Pricing Aggregate Queries in a Data Marketplace

Chao Li
University of Massachusetts
Amherst, Massachusetts, USA
chaoli@cs.umass.edu

Gerome Miklau
University of Massachusetts
Amherst, Massachusetts, USA
miklau@cs.umass.edu

ABSTRACT

In a data marketplace, producers of data provide query answers to consumers in exchange for payment. A market for data allows capital to flow from data consumers to data producers. This helps to finance the costs of data collection and dissemination, thereby increasing the availability of data throughout society. While nascent data markets exist on the Web, they are currently limited in the pricing mechanisms and interactions they support. In this paper we propose new criteria for interactive pricing in a data market: price functions should be non-disclosive, arbitrage-free, and regret-free. We study the structure of price functions meeting these criteria, and distinguish between NP-complete and PTIME cases for computing a conforming price function.

1. INTRODUCTION

Information, in the form of structured data, is increasingly considered a critical resource. It has value to a broad range of *data consumers* who derive that value by analyzing it to make business decisions, by using it as the basis of marketed services, or by transforming and reselling it. The creation or extraction of data is usually not free. Its cost is borne by a *data producer*, and can include the cost of raw collection, data cleaning, transformation, storage, and/or transmission. Examples include: the cost of sequencing to produce genomic data; the cost of a sensor infrastructure to produce meteorological data; the cost of surveying to produce census data; even the cost of crawling the Web to produce an index that is the basis of search query results.

In a data marketplace, producers of data provide query answers to consumers in exchange for payment. A Web-based data market offers a new avenue for financing data production by allowing capital to flow efficiently from interested data consumers to willing data producers. This can increase the availability of data because an active market can provide incentives for new forms of data production where there were none before. In addition, financial compensation for producers can help to overcome common obstacles to data sharing such as competitiveness concerns or privacy risks.

Markets for data are emerging on the Web, typically allowing consumers to pay for bulk downloads of data or for calls to the producer's API, which return query answers. Windows Azure Marketplace [1] and Infochimps [4] are two data marketplaces that sell data from many smaller data producers. In addition, a number of recent start-ups are focused on the paid exchange of data. For example, Factual [3] focuses on location-based data while Singly [2] and Personal [5] focus on the paid exchange of personal data.

These nascent data markets remain limited in the pricing methods and interactions they allow. In particular, current marketplaces typically sell only wholesale subscriptions to a data set or coarse-grained all-or-nothing access. In response to this limitation (among others) Balazinska, Howe, and Suciu [7] recently described a number of compelling research challenges for realizing the promise of data markets. In this paper we pursue one of these: fine-grained query pricing.

We consider a data producer who sells answers to aggregate queries. Consumers request the price of individual queries, or sets of queries, from the producer prior to purchase. We allow interaction between the producer and consumer: after paying for a set of queries, and receiving their answers, the consumer may choose to purchase additional queries. We focus on basic aggregate queries (including count, sum, and average queries with arbitrary selection conditions) evaluated on relational data.

Pricing data in a market setting is complex and sometimes counter-intuitive. We investigate the (data producer's) problem of efficiently and soundly setting prices through the specification of a pricing function for queries. Our goal is to devise price functions that satisfy three key properties. First, the price function should be non-disclosive, so that it is not possible to infer unpaid query answers by analyzing the publicly available prices of queries. Second, the price function should be arbitrage-free, so that a consumer cannot combine queries to achieve a price lower than intended. Third, the price function should (ideally) be regret-free, so that the price of asking a sequence of queries in multiple interactions is not higher than asking them all-at-once.

We provide algorithms and complexity results for the computational problems that are at the core of pricing mechanisms for aggregate queries. In some cases, we find that computing sound prices for queries is intractable, but we also show important cases where price functions can be computed efficiently. In addition, we show a surprising result about the structure of any price function that simultaneously satisfies all three properties.

2. PRICING AGGREGATE QUERIES

In this section we state assumptions, describe our class of aggregate queries, define a price function, and propose key properties a pricing function should satisfy.

2.1 Assumptions

We assume throughout the paper that the database instance is fixed. Therefore all queries are answered with respect to a single instance and we do not consider releasing revisions to released query answers. Further, we ignore risks posed by colluding users. Clearly Alice could pay for a query answer, share it with Bob, and they could divide the cost. We assume users are trusted not to subvert the pricing mechanism in this way, or prevented from doing so by other means. Finally, because we consider a class of common aggregate queries, we treat the computational cost of queries as negligible compared to the inherent value of query answers.

2.2 Linear Aggregation Queries

We consider a general class of *linear aggregation queries* over a single relation. Included in this class are queries that count the number of tuples satisfying an arbitrary predicate, as well as sums and averages over discrete numerical attributes. This class can therefore support basic SQL aggregates SUM-AVG-COUNT, group-by queries, data cube queries, and it can be used to compute histograms, marginals, or the sufficient statistics for more complex tasks like decision trees or log-linear models.

Our queries are *linear* because they can be represented as a linear combination of cell counts which are derived from the database instance. Each linear query returns a single numeric value and is described by a vector of coefficients. Accordingly a set of queries can be represented as a matrix. This query representation is critical to achieving our results. The coefficients show precisely the parts of the database on which the query results depend. Further, the techniques of linear algebra allow us to precisely define the queries that are computable from a purchased query set and, as we will see, provide a foundation for the analysis of price functions.

Let the database I be an instance of a single-relation schema $R(\mathbb{A})$, with attributes $\mathbb{A} = \{A_1, A_2, \ldots, A_k\}$. The crossproduct of the attribute domains, written $dom(\mathbb{A})$, is the set of all possible tuples that may occur in I.

In order to formulate linear queries, we first transform the instance I into a *data vector* $\mathbf{x}$ of cell counts. To formally define the data vector we associate, with each element x_i of $\mathbf{x}$, a Boolean *cell condition* ϕ_i, which evaluates to True or False for any tuple in $dom(\mathbb{A})$. The cell conditions are pairwise unsatisfiable: any tuple in $dom(\mathbb{A})$ satisfies exactly one cell condition. Then x_i is defined to be the count of the tuples from I which satisfy ϕ_i.

DEFINITION 1 (DATA VECTOR). *Given an ordered list of cell conditions $\phi_1, \phi_2 \ldots \phi_n$ the data vector $\mathbf{x}$ is a length-n column vector defined by n positive integral counts $x_i = |\{t \in I \mid \phi_i(t) \text{ is True}\}|$.*

We could represent instance I by defining the vector $\mathbf{x}$ with one cell for every element of $dom(\mathbb{A})$. In this case, $\mathbf{x}$ is a bit vector of size $|dom(\mathbb{A})|$ with nonzero counts for every tuple in I. It is often possible to represent a set of queries with fewer cell counts. For example, a common way to form a vector of base counts over larger cells is to partition each $dom(A_i)$ into d_i regions (which could correspond to ranges over an ordered domain or individual elements in a categorical domain). Then we define individual cells by taking the cross-product of the regions in each attribute. Ultimately, the choice of cell conditions is determined by the set of queries requested by users. In the sequel, we assume a fixed set of n cell conditions.

EXAMPLE 1. *Consider the relational schema $R(name, industry, cost, rank)$ describing financial assessments of firms in various industries. If we wish to form queries only over cost (high, $> 1M$ or low, $\leq 1M$), and ranges of the rank attribute $[1, 10]$, $[11, 100]$, $[101, 500]$, $[501, +\infty]$, then we can define the 8 cell conditions enumerated in Figure 1(a).*

A linear aggregation query computes a specified linear combination of the elements of the data vector $\mathbf{x}$.

DEFINITION 2 (LINEAR QUERY). *A* linear query *is a length-n row vector $\mathbf{q} = [q_1 \ldots q_n]$ with each $q_i \in \mathbb{R}$. The answer to a linear query $\mathbf{q}$ on $\mathbf{x}$ is the vector product $\mathbf{qx} = q_1x_1 + \cdots + q_nx_n$. We use $\mathcal{Q}$ to denote the set of all linear queries.*

Using only coefficients of 0 or 1, a linear query can express any predicate counting query, counting the number of tuples satisfying any disjunction of the cell conditions defining $\mathbf{x}$.

Note that data cubes, histograms, and group-by queries, which consist of multiple numerical results, will be represented as a set of linear queries. A *query set* is represented as a matrix where each row is a single linear query.

DEFINITION 3 (QUERY MATRIX). *A* query matrix *is a collection of m linear queries, arranged by rows to form an $m \times n$ matrix.*

If $\mathbf{Q}$ is an $m \times n$ query matrix, the answer to $\mathbf{Q}$ is a length m column vector of numerical query results, which can be computed as the matrix product $\mathbf{Qx}$. To simplify notation, we use $\mathbf{Q}$ to denote both a set of queries and its representation as a matrix, and similarly for a query $\mathbf{q}$. Accordingly, we use $\mathbf{Q}_1 \cup \mathbf{Q}_2$ to denote the union of two query sets.

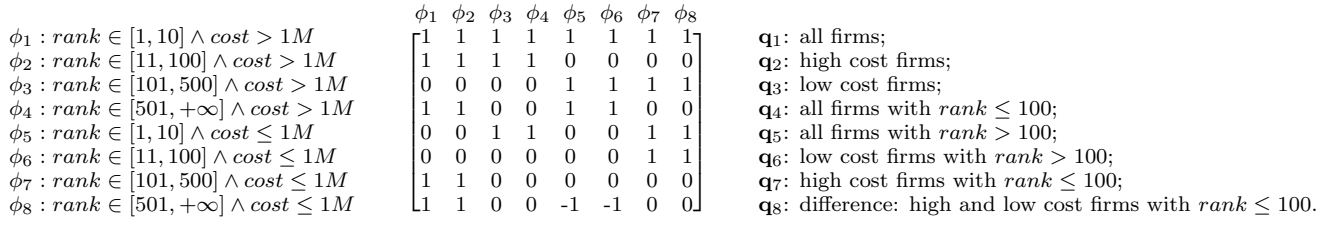
EXAMPLE 2. *Figure 1(b) shows a query matrix representing a query set of 8 linear queries. Figure 1(c) describes the meaning of the queries w.r.t. the cell conditions in Figure 1(a).*

Representing queries in this manner is common in the privacy literature [8, 14, 13] and has also appeared in the context of provenance for aggregates [6].

2.3 Interactive Query Pricing

One objective of fine-grained query pricing is to allow users to pay for just the information they wish to acquire from a database. However users often have imperfect knowledge of the queries of interest to them or the aggregate statistics that are sufficient for carrying out their task. Therefore, in practice, it is important to permit a user to interact with the data owner: to purchase queries, to receive answers, and perhaps to purchase further queries.

Our definition of a query pricing function incorporates an extension allowing for *conditional pricing* in which the price of query set $\mathbf{Q}$ can depend on the queries for which a consumer has already paid. Below, $\mathcal{P}(\mathcal{Q})$ is the set of all linear query sets.



(a) Cell conditions Φ (b) A query set $\mathbf{Q}$ (c) Counting queries defined by rows of $\mathbf{Q}$

Figure 1: For schema $R(name, industry, cost, rank)$, (a) shows 8 cell conditions on attributes *cost* and *rank*. The database vector x (not shown) will accordingly consist of 8 counts; (b) shows a sample query set Q consisting of 8 counting queries, each described informally in (c).

Definition 4 (Conditional price function). *A price function p is a function $p : \mathcal{P}(\mathcal{Q}) \times \mathcal{P}(\mathcal{Q}) \to \mathbb{R}^+ \cup \{0\}$. A consumer who has previously paid for query set $\mathbf{Q}_1$ can purchase query set $\mathbf{Q}_2$ for price $p(\mathbf{Q}_2|\mathbf{Q}_1)$.*

A consumer who has not purchased any queries must pay $p(\mathbf{Q}|\emptyset)$ for query set $\mathbf{Q}$, which we abbreviate simply as $p(\mathbf{Q})$. We assume that every query that provides some information about the database has a positive cost. This means $p(\mathbf{Q}_2|\mathbf{Q}_1) = 0$ if and only if the answer to all queries of $\mathbf{Q}_2$ can be derived using the answer to queries of $\mathbf{Q}_1$. In particular, $p(\mathbf{Q}) = 0$ if and only if $\mathbf{Q} = \mathbf{0}$ (i.e. $\mathbf{Q}$ contains only the query with all zero coefficients).

2.4 Properties of the Price Function

Next we define *non-disclosive*, *arbitrage-free*, and *regret-free* price functions. We always require price functions to be non-disclosive and arbitrage-free, but we will sometimes consider forgoing the regret-free property.

Non-disclosive pricing

In a marketplace, users are typically permitted to inquire freely about the prices of goods before their purchase. We therefore assume the pricing function (and its key properties) are public information. A non-disclosive pricing function does not allow the user to deduce query answers through inspection of the values of the price function.

Notice that the price function in Definition 4 is independent of the database instance. Because the price of queries does not change for different instances, inspection of the price function cannot reveal information about the database. Such functions assign a price to a query based on the meaning of the query and never based on the current answer. For example, if the price of query $\mathbf{q}_4$: Count(all firms with $rank \leq 100$) is less than $\mathbf{q}_5$: Count(all firms with $rank > 100$) it should reflect the relative value of statistics about low rank firms and high rank firms, regardless of the magnitude of the query answers for any particular instance. In particular, the pricing relationship holds even if the answer to $\mathbf{q}_5$ is zero. It follows that consumers must pay even for queries that reveal that there are no tuples of a certain type in the database.

Arbitrage-free pricing

Arbitrage is possible when equivalent assets are available for two different prices. The nature of information products makes arbitrage particularly challenging to avoid. When an information product is used as the input to a computation, it is not consumed—it remains for use in other computations. (This is in contrast to, for example, physical products which can serve only once as inputs to a manufacturing process.)

In our setting this means that a collection of purchased query answers can be combined or post-processed in order to derive many other query answers not explicitly purchased. We denote by $\text{span}(\mathbf{Q})$ the set of all aggregate queries that can be precisely derived from the answers to queries in $\mathbf{Q}$. Because we consider linear queries, the span of $\mathbf{Q}$ is simply the set of all queries that can be expressed as a linear combination of the queries of $\mathbf{Q}$.

Definition 5 (Span of a query set). *Given a set of linear queries $\mathbf{Q} = \{\mathbf{q}_1 \dots \mathbf{q}_m\}$,* $\text{span}(\mathbf{Q}) = \{c_1\mathbf{q}_1 + \dots + c_m\mathbf{q}_m \mid c_1 \dots c_m \in \mathbb{R}\}$

Example 3. *Consider the queries in Figure 1. Let query set $\mathbf{Q}' = \{\mathbf{q}_2, \mathbf{q}_3, \mathbf{q}_4\}$. Notice that $\mathbf{q}_1$ can be expressed as $\mathbf{q}_2 + \mathbf{q}_3$ and that $\mathbf{q}_5$ can be expressed as $\mathbf{q}_2 + \mathbf{q}_3 - \mathbf{q}_4$. Thus $\mathbf{q}_1, \mathbf{q}_5 \in \text{span}(\mathbf{Q}')$.*

Noticing that there are distinct query sets with the same span as $\mathbf{Q}$, we are also interested in the smallest size among those sets.

Definition 6 (Rank of a query set). *Given a set of linear queries $\mathbf{Q}$, the rank of query set $\mathbf{Q}$, denoted as* $\text{rank}(\mathbf{Q})$*, is the size of the smallest query set whose span is* $\text{span}(\mathbf{Q})$.

In order to avoid arbitrage, the price of a collection of queries $\mathbf{Q}$ must reflect the value of all queries in the span of $\mathbf{Q}$. More precisely, if the span of $\mathbf{Q}_1$ is contained in the span of $\mathbf{Q}_2$, then $\mathbf{Q}_2$ should never have a lower price than $\mathbf{Q}_1$. Conditional query pricing adds another potential arbitrage opportunity. The consumer should not be able to achieve a pricing advantage by decomposing a set of queries and asking each individually in separate interactions. We include both of these requirements in our definition of an arbitrage-free pricing function:

Property 1 (Arbitrage-free pricing). *A pricing function p is arbitrage-free if, for any query sets $\mathbf{Q}$, $\mathbf{Q}_1$, and $\mathbf{Q}_2$, the following hold:*

(i) If $\text{span}(\mathbf{Q}_1 \cup \mathbf{Q}) \subseteq \text{span}(\mathbf{Q}_2 \cup \mathbf{Q})$, $p(\mathbf{Q}_1|\mathbf{Q}) \leq p(\mathbf{Q}_2|\mathbf{Q})$.

(ii) $p(\mathbf{Q}_2) \leq p(\mathbf{Q}_2|\mathbf{Q}_1) + p(\mathbf{Q}_1)$.

Example 4. *Referring to Figure 1, let $\mathbf{Q}_1 = \{\mathbf{q}_1, \mathbf{q}_5\}$ and $\mathbf{Q}_2 = \{\mathbf{q}_2, \mathbf{q}_3\}$. If $p(\mathbf{Q}_1|\{\mathbf{q}_4\}) > p(\mathbf{Q}_2|\{\mathbf{q}_4\})$, a user interested in $\mathbf{Q}_1$ can ask $\mathbf{Q}_2$ with a lower cost and derive the answers to all queries in $\mathbf{Q}_1$ using a linear combination of $\mathbf{q}_2, \mathbf{q}_3, \mathbf{q}_4$.*

Arbitrage-free pricing has been considered previously in the context of data markets [7, 12]. The definition above extends prior notions to include conditional pricing and differs from [12] because it is based on data independent pricing. In particular, $p(\mathbf{Q}_1) < p(\mathbf{Q}_2)$ only violates the arbitrage-free condition if the answer to $\mathbf{Q}_1$ implies the answer to $\mathbf{Q}_2$ for all databases.

Regret-free pricing

Interactive query markets allow consumers to cope with imperfect knowledge of their task by adapting or evolving their queries as they go. Proposition 1(ii) suggests that querying interactively costs at least as much as asking all queries together. An important aspect of pricing functions in this context is whether the consumer will be penalized for requesting queries interactively instead of all-at-once. We call a pricing function *regret-free* if there is no such penalty.

PROPERTY 2 (REGRET-FREE PRICING). *A pricing function is regret-free if for any query sets $\mathbf{Q}_1$ and $\mathbf{Q}_2$ such that* $\text{span}(\mathbf{Q}_1) \subseteq \text{span}(\mathbf{Q}_2)$, $p(\mathbf{Q}_2) = p(\mathbf{Q}_2|\mathbf{Q}_1) + p(\mathbf{Q}_1)$.

EXAMPLE 5. *In Figure 1, let $\mathbf{Q}_1 = \{\mathbf{q}_1, \mathbf{q}_5\}$, a consumer interested in $\mathbf{Q}_1$ can either ask $\mathbf{Q}_1$ directly or ask $\mathbf{q}_5$ before $\mathbf{Q}_1$. A regret-free pricing function p must guarantee that $p(\mathbf{Q}_1|\{\mathbf{q}_5\}) + p(\mathbf{q}_5) = p(\mathbf{Q}_1)$ so that the consumer is not penalized for the interaction.*

3. ARBITRAGE-FREE PRICING

In this section, we consider the design of arbitrage-free price functions. We consider both inductive and deductive pricing. In each case, we prove that computing the pricing function can be NP-hard, but we also identify PTIME cases. Due to space constraints we omit proofs of theorems.

3.1 Arbitrage-free inductive pricing

In inductive pricing we assume the data producer sets prices for all individual queries. We assume these prices are specified using a base price function denoted by $\bar{p}$, which describes the price that the data producer would like to charge for each query. Since using $\bar{p}$ directly may lead to arbitrage, one needs to derive a pricing function from $\bar{p}$ for sets of queries, including conditional prices, that are arbitrage-free.

Intuitively, to avoid arbitrage, we must set the price of query set $\mathbf{Q}_2$ to be the price of the least expensive set of queries the consumer could use to compute $\mathbf{Q}_2$. If the consumer previously purchased query set $\mathbf{Q}_1$, then we choose the least expensive set of queries to add to $\mathbf{Q}_1$ to enable computation of $\mathbf{Q}_2$. This is formalized in the following definition.

THEOREM 1 (PRICE INDUCTION). *Given a base price function $\bar{p}$ from $\mathcal{Q}$ to $\mathbb{R}^+ \cup \{0\}$ the induced price function p, defined as follows, is arbitrage-free.*

$$p(\mathbf{Q}_2|\mathbf{Q}_1) = \min_{\mathbf{Q} \in \mathcal{P}(\mathcal{Q})} \{\sum_{\mathbf{q} \in \mathbf{Q}} \bar{p}(\mathbf{q}) | \text{span}(\mathbf{Q}_2) \subseteq \text{span}(\mathbf{Q} \cup \mathbf{Q}_1)\}.$$

Moreover, p is the largest pricing function among all arbitrage-free price functions p' with $p'(\mathbf{q}) \leq \bar{p}(\mathbf{q})$ for all $\mathbf{q}$. Once p has been computed, the data producer can later scale p with a constant to make the price for each query $\mathbf{q}$ at least $\bar{p}(\mathbf{q})$.

Unfortunately, given a function $\bar{p}$, there is no polynomial time method to derive the price $p(\mathbf{Q}_2|\mathbf{Q}_1)$, even if $\mathbf{Q}_1 = \emptyset$ and $\mathbf{Q}_2$ consists of one query. In this case, there is a reduction from the problem of exact cover by 3-sets [10] to the problem of computing the induced price.

THEOREM 2. *Given a base price function $\bar{p}$, let p be the pricing function deriving from $\bar{p}$ as described in Theorem 1. Computing $p(\mathbf{q})$ for arbitrary query $\mathbf{q}$ is NP-complete.*

Cell-based inductive pricing

A perhaps more natural approach to specifying a pricing function inductively is to price simple components of queries and then derive prices for complete queries from those. We next assume the data producer sets a price measure for each cell of the data vector and also specifies a pricing relationship between pairs of cells. This input can be represented an $n \times n$ symmetric *pricing matrix*, $\mathbf{P}$, in which the i-th diagonal entry $\mathbf{P}_{ii}$ of $\mathbf{P}$ is the price measure for cell i, and an off-diagonal entry $\mathbf{P}_{ij} = \mathbf{P}_{ji}$ controls (along with $\mathbf{P}_{ii}$ and $\mathbf{P}_{jj}$) the price of queries that contain both cell i and j. Given this natural restriction on the specification, we show next that we can induce an arbitrage-free price function in polynomial time.

The pricing matrix represents a base pricing function $\bar{p}$ defined as follows, for any query $\mathbf{q}$:

$$\bar{p}(\mathbf{q}) = c_0 \exp(\frac{1}{||\mathbf{q}||_2^2} \mathbf{q}\mathbf{P}\mathbf{q}^T).$$

Here c_0 is a constant which can be used to linearly scale the base price function. Moreover, increasing $\mathbf{P}_{ii}$ boosts the price of all queries that contain cell x_i and increasing $\mathbf{P}_{ij}$ augments the price when cells x_i and x_j appear together.

EXAMPLE 6. *Recall the cells in Figure 1, and let $\mathbf{P}_{11} = \mathbf{P}_{22} = \mathbf{P}_{33} = \ln 10$, $\mathbf{P}_{12} = \mathbf{P}_{21} = 0$, $\mathbf{P}_{23} = \mathbf{P}_{32} = -\ln 2$, $\mathbf{P}_{13} = \mathbf{P}_{31} = \ln 2$. The parameters above indicate that the prices on x_1, x_2, x_3 are the same; the price of x_1 is independent of the price of x_2 and positively correlated with x_3; the price of x_2 is negatively correlated with the price of x_3. In order to compute the prices of queries $x_1 + x_2$, $x_1 + x_3$ and $x_2 + x_3$, we find that $p(x_1) = p(x_2) = p(x_3) = 10$, $p(x_1 + x_2) = 10$, $p(x_1 + x_3) = 20$ and $p(x_2 + x_3) = 5$.*

Given a pricing matrix $\mathbf{P}$, the base price function $\bar{p}$ defined above is not necessarily arbitrage-free so we still use Theorem 1 to define the arbitrage-free price function, as shown in the following theorem.

THEOREM 3. *Given a function $\bar{p}$ defined with the pricing matrix $\mathbf{P}$ as above, let $\psi = \min_{\mathbf{q} \in \mathcal{Q}} \bar{p}(\mathbf{q})$, $\psi' = \min_{\mathbf{q} \in \text{span}(\mathbf{Q}_2) - \text{span}(\mathbf{Q}_1)} \bar{p}(\mathbf{q})$ and r be the rank of* $\text{span}(\mathbf{Q}_2) - \text{span}(\mathbf{Q}_1)$. *Then $p(\mathbf{Q}_2|\mathbf{Q}_1) = \min((r+1)\psi, r\psi')$.*

Above, ψ and ψ' are equivalent to the smallest eigenvalues of $\mathbf{P}$ and the projection of $\mathbf{P}$ on $\text{span}(\mathbf{Q}_2) - \text{span}(\mathbf{Q}_1)$, respectively. Thus $p(\mathbf{Q}_2|\mathbf{Q}_1)$ can be computed in $O(n^3)$ time.

The main idea behind Theorem 3 is that, given query $\mathbf{q}_0$, we can represent any query $\mathbf{q}_1$ using $\mathbf{q}_0$ and $\mathbf{q}_0 + \epsilon\mathbf{q}_1$ with arbitrarily small ϵ. It is possible to show that Theorem 3 is correct for any continuous function p. But the theorem is not true if we limit our focus to predicate queries, (linear queries with only 0-1 coefficients) since the queries can not be combined freely and the query $\mathbf{q}_0 + \epsilon\mathbf{q}_1$ may not be a predicate query. In this case, computing $p(\mathbf{q})$ becomes an integer programming problem and is therefore NP-complete.

3.2 Arbitrage-free deductive pricing

Instead of specifying base prices for individual queries, the data producer may instead have prices in mind for certain query sets that are of primary interest. The challenge is then to determine whether there is an arbitrage-free pricing

Algorithm 1 Arbitrage-free deductive pricing for continuous price functions.

Input: Query sets $\mathbf{Q}$, $\mathbf{Q}'$, $\mathbf{Q}_1, \ldots, \mathbf{Q}_k$, prices $\psi_1, \ldots, \psi_k$
Output: Range of $p(\mathbf{Q}|\mathbf{Q}')$

1: Let $r_i = \text{rank}(\mathbf{Q}_i)$, $\psi_i' = \frac{\psi_i}{r_i}$ and $\psi_i'' = \frac{\psi_i}{r_i+1}$, $i = 1, \ldots, k$.
2: For each pair of $1 \le i \ne j \le k$,
3: Return **Inconsistenct** if one of followings is true:
4: $\text{span}(\mathbf{Q}_i) \subseteq \text{span}(\mathbf{Q}_j)$ **and** $((\psi_i' < \psi_j')$ **or** $(\psi_i > \psi_j))$
5: $\psi_i'' > \psi_j'$
6: Let $r = \text{rank}(\mathbf{Q} \cup \mathbf{Q}') - \text{rank}(\mathbf{Q}')$.
7: If $r = 0$, return 0.
8: Let $\psi' = \max(\max_{i:\mathbf{Q} \subseteq \text{span}(\mathbf{Q}_i)}(\psi_i'), \max_i(\psi_i''))$ and $\psi = \min_i(\psi_i')$.
9: If $(r+1)\psi \le r\psi'$, return $(r+1)\psi$. Otherwise, return $[r\psi', (r+1)\psi]$.

function consistent with the provided prices, and if so, to deductively compute it. The pricing function is determined implicitly by those queries or query sets, using an approach similar to Theorem 1, where we choose the cheapest set of priced queries that supports the input set.

THEOREM 4 (PRICE DEDUCTION). *Given a set of query sets $\mathbf{Q}_1, \ldots, \mathbf{Q}_k$ and their prices $\psi_1, \ldots, \psi_k$, let $\tilde{\mathcal{P}}(\mathcal{Q})$ be the set of all subsets of $\{\mathbf{Q}_1, \ldots, \mathbf{Q}_k\}$. The following deduced price function is arbitrage-free:*

$$p(\mathbf{Q}|\mathbf{Q}') = \min_{\tilde{\mathcal{Q}} \in \tilde{\mathcal{P}}(\mathcal{Q})} \{ \sum_{\mathbf{Q}_i \in \tilde{\mathcal{Q}}} \psi_i | \text{span}(\mathbf{Q}) \subseteq \text{span}(\mathbf{Q}' \cup \bigcup_{\mathbf{Q}_i \in \tilde{\mathcal{Q}}} \mathbf{Q}_i) \}.$$

A set of query sets $\mathbf{Q}_1, \ldots, \mathbf{Q}_k$ and prices $\psi_1, \ldots, \psi_k$ is called *consistent* if the pricing function p defined above satisfies $p(\mathbf{Q}_i) = \psi_i$, $i = 1, \ldots, k$. Thus checking consistency is a special case of computing $p(\mathbf{Q}|\mathbf{Q}')$. A reduction similar to that of Theorem 2 indicates that both checking consistency and general deductive price computation are both NP-complete.

THEOREM 5. *Given query sets $\mathbf{Q}_1, \ldots, \mathbf{Q}_k$ and prices $\psi_1, \ldots, \psi_k$, and pricing function p defined as in Theorem 4. Both verifying whether $p(\mathbf{Q}_i) = \psi_i$ for $i = 1, \ldots, k$ and computing $p(\mathbf{Q}'|\mathbf{Q})$ are NP-complete.*

However, under the assumption that p is a continuous function, one can estimate the price of a query set deductively in polynomial time, as described in Algorithm 1. The key idea follows the property of p as described in Theorem 3: given query sets $\mathbf{Q}_1, \ldots, \mathbf{Q}_k$ and prices $\psi_1, \ldots, \psi_k$, the algorithm estimates the cheapest query in $\mathcal{Q}$ and the cheapest query in $\mathbf{Q}_i$ for each $i = 1, \ldots, k$. The algorithm then checks consistency using those prices and derives a range of possible values for the conditional query set price $p(\mathbf{Q}|\mathbf{Q}')$.

The complexity of Algorithm 1 is dominated by checking, for each pair of sets, if one query set is a subset of the other.

THEOREM 6. *Let m be the number of queries in the query set with largest number of queries among $\mathbf{Q}$, $\mathbf{Q}'$, $\mathbf{Q}_1, \ldots, \mathbf{Q}_k$. The time complexity of Algorithm 1 is $O(m^2 n + mk^2)$.*

4. ARBITRAGE-FREE AND REGRET-FREE PRICING

Ideally we would like to design price functions that are also regret-free in order to avoid complex reasoning by the consumer about their interactions in a data market. We show next that, when a price function is both arbitrage-free and regret-free the feasible pricing functions have a surprising, highly specialized structure that is determined by no more than two distinct prices.

THEOREM 7. *A price function p is arbitrage-free and regret-free if and only if there exists a set of queries $\mathbf{q}_1, \ldots, \mathbf{q}_k$, with $\text{span}(\mathbf{q}_1, \ldots, \mathbf{q}_k) = \mathbf{U}$, and two prices, ψ_{low} and ψ_{high} where $\psi_{low} \le \psi_{high}$, such that:*

$$p(\mathbf{q}) = \begin{cases} \psi_{low} & \mathbf{q} \in \mathbf{U} \\ \psi_{high} & \mathbf{q} \notin \mathbf{U} \end{cases}$$

The theorem shows that every query is either "cheap", costing price ψ_{low}, or "expensive", costing price ψ_{high}. Further, the spanning space $\mathbf{U}$ in the theorem is the set of all cheap queries. The answers to all cheap queries can be derived from the answers to any set of basis queries whose span is $\mathbf{U}$. Such a set of basis queries never needs to be larger than $\text{rank}(\mathbf{U})$. This implies that the purchase of all cheap queries will cost $\psi_{low} * \text{rank}(\mathbf{U})$. Similarly, the set of all expensive queries will cost $\psi_{high}(n - \text{rank}(\mathbf{U}))$. More generally, the price of *any* set of queries $\mathbf{Q}$ is determined by the rank of the cheap spanning space (say r_{low}) and the rank of expensive spanning space (say r_{high}) which together make up the span of $\mathbf{Q}$. Then $p(\mathbf{Q}) = r_{low}\psi_{low} + r_{high}\psi_{high}$.

EXAMPLE 7. *Referring to the example in Fig. 1, a price function in which all queries related to top-100 firms are priced at 25 and other queries are 1 (i.e. $\psi_{low} = 1$, $\psi_{high} = 25$ and $\mathbf{U} = \text{span}(\{x_3, x_4, x_7, x_8\})$) is regret-free. Since $\mathbf{Q}$ is the set of all queries in Figure 1(c), $\text{rank}(\mathbf{Q}) = 4$, $\text{rank}(\mathbf{U}) = 4$, $\text{rank}(\mathbf{Q} \cup \mathbf{U}) = 6$ and $p(\mathbf{Q}) = 2\psi_{high} + 2\psi_{low} = 52$.*

The proof of Theorem 7 is based on two facts. First, if there are queries $\mathbf{q}_1$, $\mathbf{q}_2$ such that $p(\mathbf{q}_1) < p(\mathbf{q}_2)$, then the properties of arbitrage-free and regret-free imply that $p(c_1\mathbf{q}_1 + c_2\mathbf{q}_2) = p(\mathbf{q}_2)$ for any nonzero scalars c_1, c_2. In addition, whenever there are three queries with distinct prices, one can construct two query sets $\mathbf{Q}_1$, $\mathbf{Q}_2$ such that $\mathbf{Q}_1 \subseteq \mathbf{Q}_2$ and $p(\mathbf{Q}_2|\mathbf{Q}_1) + p(\mathbf{Q}_1) > p(\mathbf{Q}_2)$, which contradicts the regret-free property.

Inductive pricing

Theorem 7 has important implications for the specification of pricing functions. To define a regret-free pricing function inductively, one need only provide a set of queries as a basis for $\mathbf{U}$ and two prices ψ_{low} and ψ_{high}. The arbitrage-free, regret-free price for any query set $\mathbf{Q}$ can be computed using the rank of matrices, as shown in Algorithm 2. The running time of Algorithm 2 is determined by the cost of computing the rank of the query sets.

Algorithm 2 Regret-free, arbitrage-free inductive pricing

Input: Query set $\mathbf{Q}_2$ and previous purchase $\mathbf{Q}_1$; A set of basis queries $\mathbf{v}_1, \ldots, \mathbf{v}_k$ such that $\text{span}(\mathbf{v}_1, \ldots, \mathbf{v}_k) = \mathbf{U}$; ψ_{low}, ψ_{high} where $\psi_{low} \le \psi_{high}$.
Output: $p(\mathbf{Q}_2|\mathbf{Q}_1)$

1: Let $r_1 = \text{rank}(\mathbf{Q}_1)$, $r_2 = \text{rank}(\mathbf{Q}_2)$ and $r_{12} = \text{rank}(\mathbf{Q}_1 \cup \mathbf{Q}_2)$.
2: Let $r_{1v} = \text{rank}(\mathbf{Q}_1 \cup \{\mathbf{v}_1, \ldots, \mathbf{v}_k\})$ and $r_{12v} = \text{rank}(\mathbf{Q}_1 \cup \mathbf{Q}_2 \cup \{\mathbf{v}_1, \ldots, \mathbf{v}_k\})$.
3: return $(r_{12v} - r_{1v})\psi_{high} + (r_1 + r_2 - r_{12} + r_{1v} - r_{12v})\psi_{low}$.

THEOREM 8. *Let the number of queries in $\mathbf{Q}_1$ and $\mathbf{Q}_2$ be m_1 and m_2 respectively. The time complexity of Algorithm 2 is $O(n(m_1 + m_2)\min(m_1 + m_2, n))$.*

Deductive pricing

Despite the special pricing structure that follows from the property of regret-free pricing, verifying whether there exists a pricing function consistent with a given set of priced query sets remains NP-hard.

THEOREM 9. *Given query sets $\mathbf{Q}_1, \ldots, \mathbf{Q}_k$ and prices $\psi_1, \ldots, \psi_k$, the problem of deciding whether there exists a regret-free, arbitrage-free pricing function p such that $p(\mathbf{Q}_i) = \psi_i$, $i = 1, \ldots, k$ is NP-complete.*

For any regret-free pricing function p, consistency of query set prices can be verified in polynomial time using Algorithm 2. On the other hand, the hardness of deductive pricing holds even if we only consider query sets consisting of queries with at most three cells, in which case it can be used to solve the 3-SAT problem. Thus the complexity of constructing a regret-free pricing function, if one exists, is at least NP-hard. A non-trivial upper bound on its complexity is still open.

5. RELATED WORK AND DISCUSSION

Our work is inspired by the vision paper of Balazinska, Howe and Suciu [7], which describes challenges associated with data markets including fine-grained pricing, fairness, and the dangers of arbitrage. The authors propose query provenance as one possible foundation for deriving pricing functions for queries. Provenance methods alone do not appear to provide easy answers to pricing problems. But the representation of our linear aggregation queries exposes each of the cells on which the query answer could depend. In this sense, we have relied on a simple form of provenance for aggregation queries. More sophisticated models of provenance for aggregation queries have been studied by Amsterdamer et al. [6] and future applications to pricing are possible.

The closest work to our own is the recent paper by Koutris et al. proposing a query-based pricing mechanism for data markets [12]. The authors consider full relational queries (not aggregates) over a dynamic (not static) database. They study a different set of basic properties: arbitrage-free pricing is considered, but the model does not include interactive pricing, and the price of a query depends on its answer with respect to the current state of the database. Such price functions are therefore disclosive. The authors consider a version of deductive pricing in which prices for a set of relational views are given and a price for any input query is derived in a manner consistent with the priced input views. For a class of priced selection views, they show it is NP-hard (in the size of the input database) to deduce the price of some conjunctive queries. The authors describe a sub-class of conjunctive queries that can be priced with polynomial data complexity, among other complexity results. Our Thm. 1 is a data-independent version of their "fundamental pricing formula", adapted to linear queries and interactive pricing.

There is an interesting relationship between data privacy and pricing in data markets. In particular, the model of differential privacy [9] allows for the accurate release of aggregate statistics while prohibiting the release of information about individuals. Differentially private mechanisms achieve this balance through the addition of random noise to query answers. Selling noisy query answers, for which the consumer pays more for greater accuracy, may make sense in some data markets and could expand the range of price functions that can be considered. In fact, Ghosh and Roth attempt to use differential privacy as a foundation for pricing personal data [11] in the context of auctions. Under the assumption of rational and truthful auction participants, the authors designed two different algorithms: one to maximize accuracy with bounded privacy budget and the other to minimize the privacy budget with desired accuracy.

We note that aggregate queries are irreversible functions of their inputs because it is not possible to exactly deduce the inputs from the output. The price of an irreversible function $f(x_1, \ldots, x_k)$ can be lower than the prices of its inputs $x_1, \ldots, x_k$, which is unusual in the pricing of physical goods. If it is possible to precisely recover the inputs from the output (as is the case for operations like sorting, restructuring, data cleaning and any deterministic computation on a single variable) then f is a *reversible* function. Pricing for reversible operations in a data market would be substantially different. In this case, the price of the output is usually larger than the price of the inputs by a factor that involves the cost of computing the function.

In the future we hope to expand our results to a broader class of queries, explore pricing for approximate query answers, and identify additional polynomial-time cases for computing sound price functions.

Acknowledgements. We thank Paris Koutris and Dan Suciu for helpful discussions. This work was supported by the NSF through grants CNS-1012748 and IIS-0643681.

6. REFERENCES

[1] http://datamarket.azure.com/.
[2] http://singly.com.
[3] http://www.factual.com/.
[4] http://www.infochimps.com/.
[5] http://www.personal.com.
[6] Y. Amsterdamer, D. Deutch, and V. Tannen. Provenance for aggregate queries. In *PODS*, 2011.
[7] M. Balazinska, B. Howe, and D. Suciu. Data markets in the cloud: An opportunity for the database community. In *PVLDB*, 2011.
[8] B. Barak, K. Chaudhuri, C. Dwork, S. Kale, F. McSherry, and K. Talwar. Privacy, accuracy, and consistency too: a holistic solution to contingency table release. In *PODS*, pages 273–282, 2007.
[9] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Third Theory of Cryptography Conference*, 2006.
[10] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.
[11] A. Ghosh and A. Roth. Selling privacy at auction. In *Electronic Commerce*, 2011.
[12] P. Koutris, P. Upadhyaya, M. Balazinska, B. Howe, and D. Suciu. Query-based data pricing. In *PODS*, 2012.
[13] C. Li, M. Hay, V. Rastogi, G. Miklau, and A. McGregor. Optimizing linear counting queries under differential privacy. In *PODS*, 2010.
[14] X. Xiao, G. Wang, and J. Gehrke. Differential privacy via wavelet transforms. In *ICDE*, 2010.