Querying Priced Information in Databases: the Conjunctive Case

RENATO CARMO
Departamento de Informática da Universidade Federal do Paraná
and
TOMÁS FEDER
Computer Science Department, Stanford University
and
YOSHIHARU KOHAYAKAWA
Instituto de Matemática e Estatística da Universidade de São Paulo
and
EDUARDO LABER
Departamento de Informática da Pontificia Universidade Católica do Rio de Janeiro
and
RAJEEV MOTWANI and LIADAN O'CALLAGHAN
Computer Science Department, Stanford University
and
RINA PANIGRAHY
Cisco Systems
and
DILYS THOMAS
Computer Science Department, Stanford University

Some of the results in this paper appeared as an extended abstract at the Latin American Theoretical Informatics 2004 (LATIN 2004) [Laber et al. 2004].
Renato Carmo was partially supported by CAPES (PICDT) and CNPq (Proc. 476817/2003-0).
Eduardo Laber was partially supported by FAPERJ (Proc. E-26/150.715/2003) and CNPq (Proc. 476817/2003-0).
Yoshiharu Kohayakawa was partially supported by MCT/CNPq (ProNEx, Proc. CNPq 664107/1997-4) and CNPq (Proc. 300334/93–1, 468516/2000–0 and Proc. 476817/2003-0)
Author's addresses: renato@ime.usp.br, tomas@theory.stanford.edu, yoshi@ime.usp.br, laber@info.puc-rio.br, rajeev@cs.stanford.edu, loc@cs.stanford.edu, rinap@cisco.com, dilys@cs.stanford.edu.

Query optimization that involves expensive predicates has received considerable attention in the database community. Typically, the output to a database query is a set of tuples that satisfy certain conditions, and, with expensive predicates, these conditions may be computationally costly to verify. In the simplest case, when the query looks for the set of tuples that simultaneously satisfy k expensive predicates, the problem reduces to ordering the evaluation of the predicates so as to minimize the time to output the set of tuples comprising the answer to the query. We study different cases of the problem: the *sequential case*, in which a single processor is available to evaluate the predicates and the *distributed case*, in which there are k processors available, each one dedicated to each different attribute (column) of the database, and there is no communication cost between the processors.

For the sequential case, we give a simple and fast deterministic k-approximation algorithm, and prove that k is the best possible approximation ratio for a deterministic algorithm, even if exponential time algorithms are allowed. We also propose a randomized, polynomial time algorithm with expected approximation ratio $1 + \sqrt{2}/2 \approx 1.707$ for $k = 2$, and prove that $3/2$ is the best possible expected approximation ratio for randomized algorithms. We also show that given $0 \leq \varepsilon \leq 1$, no randomized algorithm achieves approximation ratio smaller than $1 + \varepsilon$ with probability larger than $(1 + \varepsilon)/2$.

For the distributed case we consider two different models: the *preemptive model*, in which a processor is allowed to interrupt the evaluation of a predicate in the middle, and the *non-preemptive model*, in which the evaluation of a predicate must be carried out through the end once it is started. We show that k is the best possible approximation ratio for a deterministic algorithm, even if exponential time algorithms are allowed. For the preemptive model we introduce a polynomial time k-approximation algorithm. For the non-preemptive model, we introduce an exponential time k-approximation algorithm and a polynomial time $O(k \log^2 k)$-approximation algorithm.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Computations on discrete structures*; *Sequencing and scheduling*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*Graph algorithms*; H.2.4 [**Database Management**]: Systems—*Query processing*; *Relational databases*; *Multimedia databases*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Information filtering*; *Search process*; *Selection process*; G.3 [**Probability and Statistics**]: Probabilistic algorithms

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: Competitive Analisys, Online Algorithms

1. INTRODUCTION

The main goal of *query optimization in databases* is to determine how a query over a database should be processed in order to minimize the user response time. A typical query extracts the tuples from a database relation that satisfy a set of conditions, or *predicates*, in database terminology. For example, consider the set of tuples $D = \{(a_1, b_1), (a_1, b_2), (a_1, b_3), (a_2, b_1)\}$ (see Fig. 1(a)) and a conjunctive query that seeks to extract the subset of tuples (a_i, b_j) for which the a_i satisfy predicate P_1 and the b_j satisfy predicate P_2. These predicates can be viewed together as a 0/1-valued function δ defined on the set of tuple elements $\{a_1, a_2, b_1, b_2, b_3\}$, with the convention that, $\delta(a_i) = 1$ if and only if $P_1(a_j)$ holds and $\delta(b_j) = 1$ if and only if $P_2(b_j)$ holds. The answer to the query is the set of pairs (a_i, b_j) with $\bar{\delta}(a_i, b_j) = \delta(a_i)\delta(b_j) = 1$. The query optimization problem that we consider is the one of determining a strategy for evaluating $\bar{\delta}$ so as to compute this set of tuples by evaluating as few values of the function δ as possible (or, more generally, with the total cost for evaluating the function $\bar{\delta}$ minimal).

It is usually the case that the cost (measured as the computational time) to evaluate the predicates of a query can be assumed to be bounded by a constant so that the query can be answered by just scanning through all the tuples in D while evaluating the corresponding predicates.

This is not the case with computationally expensive predicates. For instance, when the database holds complex data as images, tables, long sequences etc, this constant may happen to be so large as to render the usual strategy impractical. In such cases, the different costs involved in evaluating each predicate must also be taken into account in order to keep user response time within reasonable bounds.

Among several proposals to model and solve this problem (see, for example [Bouganim et al. 2001; Chaudhuri and Shim 1993; Hellerstein 1998]), we focus on the improvement of the approach proposed in [Porto 2001] where, differently from the others, the query evaluation problem is reduced to an optimization problem on a hypergraph (see Fig. 1). We study different cases of the problem: the *sequential case*, in which a single processor is available to evaluate the predicates and the *distributed case*, in which there are k processors available, each one dedicated to each different attribute (column) of the database and there is no communication cost between the processors.

1.1 Problem Statement

A *hypergraph* is a pair $G = (V(G), E(G))$ where $V(G)$, the set of *vertices* of G, is a finite set and each *edge* $e \in E(G)$ is a non-empty subset of $V(G)$.

The *rank of* G, denoted $r(G)$, is the maximum cardinality of an edge in G. A hypergraph G is said to be r-*uniform* if each edge has $r(G)$ vertices. Hypergraph G is said to be k-partite if there is a partition $\{V_1, \ldots, V_k\}$ of $V(G)$ such that no edge contains more than one vertex in the same partition class. Unless otherwise noted, every hypergraph G in this work will be uniform, $r(G)$-partite and $\{V_i(G)\colon\ 1 \leq\ i\ \leq\ r(G)\}$ denotes the partition of $V(G)$ under consideration.

The *neighborhood* of $v \in V(G)$ is the set

$$\Gamma(v) = \{u \in V(G) - \{v\}\colon \{u, v\} \subseteq e \text{ for some } e \in E(G)\}\,.$$

A *matching* in a hypergraph G is a set $M \subseteq E(G)$ with no two edges in M sharing a common vertex. A hypergraph G is said to be a *matching* if $E(G)$ is a matching. A *cover* for G is a set $C \subseteq V(G)$ such that every edge of G has at least one vertex in C.

Given a hypergraph G and a function $\delta : V(G) \to \{0, 1\}$ we define an *evaluation* of (G, δ) as a set $\mathsf{E} \subseteq V(G)$ such that, knowing the value of $\delta(v)$ for each $v \in \mathsf{E}$, one may determine, for each $e \in E(G)$, the value of

$$\bar{\delta}(e) = \prod_{v \in e} \delta(v). \tag{1}$$

For any $X \subseteq V(G)$, we let

$$\begin{aligned} \Gamma(X) &= \bigcup_{v \in X} \Gamma(v) \\ \Gamma_1(X) &= \{\Gamma(v)\colon v \in X \text{ and } \delta(v) = 1\}\,. \end{aligned}$$

Note that an evaluation for (G, δ) must contain both, the set $\Gamma_1(V(G))$ and a cover for G, otherwise there will be an edge for which the value of $\bar{\delta}$ cannot be determined.

Given a hypergraph G and a function $\gamma : V(G) \to \mathbb{R}$ we associate a *cost* $\gamma(X)$ to each subset X of $V(G)$, as a function of the values of $\gamma(x), x \in X$, for example,

$$\gamma(X) = \sum_{v \in X} \gamma(v).$$

See below for the different definitions of $\gamma(X)$ to be used in later sections.

Given an integer $k > 0$, an instance of the $\mathsf{Dynamic}$ k-$\mathsf{partite\ Ordering}$ problem ($\mathsf{D}k\mathsf{O}$) is a k-partite, k-uniform hypergraph G, together with functions δ and γ as above. The goal in $\mathsf{D}k\mathsf{O}$ is to determine an evaluation of minimum cost for (G, δ, γ). Observe that while the value of $\gamma(v)$ is known in advance for each $v \in V(G)$, the function δ is 'unknown' to us at first. More precisely, the value of $\delta(v)$ becomes known only when $\delta(v)$ is actually evaluated, and this evaluation costs $\gamma(v)$.

Before we proceed, let us observe that $\mathsf{D}k\mathsf{O}$ models our database problem as follows: the vertex classes $V_i(G)$, $1 \leq i \leq k$ correspond to the k different attributes of the relation that is being queried. Each vertex of G corresponds to a distinct attribute value (tuple element). The edges correspond to tuples in the relation, $\gamma(v)$ is the cost to evaluate δ on v and $\delta(v)$ corresponds to the result of a predicate evaluated at the corresponding tuple element. For this reason, we say that a vertex v is a *false vertex* or a *true vertex* when $\delta(v) = 0$ or $\delta(v) = 1$, respectively.

As described up to this point, $\mathsf{D}k\mathsf{O}$ stands for a whole family of related computational problems rather than a unique problem, since each different definition of the cost $\gamma(\mathsf{E})$ of an evaluation E may define a different problem. In every case, we will refer to a minimum cost evaluation as an *optimal evaluation.* If $\mathcal{I}$ is an instance of $\mathsf{D}k\mathsf{O}$ we will denote an optimal evaluation for $\mathcal{I}$ by $\mathsf{E}^*_{\mathcal{I}}$. We are interested in modelling our database problem in two different contexts.

The case in which there is only one processor available to evaluate the predicates will be called *the sequential case* of $\mathsf{D}k\mathsf{O}$, denoted $\mathsf{sD}k\mathsf{O}$. In this case, the *cost of an evaluation* E is defined as

$$\gamma(\mathsf{E}) = \sum_{v \in \mathsf{E}} \gamma(v).$$

The case in which there are k processors available, one dedicated to evaluate the predicates referring to each attribute of the database, will be called *the distributed case* of $\mathsf{D}k\mathsf{O}$, and will be denoted by $\mathsf{dD}k\mathsf{O}$. In the distributed case, we address both the *preemptive* and the *non-preemptive* models. In the former, we are allowed to interrupt the evaluating of a vertex before completed, while in the latter such interruption is not allowed.

Figure 1(b) shows an instance of $\mathsf{sD2O}$. The value of $\delta(v)$ is indicated inside each vertex v. Suppose that $\gamma(a_1) = 3$ and $\gamma(b_1) = \gamma(b_2) = \gamma(b_3) = 2$. In this case, any strategy that starts evaluating $\delta(a_1)$ will return the evaluation $\{a_1, b_1, b_2, b_3\}$, of cost 9. However, the evaluation of minimum cost for this instance is $\{b_1, b_2, b_3\}$, of cost 6. This example highlights the key point: the problem is to devise a strategy for dynamically choosing, based on the function γ and the values of δ already revealed, the next vertex v whose δ-value should be evaluated, so as to minimize the final, overall cost.

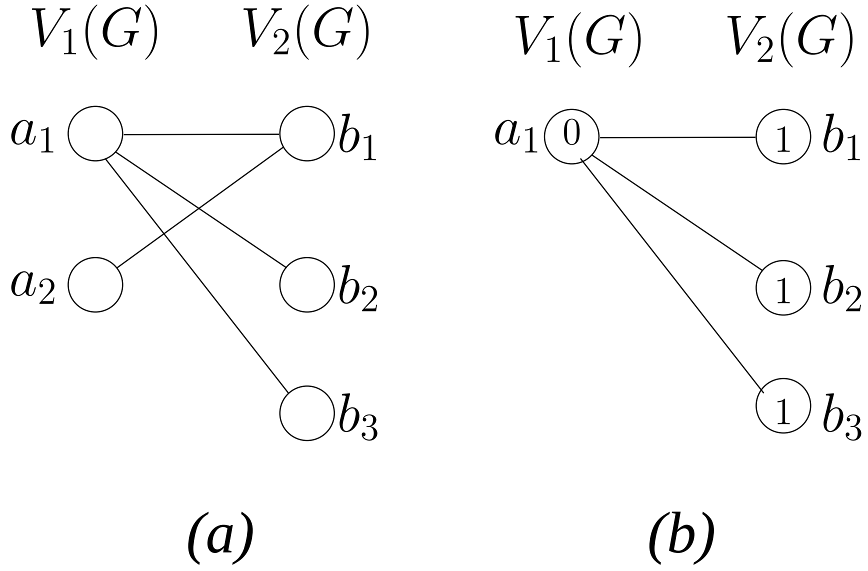

Fig. 1. The set of tuples $\{(a_1, b_1), (a_1, b_2), (a_1, b_3), (a_2, b_1)\}$ and an instance of D2O

In the distributed case, an evaluation is an object more complex than a set of vertices, because of the scheduling information involved. In order to remain close to the motivating problem, we allow for processors in idle state while waiting for others to complete their respective evaluations, or for processors aborting the evaluation of a vertex before completed, to start the evaluation of another vertex (preemption).

To illustrate the subtleties involved in these scheduling issues, let us consider an instance $\mathcal{I} = (G, \delta, \gamma)$ of dD2O, in which G has a single edge $e = \{v_1, v_2\}$ with costs $\gamma(v_1) = 1$ and $\gamma(v_2) = 1000$:

—In the non-preemptive model, if both processors start to evaluate simultaneously, the cost would be 1000, whatever the values of $\delta(v_1)$ and $\delta(v_2)$; if the processor in charge of $V_2(G)$ waits for the result of the evaluation of v_1 before processing v_2, the cost of the evaluation would be either 1001, if $\delta(v_1) = 1$ or 1, if $\delta(v_1) = 0$.

—In the preemptive model the cost would be either 1 if $\delta(v_1) = 0$, or 1000 if $\delta(v_1) = 1$.

The results we are concerned with here do not require us to explicitly formalize the scheduling information contained in an evaluation. It will be enough for our purposes to define the *cost of an evaluation* E as the time elapsed between the start of evaluation by the first processor and the end of the evaluation by the last processor.

1.2 Measuring the Performance of Algorithms for DkO

Let $\mathcal{A}$ be an algorithm for DkO and let $\mathcal{I} = (G, \delta, \gamma)$ be an instance of DkO. We will denote the evaluation computed by $\mathcal{A}$ on input $\mathcal{I}$ by $\mathcal{A}(\mathcal{I})$. Establishing a measure for the performance of a given algorithm $\mathcal{A}$ for DkO is somewhat delicate: for example, a worst case analysis of $\gamma(\mathcal{A}(\mathcal{I}))$ is not suitable since any correct algorithm should output an evaluation comprising all vertices in $V(G)$ when $\delta(v) = 1$ for every $v \in V(G)$. Hence the following definition.

Let $\mathcal{I} = (G, \delta, \gamma)$ be an instance of DkO and let E be an evaluation for $\mathcal{I}$. The *deficiency of evaluation* E *(with respect to* $\mathcal{I}$*)* is the ratio

$$d(\mathsf{E}, \mathcal{I}) = \frac{\gamma(\mathsf{E})}{\gamma(\mathsf{E}^*_{\mathcal{I}})}.$$

Given an algorithm $\mathcal{A}$ for DkO, we define the *deficiency of* $\mathcal{A}$ as the worst case deficiency of the evaluation $\mathcal{A}(\mathcal{I})$, where $\mathcal{I}$ ranges over all possible instances of the problem, that is,

$$d(\mathcal{A}) = \max \{d(\mathcal{A}(\mathcal{I}), \mathcal{I}) \colon \mathcal{I} \text{ is an instance of } \mathsf{D}k\mathsf{O}\}.$$

If $\mathcal{A}$ is a randomized algorithm, then $d(\mathcal{A}(\mathcal{I}), \mathcal{I})$ is a random variable, and the *deficiency* of $\mathcal{A}$ is defined as the maximum over all instances of the expected value of this random variable, that is,

$$d(\mathcal{A}) = \max \{\mathbb{E}\left[d(\mathcal{A}(\mathcal{I}), \mathcal{I})\right] : \mathcal{I} \text{ is an instance of } \mathsf{D}k\mathsf{O}\}$$

We say that algorithm $\mathcal{A}$ is a *deficiency-optimal algorithm for* DkO if its deficiency is the best possible. Clearly, in all scenarios we wish to devise fast algorithms whose deficiency is as close to 1 as possible. In this paper, we are concerned with designing algorithms for DkO, analyzing them and establishing bounds for their deficiency.

1.3 Statement of Results

The results in this work are organized as follows. Section 2 deals with the sequential case of DkO (sDkO) and Section 3 deals with the distributed case (dDkO). In Section 4 we indicate some open problems.

The study of the sequential case in Section 2 starts with the establishment of lower bounds on the deficiency of deterministic and randomized algorithms for sDkO (Section 2.1). These bounds apply for exponential time algorithms as well.

We then introduce a deficiency-optimal deterministic algorithm for sDkO with time complexity $O(k|E(G)| \log |V(G)|)$ in Section 2.2. This algorithm does not need to know the whole hypergraph in advance in order to solve the problem, since it scans the edges (tuples), evaluating each of them as they become available. This is a convenient feature for the database application that motivates this work.

Section 2.3 focuses on the study of the sequential case of the *Dynamic* 2*-partite Ordering Problem* (sD2O). We introduce $\mathcal{R}_\varepsilon$, a randomized algorithm for sD2O with time complexity $O(|V(G)|^3)$ whose expected deficiency is at most $2 - \varepsilon$ for any given $0 \leq \varepsilon \leq 1 - \sqrt{2}/2$. Clearly, the best bound for the expected deficiency of $\mathcal{R}_\varepsilon$ is achieved when $\varepsilon = 1 - \sqrt{2}/2$. However, the greater the value of ε, the greater the probability that a particular execution of $\mathcal{R}_\varepsilon$ will return a poor result. In Section 2.3.1 we prove that the algorithm's deficiency is always bounded from above by $1 + 1/(1 - \varepsilon)$.

The deficiency $\mathcal{R}_\varepsilon$ is not assured to be highly concentrated around the expectation. In Section 2.3.2, we show that this limitation is inherent to the problem, rather than a weakness of our approach. More precisely, we show that for any $0 \leq \varepsilon \leq 1$, no randomized algorithm can have deficiency smaller than $1 + \varepsilon$ with probability larger than $(1+\varepsilon)/2$. The proof of this fact makes use of Yao's Minimax Principle (see Theorem 2.2).

The study of the distributed case in Section 3 also starts with lower bounds on the deficiency of deterministic and randomized algorithms for d$\mathsf{D}k\mathsf{O}$ (Section 3.1). Again, these lower bounds for exponential time algorithms as well.

Section 3.2 introduces deterministic algorithms for d$\mathsf{D}k\mathsf{O}$. In Section 3.2.1, we present a deficiency-optimal algorithm with time complexity $O(k|E(G)|\log|V(G)|)$ for the preemptive model and in Section 3.2.2 we give a deficiency-optimal exponential time algorithm and an algorithm with time complexity $O(k|E(G)|\log|V(G)|)$ and deficiency $O(k\log^2 k)$ for the non-preemptive model.

1.4 Related Work

The problem of optimizing queries with expensive predicates has received some attention in the database community [Avnur and Hellerstein 2000; Bouganim et al. 2001; Chaudhuri and Shim 1993; Hellerstein 1998; Laber et al. 2002; Porto 2001]. However, most of the proposed approaches [Bouganim et al. 2001; Chaudhuri and Shim 1993; Hellerstein 1998] do not take into account the fact that an attribute value may appear in different tuples. In other words, these approaches only address those instances (G,δ,γ) of DkO where G is a matching. Indeed, the *Predicate Migration* algorithm proposed in [Hellerstein 1998], the main reference in this subject, may be viewed as a deficiency-optimal algorithm for the variant of DkO, in which the input hypergraph G is a matching, the probability of a vertex in $V_i(G)$ being true is known in advance, and the goal is to compute an evaluation of minimum expected cost.

The non-preemptive model is studied in [Laber et al. 2002] and [Bouganim et al. 2001] study the dD2O in the non-preemptive model. Laber *et al.* [Laber et al. 2002] prove a lower bound of 2 for the deficiency of deterministic algorithms and 1.5 for the deficiency of randomized algorithms. The best known upper bound for the deficiency of polynomial-time deterministic algorithms is 3 (a folklore result) whereas it is 2.67 for randomized algorithms [Laber et al. 2002].

The idea of processing the hypergraph induced by the input relation of the database appears first in [Porto 2001], but no theoretical analysis is made.

The non-preemptive version of dD2O is studied in [Laber et al. 2002], where the following results are presented

—a lower bound of 3/2 on the deficiency of any (not only polynomial) randomized algorithm;

—a randomized polynomial time algorithm with deficiency 8/3;

—a linear time algorithm of deficiency 2 for the particular case of dD2O in which all the vertices have the same cost.

In the present work, we restrict our attention to *conjunctive* queries (in the sense of (1)). However, much more general queries could be considered. For example, $\bar{\delta}\colon E(G) \to \{0,1\}$ could be any formula in the first order propositional calculus involving the predicates represented by δ. In [Charikar et al. 2002], the problem of querying priced information is considered. In particular, they considered the problem of evaluating a query that can be represented by an "AND/OR tree" over a set of variables, where the cost of evaluating each variable may be different. This problem is further studied in [Laber 2004] and [Ferdinando Cicalese and Eduardo Laber 2005]. The framework for querying priced information proposed in that paper can be viewed as a restricted version of the problem described in this paragraph, where the input hypergraph has one single edge.

We finally observe that for different classes of k-partite hypergraphs, including trees and graphs with perfect flows, better bounds have been obtained [Feder et al. 2003].

2. THE SEQUENTIAL CASE

In this section we study sDkO, the sequential case of DkO. We start with lower bounds on the deficiency of deterministic and randomized algorithms for sDkO and then introduce a deficiency-optimal deterministic algorithm for sDkO with time complexity $O(k|E(G)|\log|V(G)|)$.

Next we focus on the study of the sequential case of the *Dynamic* 2*-partite Ordering Problem* (sD2O) and introduce a randomized algorithm for sD2O with time complexity $O(|V(G)|^3)$ whose expected deficiency is at most $2-\varepsilon$ for any given $0 \leq \varepsilon \leq 1-\sqrt{2}/2$. We then prove that this algorithm's deficiency is always bounded by $1+1/(1-\varepsilon)$. Finally, we show that that for any $0 \leq \varepsilon \leq 1$, no randomized algorithm can have deficiency smaller than $1+\varepsilon$ with probability larger than $(1+\varepsilon)/2$.

Recall that, since this section deals only with the sequential case of DkO, the cost of an evaluation E is defined as $\gamma(\mathsf{E}) = \sum_{v\in\mathsf{E}} \gamma(v)$.

2.1 Lower Bounds

We start with some lower bounds for the deficiency of algorithms for sDkO. As noted before, these bounds apply to exponential time/space algorithms as well.

THEOREM 2.1. *Let an integer $k > 0$ be given. Given a deterministic algorithm $\mathcal{A}$ for* sDkO *and a hypergraph G with at least one edge and $r(G) = k$, there exist functions γ and δ such that, for $\mathcal{I} = (G, \delta, \gamma)$, we have $d(\mathcal{A}(\mathcal{I}), \mathcal{I}) = k$.*

PROOF. Let $\mathcal{A}$ and G be as above and let e be an edge of G. Let γ and δ' be given by

$$\gamma(v) = \delta'(v) = \begin{cases} 1, & \text{if } v \in e; \\ 0, & \text{otherwise.} \end{cases}$$

Consider the execution of $\mathcal{A}(G, \delta', \gamma)$. Let u be the last vertex of e evaluated in this execution and let

$$\delta(v) = \begin{cases} 0, & \text{if } v = u; \\ \delta'(v), & \text{otherwise.} \end{cases}$$

It is not difficult to see that $\gamma(\mathcal{A}(G, \delta, \gamma)) = r(G) = k$ and that $\mathsf{E}^* = (V(G) - e) \cup \{u\}$ is an evaluation of cost 1 of $\mathcal{I} = (G, \delta, \gamma)$. Therefore $d(\mathcal{A}(\mathcal{I}), \mathcal{I}) = \gamma(\mathcal{A}(\mathcal{I}))/\gamma(\mathsf{E}^*) = k$. $\square$

To prove the corresponding result for randomized algorithms, we will apply Yao's minimax principle, in the following form.

THEOREM 2.2 [YAO 1977]. *Let a probability distribution π on the set of instances $\mathcal{I}$ of* sDkO *be given and let $\mathcal{A}$ be an optimal deterministic algorithm for* sDkO *under this distribution. The value of $\mathbb{E}_\pi[\gamma(\mathcal{A}(\mathcal{I}))]$ is a lower bound on the value of $\mathbb{E}[\gamma(\mathcal{R}(I))]$ for any randomized algorithm $\mathcal{R}$ for* sDkO.

THEOREM 2.3. *Let $k > 0$. Given a randomized algorithm $\mathcal{A}$ for* sDkO *and a hypergraph G with $r(G) = k$, there exist functions γ and δ such that, for $\mathcal{I} = (G, \delta, \gamma)$ we have $d(\mathcal{A}(\mathcal{I}), \mathcal{I}) = (k+1)/2$.*

PROOF. Consider the probability space of all instances (G, δ, γ) of sDkO where the only instances with positive probability are those with one single edge, having exactly one false vertex, and with $\gamma(v) = 1$ for all $v \in V(G)$. Let each of these instances have the same probability, namely, $1/k$. It is easy to see that 1 is the cost of an optimal evaluation for any instance of positive probability in this distribution.

Let $\mathcal{A}$ be a deficiency-optimal deterministic algorithm for this distribution of instances of sDkO and let $\mathcal{I}$ be an instance of positive probability in this distribution. Note that the value of $\gamma(\mathcal{A}(\mathcal{I}))$ ranges through all the possible values $1, \ldots, k$, as $\mathcal{I}$ ranges through its k possible values, and that $\mathcal{A}$ always evaluates the vertices of the hypergraph in the same order until it evaluates the false vertex, at which point it stops.

As there are exactly k instances of positive probability, all of them equiprobable, the expected value of $\gamma(\mathcal{A}(\mathcal{I}))$ for a given deterministic algorithm $\mathcal{A}$ is then

$$\mathbb{E}[\gamma(\mathcal{A}(\mathcal{I}))] = \sum_{i=1}^{k} \frac{1}{k} i = \frac{k+1}{2}.$$

We can conclude that any deficiency-optimal deterministic algorithm $\mathcal{A}$ for this distribution of instances of sDkO has its expected deficiency bounded by $(k+1)/2$.

Theorem 2.2 together with this remark lead to the conclusion that any randomized algorithm for sDkO has its deficiency bounded below by $(k+1)/2$. $\square$

The above results can be summarized as follows.

COROLLARY 2.4. *No deterministic algorithm for* sDkO *can achieve deficiency smaller than k and no randomized algorithm for* sDkO *can achieve deficiency smaller than $(k+1)/2$.*

2.2 A Deficiency-optimal Polynomial Deterministic Algorithm

In this section we introduce a deficiency-optimal polynomial time deterministic algorithm for sDkO, that is, one which has deficiency k. The algorithm is based on the primal-dual approximation method [Vazirani 2001].

The algorithm works as follows. Given an instance (G, δ, γ) of DkO, the algorithm keeps track of a 'dual variable' $\psi(e)$, initially set to 0, for each $e \in E(G)$ and a 'primal variable' $r(v) = \sum_{v \in e} \psi(e)$, for each $v \in V(G)$. Let us define the *gap of vertex v* as the difference

$$\Psi(v) = \gamma(v) - r(v). \tag{2}$$

The main loop of the algorithm scans $E(G)$. For each examined edge e, if the value of $\bar{\delta}(e)$ cannot be determined (because some of its vertices have not yet been evaluated), the algorithm "evaluates" this edge (Algoritm EVAL): the vertices of e which have not been evaluated up to this point, are evaluated in increasing order of their gaps. This "evaluation of e" finishes when either one of its vertices is evaluated and is found to be false or when the last vertex of e is evaluated and is found to be true.

Algorithm $\mathcal{P}(G, \delta, \gamma)$

for each $e \in E(G)$, $\psi(e) \leftarrow 0$
for each $v \in V(G)$, $r(v) \leftarrow 0$
for each $e \in E(G)$
  if $\bar{\delta}(e)$ is undefined
    EVAL(e)

Algorithm EVAL(e)

while $\bar{\delta}(e)$ is undefined
(1) $v \leftarrow$ a vertex of minimum gap in e which has not yet been evaluated
(2) $g \leftarrow \gamma(v) - r(v)$
(3) evaluate v
(4) $\psi(e) \leftarrow \psi(e) + g$
(5) for each $u \in e$
    $r(u) \leftarrow r(u) + g$

THEOREM 2.5. *Algorithm $\mathcal{P}$ is a deficiency-optimal algorithm for* $\mathsf{sD}k\mathsf{O}$ *with time complexity* $O(k|E(G)| \log |V(G)|)$

PROOF. Let $\mathcal{I} = (G, \delta, \gamma)$ be an instance of $\mathsf{sD}k\mathsf{O}$, let E^* be an optimal evaluation for $\mathcal{I}$ and let

$$J = \{e \in E(G)\colon e \text{ is passed to EVAL}\}.$$

As E^* is a cover for G, we have $\mathsf{E}^* \cap e \neq \emptyset$ for every $e \in J$. For each $e \in J$, let v' be the vertex of $\mathsf{E}^* \cap e$ with maximum gap right before the execution of EVAL(e). Since $\mathsf{E}^* \cap e$ is an evaluation for e, we must have that either v' is the last vertex processed by the loop in EVAL or that v' is not processed at all. In either case, we have

$$\Psi(v') = \max\{\Psi(u)\colon u \in e\}$$

right after the execution of EVAL and we can conclude that for every $e \in J$,

$$\sum_{v \in \mathsf{E}^* \cap e} \Psi(v) \geq \max\{\Psi(v)\colon v \in e\}.$$

Furthermore, upon the termination of $\mathcal{P}$, we have $\gamma(v) \geq \sum_{e \in J} \psi(e)$, and then,

$$\gamma(\mathsf{E}^*) = \sum_{v \in \mathsf{E}^*} \gamma(v) \geq \sum_{e \in J} \sum_{v \in \mathsf{E}^* \cap e} \Psi(v) \geq \sum_{e \in J} \max\{\Psi(v)\colon v \in e\}. \tag{3}$$

If v is indeed evaluated then $\gamma(v) = \sum_{e \in J} \psi(e)$, and it follows that

$$\gamma(\mathcal{P}(\mathcal{I})) = \sum_{v \in \mathcal{P}(\mathcal{I})} \sum_{e \in J} \psi(e) = \sum_{e \in J} \sum_{v \in \mathcal{P}(\mathcal{I})} \Psi(v) \leq \sum_{e \in J} |e| \max \{\Psi(v) \colon v \in e\} . \quad (4)$$

From (3) and (4), follows

$$d(\mathcal{P}(\mathcal{I}), \mathcal{I}) = \frac{\gamma(\mathcal{P}(\mathcal{I}))}{\gamma(\mathsf{E}^*)} \leq |e| = k,$$

therefore, $d(\mathcal{P}) = k$.

We now approach the time complexity of algorithm $\mathcal{P}$. The initialization step requires time $O(|V(G)| + |E(G)|)$. In the next step, Algorithm EVAL is invoked at most $|E(G)|$ times, as each edge e is read from the input.

At each invocation, EVAL(e) builds a priority queue for the non-evaluated vertices in e and then extracts these vertices one by one from the queue. This takes $O(k \log k)$ time in the worst case. Finally, the last step in Algorithm EVAL can be executed in time $O(k \log |V(G)|)$, if a balanced search tree for $V(G)$ is mantained throughout the execution.

Therefore, the overall time complexity of algoritm $\mathcal{P}$ is

$$\begin{aligned} &O(\max \{|V(G)| + |E(G)|, |E(G)|(k \log k + k \log |V(G)|\}) \\ &\qquad = O(k|E(G)| \max \{|V(G)|/(k|E(G)|), \log k|V(G)|\}) \\ &\qquad\qquad = O(k|E(G)| \log |V(G)|), \end{aligned}$$

as required. □

2.3 The Bipartite Case and a Randomized Algorithm

In this section we restrict our attention to sD2O, the restricted case of sDkO in which G is a bipartite graph.

Let $\mathcal{I} = (G, \delta, \gamma)$ be an instance of sD2O, and let C be a cover for G. We define the *C-evaluation for $\mathcal{I}$* as the set $\mathsf{E}(C) = C \cup \Gamma_1(C)$. It is not difficult to see that the C-evaluation for $\mathcal{I}$ is indeed an evaluation for $\mathcal{I}$. We call an algorithm for sD2O that always outputs $\mathsf{E}(C)$ for some cover C a *cover-oriented algorithm for* sD2O.

As any evaluation for (G, δ) must contain some cover for G and also must contain $\Gamma_1(V(G))$, it is not difficult to conclude that a C-evaluation for an instance of sD2O has deficiency at most 2, whenever C is a minimum cover for (G, γ). This observation appears in [Laber et al. 2002] with respect to dD2O. We call a cover-oriented algorithm for sD2O that always outputs $\mathsf{E}(C)$ for some minimum cover C, a *minimum-cover-oriented algorithm for* sD2O. Since 2 is a lower bound for the deficiency of any deterministic algorithm for sD2O (see Section 2.2), we have that any deterministic cover-oriented algorithm for sD2O is deficiency-optimal.

Moreover, when G is a bipartite graph, a minimum cover C for (G, γ) may be computed in time $O(|V(G)|^3)$, by reducing it problem to of determining a minimum cut (maximum flow) on a directed graph with $|V(G)|+2$ vertices and $|E(G)|+|V(G)|$ arcs (see [Cook et al. 1997]). The evaluation $\mathsf{E}(C)$ can then be computed in time $O(|V(G)|)$. Therefore, it is possible to devise a minimum-cover-oriented algorithm for sD2O with time complexity $O(|V(G)|^3)$.

Let $0 \leq \varepsilon \leq 1-\sqrt{2}/2$. In this section, we introduce $\mathcal{R}_\varepsilon$, a polynomial time randomized algorithm for sD2O whose deficiency satisfies, for every instance $\mathcal{I}$,

$$\mathbb{E}\left[d(\mathcal{R}_\varepsilon(\mathcal{I}),\mathcal{I})\right] \leq 2-\varepsilon \tag{5}$$

and

$$d(\mathcal{R}_\varepsilon(\mathcal{I}),\mathcal{I}) \leq 1+\frac{1}{1-\varepsilon}. \tag{6}$$

Algorithm $\mathcal{R}_\varepsilon$ provides a trade-off between expected deficiency and worst case deficiency. At one extreme, where $\varepsilon = 1-\sqrt{2}/2$, we have expected deficiency $1.707\ldots$ and worst case deficiency of 2.41. At the other extreme, where $\varepsilon = 0$, $\mathcal{R}_\varepsilon$ becomes actually a deterministic algorithm with deficiency 2.

The key idea in the design of $\mathcal{R}_\varepsilon$ comes from trying to understand under which conditions a cover-oriented algorithm shows a bad performance. More exactly, given an instance $\mathcal{I}$ of sD2O, a minimum cover C for (G,δ), and $\varepsilon > 0$, we turn our attention to the instances $\mathcal{I}$ having $d(\mathsf{E}(C),\mathcal{I}) \geq 2-\varepsilon$.

As suggested by the proofs of Theorem 2.1 and Theorem 2.3, one family of such instances can be constructed as follows. Consider an instance $\mathcal{I} = (G,\delta,\gamma)$ of sD2O where G is a matching of n edges, $\delta(v) = 1$ for every $v \in V_1(G)$, $\delta(v) = 0$ for every $v \in V_2(G)$ and $\gamma(v) = 1$ for every $v \in V(G)$. Clearly, $V_2(G)$ is an optimum evaluation for $\mathcal{I}$, with cost n. On the other hand, note that the deficiency of a deterministic cover-oriented algorithm for sD2O depends on which of the 2^n minimum covers of G is chosen. In the particular case in which $C = V_1(G)$ is chosen, we have $d(\mathsf{E}(C),\mathcal{I}) = 2n/n = 2$.

This example suggests the following idea. If C is a minimum cover for (G,γ) and nonetheless $\mathsf{E}(C)$ is not a "good evaluation" for $\mathcal{I} = (G,\delta,\gamma)$, then there must be another cover C' of G whose intersection with C is "small" and still C' is not "far from being" a minimum cover for G. The following lemma captures this idea formally.

Lemma 2.6. *Let $\mathcal{I} = (G,\delta,\gamma)$ be an instance of* sD2O*, let $T \subseteq \Gamma_1(V(G))$, and let C_T be a minimum cover of $G-T$. Then $\gamma(C_T)+\gamma(T) \leq \gamma(\mathsf{E}^*_{\mathcal{I}})$.*

Proof. From $T \subseteq \Gamma_1(V(G))$ it follows that $T \subset \mathsf{E}^*$. Furthermore, E^*-T is a cover for $G-T$, for otherwise $\overline{\delta}(uv)$ cannot be determined for some $uv \in E(G-T)$. Since C_T is a minimum cover for $G-T$, it follows that $\gamma(C_T) \leq \gamma(\mathsf{E}^*-T)$. Therefore, $\gamma(C_T)+\gamma(T) \leq \gamma(\mathsf{E}^*_{\mathcal{I}}-T)+\gamma(T) = \gamma(\mathsf{E}^*_{\mathcal{I}})$. □

Lemma 2.7. *Let $\mathcal{I} = (G,\delta,\gamma)$ be an instance of* sD2O*, let C be a minimum cover for (G,γ) and let $0 < \varepsilon < 1$. If $d(\mathsf{E}(C),\mathcal{I}) \geq 2-\varepsilon$, then there is a vertex cover C_ε for G such that*

$$\gamma(C_\varepsilon) \leq \frac{\gamma(C-C_\varepsilon)}{1-\varepsilon}. \tag{7}$$

Proof. Let $T = \Gamma_1(C)-C$. Since $d(\mathsf{E}(C),\mathcal{I}) \geq 2-\varepsilon$, it follows that $2-\varepsilon \leq (\gamma(C)+\gamma(T))/\gamma(\mathsf{E}^*_{\mathcal{I}})$.

Let C_T be a minimum cover for $G-T$. Since $T \subseteq \Gamma_1(V(G))$, it follows from Lemma 2.6 that $\gamma(C_T)+\gamma(T) \leq \gamma(\mathsf{E}^*_{\mathcal{I}})$.

Simple calculations give $\gamma(T) \leq \left(\gamma(C)-(2-\varepsilon)\gamma(C_T)\right)/(1-\varepsilon)$.

Take $C_\varepsilon = C_T \cup T$ and note that C_ε is a cover for G with $(C_\varepsilon \cap C) \subset C_T$. Then,

$$\gamma(C_\varepsilon) \le \gamma(C_T) + \gamma(T) \le \frac{\gamma(C) - \gamma(C_T)}{1-\varepsilon} \le \frac{\gamma(C) - \gamma(C_\varepsilon \cap C)}{1-\varepsilon} = \frac{\gamma(C - C_\varepsilon)}{1-\varepsilon},$$

as required. $\square$

Let $\mathcal{I} = (G, \delta, \gamma)$ and let C and ε be as in the statement of Lemma 2.7. Let C' be a minimum cover for $(G, \gamma_{C,\varepsilon})$, where $\gamma_{C,\varepsilon}$ is given by

$$\gamma_{C,\varepsilon}(v) = \begin{cases} (1-\varepsilon)\gamma(v), & \text{if } v \notin C; \\ (2-\varepsilon)\gamma(v), & \text{otherwise.} \end{cases}$$

Formulating the problem of finding a cover C_ε satisfying (7) as a linear system we get to

$$\begin{aligned} \sum_{v \in V(G)} \gamma(v) x_v \ &\le \ \frac{\sum_{v \in C} \gamma(v)(1 - x_v)}{1-\varepsilon} \\ x_u + x_v \ &\ge \ 1, \ \text{for every } uv \in E(G), \end{aligned}$$

which has a solution in $\{0,1\}^{|V(G)|}$ if and only if such a cover C_ε exists. However, through simple algebraic manipulations we realize that the previous system is equivalent to

$$\begin{aligned} \sum_{v \in C} (2-\varepsilon) x_v + \sum_{v \in V(G) - C} (1-\varepsilon)\gamma(v) x_v \ &\le \ \gamma(C) \\ x_u + x_v \ &\ge \ 1, \ \text{for every } uv \in E(G), \end{aligned}$$

and this linear system has a solution in $\{0,1\}^{|V(G)|}$ if and only if $\gamma_{C,\varepsilon}(C') \le \gamma(C)$. Furthermore if $\gamma_{C,\varepsilon}(C') \le \gamma(C)$ then

$$\gamma(C') \le \frac{\gamma(C - C')}{1-\varepsilon}. \tag{8}$$

This last remark, together with Lemma 2.7, provides an efficient way to verify whether or not $\mathsf{E}(C)$ is a good evaluation for (G, δ, γ), given a minimum cover C for (G, γ).

As a cover C' as above can be computed in polynomial time, we can devise a polynomial time randomized algorithm for sD2O which works as follows.

At the first step, the algorithm determines a minimum cover C for (G, γ) and a minimum cover C' for $(G, \gamma_{C,\varepsilon})$. If $\gamma_{C,\varepsilon}(C') > \gamma(C)$, then Lemma 2.7 assures that $\mathsf{E}(C)$ will be a good evaluation for (G, δ, γ) and the algorithm returns $\mathsf{E}(C)$. Otherwise, the algorithm returns $\mathsf{E}(C)$ with probability $p = p(\varepsilon)$ or $\mathsf{E}(C')$ with probability $1 - p$ as the solution.

Algorithm $\mathcal{R}_\varepsilon(G, \delta, \gamma)$

(1) $C \leftarrow$ a minimum cover for (G, γ)
(2) $C' \leftarrow$ a minimum cover for $(G, \gamma_{C,\varepsilon})$
(3) If $\gamma_{C,\varepsilon}(C') > \gamma(C)$, return $\mathsf{E}(C)$
(4) $p \leftarrow \frac{1-3\varepsilon+\varepsilon^2}{1-2\varepsilon}$
(5) $x \leftarrow$ a random number uniformly chosen from $[0, 1]$.
(6) if $x < p$, return $\mathsf{E}(C)$, otherwise return $\mathsf{E}(C')$

As C is a minimum cover for (G, γ), we have that $\gamma(C) \leq \gamma(C')$. It sounds reasonable to select C with probability higher than $1/2$, that is $p \geq 0.5$. Imposing this condition on the value of p leads to $0 \leq \varepsilon \leq 1 - \sqrt{2}/2$.

2.3.1 *Algorithm Analysis.* The correctness of Algorithm $\mathcal{R}_\varepsilon$ follows from the fact that $\mathcal{R}_\varepsilon$ is a cover-oriented algorithm. Besides, the time needed for Algorithm $\mathcal{R}_\varepsilon$ is clearly asymptotically the same as the time needed for computing two minimum-weight covers on a bipartite graph. Therefore, the time complexity of $\mathcal{R}_\varepsilon$ is $O(|V(G)|^3)$. Properties (5) and (6) of the evaluation computed by $\mathcal{R}_\varepsilon$, claimed at the beginning of Section 2.3, are proven in this section.

Let $\mathcal{I} = (G, \delta, \gamma)$, C, C' and ε be as in the statement of algorithm $\mathcal{R}_\varepsilon$. It will be convenient to define (see Figure 2 for a schematic representation

$$\begin{aligned} H &= \Gamma_1(C \cap C') - (C \cup C'), \\ H_C &= \Gamma_1(C') \cap (C - C'), \\ H_{C'} &= \Gamma_1(C) \cap (C' - C). \end{aligned}$$

The next result shows that any edge having an endpoint outside $C \cup C'$ must have its other endpoint in $C \cap C'$.

LEMMA 2.8. *If C and C' are two covers for G, then $\Gamma(V(G) - (C \cup C')) \subseteq C \cap C'$.*

PROOF. Let $v \in V(G) - (C \cup C')$ and $u \in \Gamma(v)$. We must have $u \in C$, otherwise C would not cover uv. The same argument allows us to conclude $u \in C'$ and hence the result. □

With Lemma 2.8 in mind, we can write

$$\begin{aligned} H \cup H_{C'} &= \Gamma_1(C) - C = \Gamma_1(C) - (C \cap \Gamma_1(C)), \\ H \cup H_C &= \Gamma_1(C') - C' = \Gamma_1(C') - (C' \cap \Gamma_1(C')), \end{aligned}$$

and, as H, $H_C, H_{C'} \subseteq \Gamma_1(V(G))$ are disjoint sets, we have

$$\gamma(\mathsf{E}(C)) = \gamma(C) + \gamma(H_{C'}) + \gamma(H), \tag{9}$$

$$\gamma(\mathsf{E}(C')) = \gamma(C') + \gamma(H_C) + \gamma(H), \tag{10}$$

$$\gamma(\mathsf{E}^*_{\mathcal{I}}) \geq \gamma(H) + \gamma(H_C) + \gamma(H_{C'}). \tag{11}$$

THEOREM 2.9. *Let $0 \leq \varepsilon \leq 1 - \sqrt{2}/2$. For any instance $\mathcal{I} = (G, \delta, \gamma)$ of* sD2O *we have $\mathbb{E}\left[d(\mathcal{R}_\varepsilon(\mathcal{I}), \mathcal{I})\right] \leq 2 - \varepsilon$.*

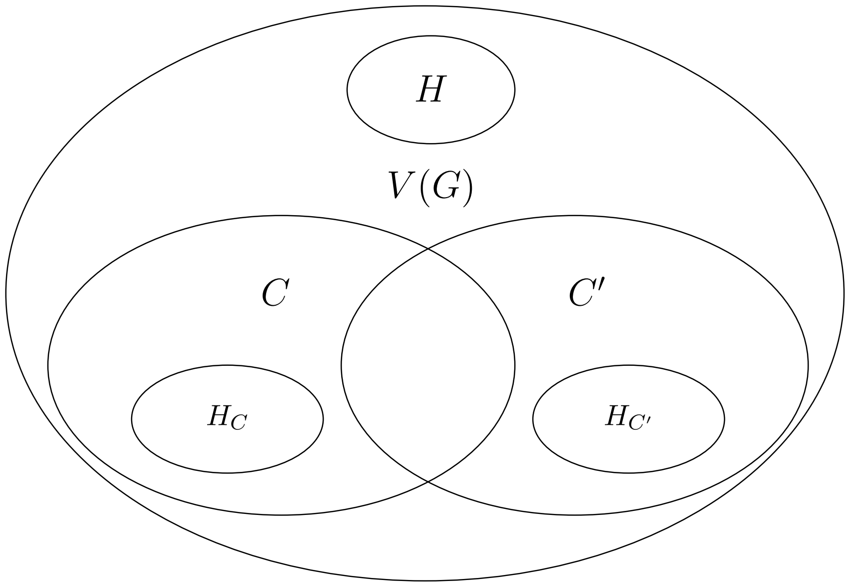

Fig. 2. Schematic view of the sets $V(G)$, C, C', H_C, $H_{C'}$ and H

PROOF. Let $\mathcal{I} = (G, \delta, \gamma)$ be an instance of sD2O and consider the execution of $\mathcal{R}_\varepsilon(\mathcal{I})$. If $\gamma_{C,\varepsilon}(C') > \gamma(C)$ at Step 4 of Algorithm $\mathcal{R}_\varepsilon$, it follows from Lemma 2.7 that $\mathbb{E}\left[d(\mathcal{R}(\mathcal{I}), \mathcal{I})\right] \leq 2 - \varepsilon$.

For the case in which $\gamma_{C,\varepsilon}(C') \leq \gamma(C)$, we have that

$$\mathbb{E}\left[d(\mathcal{R}_\varepsilon(\mathcal{I}), \mathcal{I})\right] = \mathbb{E}\left[\gamma(\mathcal{R}_\varepsilon(\mathcal{I}))\right] / \gamma(\mathsf{E}^*_{\mathcal{I}}).$$

From (9) and (10) we have

$$\begin{aligned} \mathbb{E}\left[\gamma(\mathcal{R}_\varepsilon(\mathcal{I}))\right] &\leq p\gamma(\mathsf{E}(C)) + (1-p)\gamma(\mathsf{E}(C')) \\ &= p\gamma(C) + (1-p)\gamma(C') + (1-p)\gamma(H_C) + p\gamma(H_{C'}) + \gamma(H). \end{aligned} \quad (12)$$

Let C_H be a minimum cover for $(G - H, \gamma)$. As every edge in $G - H$ is covered by C_H and every edge incident to H is covered by a vertex in $C \cap C'$, we can apply Lemma 2.8 to $V(H)$ and conclude that $C_H \cup (C \cap C')$ is a cover for G,

As C is a minimum cover for G, we have that

$$\gamma(C) \leq \gamma(C_H \cup (C \cap C')) \leq \gamma(C_H) + \gamma(C \cap C'),$$

or, equivalently,

$$\gamma(C_H) \geq \gamma(C) - \gamma(C \cap C') \geq \gamma(C - C').$$

As $H \subseteq \Gamma_1(V(G))$, it follows from Lemma 2.6 that

$$\gamma(\mathsf{E}^*_{\mathcal{I}}) \geq \gamma(H) + \gamma(C - C'), \quad (13)$$

so that, from the fact that C is a minimum cover and inequalities (11) and (13), we have,

$$\gamma(\mathsf{E}^*_{\mathcal{I}}) \geq \max\{\gamma(C), \gamma(H) + \gamma(H_C) + \gamma(H_{C'}), \gamma(H) + \gamma(C - C')\} \quad (14)$$

As the choice of ε implies $p \geq 0.5$, it follows from (12) and (14) that

$$\mathbb{E}\left[d(\mathcal{R}_\varepsilon(\mathcal{I}),\mathcal{I})\right] \leq 2p + (1-p) + \max\left\{\frac{\gamma(H)}{\gamma(H)}, \frac{\gamma(C')}{\gamma(C-C')}\right\}$$
$$\leq 2p + (1-p)\frac{\gamma(C') + \gamma(H)}{\gamma(H) + \gamma(C-C')}.$$

For $p = (1 - 3\varepsilon + \varepsilon^2)/(1 - 2\varepsilon)$ and recalling inequality (8) we get $\mathbb{E}\left[d(\mathcal{R}_\varepsilon(\mathcal{I}),\mathcal{I})\right] \leq 2 - \varepsilon$. □

THEOREM 2.10. *Let $0 \leq \varepsilon \leq 1 - \sqrt{2}/2$. For any instance $\mathcal{I} = (G, \delta, \gamma)$ of* sD2O *we have $d(\mathcal{R}_\varepsilon(\mathcal{I}),\mathcal{I}) \leq 1 + 1/(1-\varepsilon)$.*

PROOF. Let C and C' be as in Algorithm $\mathcal{R}_\varepsilon$. If $\mathcal{R}_\varepsilon(\mathcal{I}) = \mathsf{E}(C)$, then

$$d(\mathcal{R}_\varepsilon(\mathcal{I}),\mathcal{I}) \leq 2 \leq 1 + \frac{1}{1-\varepsilon},$$

because C is a minimum cost cover. On the other hand, if $\mathcal{R}_\varepsilon(\mathcal{I}) = \mathsf{E}(C')$, then

$$d(\mathcal{R}_\varepsilon(\mathcal{I}),\mathcal{I}) \leq \frac{\gamma(C') + \gamma(H_C) + \gamma(H)}{\max\{\gamma(C), \gamma(H_C) + \gamma(H)\}},$$

and then, recalling inequality (8) we have

$$d(\mathcal{R}_\varepsilon(\mathcal{I}),\mathcal{I}) \leq \frac{2-\varepsilon}{1-\varepsilon} = 1 + \frac{1}{1-\varepsilon},$$

as required. □

To see that the bound given by inequality (5) is tight when $\varepsilon = 1 - \sqrt{2}/2$, let $\alpha \approx \sqrt{2}$ and consider the instance $\mathcal{I}_\alpha = (G, \delta, \gamma)$, where G is a complete bipartite graph with $|V_2(G)| = \alpha|V_1(G)|$, $\delta(v) = 0$ for every $v \in V_1(G)$, $\delta(v) = 1$ for every $v \in V_2(G)$, and $\gamma(v) = 1$ for every $v \in V(G)$. Clearly, $V_1(G)$ is a minimum cover for (G, γ). The set $V_2(G)$, however, is a minimum cover for $(G, \gamma_{C,\varepsilon})$ and $\gamma_{C,\varepsilon}(V_2(G)) \leq \gamma(V_1(G))$. Hence, $\mathcal{R}_\varepsilon(\mathcal{I})$ returns $\mathsf{E}(V_1(G)) = V_1(G)$ with probability 1/2 and $\mathsf{E}(V_2(G)) = V(G)$ with probability 1/2, so that the deficiency is $d(\mathcal{R}_\varepsilon(\mathcal{I}),\mathcal{I}) = 1 + \alpha/2$. By choosing the value of α conveniently close to $\sqrt{2}$, the deficiency of $\mathcal{R}_\varepsilon(\mathcal{I}_\alpha)$ can get as close to $1 + \sqrt{2}/2$ as desired.

2.3.2 *Lower Bound for Randomized Algorithms.* We have proved in the Section 2.3.1 that Algorithm $\mathcal{R}_\varepsilon$, with $\varepsilon = 1 - \sqrt{2}/2$, has its deficiency bounded from above by $1 + \sqrt{2}/2 = 1.707\ldots$. However, $\mathcal{R}_\varepsilon$ does not achieve this deficiency with high probability. For the family of instances $\mathcal{I}_\alpha$ described at the end of previous section, $\mathcal{R}_\varepsilon$ attains deficiency $1 + \alpha/2$ with probability 1/2 and deficiency 1 with probability 1/2. One can speculate whether a more dynamic algorithm would not have deficiency closer to the minimum possible value of 1.5 with higher probability. In this section, we prove (Theorem 2.12) that no randomized algorithm for sD2O can have deficiency smaller than μ for any given $1 \leq \mu \leq 2$ with probability close to 1.

We shall prove this considering instances $\mathcal{I} = (G, \delta, \gamma)$ in which G is a complete bipartite graph on n vertices and $|V_1(G)| = |V_2(G)|$ and with $\gamma(v) = 1$ for all $v \in V(G)$. All instances in this section are assumed to be of this form.

Let $\mathcal{R}$ be a randomized algorithm for sD2O and let $1/2 \leq \lambda \leq 1$. Given an instance $\mathcal{I} = (G, \delta, \gamma)$, let

$$\begin{aligned} P(\mathcal{R}, \mathcal{I}, \lambda n) &= \mathbb{P}\left(\gamma(\mathcal{R}(\mathcal{I})) \geq \lambda n\right), \\ P(\mathcal{R}, \lambda n) &= \max\left\{P(\mathcal{R}, \mathcal{I}, \lambda n) \colon \mathcal{I} \text{ is an instance of sD2O}\right\}. \end{aligned}$$

Given a deterministic algorithm $\mathcal{D}$ and an instance $\mathcal{I}$ of sD2O, we define the *payoff of $\mathcal{D}$ with respect to $\mathcal{I}$* as

$$g(\mathcal{D}, \mathcal{I}) = \begin{cases} 1, & \text{if } \gamma(\mathcal{D}(\mathcal{I})) \geq \lambda n; \\ 0, & \text{otherwise.} \end{cases}$$

One may deduce from Yao's minimax principle (see Theorem 2.2) that, for any randomized algorithm $\mathcal{R}$, we have

$$\max\left\{\mathbb{E}\left[g(\mathcal{R}, \mathcal{I})\right] : \mathcal{I} \text{ is an instance of sD2O}\right\} \geq \max\left\{\mathbb{E}\left[g(\mathcal{D}^*, \mathcal{I}_p)\right]\right\}. \tag{15}$$

where $\mathcal{D}^*$ is a deficiency-optimal deterministic algorithm, in the average case sense, for the probability distribution p over the set of possible instances for sD2O. (In the inequality above, the expectation is taken with respect to the coin flips of $\mathcal{R}$ on the left-hand side and with respect to p on the right-hand side; we write $\mathcal{I}_p$ for an instance generated according to p).

Since a randomized algorithm can be viewed as a distribution probability over the set of deterministic algorithms, we have $\mathbb{E}\left[g(\mathcal{R}, \mathcal{I})\right] = P(\mathcal{R}, \mathcal{I}, \lambda n)$ and hence,

$$\max\left\{\mathbb{E}\left[g(\mathcal{R}, \mathcal{I})\right] : \mathcal{I} \text{ is an instance of sD2O}\right\} = P(\mathcal{R}, \lambda n).$$

Moreover, $\mathbb{E}\left[g(\mathcal{D}^*, \mathcal{I}_p)\right]$ is the probability that the cost of the evaluation computed by a deficiency-optimal algorithm for the distribution p is at least λn, that is,

$$\mathbb{P}\left(\gamma(\mathcal{D}^*(\mathcal{I})) \geq \lambda n\right) = \mathbb{E}\left[g(\mathcal{D}^*, \mathcal{I}_p)\right].$$

Thus, if we are able to define a probability distribution p over the set of possible instances of sD2O and analyze a deficiency-optimal algorithm for such a distribution, we can obtain a lower bound for $P(\mathcal{A}, \lambda n)$.

Let n be an even positive integer and let G be a complete bipartite graph with $V(G) = \{1, \ldots, n\}$ with $V_1(G) = \{1, \ldots, n/2\}$ and $V_2(G) = \{n/2 + 1, \ldots, n\}$. Let $\gamma(v) = 1$ for all $v \in V(G)$ and for $1 \leq i \leq n$, let

$$\delta_i(v) = \begin{cases} 1, & \text{if } i = v; \\ 0, & \text{otherwise.} \end{cases}$$

Consider the probability distribution p where the only instances with positive probability are $\mathcal{I}_i = (G, \delta_i, \gamma)$, $1 \leq i \leq n$ and all these instances are equiprobable, with probability $1/n$ each. A key property of these instances is that the cost of the optimum evaluation for all of them is $n/2$, since all the vertices of the vertex class of the graph that does not contain the only true vertex must be evaluated. We have the following lemma.

LEMMA 2.11. *Let $\mathcal{D}^*$ be a deficiency-optimal algorithm for the distribution probability p. Then,*

$$\mathbb{E}\left[g(\mathcal{D}^*, \mathcal{I}_p)\right] \geq 1 - \lambda.$$

PROOF. First of all we observe that if $\mathcal{D}^*$ evaluates the true vertex v, then it must evaluate all vertices in the other vertex class before it finishes, for G is a complete bipartite graph.

Let j be the integer such that $\mathcal{D}^*$ evaluates the vertex j at the moment it completes the evaluation of *all* vertices from one vertex class of the graph, say $V_1(G)$, having obtained $\delta(v) = 0$ *for all vertices* $v \in V_1(G)$ *before* vertex j. (To determine j simply run $\mathcal{D}^*$ and reply $\delta(v) = 0$ for all vertices until it is about to evaluate all vertices from one vertex class.)

Let $v_1, v_2, \ldots, v_{n/2} = j$ be the vertices of $V_1(G)$ in the order they were evaluated by $\mathcal{D}^*$. Let $t = \lambda - 1/2$ and consider the instances $\mathcal{I}_{v_{\lceil tn \rceil}}, \mathcal{I}_{v_{\lceil tn \rceil + 1}}, \ldots, \mathcal{I}_{v_{n/2}}$. In all these instances, $\mathcal{D}^*$ evaluates at least $\lceil tn \rceil$ vertices in $V_1(G)$ and all vertices in $V_2(G)$. Thus, for at least $n/2 - \lceil tn \rceil + 1$ instances, $\mathcal{D}^*$ costs at least $n/2 + \lceil tn \rceil = \lceil \lambda n \rceil$. Hence,

$$\mathbb{E}\left[g(\mathcal{D}^*, \mathcal{I}_p)\right] \geq \frac{n/2 - \lceil tn \rceil + 1}{n} \geq 1 - \lambda,$$

as required. $\square$

For all $1 \leq j \leq n$, the cost of an optimal evaluation for $\mathcal{I}_j$ is $n/2$, and we have the following result.

THEOREM 2.12. *Let $\mathcal{R}$ be a randomized algorithm for* sD2O *and let* $1 \leq \mu \leq 2$ *be a real number. There is an instance $\mathcal{I}$ for which* $\mathbb{P}\left(d(\mathcal{R}(\mathcal{I}), \mathcal{I}) \geq \mu\right) \geq 1 - \mu/2$.

3. THE DISTRIBUTED CASE

In this section we study d$\mathrm{D}k$O the Dynamic k-partite Ordering problem, in the setting where k processors are available, each one dedicated to the processing of the vertices of each vertex class of the input hypergraph. In this model we assume that there is no communication cost whatsoever between the different processors.

In Section 3.1 we show that k is a lower bound on the deficiency of any (not only polinomial) deterministic algorithm for d$\mathrm{D}k$O. In Section 3.2 we introduce deterministic algorithms for d$\mathrm{D}k$O. Section 3.2.1 introduces a slight modification to Algorithm $\mathcal{P}$ (from Section 2.2) which results in a deficiency-optimal deterministic algorithm with time complexity $O(k|E(G)| \log |V(G)|)$ for d$\mathrm{D}k$O, under the preemptive model. In Section 3.2.2, we introduce a deficiency-optimal deterministic exponential time algorithm and a deterministic algorithm with time complexity $O(k|E(G)| \log |V(G)|)$ algorithm with deficiency $O(k \log^2 k)$, both under the non-preemptive model of d$\mathrm{D}k$O.

Recall that, as discussed at the end of Section 1.1, an evaluation E in the distributed case is a complex object which holds, besides the vertices to be evaluated, information defining the scheduling of the evaluation of these vertices in the corresponding processors. We will not need an analytic expression for the cost $\gamma(\mathsf{E})$ of an evaluation E. Throughout this section, $\gamma(\mathsf{E})$ represents the time elapsed between the start of evaluation by the first processor and the end of the evaluation by the last processor.

3.1 Lower Bounds

THEOREM 3.1. *Given a deterministic algorithm $\mathcal{A}$ for* d$\mathrm{D}k$O*, there is an instance $(G_\mathcal{A}, \gamma_\mathcal{A}, \delta_\mathcal{A})$ such that the execution of $\mathcal{A}(G_\mathcal{A}, \gamma_\mathcal{A}, \delta_\mathcal{A})$ evaluates all the ver-*

tices of some vertex class.

Proof. Let $\mathcal{A}$ be a deterministic algorithm for dDkO. Let $G_{\mathcal{A}}$ be a matching and let $\gamma_{\mathcal{A}}(v) = 1$ for every $v \in V(G_{\mathcal{A}})$.

Note that since $G_{\mathcal{A}}$ is a matching, each vertex of $G_{\mathcal{A}}$ can be uniquely described by a pair $(i,\ a) \in \{1,\ \ldots,\ k\} \times E(G_{\mathcal{A}})$, and let us define B as the complete bipartite graph where $V_1(B) = \{1, \ldots, k\}$ and $V_2(B) = E(G_{\mathcal{A}})$, so that there is a natural one-to-one correspondence between the edges of B and the vertices of $G_{\mathcal{A}}$.

Our aim is to prove that there is a function $\delta_{\mathcal{A}} \colon V(G_{\mathcal{A}}) \to \{0, 1\}$ such that the execution of $\mathcal{A}(G_{\mathcal{A}}, \gamma_{\mathcal{A}}, \delta_{\mathcal{A}})$ evaluates all the vertices in $V_i(G)$ for some $1 \leq i \leq k$. As $\delta_{\mathcal{A}}$ is uniquely described by the set $\delta_{\mathcal{A}}^{-1}(0) = \{v \in V(G_{\mathcal{A}}) \colon \delta(v) = 0\}$, we will think of $\delta_{\mathcal{A}}$ in terms of the set of edges of B corresponding to $\delta_{\mathcal{A}}^{-1}(0)$.

To describe a worst case scenario for $\mathcal{A}$ we define a two-player game in which the algorithm is one of the players and its adversary forces a "bad performance" of $\mathcal{A}$. For convenience, we describe our game in terms of the graph B defined above.

As noted above, any 0-1-function on $V(G_{\mathcal{A}})$ is uniquely described by a subset of $V(G_{\mathcal{A}})$, which, in turn, uniquely corresponds to a subset of $E(B)$. The idea of our game will be that the adversary chooses a set $F \subseteq E(B)$ and the algorithm has to discover the set F by picking edges of B, one by one, and asking the adversary at each move whether that edge is in F or not.

Given a complete bipartite graph, and an integer $0 < t \leq |V_2(B)|$, let us describe the game $\mathcal{G}(B, t)$ between two players, the hider and the seeker. In this game, B and t are given to both players and the hider chooses $F \subseteq E(B)$ (which she keeps hidden from the seeker) such that

$$\deg_{B[F]}(v) = \begin{cases} t, & \text{if } v \in V_1(B); \\ 1, & \text{if } v \in V_2(B). \end{cases}$$

A *move of the game* is a pair $(e, a) \in E(B) \times \{\mathsf{yes}, \mathsf{no}\}$, which is interpreted as the seeker querying "*does $e \in F$?*" and the hider answering yes or no, accordingly. The game finishes when the seeker has determined F. A *realization of the game* is a sequence of moves of the game from beginning to end.

The goal of the seeker is to determine F. However, note that since F does not have to be disclosed until the very end, the hider does not have to make up her mind about which F to pick at the beginning; she may answer the queries as the game evolves, just making sure that her answers are consistent with *some* choice of F.

The following theorem is stated without proof for space considerations.

Theorem 3.2. *Let B, t and $\mathcal{G}$ be as above. There is a strategy for the* hider *in the game $\mathcal{G}(B, t)$ which forces the* seeker *to query every edge adjacent to some vertex $v \in V_1(B)$ in any realization of game $\mathcal{G}(B, t)$.*

Proof. For the proof, we refer the reader to [Carmo et al. 2004]. □

Translated back into the setting of the original problem, Theorem 3.2 states that for every deterministic algorithm $\mathcal{A}$ for dDkO, there is a function $\delta_{\mathcal{A}} \colon V(G_{\mathcal{A}}) \to \{0, 1\}$ which forces $\mathcal{A}(G_{\mathcal{A}}, \gamma_{\mathcal{A}}, \delta_{\mathcal{A}})$ to evaluate $\delta_{\mathcal{A}}(v)$ for every $v \in V_i(G)$ for some $1 \leq i \leq k$.

Corollary 3.3. *Every deterministic algorithm for* dDkO*, whether in the preemptive or in the non-preemptive model, has deficiency at least* k*.*

Proof. Let $\mathcal{A}$ be a deterministic algorithm for dDkO. Let $t > 0$ be given and let $\mathcal{I}_{\mathcal{A}} = (G_{\mathcal{A}}, \gamma_{\mathcal{A}}, \delta_{\mathcal{A}})$ be an instance of dDkO, as in the proof of Theorem 3.1, with $|E(G_{\mathcal{A}})| = tk$ (t will play the same rôle as in the game $\mathcal{G}(B, t)$ defined above). Note that $G_{\mathcal{A}}$ has tk^2 vertices, with exactly tk of them in each vertex class. Besides, each vertex class has exactly t false vertices.

Let E be the evaluation of $\mathcal{I}_{\mathcal{A}}$ in which only the false vertices are evaluated, all of them in parallel with no idle processing. As each vertex class of $G_{\mathcal{A}}$ has exactly t false vertices, we have that $\gamma(\mathsf{E}) = t$. However, according to Theorem 3.1, when given $\mathcal{I}_{\mathcal{A}}$ as input, algorithm $\mathcal{A}$ evaluates $\delta_{\mathcal{A}}$ for every vertex in some of the vertex classes, say $V_1(G_{\mathcal{A}})$, and hence

$$d(\mathcal{A}) \geq d(\mathcal{A}(\mathcal{I}_{\mathcal{A}}), \mathcal{I}_{\mathcal{A}}) \geq \frac{\gamma(\mathcal{A}(\mathcal{I}_{\mathcal{A}}))}{\gamma(\mathsf{E})} \geq \frac{|V_1(G_{\mathcal{A}})|}{t} = \frac{tk}{t} = k,$$

as required. ☐

3.2 Deterministic Algorithms for dDkO

3.2.1 *The Preemptive Model.* The following result formalizes a natural lower bound for the cost of the optimal solution of an instance of dDkO in terms of the cost of the optimal solution of the same instance of sDkO. This bound will be used in the proof of Theorem 3.5.

Lemma 3.4. *Let* $\mathcal{I}$ *be an instance of* dDkO *and let* $\gamma^*_{\mathcal{I}}$ *denote the cost of an optimal evaluation for* $\mathcal{I}$ as an instance of sDkO*. Then*

$$\gamma(E^*_{\mathcal{I}}) \geq \frac{\gamma^*_{\mathcal{I}}}{k}. \tag{16}$$

Proof. Let $\mathcal{I} = (G, \delta, \gamma)$ be an instance of dDkO and let E^* be an optimal evaluation for $\mathcal{I}$. For each $1 \leq i \leq k$, let $\gamma_i(\mathsf{E}^*)$ denote the cost incurred by the processor in charge of $V_i(G)$ in E^*. We clearly have

$$\sum_{i=1}^{k} \gamma_i(\mathsf{E}^*) \geq \gamma^*_{\mathcal{I}},$$

and hence

$$\gamma(\mathsf{E}^*_{\mathcal{I}}) = \max\{\gamma_i(\mathsf{E}^*) \colon 1 \leq i \leq k\} \geq \frac{1}{k}\sum_{i=1}^{k} \gamma_i(\mathsf{E}^*) \geq \frac{\gamma^*_{\mathcal{I}}}{k},$$

as required. ☐

Let us define $\mathcal{P}'$ as the algorithm resulting by replacing line (3) of Algorithm $\mathcal{P}$ (see Section 2.2) by the following

Algorithm $\mathcal{P}'$

...

3 evaluate in parallel every vertex of e (whose evaluation has not completed yet) for $g = \gamma(v) - r(v)$ units of time.

...

THEOREM 3.5. *Algorithm $\mathcal{P}'$ is a deficiency-optimal algorithm for* $\mathsf{dD}k\mathsf{O}$ *in the preemptive model with time complexity* $O(k|E(G)|\log|V(G)|)$.

PROOF. Let $\mathcal{I} = (G, \delta, \gamma)$ be an instance of $\mathsf{dD}k\mathsf{O}$, let $\gamma^*_{\mathcal{I}}$ denote the cost of the optimal evaluation for $\mathcal{I}$ *as an instance of* $\mathsf{sD}k\mathsf{O}$. For each $1 \le i \le k$, let $\gamma_i(\mathcal{P}'(\mathcal{I}))$ denote the cost incurred by the processor in charge of $V_i(G)$ in the evaluation $\mathcal{P}'(\mathcal{I})$. Without loss of generality, we assume $\gamma_1(\mathcal{P}'(\mathcal{I})) = \max\{\gamma_i(\mathcal{P}'(\mathcal{I}))\colon 1 \le i \le k\}$ and, from (3) we have

$$\gamma_1(\mathcal{P}'(\mathcal{I})) \le \sum_{e \in J} \max\{\Psi(x)\colon x \in e\} \le \gamma^*_{\mathsf{s}}(\mathcal{I}),$$

where $J = \{e \in E(G)\colon e \text{ is passed to } \textsc{Eval}\}$. Hence with (16) we get

$$d(\mathcal{P}'(\mathcal{I}), \mathcal{I}) = \frac{\gamma(\mathcal{P}'(\mathcal{I}))}{\gamma(\mathsf{E}^*_{\mathcal{I}})} \le \frac{\gamma_1(\mathcal{P}'(\mathcal{I}))}{\gamma^*_{\mathcal{I}}/k} \le \frac{\gamma^*_{\mathcal{I}}}{\gamma^*_{\mathcal{I}}/k} = k,$$

as required. □

3.2.2 *The Non-Preemptive Model*

THEOREM 3.6. *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* $k \ge 13$, *with* $0.2229 < c_k < 0.5177$.
$\min(k\lceil\log_2 W\rceil, k^2-k+1)$, *where W is the largest cost.*

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 $\lceil x\rceil \le k-1$ 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 $0.2229 < \sqrt{6} - 2\log_5 6 \leq c \leq \sqrt{13} - 2\log_5 12 < 0.5177$. □

4. OPEN PROBLEMS

Several problems remain open. We single out the following.

—Is there a randomized algorithm for sDkO with deficiency $(k+1)/2$? We do not know the answer even if exponential time is allowed and $k = 2$.

—Is there a polynomial time algorithm for dDkO with deficiency k for the non-preemptive model?

As noted in Section 1.4, in this work, we restricted our attention to *conjunctive* queries (in the sense of (1)). However, much more general queries could be considered. For example, $\bar{\delta}\colon E(G) \to \{0,1\}$ could be any formula in the first order propositional calculus involving the predicates represented by δ. It would be very interesting to investigate DkO with such generalized queries.

REFERENCES

Avnur, R. and Hellerstein, J. M. 2000. Eddies: continuously adaptive query processing. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data: May 16–18, 2000, Dallas, Texas*, W. Chen, J. Naughton, and P. A. Bernstein, Eds. SIGMOD Record (ACM Special Interest Group on Management of Data), vol. 29(2). ACM Press, New York, NY 10036, USA, 261–272.

Bouganim, L., Fabret, F., Porto, F., and Valduriez, P. 2001. Processing queries with expensive functions and large objects in distributed mediator systems. In *Proc. 17th Intl. Conf. on Data Engineering, April 2-6, 2001, Heidelberg, Germany.* 91–98.

Carmo, R., Feder, T., Kohayakawa, Y., Laber, E., Motwani, R., O'Callaghan, L., Panigrahy, R., and Thomas, D. 2004. A two–player game on graph factors. Tech. Rep. RT-MAC 2004-05, IME-USP. available at `http://www.ime.usp.br/~renato/text/A_Two-Player_Game_on_Graph_Factors`.

Charikar, M., Fagin, R., Guruswami, V., Kleinberg, J., Raghavan, P., and Sahai, A. 2002. Query strategies for priced information. *J. Comput. System Sci. 64,* 4, 785–819. Special issue on STOC 2000 (Portland, OR).

Chaudhuri, S. and Shim, K. 1993. Query optimization in the presence of foreign functions. In *Proc. 19th Intl. Conf. on Very Large Data Bases, August 24-27, 1993, Dublin, Ireland.* 529–542.

Cook, W. J., Cunningham, W. H., Pulleyblank, W. R., and Schrijver, A. 1997. *Combinatorial Optimization.* John Wiley, New York.

Feder, T., Motwani, R., O'Callaghan, L., Panigrahy, R., and Thomas, D. 2003. Online distributed predicate evaluation. Tech. Rep. 2003-81, Stanford University. available at `http://dbpubs.stanford.edu/pub/2003-81`.

Ferdinando Cicalese and Eduardo Laber. 2005. A new strategy for qurying priced information. In *STOC 2005*.

Hellerstein, J. M. 1998. Optimization techniques for queries with expensive methods. *ACM Transactions on Database Systems 23,* 2 (June), 113–157.

Laber, Carmo, and Kohayakawa. 2004. Querying priced information in databases: The conjunctive case: Extended abstract. In *LATIN: Latin American Symposium on Theoretical Informatics*.

Laber, E. 2004. A randomized competitive algorithm for evaluating priced and/or trees. In *STACS 2004*.

Laber, E. S., Parekh, O., and Ravi, R. 2002. Randomized approximation algorithms for query optimization problems on two processors. In *Proceedings of ESA 2002*. Rome, Italy, 136–146.

Porto, F. 2001. Estratégias para a execução paralela de consultas em bases de dados científicos distribuídos (strategiies for the parallel execution of queries in distributed scientific databases). Ph.D. thesis, Departamento de Informática, PUC-Rio.

Vazirani, V. V. 2001. *Approximation Algorithms*. Springer-Verlag.

Yao, A. C. 1977. Probabilistic computations : Toward a unified measure of complexity. In *18th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society Press, Long Beach, Ca., USA, 222–227.

Received Month Year; revised Month Year; accepted Month Year