Revenue Maximization for Query Pricing

Shuchi Chawla, Shaleen Deep, Paraschos Koutris, Yifeng Teng
University of Wisconsin-Madison
Madison, WI, USA
{shuchi, shaleen, paris, yifengt}@cs.wisc.edu

ABSTRACT

Buying and selling of data online has increased substantially over the last few years. Several frameworks have already been proposed that study query pricing in theory and practice. The key guiding principle in these works is the notion of *arbitrage-freeness* where the broker can set different prices for different queries made to the dataset, but must ensure that the pricing function does not provide the buyers with opportunities for arbitrage. However, little is known about revenue maximization aspect of query pricing. In this paper, we study the problem faced by a broker selling access to data with the goal of maximizing her revenue. We show that this problem can be formulated as a revenue maximization problem with single-minded buyers and unlimited supply, for which several approximation algorithms are known. We perform an extensive empirical evaluation of the performance of several pricing algorithms for the query pricing problem on real-world instances. In addition to previously known approximation algorithms, we propose several new heuristics and analyze them both theoretically and experimentally. Our experiments show that algorithms with the best theoretical bounds are not necessarily the best empirically. We identify algorithms and heuristics that are both fast and also provide consistently good performance when valuations are drawn from a wide variety of distributions.

PVLDB Reference Format:
Chawla et al. Revenue Maximization for Query Pricing. *PVLDB*, 13(1): 1-14, 2019.
DOI: https://doi.org/10.14778/3357377.3357378

1. INTRODUCTION

The last decade or so has seen an explosion of data being collected from a variety of sources and across a broad range of areas. Many companies, including Bloomberg [4], Twitter [10], Lattice Data [7], DataFinder [5], and Banjo [2] collect such data, which they then sell as structured (relational) datasets. These datasets are also often sold through online *data markets*, which are web platforms for buying and selling data: examples include BDEX [3], Salesforce [9] and QLik DataMarket [8]. Even though data sellers and data markets offer an abundance of data products, the pricing schemes currently used are very simplistic. In most cases, a data buyer has only one option, to buy the whole dataset at a fixed price. Alternatively, the dataset is split into multiple disjoint chunks, and each chunk is sold at a separate price.

However, buyers are often interested in extracting specific information from a dataset and not in acquiring the whole dataset. Accessing this information can be concisely captured through a *query*. Selling the whole dataset at a fixed price forces the buyer to either pay more for the query than it is valued, or to not buy at all. This means that valuable data is often not accessible to entities with limited budgets, and also that data-selling companies and marketplaces behave suboptimally with respect to maximizing their revenue. Indeed, popular cloud database providers such as Google [6] and Amazon [1] also follow a coarse grained pricing model where the user is charged based on the number of bytes scanned rather than information content of the requested query.

Query-Based Pricing. To address this problem, a recent line of research [37, 39, 30] introduced the framework of query-based pricing. A *query-based pricing scheme* tailors the purchase of the data to the user's needs, by assigning a different price to each issued query. Given a dataset $\mathcal{D}$ and a query Q over the dataset, the user must pay a price $p(Q, \mathcal{D})$ to obtain the answer $Q(\mathcal{D})$. This price reflects only the value of the information learned by obtaining the query answer, and not the computational cost of executing the query. The work on query-based pricing has mainly focused on how to define a well-behaved pricing function, and how to develop system support for efficiently implementing a data marketplace. In particular, a key property that a pricing function must obey is that of *arbitrage-freeness*: it should not be possible for the buyer to acquire a query for a cheaper price through the combination of other query results. The arbitrage-freeness constraint makes the design of pricing functions a challenging task, since deciding whether a query is more informative than another query (or set of queries) is generally computationally hard, and for practical applications it is critical that the price computation can be performed efficiently.

To overcome this barrier, [30] proposes a setup where we start with a set $\mathcal{S}$ consisting of multiple "candidate" databases instances; this set is called the *support*. Each query Q can then be thought of as a function that classifies instances from $\mathcal{S}$: ones that return the same answer as

Proceedings of the VLDB Endowment, Vol. 13, No. 1
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3357377.3357378

$Q(\mathcal{D})$, and ones that do not. Whether a query is more informative than another then amounts to whether it classifies more inconsistent instances than the other. The benefit of this approach is that in order to find the price of a query, it suffices to only examine the instances in the set $\mathcal{S}$, which is a computationally feasible problem.

The Revenue Maximization Problem. Although prior work provides a framework to reason about the formal properties of pricing functions, it does not address the following fundamental question:

How do we assign prices to the queries in order to maximize the seller's revenue while ensuring arbitrage freeness?

This is the main problem we study in this paper. Our key observation is that query pricing can be cast as a problem of pricing subsets (*bundles*) over a ground set of *items*, where each item corresponds to a database instance in $\mathcal{S}$. The arbitrage-freeness constraint corresponds to the pricing function (which is a set function over $\mathcal{S}$) being *monotone* and *subadditive*. Since there is no limit to how many times a seller can sell a query (a digital good), we can model the seller as having *unlimited supply* for each query answer. We further consider *single minded* buyers, which means that each buyer wants to buy the answer to a single set of queries.

Finding the monotone and subadditive pricing function that maximizes revenue in this setting is a computationally hard problem. Furthermore, a subadditive function, even if we manage to find one, can take exponential space (w.r.t. $\mathcal{S}$) to store. Therefore, *for practical applications we must seek a simple and concise pricing function that approximates the optimal subadditive pricing in terms of revenue.* In this paper, we explore such succinct families of pricing functions that are appropriate for use in a data market, and answer the following questions:

- What is the theoretical gap between optimal revenue and the revenue obtained through succinct families of pricing functions?
- Which revenue maximization algorithms are best suited for query pricing and what guarantees do they offer?
- How well do the theoretical revenue and performance bounds translate to real-world query workloads?

Our Contributions. We now discuss our contributions in detail.

Succinct Pricing Functions. We study three types of succinct pricing functions (Section 4). The first, *uniform bundle pricing*, assigns the same price to every bundle (query) and is the default pricing scheme in many data markets. The second, additive or *item pricing*, assigns a price to each item (instance in the support) and charges a price for each bundle equal to the sum of prices for the items in the bundle. Item pricing has been studied extensively from a theoretical perspective and a number of approximation algorithms are known (see, e.g.,[34, 15, 19]). Third, we consider a much more general class of pricings, namely *XOS or fractionally subadditive pricings*. These pricing functions are more expressive than item or uniform bundle pricings, while at the same time having a small representation size. Our key finding is that XOS pricing functions can achieve a logarithmic factor larger revenue than the better of item and uniform bundle pricing.

Revenue Maximization Algorithms. We theoretically study several algorithms for finding the revenue maximizing pricing function (Section 5). In quantifying performance, several parameters of the instance are relevant: the number of items n (which is the size of the support), the number of bundles m (which is the number of the queries issued in the market), the size of the largest bundle k, and the maximum number of bundles any item belongs to B. In the context of query pricing, it is usually the case that $B \leq m \ll k \leq n$, so algorithms with approximation factors and running time depending on B or m are generally better than those depending on k or n.

In particular, although we can always find the optimal uniform bundle pricing efficiently, it is computationally hard to find the optimal item pricing. Hence, we consider several algorithms for the latter task that come with worst case approximation factors that are logarithmic in one or more of the natural parameters of the instance. Apart from known algorithms, we also develop new algorithmic techniques that improve performance. Finally, for the family of XOS pricing functions, we propose an algorithm that simply combines multiple additive item pricing functions.

Experimental Evaluation. Finally, we perform an empirical analysis of the different pricing functions to understand how well do the algorithms hold up in practice, which of these algorithms should a practitioner use, and what features of the problem instance dictate this choice (Section 6). We compare the pricing algorithms using both synthetic and real world query workloads.

Our study shows that the worst-case analysis of pricing algorithms does not capture how well the algorithms behave in real-world instances in terms of approximating the optimal revenue. In particular, we observe that the structure of the bundles induced by different query workloads, as well as the distribution of buyer valuations, heavily influences the quality of approximation. For example, the algorithm that obtains the best known worst case approximation ratio does not achieve the best performance of the algorithms we tested in any of our setups. Our experiments also show that it is possible to efficiently extract most of the available revenue using succinct pricing functions, and in particular item pricings. Hence, succinct pricing functions seem a good practical choice for a data marketplace.

Lessons and Open Problems. Finally, we discuss our takeaway lessons, as well as several exciting open questions (Section 7).

2. RELATED WORK

Query-Based Pricing. There exist various simple mechanisms for data pricing (see [42] for a survey on the subject), including a flat fee tariff, usage-based and output-based prices. These pricing schemes do not provide any guarantees against arbitrage. The vision for arbitrage-free query-based pricing was first introduced by Balazinska et al. [14], and was further developed in a series of papers [37, 39, 38]. The proposed framework requires that the seller sets fine-grained price points, which are prices assigned to a specific type of queries over the dataset; these price points are used as a guide to price the incoming queries. Even though the pricing problem in this setting is in general NP-hard, the QueryMarket prototype [39] showed that it is feasible to compute the prices for small datasets, albeit not in real-time

or maximizing the revenue. Further work on data pricing proposed new criteria for interactive pricing in data markets [40], and described new necessary arbitrage conditions along with several negative results related to the trade-off between flexible pricing and arbitrage avoidance [41]. Upadhyaya et al. [47] investigated history-aware pricing using refunds. More recently, [29] characterized the possible space of pricing functions with respect to different arbitrage conditions. The theoretical framework was then implemented as part of the QIRANA system [30, 31], which can support pricing in real-time. The framework we use in this paper for query-based pricing is the same one from [30].

Revenue Maximization. Revenue-maximizing mechanisms have been well understood in single-item auctions, where the posted pricing mechanism is optimal [43]. However, in general multi-parameter settings, revenue-maximizing mechanisms are considered hard to characterize. In the past few decades, many researchers started to focus on simple and approximately optimal solutions, especially posted-pricing mechanisms. Recent line of work shows that in Bayesian setting with limited supply, posted pricing achieves constant approximation when there is single buyer [11, 24, 25, 26, 45], and logarithmic approximation (with respect to the number of items) when there are multiple buyers [20, 27, 21].

In this paper, we focus on the case where all valuations of buyers are revealed to the seller. The study of this setting was initiated by [34], which shows that item pricing gives $O(\log n)$-approximation for unit-demand buyers in limited-supply setting, and $O(\log n + \log m)$-approximation for single-minded buyers in unlimited supply setting. The competitive ratio for unlimited supply setting was improved to $O(\log k + \log B)$ by [19] then to $O(\log B)$ by [28] where k denotes the size of largest bundle, and B denotes the maximum number of bundles containing a specific item. Another line of work studies how to find best possible item pricing in above setting where k is bounded. This problem is also known as the *k-hypergraph pricing* problem. [19] gave the first polynomial-time algorithm finding an approximately optimal item pricing with competitive ratio k^2. The approximation ratio is improved to k by [15], which is proven to be near-optimal: under the Exponential Time Hypothesis there is no polynomial-time algorithm that achieves competitive ratio $k^{1-\epsilon}$ [22].

Pricing information. Another line of research in economics considers the revenue maximization problem for a seller offering to sell information. See, for example, [12, 16, 17] and references therein. However, that literature differs from our work in several fundamental aspects. First, in those works, both the seller and the buyer are unaware of the true state of the information (*i.e.*, the dataset), and this state is stochastic. Second, the seller is allowed to sell queries whose results are randomized. Third, the buyer's type (which information he is interested in and his value) are unknown to the seller. In contrast, in our setting while the pricing is required to be arbitrage-free, the types of the buyers are known to the seller in advance. As such the two models lead to very different types of pricing mechanisms and algorithms.

3. THE QUERY-BASED PRICING FRAMEWORK

USER

uid	name	gender	age
1	Abe	m	18
2	Alice	f	20
3	Bob	m	25
4	Cathy	f	22

Figure 1: A relation with 4 attributes. uid is the primary key.

In this section, we present the framework of query-based pricing proposed by [29], and then formally describe the pricing problems we tackle.

3.1 Query-Based Pricing Basics

The *data seller* wants to sell an instance $\mathcal{D}$ through a data market, which functions as the broker. The instance has a fixed relational schema $\mathbf{R} = (R_1, \ldots, R_k)$. We denote by $\mathcal{I}$ the set of possible database instances. The set $\mathcal{I}$ encodes information about the data that is provided by the data seller, and is public information known to any buyer (together with the schema). We allow the set $\mathcal{I}$ to be infinite, but countable. For example, suppose that the schema consists of a single binary relation $R(A, B)$, and the domain of both attributes is $[\ell] = \{1, \ldots, \ell\}$. Then, $\mathcal{I} = 2^{[\ell]\times[\ell]}$, *i.e.* the set of all directed graphs on the vertex set $[\ell]$.

Data buyers can purchase information by issuing queries on $\mathcal{D}$ in the form of a *query vector* $\mathbf{Q} = \langle Q_1, \ldots, Q_p \rangle$. For our purposes, a query Q is a deterministic function that takes as input a database instance $\mathcal{D}$ and returns an output $Q(\mathcal{D})$. We denote the output of the query vector by $\mathbf{Q}(\mathcal{D}) = \langle Q_1(\mathcal{D}), \ldots Q_p(\mathcal{D}) \rangle$.

EXAMPLE 1. *Consider a database that consists of a single* USER *relation as shown in Figure 1. Suppose that the data seller has fixed the price of the entire relation to* \$100. *Consider a data buyer, Alice, who is a data analyst and wants to study user demographics. Since Alice has a limited budget, she cannot afford to purchase the entire database. Thus, Alice will extract information from the table by issuing relational queries over time. We will use this as a running example throughout the section.*

A *pricing function* $p(\mathbf{Q}, \mathcal{D})$ takes as input a query vector $\mathbf{Q}$ and a database instance $\mathcal{D} \in \mathcal{I}$ and assigns to it a price,[1] which is a number in $\mathbb{R}_+$. Assigning prices to query vectors without any restrictions can lead to arbitrage opportunities in the following two ways:

Information Arbitrage. The first condition captures the intuition that if a query vector $\mathbf{Q}_1$ reveals a subset of information of what a query vector $\mathbf{Q}_2$ reveals, then the price of $\mathbf{Q}_1$ must be no more than the price of $\mathbf{Q}_2$. If this condition is not satisfied, it creates an arbitrage opportunity, since a data buyer can purchase $\mathbf{Q}_2$ instead, and use it to obtain the answer of $\mathbf{Q}_1$ for a cheaper price.

Formally, we say that $\mathbf{Q}_2$ determines $\mathbf{Q}_1$ under database $\mathcal{D}$ if for every database $\mathcal{D}' \in \mathcal{I}$ such that $\mathbf{Q}_2(\mathcal{D}) = \mathbf{Q}_2(\mathcal{D}')$, we also have $\mathbf{Q}_1(\mathcal{D}') = \mathbf{Q}_1(\mathcal{D})$. We say that the pricing function p has no *information arbitrage* if for every database $\mathcal{D} \in \mathcal{I}$

[1] Allowing prices to be general functions of query vectors, rather than just additive over queries allows for more expressivity and therefore more revenue for the seller.

Table 1: Symbol definitions.

Symbol	Description
$\mathbf{Q}$	Query vector containing buyer queries
$\mathcal{D}$	Input database provided by the seller
$\mathcal{I}$	Set of all possible database instances consistent with $\mathcal{D}$
$\mathcal{S}$	Support set chosen by the framework
$p(\mathbf{Q}, \mathcal{D})$	Pricing function
$\mathcal{C}_\mathcal{S}(\mathbf{Q}, \mathcal{D})$	Conflict set of query $\mathbf{Q}$ (subset of $\mathcal{S}$)
$\mathcal{V}$	Vertex set of hypergraph
$\mathcal{E}$	Hyperedges in a hypergraph
$\mathcal{H} = (\mathcal{V}, \mathcal{E})$	Hypergraph generated by transforming buyer queries into hyperedges (bundles) containing db's in conflict set ($\mathcal{V} = \mathcal{S}$)
item j	some database $j \in \mathcal{V}$ in vertex set of $\mathcal{H}$
$v_\mathbf{Q}$	buyer valuation of query vector $\mathbf{Q}$
bundle A	Conflict set of some query in $\mathcal{H}$.
B	maximum degree over all items in $\mathcal{H}$

such that $\mathbf{Q}_2$ determines $\mathbf{Q}_1$ under $\mathcal{D}$, we have $p(\mathbf{Q}_2, \mathcal{D}) \geq p(\mathbf{Q}_1, \mathcal{D})$.

Example 2. *In our running example, suppose that Alice wants to count the number of female users in the relation. She can issue the query* Q_1 = `SELECT count(*) FROM User WHERE gender = 'f'`. *However, a different way she can learn the same information is by issuing the query* Q_2 = `SELECT gender, count(*) FROM User GROUP BY gender`. Q_2 *will return the number of users for each gender as its output. Suppose that the buyer charges* $p(Q_1) = \$10$ *and* $p(Q_2) = \$5$, *then there exists an information arbitrage opportunity. Since Alice can learn the required information from* Q_2 *at a* cheaper *cost, she has no incentive to purchase* Q_1. *Thus, if the seller wants to prevent this arbitrage, he needs to ensure that* $p(Q_1) \leq p(Q_2)$. *Thus, the seller now sets the price of* Q_2 *to* \$10, *i.e,* $p(Q_2) = \$10$.

Combination Arbitrage. The second condition regards the scenario where a data buyer wants to obtain the answer for the query vector $\mathbf{Q} = \mathbf{Q}_1 \| \mathbf{Q}_2$, where $\|$ denotes vector concatenation. Instead of asking $\mathbf{Q}$ as one, the buyer can create two separate accounts, and use one to ask for $\mathbf{Q}_1$ and the other to ask for $\mathbf{Q}_2$. To avoid such an arbitrage situation, we must make sure that the price of $\mathbf{Q}$ is at most the sum of the prices for $\mathbf{Q}_1$ and $\mathbf{Q}_2$. Formally, we say that the price function p has no *combination arbitrage* if for every database $\mathcal{D} \in \mathcal{I}$, we have $p(\mathbf{Q}_1 \| \mathbf{Q}_2, \mathcal{D}) \leq p(\mathbf{Q}_1, \mathcal{D}) + p(\mathbf{Q}_2, \mathcal{D})$.

Example 3. *Alice now wants to find the average age of female users in the relation. She can issue the query* Q_3 = `SELECT AVG(age) FROM User WHERE gender = 'f'`. *Suppose the seller decides to price* $p(Q_3) = \$20$. *However, Alice could have also chosen to ask* Q_4 = `SELECT SUM(age) FROM User WHERE gender = 'f'`. *Now, she can obtain her desired result by combining the answers of* Q_4 *and* Q_2 [2]. *If the seller prices* $p(Q_4) = \$5$, *then there exists a combination arbitrage opportunity, since* $p(Q_3) > p(Q_4) + p(Q_2)$ *which gives Alice an incentive to split her query. To avoid this, the seller needs to ensure that* $p(Q_3) \leq p(Q_4) + p(Q_2)$.

We say that the pricing function p is *arbitrage-free* if it has no information arbitrage and no combination arbitrage.

[2]We assume that it is public information that gender in this relation takes only two values: m and f

3.2 From Pricing Queries to Pricing Bundles

In general, computing whether $\mathbf{Q}_2$ determines $\mathbf{Q}_1$ under some $\mathcal{D}$ is an intractable problem. To overcome this obstacle, we take a different view of a query vector. Let $\mathcal{S} \subseteq \mathcal{I}$ be any subset of $\mathcal{I}$, called the *support*, and define the *conflict set* of $\mathbf{Q}$ with respect to $\mathcal{S}$ as:

$$\mathcal{C}_\mathcal{S}(\mathbf{Q}, \mathcal{D}) = \{\mathcal{D}' \in \mathcal{S} \mid \mathbf{Q}(\mathcal{D}) \neq \mathbf{Q}(\mathcal{D}')\}.$$

Intuitively, the conflict set contains all the instances from $\mathcal{S}$ for which the buyer knows that cannot be the underlying instance $\mathcal{D}$ once she learns the answer $\mathbf{Q}(\mathcal{D})$. This construction maps each query vector to a *bundle* $\mathcal{C}_\mathcal{S}(\mathbf{Q}, \mathcal{D})$ over the set $\mathcal{S}$. We should remark here that the task of computing the bundle $\mathcal{C}_\mathcal{S}(\mathbf{Q}, \mathcal{D})$ is computationally feasible if we choose $\mathcal{S}$ to be small enough, since we can simply iterate through all the items $\mathcal{D}' \in \mathcal{S}$, and for each item check the condition $\mathbf{Q}(\mathcal{D}) \neq \mathbf{Q}(\mathcal{D}')$.

Example 4. *Consider the support set* $\mathcal{S} = \{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3\}$ *as shown below. The colored font highlights the values that are changed with respect to* $\mathcal{D}$.

$\mathcal{D}_1$

uid	name	gender	age
1	Abe	m	18
2	Alice	f	**30**
3	Bob	m	25
4	Cathy	f	22

$\mathcal{D}_2$

uid	name	gender	age
1	Abe	**f**	18
2	Alice	f	20
3	Bob	m	25
4	Cathy	f	22

$\mathcal{D}_3$

uid	name	gender	age
1	Abe	m	18
2	Alice	f	20
3	**Ben**	m	25
4	Cathy	f	22

For query Q_1 *from Example 2,* $Q_1(\mathcal{D}) = Q_1(\mathcal{D}_1) = Q(\mathcal{D}_3)$ *but* $Q_1(\mathcal{D}) \neq Q_1(\mathcal{D}_2)$. *Thus,* $\mathcal{C}_\mathcal{S}(Q_1, \mathcal{D}) = \{\mathcal{D}_2\}$. *Similarly,* $\mathcal{C}_\mathcal{S}(Q_3, \mathcal{D}) = \{\mathcal{D}_1, \mathcal{D}_2\}$.

We can now compute a price for $\mathbf{Q}$ by applying a set function $f : 2^\mathcal{S} \to \mathbb{R}_+$ to $\mathcal{C}_\mathcal{S}(\mathbf{Q}, \mathcal{D})$. A set function f is *monotone* if for sets $A \subseteq B$ we always have $f(A) \leq f(B)$, and *subadditive* if for every set A, B we have $f(A) + f(B) \geq f(A \cup B)$. By choosing f to be monotone and subadditive, we can guarantee that the pricing function is arbitrage-free.

Theorem 1 ([29]). *Let* $\mathcal{S} \subseteq \mathcal{I}$, *and* f *be a set function* $f : 2^\mathcal{S} \to \mathbb{R}_+$. *Then, the pricing function* $p(\mathbf{Q}, \mathcal{D}) = f(\mathcal{C}_\mathcal{S}(\mathbf{Q}, \mathcal{D}))$ *is arbitrage-free if and only if the function* f *is monotone and subadditive.*

We emphasize that the arbitrage-freeness guarantee holds for any support $\mathcal{S}$. The choice of $\mathcal{S}$ impacts the granularity of prices that can be assigned to queries which in turn affects the revenue. Observe that in the extreme case when $\mathcal{S} = \emptyset$, $\mathcal{C}_\mathcal{S}(\mathbf{Q}, \mathcal{D}) = \emptyset$ for any $\mathbf{Q}$ implying that all queries have exactly the same price. On the other hand, a large support set can make the computation of the conflict set prohibitively expensive. Section 6.5 explores the tradeoff between the revenue obtained and $\mathcal{S}$ in more detail.

3.3 Revenue Maximization

We consider the unlimited supply setting, where the seller can sell any number of units of each query. Additionally, we assume that the buyers are single-minded: each buyer is interested in buying only a single query vector $\mathbf{Q}$; the buyer will purchase $\mathbf{Q}$ only if $p(\mathbf{Q}, \mathcal{D}) \leq v_\mathbf{Q}$, where $v_\mathbf{Q}$ is the valuation that the buyer has for $\mathbf{Q}$. Note that the single-minded buyer assumption is not restrictive; a buyer who wishes to purchase multiple queries (say $Q_1, \ldots, Q_\lambda$) can be modeled as λ separate buyers where each buyer $i \in [\lambda]$ wants

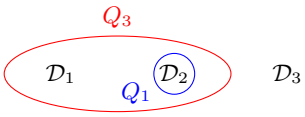


Figure 2: Hypergraph instance for Example 4. Q_1 and Q_3 are represented as hyperedges containing the databases in their conflict sets.

to purchase query $\mathbf{Q}_i = \langle Q_i \rangle$. Similarly, if the buyer wants to buy all λ queries together, then we can already express it as a bundle $\mathbf{Q} = \langle Q_1, \ldots, Q_\lambda \rangle$.

The problem setup is as follows. We are given as input a set of m buyers, where each buyer i is interested of purchasing a query vector $\mathbf{Q}_i$ with valuation v_i. These valuations can be found by performing market research in order to understand the demand and price buyers are willing to pay for queries of interest. Defining and using these demand curves is a standard practice in the study of economics for digital goods [33]. We pick a support set $\mathcal{S} \subseteq \mathcal{I}$ of size $n = |\mathcal{S}|$. By using the transformation of query vectors to bundles over $\mathcal{S}$, we can construct a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, with vertex set $\mathcal{V} = \mathcal{S}$, and hyperedges $\mathcal{E} = \{e_i \mid i = 1, \ldots, m\}$, where $e_i = \mathcal{C}_\mathcal{S}(\mathbf{Q}_i, \mathcal{D})$. Figure 2 shows an example of hypergraph instance constructed from the queries and their conflict sets.

A pricing function p is a set function that maps subsets of S to prices in $\mathbb{R}_+$[3]. The task at hand is to find a monotone and subadditive pricing p that maximizes the seller's revenue. The revenue of a pricing function p is given by:

$$R(p) = \sum_{i: v_i \geq p(e_i)} p(e_i)$$

The optimal revenue is:

$$\text{OPT} = \max_{\text{monotone; subadditive } p} R(p)$$

Many of the approximation results in the literature use a simpler (and weaker) upper bound on OPT, namely the sum of all bundle values $\sum_i v_i$, as a basis of comparison for algorithms' performance. Throughout the paper, we will use the term hypergraph to refer to the instance created by the transformation as described above and the term *item* to refer to some instance in the vertex set $\mathcal{V}$ of the hypergraph.

3.4 Simple Pricing Functions

For practical applications (e.g., QIRANA [30]), we must only consider functions that can be both concisely represented and also efficiently computable. For example, it is not desirable to come up with a function p where we need to explicitly store all the 2^n values for all input bundles from $\mathcal{S}$. For this reason, we focus on a few important subclasses of monotone and subadditive set functions:

- The *uniform bundle price* $p^b(\cdot)$ assigns the same price to every hyperedge, *i.e.* $p^b(e) = P$ for some number $P \geq 0$.
- The *additive price* $p^a(\cdot)$ assigns a weight $w_j \geq 0$ to every item $j \in \mathcal{S}$, and then defines $p^a(e) = \sum_{j \in e} w_j$. Such a pricing function is also commonly known as an *item pricing*.

[3]Here we have overloaded p to also be a pricing function with input a bundle of items.

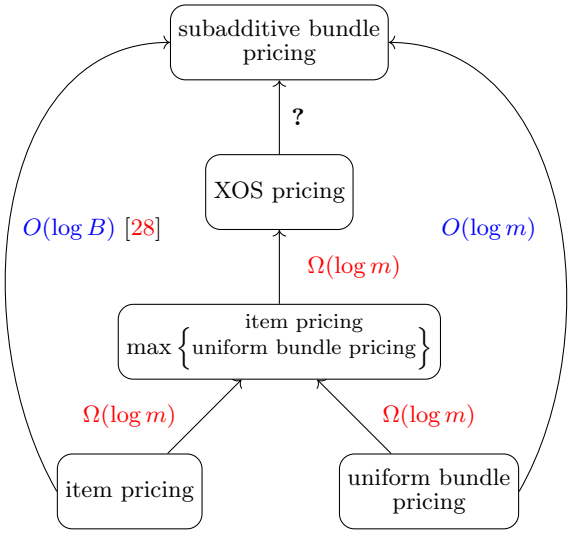


Figure 3: Summary of the lower and upper bounds between different subclasses of pricing functions. The red font show results in this paper; blue font shows known results.

- The *XOS price* $p^x(\cdot)$ defines k weights $w_j^1, w_j^2, \ldots, w_j^k$ for each item $j \in \mathcal{S}$, and sets the price to $p^x(e) = \max_{i=1}^k \sum_{j \in e} w_j^i$.

Given the above three subclasses of pricing functions, we consider the following two questions, both from a theoretical and practical point of view. First, how much do we lose in terms of revenue by replacing the optimal monotone and subadditive pricing function with a uniform, additive or XOS pricing function? In other words, we seek to understand what is the revenue we lose for the sake of computational efficiency. Second, we want to develop algorithms that can optimize the prices for each subclass and achieve a good approximation ratio with respect to the optimal revenue.

4. UPPER AND LOWER BOUNDS

In this section, we present worst-case guarantees on how well uniform bundle pricing and item pricing can approximate the optimal subadditive and monotone bundle pricing. Figure 3 summarizes the upper and lower bounds that are either known, or we obtain in this paper.

Upper Bounds. It is a folklore result that for any hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ with valuations $\{v_e\}_{e \in \mathcal{E}}$, one can always construct a uniform bundle price that is $O(\log m)$ away from the sum of valuations $\sum_e v_e$, which is an upper bound on the optimal subadditive and monotone bundle pricing.

LEMMA 1. *Consider a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ with valuations $\{v_e\}_{e \in \mathcal{E}}$. Then, there exists a uniform bundle price p^b that achieves revenue $O(\log m)$ away from $\sum_{e \in \mathcal{E}} v_e$, where $m = |\mathcal{E}|$.*

Similarly, we know from [28] that item pricing can achieve a $O(\log B)$ approximation of the sum of valuations. Recall that B is the maximum number of hyperedges that any vertex can be contained in, and hence $B \leq m$.

Lower Bounds. The theoretical upper bound of $O(\log m)$ is tight in the worst case for both uniform item pricing and

bundle pricing. In particular, we show the following results (proofs in full paper [23]).

LEMMA 2. *There exists a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ with additive valuations, such that any uniform bundle price produces revenue $\Omega(\log m)$ from the optimal revenue $\sum_{e \in \mathcal{E}} v_e$.*

LEMMA 3. *There exists a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ with uniform valuations, such that any item pricing solution produces revenue $\Omega(\log m)$ from the optimal revenue $\sum_{e \in \mathcal{E}} v_e$.*

LEMMA 4. *There exists a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ with submodular valuations, such that any uniform bundle pricing and any item pricing produces revenue $\Omega(\log m)$ from the optimal revenue $\sum_{e \in \mathcal{E}} v_e$.*

Note that for each of the above result, there exists a subadditive pricing function that can extract the full revenue. The above lower bounds tell us that there are problem instances where uniform bundle pricing will be optimal, but item pricing will behave poorly, and vice versa. Moreover, there are instances where both subclasses of pricing functions will not perform well with respect to the optimal submodular monotone function (which is a subset of subadditive and monotone bundle pricing). A straightforward corollary of the lower bound of Lemma 4 is that even an XOS pricing function that combines a constant number of item pricing functions suffers from the $\Omega(\log m)$ revenue gap. An open question here is whether an XOS pricing that uses a non-constant (but still small enough) number of item pricings can obtain a better approximation guarantee with respect to the optimal subadditive and monotone bundle pricing.

5. APPROXIMATION ALGORITHMS

In this section, we present the various approximation algorithms that we consider in our experimental evaluation. We consider algorithms from two subclasses of subadditive and monotone pricing schemes: (*i*) uniform bundle pricing, and (*ii*) item (additive) pricing.

5.1 Uniform Bundle Pricing

In uniform bundle pricing, the algorithm sells every hyperedge at a fixed price P. Then, if the buyer has valuation $v_e \geq P$, the hyperedge (and thus, the query bundle corresponding to the hyperedge) can be sold. To compute the optimal uniform bundle price P, we use a folklore algorithm that we call UBP. The algorithm first sorts the hyperedges in $\mathcal{E}$ in decreasing order of valuation. Then, it makes a linear pass over the ordered valuations, and for every hyperedge $e \in E$ computes the revenue R_e obtained if we set the price $P = v_e$. In the end, it outputs $\max_e\{R_e\}$. It is easy to see that the algorithm runs in time $O(m \log m)$, and that it achieves an approximation ratio of $O(\log m)$.

Uniform bundle pricing is very attractive as a practical pricing scheme, since it has a single parameter (and thus it has a concise representation), and computing the price of a new query is a trivial task. However, because it is insensitive to the structure of the bundle (and hence the query), it will perform very poorly when the valuations have a large difference across the hyperedges. We should remark here that we expect this to be the case in real-world scenarios: for instance, consider a large table and two queries: one that returns the whole dataset, and one that returns only a single row of the table. Then, it is reasonable to expect that the valuation for these two queries will generally differ by a large margin.

5.2 Item Pricing

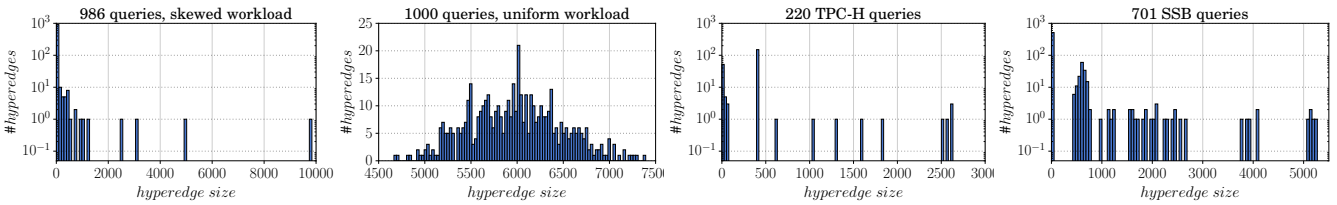
In item pricing, we seek to assign a weight $w_j \geq 0$ to each vertex in the hypergraph. Then, the price of any hyperedge e is given by $p(e) = \sum_{j \in e} w_j$. The representation size of item pricing is $O(n)$, so we can guarantee that it will have a concise representation as long as we pick the support set to be small enough (recall that $n = |\mathcal{S}|$). Unlike uniform bundle pricing, item pricing can capture large differences between the valuation of different queries. On the other hand, we also need to choose a large enough support size, so that we can extract a reasonably good revenue from our pricing function. A large support size also guarantees that that new queries that arrive will have non-empty hyperedges and hence will not be priced to 0. In our experiments, we evaluate four item pricing algorithms.

The Uniform Item Pricing (UIP) Algorithm. Our first item pricing algorithm is an $O(\log n + \log m)$-approximation algorithm given by Guruswami et. al [34]. UIP outputs a uniform item pricing, where every w_j is set to the same value w. The algorithm sorts all hyperedges on the value $q_e = \frac{v_e}{|e|}$, computes for every hyperedge e the revenue R_e that we can obtain if we set $w_j = q_e$ for every $j \in \mathcal{V}$, and finally outputs the pricing that gives $\max_e\{R_e\}$. Its running time is also $O(m \log m)$.

The LP Item Pricing (LPIP) Algorithm. The second item pricing algorithm we consider builds upon the UIP algorithm to construct a non-uniform item pricing. LPIP constructs a separate linear program $LP(e)$ for every hyperedge $e \in \mathcal{E}$ as follows. Let $F_e \subseteq \mathcal{E}$ be the set of hyperedges e' such that $v_{e'} \geq v_e$. Then, $LP(e)$ has the objective of maximizing the revenue, with the constraint that every edge in F_e must be sold: in other words, for every $e' \in F_e$, we must have $\sum_{j \in e'} w_j \leq v_{e'}$. Observe that the uniform item pricing solution that sets each weight w_j to v_e is a feasible solution for $LP(e)$, so the output of the linear program can only give a better item pricing. LPIP outputs the revenue-maximizing solution across all $LP(e)$. The worst-case approximation guarantee of LPIP is also $O(\log m)$; however, as we will see in the experimental section, it often outperforms UIP.

The Capacity Item Pricing (CIP) Algorithm. This algorithm is an $O(\log B)$-approximation algorithm given by [28]. Although this primal-dual algorithm was presented in the context of item pricing with limited supply, it readily extends to the unlimited supply setting. Intuitively, CIP sets a uniform capacity constraint k on how many times each item (vertex) can be sold, and for that k solves a linear program that optimizes for the welfare-maximization problem. The dual solution of this LP gives the prices of items such that at least k copies of each item are sold. [28] proves that if we search through the possible capacity constraint using a step-size of $(1+\epsilon)$ – so $k = 1, (1+\epsilon), (1+\epsilon)^2, \ldots$ – then the revenue-maximizing item pricing across all k's achieves an approximation ratio of $O((1+\epsilon)\log B)$.

The Layering Algorithm. Since the previous algorithms require solving multiple linear programs, they can be slow when the size of input is large. We thus also consider a fast



(a) Skewed workload **(b)** Uniform workload **(c)** TPC-H workload **(d)** SSB workload

Figure 4: Hyperedge size distribution

greedy algorithm that achieves an $O(B)$-approximation in the worst case (but as we will see, a much better approximation in practice).

Algorithm 1: The Layering Algorithm

input : Hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ and valuation $\{v_e\}_{e\in\mathcal{E}}$
output: Item pricing w_j for each item $j \in \mathcal{V}$

$Rev \leftarrow 0$, $S \leftarrow \emptyset$, $w_j \leftarrow 0$ for each $j \in \mathcal{V}$;
while $\mathcal{E} \neq \emptyset$ **do**
  Let $\mathcal{E}' \subseteq \mathcal{E}$ be a minimal set cover of the items in $\bigcup_{e\in\mathcal{E}} e$;
  if $\sum_{e\in\mathcal{E}'} v_e > Rev$ **then**
    $S \leftarrow \mathcal{E}'$;
    $Rev \leftarrow \sum_{e\in\mathcal{E}'} v_e$;
  $\mathcal{E} \leftarrow \mathcal{E} \setminus \mathcal{E}'$;
for $e \in S$ **do**
  Find item $j \in e$ such that $j \notin e'$, $\forall e' \in S$, $e' \neq e$;
  $w_j \leftarrow v_e$;
return $\{w_j\}_{j\in\mathcal{V}}$

The key idea of the algorithm is to arrange the hyperedges in a layered fashion such that in each layer, every hyperedge has a unique item. Then, setting the weight for unique items to the valuation of the edge and all other items to zero can extract the full revenue in a particular layer. The following theorem proves the correctness of Algorithm 1, and analyzes its performance.

THEOREM 2. *Algorithm 1 outputs a B-approximation item pricing in $O(Bm)$ time.*

PROOF. Each step the algorithm finds a minimal set cover of the remaining items, call the set cover a *layer*. On one hand, since each item presents in at most B hyperedges, there are at most B layers as at each step the degree of each item decreases by at least 1. On the other hand, each hyperedge e in a minimal set cover $\mathcal{E}'$ must contain at least one unique item that is not contained in other sets: otherwise $\mathcal{E}' \setminus \{e\}$ is still a set cover, which contradicts the minimality of $\mathcal{E}'$. Pricing these unique items at price equal to the value of corresponding sets can extract full revenue from the hyperedges in this layer. There must exist a layer such that the item pricing can achieve $\Omega(\frac{1}{B})$ of the total value of all the hyperedges, thus this item pricing algorithm has approximation ratio $O(B)$. The running time for each step is $O(m)$, and the total running time is $O(Bm)$. □

The XOS Pricing (XOS) Algorithm. The last pricing algorithm we consider is the XOS function obtained by computing the bundle price using pricing vector from LPIP , CIP and then using the higher of the two as price offered by the seller.

6. EXPERIMENTAL EVALUATION

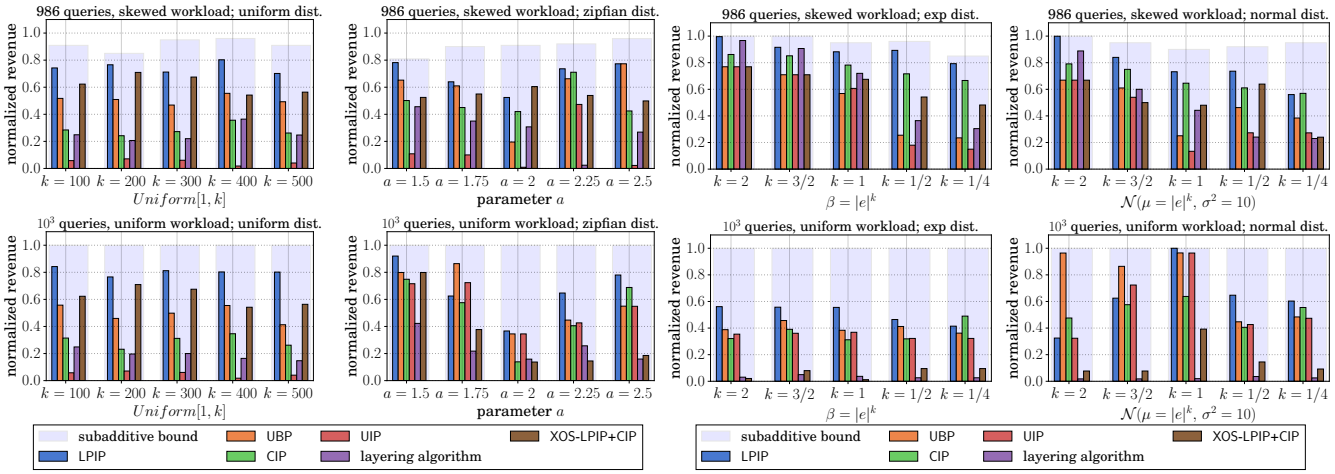
In this section, we empirically evaluate the performance of the five pricing algorithms presented in Section 5. We evaluate the performance across two measures: (i) the runtime of the algorithm, and (ii) the revenue that the algorithm can generate. All pricing algorithms run on hypergraph structures that are generated from a workload of SQL queries executed over a real-world dataset. The valuations are obtained using different generative random processes, so as to observe the algorithmic behavior under different scenarios. These generative models are motivated by studies modeling valuations for digital goods in online platforms and their pricing [44, 18, 35, 46, 48, 13, 49].

6.1 Experimental setup

We perform all our experiments on Intel Core i7 processor machine and 16 GB main memory running OS X 10.10.5. We use MySQL as the underlying database for query processing and evaluation. Our implementation is written in Python on top of the QIRANA query pricing system [30]. For all our experiments, we use Cvxpy [32] optimization toolkit for running linear programs. QIRANA generates a support set $\mathcal{S}$ by randomly sampling "neighboring" databases of the underlying database $\mathcal{D}$, *i.e.* databases from $\mathcal{I}$ that differ from $\mathcal{D}$ only in a few places. The advantage of this strategy is that it is possible to succinctly represent the support set by storing only the differences from $\mathcal{D}$, which is efficient in terms of storage. For every query bundle $\mathbf{Q}$, QIRANA computes the conflict set $\mathcal{C}_\mathcal{S}(\mathbf{Q}, \mathcal{D})$, which is the bundle (or hyperedge) that we use as input to the pricing algorithms.

Table 2 shows the design space of our experimental evaluation. Our experiments are over the world dataset, TPC-H benchmark and SSB benchmark. Each query workload will generate a hypergraph; to assign valuations over the hyperedges, we sample from different types of distributions, which we describe later in detail. We evaluate our algorithms for each instance that is generated in this fashion.

In order to compare how well our algorithms perform in terms of revenue, we use two upper bounds: (i) sum of valuations, and (ii) an upper bound on the optimal subadditive valuation. We find an upper bound on the optimal subadditive valuations by computing a linear program whose constraints encode the arbitrage constraints. Since the number of constraints can be exponential in the number of hyperedges, we optimize by greedily adding constraints for bundles with largest valuations and finding a set of bundles that cover the hyperedge with small valuations. As we will see



(a) Sampling bundle valuations

(b) Scaling bundle valuations

Figure 5: skewed and uniform workload

Table 2: Experimental Design Space

Dataset	Algorithms	Query Workload	Valuation Model
`world` dataset	UBP	uniform	sampled bundle
	UIP	skewed	scaled bundle
`SSB` benchmark	LPIP	`SSB` queries	additive bundle
	CIP	`TPC-H` queries	
`TPC-H` benchmark	Layering		

Table 3: Hypergraph Characteristics

Query Workload	# Queries (m)	Max degree (B)	Avg edge size
uniform	1000	400	5982.07
skewed	986	22	41.67
`SSB`	701	257	278.72
`TPC-H`	220	151	375.48

later, this helps us compare the performance of algorithms with respect to the subadditive bound, which can generally be much smaller than the sum of valuations. In all our experiments, we report each data point as an average over 5 runs, where we discard the first run to minimize cache latency effects on running time of the algorithms.

6.2 Workload and Dataset Characteristics

We now describe briefly the characteristics of the query workload and datasets. The first dataset we consider is `world` dataset, a popular database provided for software developers. It consists of 3 tables, which contain 5000 tuples and 21 attributes. We construct a support set of size $n = |\mathcal{S}| = 15000$. For `TPC-H` and `SSB`, we generate data for scale factor of one ($\approx$ 10 million rows) and support set of size 100000.

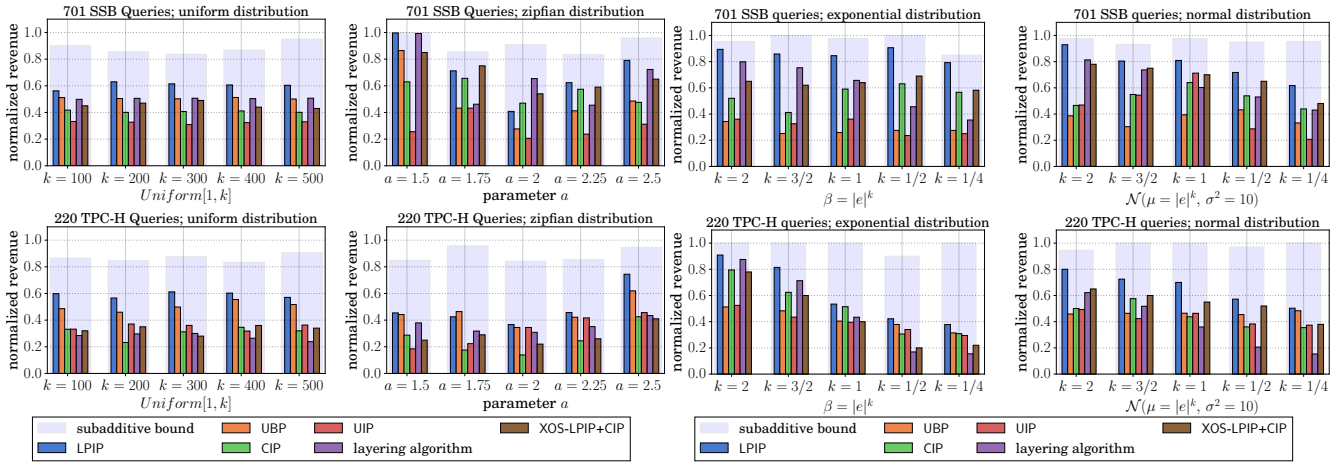
We consider four different query workloads, which create different hypergraphs that fundamentally differ in structure:

- The *skewed* query workload [36] consists of $m = 986$ SQL queries containing selection, projections and join queries with aggregation. The list of queries in this workload is presented in the full paper [23].
- The *uniform* query workload consists of only selection and projection SQL queries with the same selectivity (which means that the output of each query is about the same).
- The `SSB` query workload is generated by using the standard twelve queries as templates where we change the constants in the predicates.
- The `TPC-H` query workload is generated by using seven of the 22 queries that are supported by [30] as templates where we change the constants in the predicates. The query generation process is described in the full version of the paper [23].

Table 3 summarizes the characteristics of the two hypergraphs generated by each query workload. Both hypergraphs have the same number of vertices and hyperedges. On the other hand, their structure is very different, as can be seen in Figures 4a and 4b, which depict the distribution of the hyperedge size. For the uniform query workload, the average size of each hyperedge is around 6000, and it is normally distributed around that value. This means that there is a high overlap among the vertices of the hyperedges. For the skewed query workload, most of the hyperedges contain only a very small number of vertices, while only a few hyperedges contain a large number of vertices. Observe also that the average hyperedge size is around 40, so the hypergraph is more sparse compared to the uniform query workload. `TPC-H` and `SSB` workloads also have skew in their hyperedge distribution (Figures 4d and 4c). `SSB` workload has exactly one hyperedge with size zero and has close to half of the edges with a unique item in it. `TPC-H` workload has eleven edges with size zero but only a quarter of edges have a unique item in them.

6.3 Measuring the Revenue

We first focus on the behavior of the pricing algorithms with respect to the goal of maximizing the revenue. We will examine separately the algorithmic behavior for different structure of the valuations.



(a) Sampling bundle valuations: SSB and TPC-H

(b) Scaling bundle valuations: SSB and TPC-H

Figure 6: SSB and TPC-H workload

Sampling Bundle Valuations. In this part of the experiment, we generate valuations for every hyperedge by sampling from a parametrized distribution.

First, we sample valuations from the uniform distribution Uniform$[1, k]$ for some parameter k. Figures 5a and 6a shows the performance of all six pricing algorithms for all workloads. We should remark that the revenue plotted is normalized with respect to the sum of valuations including the subadditive upper bound. We can observe that the LPIP algorithm performs much better in all cases for both query workloads. The second best algorithm is UBP; we expect that uniform bundle pricing performs well in this case, since the size of the bundle is not correlated with the valuation in this setup. Finally, notice the huge gap between UIP and LPIP: this is an instance where both algorithms have the same worst-case guarantees, but their revenue differs by a large margin. For SSB and TPC-H workloads, layering algorithm gets close to half and a quarter of the possible revenue for uniform bundle valuations (figure 6a), in proportion to the number of edges with unique items. For all workloads, XOS pricing function obtained using the LPIP and CIP pricing vector is close to the performance of CIP.

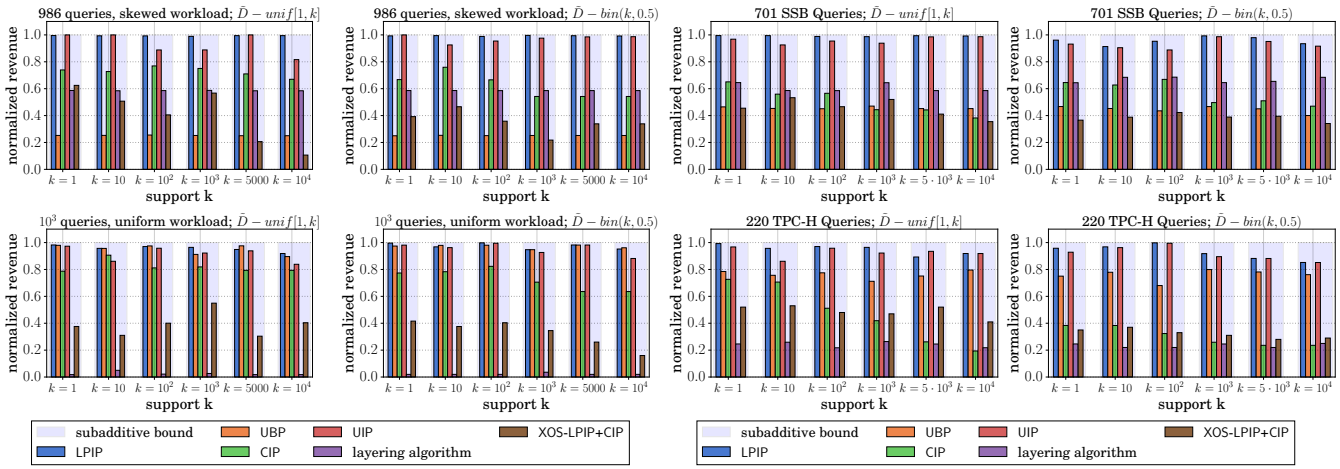
Second, we sample valuations from the zipfian distribution parametrized by a. LPIP again performs better than the other pricing algorithms, but now UBP comes a close second (and in one case performs better than LPIP).

The layering algorithm does not perform well except in the case of zipfian distribution with exponent smaller than 2. Indeed, for $a < 2$, zipfian distribution assigns a large valuation to some hyperedge that contributes significantly to the total revenue. In such cases, the layering algorithm can always extract full revenue from the layer containing high valuation edges and perform well in practice. This also explains why for SSB workload and $a = 1.5$ (figure 6a), the revenue extracted is close to 1. In that specific instance, one edge was assigned a valuation of 1428920 and the sum of all valuations was 1653537. As the zipfian exponent becomes greater than two, the spread of valuations becomes smaller and the layering algorithm performs worse.

Finally, the CIP algorithm does not perform that well, even though it is theoretically optimal. This is because going over all capacity vectors with limited supply is very expensive. In our implementation, running the linear program for a large number of capacity vectors for the uniform workload takes close to 2 hours in total (we discuss reasons for this in the next section). Thus, we reduce the number of capacity vectors that we try by increasing the $(1+\epsilon)$ parameter. This introduces a factor of $(1+\epsilon)$ in the approximation ratio but allows for the running time to be smaller. For the purpose of experiments, we set $(1+\epsilon)$ such that the running time is ~ 30 minutes. Performing this optimization allows us to complete the algorithm rather than truncating the experiment prematurely and returning the best result obtained so far. The approximation factor of CIP remains marginally inferior to LPIP (although in some cases, it outperforms LPIP) while XOS pricing is consistently worse than both LPIP and CIP.

Scaling Bundle Valuations. In the previous scenario, the valuations were sampled independently of the edge size. Our next experiment correlates the size of each hyperedge with the valuation that is assigned to it. To achieve this, we sample each valuation from the parameterized exponential and normal distribution as follows: we assign $v_e \sim$ exponential$(\beta = |e|^k)$ where β is the mean of the distribution. Similarly, for normal distribution, we assign $v_e \sim \mathcal{N}(\mu = |e|^k, \sigma^2 = 10)$. Here k is the parameter that we will vary. Figure 5b and 6b shows the revenue generated for the four query workloads and two families of distributions for different values of the parameter k.

For the skewed query workload, when $k \geq 1$, most of the revenue is concentrated in a few edges that have extremely large valuations (because there are few edges of large size). In this case, all algorithms perform very well and extract almost all of the revenue. For smaller values of k, the algorithms can extract smaller revenue, and LPIP and CIP perform best. Notice again the large revenue gap between LPIP and UIP for such values (as much as 5x). The same behavior is also observed for TPC-H and SSB . Since layering algorithm is the second best algorithm for $k > 1$, we investigated why this is the case for TPC-H. We found that all hyperedges with size greater than 500 (total of 10 as seen in figure 4c) have unique items and thus are placed in a single



(a) Sampling item prices: uniform and skewed workload

(b) Sampling item prices: SSB and TPC-H

Figure 7: Sampling item prices: all workloads

layer. Since these edges contribute the most to the revenue, layering performs close to the best. SSB workload has a more even spread of edge with unique items. Out of 36 edges with size > 1000, 22 edges have unique items.

The landscape changes for the uniform query workload. The main observation is that the layering algorithm performs extremely poorly. The revenue generated by the other four algorithms is very close, with LPIP and UBP performing the best. For the exponential distribution, all algorithms are very far from optimal. However, we believe this is not an anomaly but rather the subadditive bound not being as good as it should be.

Sampling Item Prices. The last set of experiments is to understand the behavior of the pricing algorithms when the valuation of each hyperedge is defined by an *additive generative model*. More specifically, we define k different distributions $\{D_i\}_{i=1}^{k}$ from which items will draw their prices and a special distribution $\tilde{D}$ which will assign each item which distribution it will sample from. The valuation of an edge is the defined as $v_e = \sum_{j \in e} x_j \sim D_{\ell_j}$ where $\ell_j \sim \tilde{D}$. Intuitively, this model will capture the scenario where parts of the database have non-uniform value and some parts are much more valuable than others. To see why this setting is of practical interest, consider a research analyst in banking who gives stock recommendations. While public information about companies and stocks may be cheap, the research analysts buy and sell recommendations will be of much higher value. For the purpose of experiments, we fix D_i to $\mathsf{Uniform}[i, i+1]$ and set $\tilde{D}$ to $\mathsf{Uniform}[1, k]$ or $\mathsf{Binomial}(k, 1/2)$ while varying k. Figure 7a and 7b shows the results of this experiment.

Here, LPIP outperforms all other algorithms across all workloads. For small values of k, the valuation of each hyperedge is closer to $|e|$. In this case, there is no gap between UIP and its LPIP. As the value of k increases, the gap between the two algorithms increases, since the weights of each item become less uniform.

We should remark two things that are distinct for each query workload. For the skewed query workload, UBP performs poorly, since now the valuation of each hyperedge is correlated (in an additive fashion) with the bundle structure. For the uniform query workload on the other hand, UBP does well, since the size of the edges is relatively concentrated. Finally, the layering algorithm is the worst performing out of all in the case of the uniform query workload.

For SSB and TPC-H , the first observation is that although these workloads are also skewed, UBP performs reasonably well (often beating CIP and layering for TPC-H). This is consistent with the fact that in the hyperedge distribution for TPC-H , 150 hyperedges have a size of $\sim$ 400. Thus, for smaller values of k, UBP is expected to perform well. SSB hyperedge distribution is more spread out compared to TPC-H which explains why UBP is not as well performing. We go one step further to see if a post-processing step can refine UBP prices to boost the revenue even more. To do so, we find the best item prices via a linear program where the constraints sell all edges sold by uniform bundle price that achieves the maximum revenue for $k = 1, \mathsf{Uniform}[1, k]$ in TPC-H workload. We observed that this simple step (runs in $\sim 1s$) improves the revenue from 0.78 to 0.99. Layering algorithm again performs the worst for TPC-H. This happens because none of the 150 hyperedges with size $\sim$ 400 contains a unique item. Thus, although valuation of each edge is correlated with $|e|$, the total revenue in the hyperedges with large size is not significantly more as compared to when the valuation is proportional to (say) $|e|^2$ as in the case of experiments in figure 6b. Layering performs better for SSB as the edges with unique items are more evenly spread (as noted before). Thus, layering algorithm is able to extract more revenue from the large size hyperedges as compared to TPC-H. Perhaps the most interesting observation is that XOS pricing function performs significantly worse than best of the two. This happens because the max function assigns prices to bundle that overshoots the $v_{\mathbf{Q}}$ leading to lower revenue.

6.4 Measuring the Runtime

In this section, we discuss the running time of the algorithms. Table 4 shows the runtime of all algorithms [4]. The most time efficient algorithms are uniform bundle pricing,

[4]We skip XOS in this section as it is dependent on both LPIP and CIP making it very expensive.

Table 4: Algorithm running times (in seconds) for different workloads.

Query Workload	LPIP	UBP	UIP	CIP	Layering
skewed	60.62	< 1	25.45	812.67	15.67
uniform	95.81	< 1	29.82	1800	50.19
SSB	$1300 + 3600$	< 1	$1300 + 13$	7200	$1300 + 32$
TPC-H	$2000 + 1900$	< 1	$2000 + 13$	7200	$2000 + 4$

Table 5: Algorithm running times (in seconds) for skewed workload (including hypergraph construction time)

Support Set Size	LPIP	UBP	UIP	CIP	Layering
$\lvert\mathcal{S}\rvert = 100$	< 1	< 1	< 1	< 1	< 1
$\lvert\mathcal{S}\rvert = 500$	6.16	< 1	5.25	6.87	1.6
$\lvert\mathcal{S}\rvert = 1000$	15.10	< 1	17.43	29.82	3.12
$\lvert\mathcal{S}\rvert = 5000$	30.12	< 1	29.82	189.97	8.78
$\lvert\mathcal{S}\rvert = 15000$	70.42	< 1	35.21	676.23	12.34

uniform item pricing and the layering algorithm. Uniform bundle pricing and uniform item pricing depend only on number of hyperedges and the number of items in the hypergraph. Thus, they are very fast to run in practice. Note that for all item pricing algorithms, we also include the time taken to compute the conflict set of the query. However, for uniform bundle pricing, we need not take that into account as it is independent of the conflict set. For skewed and uniform workload, the layering algorithm is slightly slower but comparable in performance. Note that the layering is faster on the skewed query workload as compared to the uniform query workload, since the maximum degree B is much smaller. Note that since the support set and the underlying database is much larger for SSB and TPC-H , running time to construct conflict set is also large ($\sim 1300s$ and $\sim 2000s$ respectively in table 4).

The two slowest running algorithms are LPIP and CIP as they require running multiple linear programs. In practice, LPIP is faster than CIP. This is because the size of the linear program is very different. In our setting, the number of edges $220 \leq m \leq 1000$ is much smaller than the number of items $n = 15000(100000)$. LPIP has at most one constraint per bundle (thus, at most m constraints) but CIP has one constraint per item (n constraints in total). This dramatically influences the running time of the two algorithms. CIP uses $(1 + \epsilon)$ as a parameter, where ϵ controls the limited supply available for each item. We adjust the value of ϵ for both workloads to ensure that the running time is at most 30 minutes. We fix $\epsilon = 0.2$ for the skewed workload and $\epsilon = 4$ for the uniform workload based on our empirical observations. For TPC-H and SSB experiments, CIP did not run to completion for values of $\epsilon \leq 0.5$ since the itemset here is much larger. In this case, we fix a value of $\epsilon = 3$ to limit the running time to a total of 2 hours. LPIP still remains the best performing algorithm for uniform and zipfian distribution.

6.5 Impact of Support Set Size

Our final set of experiments is to understand the impact of support set size, *i.e.*, the number of items n. Given a hypergraph instance, adding more items to the hypergraph is an interesting proposition since it can only increase the revenue. However, this also comes at a higher cost of running time. On the other hand, too few items in the hypergraph

Table 6: Algorithm running times (in seconds) for SSB workload (excluding hypergraph construction time)

Support Set Size	LPIP	UBP	UIP	CIP	Layering
$\lvert\mathcal{S}\rvert = 1000$	180.29	< 1	< 1	3.55	< 1
$\lvert\mathcal{S}\rvert = 5000$	363.97	< 1	2.85	21.43	< 1
$\lvert\mathcal{S}\rvert = 10000$	709.10	< 1	5.12	50.66	< 1
$\lvert\mathcal{S}\rvert = 50000$	2500	< 1	11.24	692.97	6.2
$\lvert\mathcal{S}\rvert = 100000$	3600	< 1	13.21	7200	32.58

can lead to suboptimal revenue for most algorithms (except uniform bundle pricing, since it is independent of the items). Figure 8a demonstrates the impact of changing the support set size on the revenue extracted. Unsurprisingly, uniform bundle pricing is not impacted by the support set size. As the support set size decreases for 15000 to 100, the performance of all item pricing algorithms decreases. Finally, Table 5 depicts the running time of all algorithms as a function of support size. The key takeaway is that running time is dependent on the support set size. Thus, the right tradeoff depends on the data seller requirements. It remains an interesting open problem to design algorithms for choosing the items in a smarter way. This will ensure that we can get good revenue guarantees without sacrificing running time. For instance, if we can create the support set in such a way that every hyperedge contains a unique item, then we can extract the full revenue from the buyers.

Figure 8b shows the same trend in revenue drop as the support set shrinks. The running time of the algorithms is more interesting for SSB. Observe the steep decrease in running time of CIP as we go from support size 100000 to 50000 (table 6). This drop happens because of two reasons: (i) since CIP contains one linear program *per* item, halving the support size reduces the number of linear programs by the same factor. (ii) as the number of items decrease, the maximum degree of hypergraph (i.e B) also decreases.

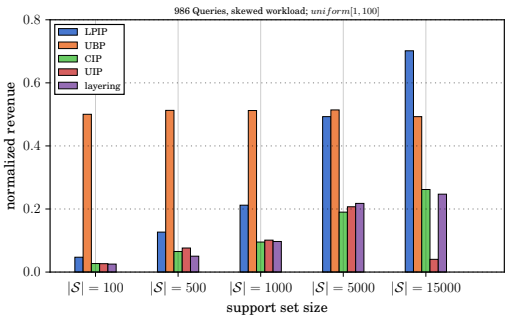
7. KEY TAKEAWAYS

The empirical study has brought forward many insights. Below, we summarize some of the most important lessons from our studies and simple rules of thumb that a data broker should follow.

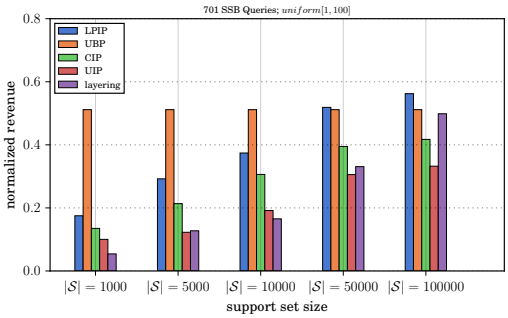
7.1 Lessons Learned

Choice of algorithm. The right choice of algorithm depends on the revenue guarantees desired by the broker, running time constraints imposed and the knowledge about query instances that need to be priced. Throughout our experiments, CIP has been the worst performing algorithm followed by layering algorithm (except for zipfian distribution on skewed workloads where it was either the best or second best) while LPIP has consistently outperformed all other choices. Thus, if the broker does not have any running time constraints, LPIP is the best pick. However, since LPIP can also be expensive (as seen for SSB and TPC-H), the better of layering and uniform bundle pricing gives the best empirical revenue guarantee.

Hypergraph structure. Knowledge about the structure of the hypergraph is crucial in predicting the performance of the algorithm. For instance, if a non-negligible fraction of the hyperedges have size zero (TPC-H workload) or have



(a) skewed workload



(b) SSB workload

Figure 8: Revenue generated with increasing itemset size

similar edge sizes (uniform query workload), then UBP performs very well. Similarly, if a large fraction of hyperedges contain a unique items, the layering algorithm and LPIP perform well.

Scalability challenges. Our experiments indicate a wide gap between the theoretical guarantees of the algorithms and their practical utility. Specifically, LP based algorithms suffer from scalability issues as the instance size grows. For LPIP there are m constraints in total while CIP contains n constraints (recall that $m \ll n$ in our setting). LPIP works well for small values of m and n. As m grows, LPIP also starts suffering from scalability issues. On the other hand, UIP, UBP and layering algorithms are both time and memory efficient.

Valuation distribution. Assumptions about valuation distribution for bundles and correlation with bundle size is also a key indicator of extracted revenue. For all query workloads, whenever the revenue is concentrated in a few edges (zipfian distribution with $a < 2$, scaled valuations and additive model), LPIP and UIP perform well. We perform experiments with different distributions and valuation models to expose the strength and weaknesses of each algorithm. However, to the best of our knowledge, there is no existing user study or datasets available that can help us validate assumptions about how buyers value different queries in the context of data pricing.

7.2 Future Work

Our empirical study has also given us many hints for future directions. We find the following tasks particularly urgent and engaging.

Choosing support set. Recall from Section 3 that our data pricing framework depends on a chosen set $\mathcal{I}$ of possible database instances. This choice is controlled entirely by the seller. However, a careful selection of these databases can affect the hypergraph structure. For instance, if there is a way to choose the items such that most hyperedges will have a unique item, then the pricing becomes significantly easier. More formally, we propose the following problem: Given a set of queries $Q_1, \ldots Q_m$, database $\mathcal{D}$, does there exist a set of databases $D_1, \ldots, D_m$ such that $Q_i(D_i) \neq Q_i(\mathcal{D})$ but $Q_i(D_j) = Q_i(\mathcal{D}), i \neq j$. Our goal is to study the data complexity of the problem and identify query fragments which admit efficient algorithms. A related variant of the problem is when the broker decides to fix the query templates for the buyers. Since the set of possible queries is restricted, the hypergraph structure can be controlled carefully to make pricing more amenable.

Learning buyer valuations. This work assumes that queries and valuations are available apriori allowing for preprocessing where we can run the revenue maximizing algorithms to identify the best pricing vector. It is also worthwhile to investigate how we can learn the prices on-the-fly. In the online setting, queries arrive and the marketplace has to dynamically vary the prices based on whether the query was bought by the buyer or not. We plan to investigate how bandit algorithms and gradient descent algorithms perform when all buyers have a fixed valuation that is unknown to the algorithm. Note that the online pricing problem requires a new model of arbitrage freeness, where the temporal aspect of the problem is also taken into account.

Maximizing revenue. From a mechanism design perspective, several interesting problems remain open. First, it remains unknown what is the gap between optimal subadditive revenue and XOSpricing functions with non-constant number of additive components. The complexity of finding the optimal item prices over graphs under the additive model where each item draws its value from a distribution and the complexity of item pricing over hypergraphs with specific structure (e.g. trees) also remains open.

User study. Finally, we believe it is worthwhile to perform a user study in order to understand how buyers interact in the data market, what queries are of interest, and how they are valued.

8. CONCLUSION

In this paper, we study the problem of revenue maximization in the context of query-based pricing. We cast the task as a bundle pricing problem for single-minded buyers and unlimited supply, and then perform a detailed experimental evaluation on the effectiveness of various approximation algorithms that provide different worst-case approximation guarantees. Our results show that the specific bundle structure often means that simple item-pricing algorithms perform much better than their worst-case guarantees. There are several interesting open questions in this space. We believe that making progress on these questions is an important step, likely to create significant impact in practice.

Acknowledgements. Shuchi Chawla and Yifeng Teng were supported in part by NSF grants CCF−1617505 and CCF−1704117. Paraschos Koutris and Shaleen Deep were supported by funding from University of Wisconsin-Madison Office of the Vice Chancellor for Research and Graduate Education and Wisconsin Alumni Research Foundation.

9. REFERENCES

[1] Amazon Redshift pricing. https://aws.amazon.com/redshift/pricing/.

[2] Banjo. ban.jo.

[3] Big Data Exchange. www.bigdataexchange.com.

[4] Bloomberg Market Data. www.bloomberg.com/enterprise/content-data/market-data.

[5] DataFinder. datafinder.com.

[6] Google BigQuery pricing. https://cloud.google.com/bigquery/pricing.

[7] Lattice Data Inc. lattice.io.

[8] QLik Data Market. www.qlik.com/us/products/qlik-data-market.

[9] Salesforce. https://www.salesforce.com/products/data/overview/.

[10] Twitter GNIP Audience API. gnip.com/insights/audience.

[11] M. Babaioff, N. Immorlica, B. Lucier, and S. M. Weinberg. A simple and approximately optimal mechanism for an additive buyer. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 21–30. IEEE, 2014.

[12] M. Babaioff, R. Kleinberg, and R. Paes Leme. Optimal mechanisms for selling information. In *Proceedings of the 13th ACM Conference on Electronic Commerce*, EC '12, pages 92–109, New York, NY, USA, 2012. ACM.

[13] Y. Bakos and E. Brynjolfsson. Bundling information goods: Pricing, profits, and efficiency. *Management science*, 45(12):1613–1630, 1999.

[14] M. Balazinska, B. Howe, and D. Suciu. Data markets in the cloud: An opportunity for the database community. *PVLDB*, 4(12):1482–1485, 2011.

[15] M.-F. Balcan and A. Blum. Approximation algorithms and online mechanisms for item pricing. In *Proceedings of the 7th ACM Conference on Electronic Commerce*, pages 29–35. ACM, 2006.

[16] D. Bergemann and A. Bonatti. Selling cookies. *American Economic Journal: Microeconomics*, 7(3):259–94, August 2015.

[17] D. Bergemann, A. Bonatti, and A. Smolin. The design and price of information. *American Economic Review*, 108(1):1–48, 2018.

[18] S. Bhattacharjee, R. D. Gopal, K. Lertwachara, and J. R. Marsden. Economic of online music. In *Proceedings of the 5th international conference on Electronic commerce*, pages 300–309. ACM, 2003.

[19] P. Briest and P. Krysta. Single-minded unlimited supply pricing on sparse instances. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 1093–1102. Society for Industrial and Applied Mathematics, 2006.

[20] Y. Cai, N. R. Devanur, and S. M. Weinberg. A duality based unified approach to bayesian mechanism design. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 926–939. ACM, 2016.

[21] Y. Cai and M. Zhao. Simple mechanisms for subadditive buyers via duality. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Teory of Computing (STOC 2017)*, pages 170–183. ACM, 2017.

[22] P. Chalermsook, B. Laekhanukit, and D. Nanongkai. Independent set, induced matching, and pricing: Connections and tight (subexponential time) approximation hardnesses. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 370–379. IEEE, 2013.

[23] S. Chawla, S. Deep, P. Koutris, and Y. Teng. Revenue Maximization for Query Pricing. *arXiv e-prints*. https://arxiv.org/pdf/1909.00845.pdf.

[24] S. Chawla, J. D. Hartline, and R. Kleinberg. Algorithmic pricing via virtual valuations. In *Proceedings of the 8th ACM conference on Electronic commerce*, pages 243–251. ACM, 2007.

[25] S. Chawla, J. D. Hartline, D. L. Malec, and B. Sivan. Multi-parameter mechanism design and sequential posted pricing. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 311–320. ACM, 2010.

[26] S. Chawla, D. Malec, and B. Sivan. The power of randomness in bayesian optimal mechanism design. *Games and Economic Behavior*, 91:297–317, 2015.

[27] S. Chawla and J. B. Miller. Mechanism design for subadditive agents via an ex ante relaxation. In *Proceedings of the 2016 ACM Conference on Economics and Computation*, pages 579–596. ACM, 2016.

[28] M. Cheung and C. Swamy. Approximation algorithms for single-minded envy-free profit-maximization problems with limited supply. In *Foundations of Computer Science, 2008. FOCS'08. IEEE 49th Annual IEEE Symposium on*, pages 35–44. IEEE, 2008.

[29] S. Deep and P. Koutris. The design of arbitrage-free data pricing schemes. In *20th International Conference on Database Theory (ICDT 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[30] S. Deep and P. Koutris. QIRANA: A framework for scalable query pricing. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 699–713. ACM, 2017.

[31] S. Deep, P. Koutris, and Y. Bidasaria. Qirana demonstration: real time scalable query pricing. *Proceedings of the VLDB Endowment*, 10(12):1949–1952, 2017.

[32] S. Diamond and S. Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[33] J. Gans, S. King, and N. G. Mankiw. *Principles of microeconomics*. Cengage Learning, 2011.

[34] V. Guruswami, J. D. Hartline, A. R. Karlin, D. Kempe, C. Kenyon, and F. McSherry. On profit-maximizing envy-free pricing. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1164–1173. Society for Industrial and Applied Mathematics, 2005.

[35] J. Hartline, V. Mirrokni, and M. Sundararajan. Optimal marketing strategies over social networks. In *Proceedings of the 17th international conference on World Wide Web*, pages 189–198. ACM, 2008.

[36] F. Henglein and K. F. Larsen. Generic multiset programming for language-integrated querying. In *ICFP-WGP*, pages 49–60, 2010.

[37] P. Koutris, P. Upadhyaya, M. Balazinska, B. Howe, and D. Suciu. Query-based data pricing. In M. Benedikt, M. Krötzsch, and M. Lenzerini, editors, *PODS*, pages 167–178. ACM, 2012.

[38] P. Koutris, P. Upadhyaya, M. Balazinska, B. Howe, and D. Suciu. Querymarket demonstration: Pricing for online data markets. *PVLDB*, 5(12):1962–1965, 2012.

[39] P. Koutris, P. Upadhyaya, M. Balazinska, B. Howe, and D. Suciu. Toward practical query pricing with querymarket. In K. A. Ross, D. Srivastava, and D. Papadias, editors, *ACMSIGMOD 2013*, pages 613–624. ACM, 2013.

[40] C. Li and G. Miklau. Pricing aggregate queries in a data marketplace. In *WebDB*, 2012.

[41] B. Lin and D. Kifer. On arbitrage-free pricing for general data queries. *PVLDB*, 7(9):757–768, 2014.

[42] A. Muschalle, F. Stahl, A. Löser, and G. Vossen. Pricing approaches for data markets. In *International Workshop on Business Intelligence for the Real-Time Enterprise*, pages 129–144. Springer, 2012.

[43] R. B. Myerson. Optimal auction design. *Mathematics of operations research*, 6(1):58–73, 1981.

[44] M. Naldi and G. DAcquisto. Performance of the vickrey auction for digital goods under various bid distributions. *Performance Evaluation*, 65(1):10–31, 2008.

[45] A. Rubinstein and S. M. Weinberg. Simple mechanisms for a subadditive buyer and applications to revenue monotonicity. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation*, pages 377–394. ACM, 2015.

[46] B. Shiller and J. Waldfogel. Music for a song: an empirical look at uniform pricing and its alternatives. *The Journal of Industrial Economics*, 59(4):630–660, 2011.

[47] P. Upadhyaya, M. Balazinska, and D. Suciu. Price-optimal querying with data apis. *PVLDB*, 9(14):1695–1706, 2016.

[48] Z. Zheng, Y. Peng, F. Wu, S. Tang, and G. Chen. An online pricing mechanism for mobile crowdsensing data markets. In *Proceedings of the 18th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, page 26. ACM, 2017.

[49] Z. Zheng, Y. Peng, F. Wu, S. Tang, and G. Chen. Trading data in the crowd: Profit-driven data acquisition for mobile crowdsensing. *IEEE Journal on Selected Areas in Communications*, 35(2):486–501, 2017.