] E. Laber, O. Parekh, and R. Ravi, "Randomized Approximation Algorithms for Query Optmization Problems on Two Processors", ESA 2002, Lecture Notes in Computer Science 2461, 649–661.

[8] E. S. Laber, F. Porto, P. Valduriez, and R. Guarino, "Cherry picking: A data-dependency driven strategy for the distributed evaluation of expensive predicates", Tech. Rep. 01, Departamento de Informatica, PUC-RJ, Rio de Janeiro, Brasil, April 2002, pp. 650, 651, 652, 654, 660.

[9] L. Lovász, "On minimax theorems of combinatorics," Doctoral Thesis, Mathematikai Lapok 26 (1975) 209–264. (Hungarian.)

[10] T. Mayr and P. Seshadri, "Client-site query extensions", ACM SIGMOD Intl. Conf. Management of Data 1999.

[11] Total Information Awareness System. http://www.darpa.mil/iao/TIASystems.htm

Appendix

Theorem 2. *The max problem on a matching of even size n with unit costs has a randomized online 1.5 approximation algorithm.*

Proof of Theorem 2. Consider an optimal solution that queries for r of the edges the endpoint in V_1, for s of the edges the endpoint in V_2, and for t of the edges both endpoints in V_1 and V_2, where $r+s+t=n$. Say $r \geq s$. Two complementary choices of the random vertices to be queried decompose $r=r_1+r_2$, $s=s_1+s_2$, and $t=t_1+t_2$, with $r_1+s_1+t_1=r_2+s_2+t_2=\frac{n}{2}$. Say $t_2 \geq t_1$.

Then the first choice of queries gives total cost $\max(r+t+s_1, s+t+r_2)=r+t+s_1$, since $s_2-r_1=(r_2+s_2+t_2)-(r+t_2) \leq \frac{n}{2}-\frac{n}{2}=0$. If $r+t_1 \geq s+t_2$, then the second choice of queries gives total cost $\max(r+t+s_2, s+t+r_1)=r+t+s_2$, since $s_1-r_2=(r_1+s_1+t_1)-(r+t_1) \leq \frac{n}{2}-\frac{n}{2}=0$. The total expected cost is then $r+t+\frac{1}{2}(s_1+s_2)=r+t+\frac{1}{2}s \leq \frac{3}{2}(r+t)$. If $r+t_1 \leq s+t_2$, then the second choice of queries gives total cost $\max(r+t+s_2, s+t+r_1)=s+t+r_1$, since

$r_2 - s_1 = (r_2 + s_2 + t_2) - (s + t_2) \le \frac{n}{2} - \frac{n}{2} = 0$. The total expected cost is then $\frac{1}{2}(r+t+s_1) + \frac{1}{2}(s+t+r_1) = r + t + \frac{1}{2}(s + s_1 - r_2) \le \frac{3}{2}(r+t)$, since $s + s_1 - r_2 = (r_1 + s_1 + t_1) + (s - r - t_1) \le \frac{n}{2} \le r + t$. Thus the expected cost $\frac{3}{2}(r+t)$ is a factor of 1.5 off from the optimal cost $r + t$. ■

Theorem 7. *In the unit cost case, if G has a perfect matching, then a randomized online algorithm achieves factor* 1.5. *In particular this factor* 1.5 *is achieved if G is regular of degree d.*

Proof of Theorem 7. If $G = (U, V, E)$ has a perfect matching, then $|U| = |V| = n$ and a minimum vertex cover contains n vertices. The algorithm first queries S_1 which can be either U or V with probability 0.5. Then the algorithm queries the set S_2 consisting of the neighbors of true vertices in S_1, and returns the set $S = S_1 \cup S_2$. The proof being straightforward is omitted for want of space.

■

Theorem 8. *With arbitrary costs, if G has a perfect flow, then a randomized online algorithm achieves factor* 1.5. *In particular this factor* 1.5 *is achieved if the cost of each vertex equals its degree.*

Proof of Theorem 8. If G has a perfect flow, then the sum w of the costs in U equals the sum of the costs in V, and also equals the cost of a minimum vertex cover. As in the unit cost case, the algorithm first queries S_1 which can be either U or V with probability 0.5. Then the algorithm queries the set S_2 of neighbors of true vertices in S_1, and returns $S = S_1 \cup S_2$. If the neighbors of true vertices have cost k in U and l in V, then the total expected cost is $w + (k+l)/2 \le 1.5 \max(w, k+l)$.

If the cost of each vertex equals its degree, then sending one unit of flow along each edge gives a perfect flow. ■

Theorem 9. *If G is a forest with unit costs, then a randomized online algorithm achieves factor* 1.5.

Proof of Theorem 9. If the forest G is a matching, then the result follows from Theorem 5. Otherwise G contains an edge $e = (u, v)$ such that v is a leaf and u is not a leaf in G. Query u. If u is false then remove both u and v from G to obtain G' and proceed analogously with G'. If u is true then query all neighbors of u and repeatedly query all neighbors of true vertices queried until no new true vertices are found, then remove the queried vertices from G to obtain G' and proceed analogously with G'.

We prove factor 1.5 in both cases u false and u true. If u is false, then u is queried in the optimal solution, because if the optimal solution queries instead v to cover the edge $e = (u, v)$, then it could have queried u instead of v at the same cost. By induction, the algorithm achieves factor 1.5 on the vertices of G' and factor 1 on the removed vertices u, v, since the optimal solution also queries u and not v. This gives a total factor 1.5.

If u is true, then u is not necessarily queried by the optimal solution, but all vertices queried and removed to obtain G' other than u must be queried by optimal solution as well, since they are neighbors of true vertices. The number k of vertices other than u removed must satisfy $k \ge 2$, since u has at least two neighbors because it is not a leaf. This gives ratio $(k+1)/k \le 1.5$ on the vertices removed, and ratio 1.5 on G' by induction, for a total factor 1.5. ■

Theorem 10. *If $G = (U, V, E)$ is a complete bipartite graph, and $\Sigma_u w(u) = a, \Sigma_v w(v) = b$, then a randomized online algorithm achieves factor $r = 1 + \frac{ab}{a^2+b^2} \le 1.5$.*

Proof of Theorem 10. Query U with probability $p = \frac{b^2}{a^2+b^2}$ and query V with probability $q = 1 - p = \frac{a^2}{a^2+b^2}$. Then query the neighbors of true. The proof being straightforward is omitted for want of space.

■

Theorem 18. *If G is a complete k-partite hypergraph, then a randomized online algorithm for the sum problem achieves factor $\frac{k+1}{2}$.*

Proof of Theorem 18. Let a_i be the total weight of V_i. Repeatedly select one V_i at random with probability proportional to $\frac{1}{a_i^2}$, and query all of V_i. If all elements of V_i are false, stop. Otherwise select one of the remaining V_i at random again, until either all V_i have been queried, or some V_i with all its elements false has been queried.

Suppose every V_i has at least one true element. Then the optimal solution must query all elements of every V_i, because each element v of V_i belongs to some hyper-edge e such that the elements of e other than v are true, so that v must be queried. Therefore the algorithm performs optimally for this case.

Suppose instead some V_i has all elements false. Then the optimal solution queries only all the elements of the V_i of least weight a_i such that all elements of V_i are false. Suppose this V_i is V_1. Then the probability that some V_i with $i \neq 1$ is queried before V_1 is $\frac{1/a_i^2}{1/a_1^2+1/a_i^2} = \frac{a_1^2}{a_1^2+a_i^2}$. The expected cost from querying V_i is thus $a_i \frac{a_1^2}{a_1^2+a_i^2} = a_1 \frac{a_1 a_i}{a_1^2+a_i^2} \leq \frac{a_1}{2}$. The expected cost from querying all V_i for $i \neq 1$ is then at most $\frac{k-1}{2} a_1$, and the expected cost from querying V_1 is at most a_1, for a total expected cost of at most $\frac{k+1}{2} a_1$, within a factor of $\frac{k+1}{2}$ of the optimal cost a_1. ■

Theorem 19. *If G is an l-forest with unit costs, then a randomized online algorithm for the sum problem achieves factor $\frac{k+l+1}{2}$. In particular, if G is a hyper-forest with unit costs, then the factor is $\frac{k}{2}+1$.*

Proof of Theorem 19. Consider the hyper-edge e that meets the remaining hyper-edges in at most l vertices. Query these l vertices v_i. If one of these v_i is false, then we have queried l vertices from e, and e requires querying at least one vertex, so the ratio for e is $l < \frac{k+l+1}{2}$. If all l vertices v_i are true, then query the remaining vertices in e, at most $k-l$ of them, in random order, until a false vertex is found in e, or all vertices in e are found to be true. If all vertices in e are true, then all vertices in e had to be queried, so the ratio for e is 1. If some vertex in e is false, say u, then for each true vertex w_j out of the remaining $k-l-1$ vertices in e other than the v_i and u, the probability that w_j is queried before u is $\frac{1}{2}$, so the expected number of vertices queried in e is $l + \frac{k-l-1}{2} + 1 = \frac{k+l+1}{2}$, for a ratio of $\frac{k+l+1}{2}$ with respect to the single vertex that must be queried in e. ■