# On Arbitrage-free Pricing for General Data Queries

Bing-Rong Lin
Dept. of Computer Science & Engineering
Penn State University

Daniel Kifer
Dept. of Computer Science & Engineering
Penn State University

## ABSTRACT

Data is a commodity. Recent research has considered the mathematical problem of setting prices for different queries over data. Ideal pricing functions need to be flexible – defined for arbitrary queries (select-project-join, aggregate, random sample, and noisy privacy-preserving queries). They should be fine-grained – a consumer should not be required to buy the entire database to get answers to simple "low-information" queries (such as selecting only a few tuples or aggregating over only one attribute). Similarly, a consumer may not want to pay a large amount of money, only to discover that the database is empty. Finally, pricing functions should satisfy consistency conditions such as being "arbitrage-free" – consumers should not be able to circumvent the pricing function by deducing the answer to an expensive query from a few cheap queries.

Previously proposed pricing functions satisfy some of these criteria (i.e. they are defined for restricted subclasses of queries and/or use relaxed conditions for avoiding arbitrage). In this paper, we study arbitrage-free pricing functions defined for *arbitrary* queries. We propose new necessary conditions for avoiding arbitrage and provide new arbitrage-free pricing functions. We also prove several negative results related to the tension between flexible pricing and avoiding arbitrage, and show how this tension often results in unreasonable prices.

## 1. INTRODUCTION

Datasets provide valuable information but are often expensive or difficult to generate. As commodities, they can be bought and sold in marketplaces like Windows Azure [13] or from individual sellers. Traditionally, data pricing is very simplistic – often a consumer must choose from a limited catalog of queries (e.g., a background check on a specific individual). To get an answer to a query not in this list (such as histogram of crimes by state from this background-check database) the consumer would need to buy the entire dataset.

Recently, Balazinska et al. [1] considered the research challenges of developing fine-grained pricing functions, where users can pay for answers to a wider variety queries without buying the entire dataset. They proposed the fundamental requirement that query pricing functions be *arbitrage-free.*

Intuitively, this requirement means that if $q$ is a query with price $\pounds(q)$ and $q_1, q_2, \dots, q_k$ are queries whose answers can be used to answer $q$, then $\pounds(q_1) + \pounds(q_2) + \cdots + \pounds(q_k) \geq \pounds(q)$. That is, one should not be able to circumvent the pricing function by finding an answer to $q$ while paying less than $\pounds(q)$ for this information.

The seminal work of Koutris et al. [5] considered how to price variants of conjunctive queries under a precise mathematical definition of arbitrage. Followup work by Li et al. [8, 7] considered how to price linear aggregate queries and a special kind of noisy queries using different definitions of arbitrage. An independently developed notion of a privacy budget, useful for automated management of certain noisy queries [9], can also be used as a pricing function. These works have provided a variety of pricing functions for various classes of queries and with different levels of protection against arbitrage. Much of this work [5, 8, 6] is concerned with computational complexity and efficient implementation of pricing algorithms. The complexity side has been well studied, however many important challenges remain unsolved. These challenges are the focus of this paper.

The first challenge is consistency. Previously proposed pricing functions are based on incompatible definitions of arbitrage, and cannot be combined (under a common definition of arbitrage) to provide an arbitrage-free pricing scheme for data access, aggregate, and noisy queries.

The second challenge is coverage. Many queries, such as retrieving a random sample of tuples, do not have fine-grained pricing. Either pricing for random samples is not supported [8, 7, 9] or the price of a random sample of a view equals the price of the entire view [5].

A third challenge is to ensure reasonable pricing - a customer should not be required to pay a large price for a useless query answer. This is clearly a subjective principle, but often we can reach a consensus: consumers shouldn't pay much to find out that the database is empty, the price for the entire database should be much larger than the price of one record, and in many (but not all) situations, negative information (such as finding out that a record $t_1$ is not in the database) should be significantly cheaper than positive information (such as finding out that a record $t_2$ is in the database).

*Proceedings of the VLDB Endowment,* Vol. 7, No. 9
Copyright 2014 VLDB Endowment 2150-8097/14/05.

We note that there is a fundamental tension between solutions to these challenges. For example, if the price of a query does not depend on the database instance, then a consumer would pay the same amount regardless of the quality of the database and resultant query answer (e.g., an empty search result). On the other hand, if query prices do depend on the database instance, then it may be possible for a savvy consumer to deduce the answer to some queries just by asking about their price (an arbitrage opportunity).

## 1.1 Contributions

In order to better understand the possibilities and limitations of arbitrage-free pricing functions for arbitrary queries, we consider in detail two pricing schemes. The first scheme prices queries independently of the database instance. The second scheme prices queries based on the answers they return. In the latter case, the consumer knows in advance the pricing function, but not the price (until the query returns and the consumer's account is charged). For example, the price may depend on the number of tuples returned and the consumer could limit its liability by specifying that at most 20 tuples should be returned. We briefly touch upon a third scheme, where the query price depends on the database and is known in advance, and show how query answers can often be deduced just from their prices (but leave solutions to this particular issue as an open problem).

For the first two pricing schemes, we catalog arbitrage opportunities and formulate conditions that must be to satisfied to avoid arbitrage. Since prices are required for *arbitrary* queries, it turns out that new conditions need to be added and previously proposed conditions need to be strengthened/generalized.

We then provide several classes of pricing functions under these pricing schemes and prove that they are arbitrage-free according to our criteria (thus they have stronger protections against arbitrage than prior work).

We generate several important negative results. First, we show that an appealing query pricing function, which sets prices base on the number of tuples returned, cannot be reasonably extended to set prices for arbitrary queries (it would have to set a high price for letting consumers query an empty database). Then we consider Bayesian pricing functions, where the price of a query answer depends on the resulting posterior distribution a consumer may form. Under general conditions, we show that the prices of retrieving a few tuples would be comparable to the price of retrieving an entire table.

In both cases, the negative results arise from the requirement that a price must be assigned to certain problematic (but natural) queries. Due to the need for avoiding arbitrage, this induces undesired dependencies between the prices of other queries. Thus we provide strong evidence that a data seller may need to choose 2 out of 3 properties (generality, arbitrage-free, reasonableness) by either carefully restricting the class of possible queries, accepting some risk of arbitrage, or assigning unreasonable/unfair prices to certain queries.

## 1.2 Outline

In Section 2, we introduce key concepts, notation, and terminology. We review related work and pricing function vulnerabilities in Section 3. In Section 4, we consider query pricing functions where the price depends on the query answer. We formulate mathematical conditions for avoiding arbitrage in Section 4.1 then provide example arbitrage-free pricing functions and prove negative results in Section 4.2. We consider pricing functions that are independent of the database instance in Section 5. Conditions for avoiding arbitrage appear in Section 5.1, followed by example arbitrage-free pricing functions in Section 5.2. We discuss dynamic pricing (where a query's price can depend on previously issued queries) in Section 6, issues in eliciting pricing functions in Section 7, and conclusions in Section 8.

# 2. NOTATION AND TERMINOLOGY

Let $\mathcal{I}$ be the set of all database instances consistent with the information provided by a data seller (e.g., a schema $\mathcal{S}$, number of records, etc.). Since our goal is to extend pricing functions to potentially arbitrary (randomized and deterministic) queries over databases in $\mathcal{I}$, it is more convenient to model queries in a language-independent way.

We view a query $q$ as a (possibly randomized) function over $\mathcal{I}$. Whether a query is randomized or deterministic, the quantity $P(q(D) = \omega)$ is well defined for all databases $D \in \mathcal{I}$ and answers $\omega \in \text{range}(q)$.

A *query bundle* $[q_1, \dots, q_k]$ is a set of queries $q_1, \dots, q_k$ that are posed at the same time. Note that a query bundle is itself just a query that simultaneously returns the results of several sub-queries (thus we use this notation for bundles when we want to emphasize the individual queries in the bundle). We use the notation $[\omega_1, \dots, \omega_k]$ to refer to the bundle of query answers that are returned (thus $\omega_i$ is the value returned by $q_i$ when executed over the data).

## 2.1 Pricing Schemes

We will consider three pricing models for query bundles: *instance-independent*, *up-front dependent* and *delayed*.

Definition 2.1 (Instance-independent pricing). *A pricing function $\pounds(\cdot)$ is instance-independent if it depends on the query bundle but not the database instance. A consumer will pay $\pounds(q)$ for the result of query bundle $q$.*

An advantage of instance-independent pricing is that the prices leak no information about the actual database instance. However, consumers will pay the same price regardless of how satisfied they are with the query answer (such as an empty search result). For this reason, instance-independent pricing schemes are suitable under the *closed-world* assumption, where empty search results (such as a lack of a criminal record for a potential employee) are meaningful.

Definition 2.2 (Up-front dependent pricing). *An up-front dependent pricing function €$(\cdot)$ depends both on the query bundle $q$ and the actual database instance $D$. The price €$(q, D)$ is available to the consumer, and based on this price, the consumer can decide whether or not to purchase the query answers.*

Up-front dependent query pricing schemes are flexible enough to provide a reduced price when the database instance has low quality (e.g., no search results) and a higher price when the database instance would provide a high quality query answer [5]. However, as we show in Section 3, clever consumers can often reconstruct large portions of the dataset (for free) just from asking about the prices of queries.

DEFINITION 2.3 (DELAYED PRICING). *A pricing function* ₵$(\cdot)$ *is* delayed *if it depends both on $q$ and on the answer $\omega$ computed by query bundle $q$ on the current database instance. The consumer knows* ₵$(\cdot)$ *in advance but does not know* ₵$(q, \omega)$ *until the answer is returned (and the consumer's account has been charged).*

The simplest delayed-pricing scheme (applicable to selection queries) is to charge based on the number of tuples returned. It is appealing because the size of the query answer roughly indicates its quality. Furthermore, a consumer can limit costs by asking queries with restricted outputs such as `SELECT name FROM Recommended_Stocks LIMIT 20` or top-$k$ queries or a fixed size random sample. Unfortunately, as we show in Section 4.2.1, even attempting to extend this pricing scheme to simple aggregate queries will lead to undesired behavior – either there is no extension (if querying empty databases should be free) or there will be a large price associated with learning that the database is empty.

Despite this drawback, delayed-pricing schemes have three favorable properties. First, unlike instance-independent pricing schemes, the price can be tailored to the quality of the query answer. Second, unlike up-front dependent pricing, query prices do not leak new information about the database instance (since they are revealed simultaneously with the query answers). Finally, since the pricing function is public, consumers can verify from the query answers that their accounts were charged correctly.

### 2.1.1 *The Probabilistic View*

The price of a query bundle depends on what it reveals about the true database instance. Clearly, the semantics of a query bundle $q$ are completely captured by the values $P(q(D) = \omega)$ for all $D \in \mathcal{I}$ and $\omega \in \text{range}(q)$. For example, suppose the database contains the table $T(A, B)$ and that the query $q$ and answer $\omega$ are such that:

- $P(q(D) = \omega) = 1$ for all $D \in \mathcal{I}$ in which table $T$ contain tuples $t_1, t_2$ (with $t_1[A] = t_2[A] = 1$) and no other tuples $t$ having $t[A] = 1$.

- $P(q(D) = \omega) = 0$ for all other database instances $D$.

Receiving this answer $\omega$ from query $q$ is clearly semantically identical to receiving the set of tuples $\{t_1, t_2\}$ in response to the query `SELECT * FROM T WHERE T.A=1`.

Thus the price of a query bundle $q$ is really a function of the probabilities $P(q(D) = \omega)$ for $D \in \mathcal{I}$ and $\omega \in \text{range}(q)$. More specifically:

- Instance independent pricing functions £$(q)$ are functions of the quantities $P(q(D) = \omega)$ for all $D \in \mathcal{I}$ and $\omega \in \text{range}(q)$. In other words they are functions of the entire matrix:

$$\begin{array}{c|ccc|} & D_1 & D_2 & \dots \\ \omega_1 & P(q(D_1) = \omega_1) & P(q(D_2) = \omega_1) & \dots \\ \omega_2 & P(q(D_1) = \omega_2) & P(q(D_2) = \omega_2) & \dots \\ \omega_3 & P(q(D_1) = \omega_3) & P(q(D_2) = \omega_3) & \dots \\ \vdots & \vdots & \vdots & \vdots \end{array}$$

- Up-front dependent pricing functions €$(q, D)$, which fix $D$ to be the true database instance, only depend on the vector $[P(q(D) = \omega_1),\ P(q(D) = \omega_2), \dots]$. This vector is the *column* of that matrix corresponding to the true database instance $D$.

- Delayed pricing functions $\mathbb{C}(q, \omega)$ must be functions of $[P(q(D_1) = \omega), P(q(D_2) = \omega), \dots]$ where we range over all $D \in \mathcal{I}$ but $\omega$ is fixed (it is the output returned by $q$). This vector is the *row* of the aforementioned matrix corresponding to the query bundle's output $\omega$. Since we will frequently be referring to this vector, we will use the following notation to represent it:

$$\vec{P}[q(\cdot) = \omega] \quad \equiv \quad \Big[P(q(D_1) = \omega),\ P(q(D_2) = \omega),\ \dots\Big]$$

and hence each delayed pricing function $\mathbb{C}(\cdot)$ satisfies $\mathbb{C}(q, \omega) \equiv f(\vec{P}[q(\cdot) = \omega])$ for some $f$.

## 2.2 Avoiding Arbitrage

An arbitrage situation exists when an attacker can obtain an answer to a query bundle without paying the price set by the data owner. We now briefly summarize different scenarios that can lead to arbitrage. In later sections, we will formalize the mathematical conditions needed to avoid each of these situations.

**Price-based arbitrage:** A consumer may ask for the price of queries rather than the answers. We must ensure that query answers cannot be deduced from prices alone.

**Separate account arbitrage [5]:** Consider the query bundle $q = [q_1, q_2]$. If the price of $q$ is larger than the sum of the prices of $q_1$ and $q_2$ then a consumer could avoid paying the nominal price for $q$ by opening two separate accounts, using one account to ask $q_1$ and another to ask $q_2$. To avoid this arbitrage scenario, the price of $q$ must be at most the sum of the prices of $q_1$ and $q_2$.[1]

**Post-processing arbitrage [8]:** If the answers to a query bundle $q'$ can always be deduced from answers to a query bundle $q$, then obtaining answers from $q$ should be at least as expensive as obtaining answers from $q'$. We do not consider the computational cost of such post-processing; this is a direction for future work.

**Serendipitous arbitrage:** This is a generalization of the arbitrage concerns in [5]. In general, a query bundle $q_1$ may not always be answerable by a query bundle $q_2$. However, depending on the actual database instance and possible randomness in $q_2$, the consumer may get lucky and get an answer that would let it deduce the answer to $q_1$ as well. For example, if a consumer pays for a query returning the maximum $M$ and minimum $m$ of a numeric attribute (such as salary) and if $m = M$ for the current database instance, then the consumer gains the ability to answer any query about that attribute. If such queries cost more than what the consumer payed to learn the fact $m = M$, then a serendipitous arbitrage opportunity exists. Another example can occur with random sampling queries. Suppose a consumer is interested in whether a database of criminal records contains Bob. If the consumer asks for a random sample of the data and the sample happens to contain Bob's record, then a serendipitous arbitrage occurs if the cost of that specific random sample is less than the cost of querying about Bob.

**Almost-Certain Arbitrage:** This situation can arise when two queries behave almost identically yet have significantly different prices. Consider an event that is so rare,

[1] We shouldn't require that the costs be equal. For example, if $q_1$ and $q_2$ are the same deterministic query, then the cost of $q$ should equal the cost of $q_1$.

that in practice we dismiss it from consideration (such as winning every single lottery for the next hundred years). Let $p$ be the probability of this event. For a query bundle $q$, consider an alternate query $\tilde{q}$ that returns the answer to $q$ with probability $1-p$ and returns $\perp$ with probability $p$. For all practical purposes $q$ and $\tilde{q}$ are interchangeable and so their prices should be nearly the same. The possibility of this kind of arbitrage was mentioned by Li et al. [7] but they did not provide criteria that are needed to avoid it. The previous discussion suggests that some notion of continuity should be imposed on the pricing function.

We formulate these conditions mathematically for delayed-pricing functions (over arbitrary queries) in Section 4. Mathematical conditions for instance-independent pricing schemes are presented in Section 5. We defer consideration of upfront-dependent pricing functions to future work – they are susceptible to price-based arbitrage (see Section 3), but a solution appears to require the data seller to carefully restrict the types of queries that can be asked (and hence is out of the scope of this paper, which investigates fine-grained pricing for arbitrary queries).

## 3. RELATED WORK

The vision paper by Balazinska et al. [1] introduced the problem of arbitrage-free fine-grained query pricing to the database community. In this section, we outline the important progress that has been achieved in this area. Our main contributions with respect to prior work are more robust guarantees against arbitrage while allowing arbitrary queries to be priced, along with negative results about the tradeoff between arbitrage protection, flexible pricing, and reasonableness of the prices.

Following [1], Koutris et al. [5] introduced a sophisticated up-front dependent pricing scheme and studied the computational complexity of pricing variations of conjunctive queries. In this pricing scheme, the data seller provides a fixed set of views $\mathbb{V} = \{V_1, \dots, V_k\}$ and their corresponding prices $p_1, \dots, p_k$. A query bundle $q$ is said to be *answerable* by a subset $\{V_{i_1}, \dots, V_{i_r}\} \subseteq \mathbb{V}$ of these views with respect to the true database instance $D$ if the following is true: whenever some instance $D' \in \mathcal{I}$ matches those selected views (i.e. $V_{i_1}(D) = V_{i_1}(D') \wedge \dots \wedge V_{i_r}(D) = V_{i_r}(D')$) then $q(D) = q(D')$ [5]. In this model, the price of $q$ is the cost of purchasing the subset of views that can answer the query bundle (when multiple such subsets exist, choose the cheapest one). This pricing scheme $€(\cdot)$ is not intended for aggregate queries (the price of the query `SELECT COUNT(*) FROM` $V_1$ will generally be equal to the price of $V_1$). To avoid price-based arbitrage, one should be careful in setting the prices for the views $V_1, \dots, V_k$ and in choosing the queries that consumers are allowed to ask. For example, suppose the database instance $D$ contains a relation $R(A, B)$ with key $B$ and a relation $S(B, C)$ with multi-attribute key $(B, C)$ (so that tuples are distinct). Suppose that the fixed set of views (specified by the seller) are the tables themselves. The price of the table $R(A, B)$ is $p_R > 0$ and the price of $S(B, C)$ is $p_S > 0$. Consider the queries:

```
q1 = SELECT A,B FROM R WHERE A=a AND B=b
q2 = SELECT C,B FROM S WHERE C=c AND B=b
q  = SELECT A,R.B,C FROM R,S WHERE R.B=S.B AND R.B=b
        AND A=a AND C=c
```

and the following four cases.

**Case 1: $q1(D) \neq \emptyset$ and $q2(D) \neq \emptyset$.** Then the price $€(q) = p_R + p_S$ (since both $R$ and $S$ are needed to answer $q$).

**Case 2: $q1(D) \neq \emptyset$ and $q2(D) = \emptyset$.** Then the price $€(q) = p_S$ because table $S$ does not contain the tuple $(b, c)$ and so we can answer the join query $q$ (an empty result) just by examining table $S$.

**Case 3: $q1(D) = \emptyset$ and $q2(D) \neq \emptyset$.** Then the price $€(q) = p_R$ using similar reasoning to Case 2.

**Case 4: $q1(D) = q2(D) = \emptyset$.** Then $€(q) = \min(p_r, p_S)$ using similar reasoning to Cases 2 and 3.

Note that Case 1 is the only situation where the tuple (a,b,c) appears in the join and is the only situation where the price $€(q) = p_R + p_S$. Thus a clever consumer can verify (for free) whether tuples are in the join of R and S simply by checking the price of parametrized queries such as $q$. Furthermore, if $p_S > p_R$, we gain additional information about table R: if the price of $q$ is $p_S + p_R$ or $p_S$, then either Case 1 or Case 2 is true and tuple $(a, b) \in R$; if the price of $q$ is $p_R$ then Case 3 or Case 4 is true and tuple $(a, b) \notin R$. A similar result holds if $p_R > p_S$. Thus repeated probing of the price function can reveal the join of two tables or even one of the base tables (more efficient attacks may also exist).

Fixing this vulnerability may require auditing the behavior of consumers along with careful selection of the initial views $V_1, \dots, V_k$, their prices, and the set of allowable consumer queries. Alternatively, a seller could switch to instance-independent or delayed pricing schemes.

An obstacle to practical implementations of query pricing schemes is the potential worst-case time complexity of computing the pricing functions. Koutris et al. [5] showed that in many circumstances, their pricing function is intractable. However, in followup work, Koutris et al. [6] showed that in practice the prices of many queries can be efficiently computed using ILP solvers. Tang et al [11] studied a related problem where the base views are individual tuples.

Li and Miklau [8] introduced an instance-independent pricing scheme for linear aggregate queries. A table is represented as a vector $\vec{x}$ and each query is represented as a vector $\vec{q}$. The query answer is the dot product $\vec{x} \cdot \vec{q}$. The definition of "arbitrage-free" used in this work is based on the linear span of queries appearing in a query bundle $q = [\vec{q1}, \dots, \vec{q_k}]$ and hence their pricing functions avoid arbitrage precisely when consumers use linear algebra to infer query answers. To see the effect of *nonlinear* processing techniques, suppose the data vector $\vec{x}$ is known to be binary-valued (e.g., each coordinate records a yes/no answer). Consider the query $\vec{q} = [1, 2, 4, 8, 16, \dots]$. The query answer $\vec{q} \cdot \vec{x}$ is a number whose binary representation is precisely $\vec{x}$. Computing the binary representation is a nonlinear operation – a consumer using matrix multiplications and additions cannot do it. We believe attacks and countermeasures will be an exciting part of the query pricing literature.

Followup work by Li et al. [7] considers the more general problem where individuals contribute their data to a market maker who allows consumers to ask noisy linear queries over the data. Consumers pay for these queries and the market maker distributes micropayments to the individuals who contributed their data (as compensation for possible loss of privacy). While specializing their results to linear queries with Laplace noise, Li et al. [7] point out inherent difficulties with setting prices for noisy queries - they provide an example of a noise distribution that has arbitrarily high variance yet returns an exact answer with probability arbitrarily close to 1. Their observation implies that setting a low price based on the high variance of such a query can lead to an almost-certain arbitrage opportunity, unless other precautions were taken. This problem was left open in [7]. Approximate answers were also studied by Tang et al. [10] but were not explicitly connected to definitions of arbitrage.

The research community has also noted connections between differential privacy [3] and the price of access to private data. To each query $q$, differential privacy assigns a value $\epsilon_q$ (which is nonnegative and possibly equal to $\infty$), that indicates (in the worst-case) how much information is leaked by the query $q$ to an attacker who knows all but one of the records in a database. For deterministic queries (e.q. SQL queries) and many randomized queries (such as random samples or aggregates with Gaussian noise), the leakage is $\epsilon = \infty$. For many other queries (such as aggregate queries with Laplace noise [3]), $\epsilon < \infty$ and is usually small. McSherry [9] popularized the term *privacy budget* to refer to the sum of the $\epsilon$ values of a series of queries. As the name suggests, $\epsilon_q$ can indeed be used for setting price of a query $q$. However, many database queries of interest have $\epsilon_q = \infty$.

Ghosh and Roth [4] consider the problem of designing auctions for compensating users when differentially private (noisy aggregate) queries are run over their data. Other types of queries (e.g., deterministic database queries) cannot be priced in this model, but this work has spurred research into designing economic incentives for individuals to contribute data to a trusted third party.

# 4. DELAYED PRICING

Delayed-pricing schemes are natural mechanisms for setting query bundle prices. The amount that a consumer pays depends on the actual answer and is correlated with the answer quality (for example, by charging \$1 for each tuple returned). The consumers have access to the pricing function (i.e. how a price is determined from a query answer), but they learn the actual price only after they commit to purchasing the query answers. This restriction is essential – otherwise a consumer could recreate much of the database simply by asking for the prices of queries (as in Section 3).

Despite not knowing the price in advance, consumers can limit their risk by choosing query bundles intelligently. For example, in a dollar-per-record pricing system they may ask for at most 20 records satisfying a `SELECT` query so that the most they pay is \$20.

The pricing function $\mathbb{C}(\cdot)$ sets a price $\mathbb{C}(q, \omega)$ based on the query bundle $q$ and the answer $\omega$ it returns. As discussed in Section 2, this price must be a function of the probability vector $\vec{P}[q(\cdot) = \omega]$ (i.e. $P(q(D) = \omega)$ for all $D \in \mathcal{I}$). In Section 4.1, we present conditions on $\mathbb{C}(\cdot)$ for avoiding arbitrage. In Section 4.2 we give examples of arbitrage-free delayed pricing functions and prove negative results about assigning reasonable prices to all queries.

## 4.1 Conditions for Avoiding Arbitrage

### 4.1.1 Avoiding price-based arbitrage

The price is a function of the query answer and a consumer learns the price and the answer simultaneously. Hence the price leaks no additional information (beyond what the consumer paid for) about the true database instance. Thus we only need to ensure prices are nonnegative (sellers do not pay consumers to ask queries) and anything the consumer can figure out with no information is free. Formally,

$$\mathbb{C}(q, \omega) \geq 0 \quad \text{(for all } q \text{ and } \omega\text{)} \tag{1}$$

$$\text{If } P(q(D) = \omega) = P(q(D') = \omega) \text{ for all } D, D' \in \mathcal{I}$$
$$\text{then } \mathbb{C}(q, \omega) = 0 \tag{2}$$

### 4.1.2 Avoiding separate-account arbitrage

Let $q_1$ and $q_2$ be query bundles. A consumer who issues them both at the same time by putting them in a query bundle $q = [q_1, q_2]$ will pay $\pounds(q)$ and a sneaky consumer who, instead, uses separate accounts to issue $q_1$ and $q_2$ will pay $\pounds(q_1) + \pounds(q_2)$. Avoiding separate account arbitrage requires:

$$\mathbb{C}([q_1, q_2], [\omega_1, \omega_2]) \quad \leq \quad \mathbb{C}(q_1, \omega_1) + \mathbb{C}(q_2, \omega_2) \tag{3}$$

and in general we expect strict inequality to hold (for example, when $q_1 = q_2$ and both are deterministic).

### 4.1.3 Avoiding post-processing arbitrage

For post-processing arbitrage, we need to consider the following scenario. A consumer is interested in the answer to a query bundle $q$. However, $q$ may be equivalent to a composite query $q_2 \circ q_1$ that first runs $q_1$ on the data and $q_2$ on the result. We can think of $q_2$ as a possibly randomized mapping that converts an output $\eta$ produced by $q_1$ into an output $\omega$ belonging to the range of $q$.[2] Thus there are several paths to getting a query answer $\omega$: the data consumer can issue query bundle $q$ and receive the answer $\omega$ while paying $\mathbb{C}(q, \omega)$, or pose query $q_1$ to get some $\eta$ at a cost of $\mathbb{C}(q_1, \eta)$ then compute $q_2(\eta)$ locally to obtain $\omega$. A post-processing arbitrage will occur if the data consumer is guaranteed to pay less by posing query $q_1$. Formally, a post-processing arbitrage scenario exists if there exists a $q_1$ and $q_2$ such that $q = q_2 \circ q_1$ and for every $\omega \in \text{range}(q)$ and every $\eta \in \text{range}(q_1)$ that may be mapped to $\omega$ by $q_2$, the inequalities $\mathbb{C}(q, \omega) \geq \mathbb{C}(q_1, \eta)$ hold (with one of the inequalities being strict). The following theorem will be useful for verifying that delayed-pricing functions avoid this arbitrage scenario:

THEOREM 4.1. *If $\mathcal{I}$ is finite, a continuous delayed-pricing function $\mathbb{C}(\cdot)$ avoids post-processing arbitrage if an only if the following two conditions hold.*

1. *If $q, q^\dagger$ and $\omega, \eta_1, \eta_2$ are such that $\vec{P}[q(\cdot) = \omega] = \vec{P}[q^\dagger(\cdot) = \eta_1] + \vec{P}[q^\dagger(\cdot) = \eta_2]$, then*
$$\mathbb{C}(q, \omega) \quad \leq \quad \max\left\{\mathbb{C}(q^\dagger, \eta_1), \quad \mathbb{C}(q^\dagger, \eta_2)\right\} \tag{4}$$
*and the inequality is strict when $\mathbb{C}(q^\dagger, \eta_1) \neq \mathbb{C}(q^\dagger, \eta_2)$.*

2. *For every pair of queries $q, q^*$ where $\vec{P}[q(\cdot) = \omega] = c\vec{P}[q^*(\cdot) = \eta]$ for some $c > 0$:*
$$\mathbb{C}(q, \omega) \quad = \quad \mathbb{C}(q^*, \omega) \tag{5}$$

PROOF. We first prove that satisfying these conditions are sufficient to avoid the post-processing arbitrage scenario. First recall from Section 2 that $\mathbb{C}(q, \omega)$ is a function $f$ of the

[2] For example, $q$ could be the query `SELECT C,COUNT(*) FROM T WHERE A=a AND B=b GROUP BY C` and $q_1$ could be the query `SELECT * FROM T WHERE A=a`. In this case, $q_2$ would filter out tuples based on attribute $B$ and then perform the group-by. In general, the queries need not be deterministic.

probability vector $\vec{P}[q(\cdot) = \omega] = [P(q(D_1) = \omega),\ P(q(D_2) = \omega), \dots]$. Using Equation 4 followed by Equation 5, we have the following:

$$\begin{aligned}
&\mathbb{C}(q, \omega)\\
&= f\left([P(q(D_1) = \omega),\ P(q(D_2) = \omega), \dots]\right)\\
&= f\left(\sum_{\eta} P(q_2(\eta) = \omega)\Big[P(q_1(D_1) = \eta),\ P(q_1(D_2) = \eta), \dots\Big]\right)\\
&\leq \max_{\eta} f\Big(P(q_2(\eta) = \omega)\Big[P(q_1(D_1) = \eta),\ P(q_1(D_2) = \eta), \dots\Big]\Big)\\
&= \max_{\eta} f\Big(\Big[P(q_1(D_1) = \eta),\ P(q_1(D_2) = \eta), \dots\Big]\Big)\\
&= \max_{\eta} \mathbb{C}(q_1, \eta)
\end{aligned}$$

with the first inequality being strict if the $\mathbb{C}(q_1, \eta)$ are not all identical. It is easy to see that this prevents the post-processing arbitrage scenario.

We now show that the conditions are necessary. To prove Equation 4, fix a query $q$ and an $\omega^* \in \text{range}(q)$. Define a query $q^\dagger$ such that $P(q(D) = \omega) = P(q^\dagger(D) = \omega)$ for all $D \in \mathcal{I}$ and all $\omega \in \text{range}(q)$ that are not equal to $\omega^*$. Let $P(q^\dagger(D) = \eta_1)$ and $P(q^\dagger(D) = \eta_2)$ be arbitrary subject to the condition that $P(q^\dagger(D) = \eta_1) + P(q^\dagger(D) = \eta_2) = P(q(D) = \omega^*)$ for all $D \in \mathcal{I}$. Define a processing algorithm $q_2$ that maps $\eta_1$ and $\eta_2$ into $\omega^*$ (and otherwise acts like the identity). Then it is easy to see that $q_2 \circ q^\dagger = q$. Clearly, $\mathbb{C}(q, \omega) = \mathbb{C}(q^\dagger, \omega)$ for all $\omega \neq \omega^*$ (since price is a function of the probability vector). Since $q_2$ maps $\eta_1$ and $\eta_2$ into $\omega^*$, there will be an arbitrage opportunity if $\mathbb{C}(q, \omega^*) \geq \mathbb{C}(q^\dagger, \eta_1)$ and $\mathbb{C}(q, \omega^*) \geq \mathbb{C}(q^\dagger, \eta_2)$ with one of those being strict inequality. If $\mathbb{C}(q^\dagger, \eta_1) = \mathbb{C}(q^\dagger, \eta_2)$ then arbitrage can only be avoided when $\mathbb{C}(q, \omega^*) \leq \max\left\{\mathbb{C}(q^\dagger, \eta_1), \mathbb{C}(q^\dagger, \eta_2)\right\}$. On the other hand, if $\mathbb{C}(q^\dagger, \eta_1) \neq \mathbb{C}(q^\dagger, \eta_2)$ then arbitrage can only be avoided when $\mathbb{C}(q, \omega^*) < \max\left\{\mathbb{C}(q^\dagger, \eta_1), \mathbb{C}(q^\dagger, \eta_2)\right\}$. This is precisely the requirement of Equation 4.

We now prove Equation 5. Note that $\mathbb{C}(q, \omega)$ is a function of the vector $\vec{P}[q(\cdot) = \omega] = [P(q(D_1) = \omega), P(q(D_2) = \omega), \dots]$ and so Equation 5 says that the price is invariant to arbitrary scaling of the vector. We will first show that this is true when scaling the vector by $1/k$, where $k$ is an integer. We will then extend it to scaling by rational numbers and then all numbers.

Fix a $q$ and $\omega^* \in \text{range}(q)$. As before, define $q^\dagger$ such that $P(q(D) = \omega) = P(q^\dagger(D) = \omega)$ for all $D \in \mathcal{I}$ and all $\omega \in \text{range}(q)$ that are not equal to $\omega^*$. Let $k$ be a nonnegative integer. For $j = 1, \dots, k$ set $\vec{P}[q^\dagger(\cdot) = \eta_j] = \frac{1}{k}\vec{P}[q(\cdot) = \omega^*]$. Essentially, $q^\dagger$ acts like $q$ with the exception that when $q$ outputs $\omega^*$, $q^\dagger$ will output one of $\eta_1, \dots, \eta_k$ with equal probability. Define $q_A$ to be the function that maps $\eta_1, \dots, \eta_k$ into $\omega^*$ (and otherwise acts like the identity). Clearly $q = q_A \circ q^\dagger$. Also define the randomized function $q_B$ that maps $\omega^*$ to $\eta_1$ with probability $1/k$, to $\eta_2$ with probability $1/k$, etc. Then it is easy to see that $q^\dagger = q_B \circ q$. Thus $q$ and $q^\dagger$ are essentially equivalent and there should be no preference among them. Since the vectors $\vec{P}[q^\dagger(\cdot) = \eta_j]$ (for $j = 1, \dots, k$) are all the same, the prices $\mathbb{C}(q^\dagger, \eta_j)$ are all the same, and it is clear that to avoid arbitrage, we need $\mathbb{C}(q^\dagger, \eta_j) = \mathbb{C}(q, \omega^*)$.

Now choose a rational number $\frac{k_1}{k_2}$. Define queries $q_A, q_B$ and outputs $\omega_1, \omega_2$ such that: $\vec{P}[q_A(\cdot) = \omega_1] = \frac{k_1}{k_2}\vec{P}[q_B(\cdot) = \omega_2]$. Clearly there exists some query $q_C$ and output $\omega_3$ such that $\frac{1}{k_1}\vec{P}[q_A(\cdot) = \omega_1] = \vec{P}[q_C(\cdot) = \omega_3]$ and $\frac{1}{k_2}\vec{P}[q_B(\cdot) = \omega_2] = \vec{P}[q_C(\cdot) = \omega_3]$. Thus $\vec{P}[q_C(\cdot) = \omega_3]$ is an integer rescaling of the other two vectors and by the previous results, we must have $\mathbb{C}(q_A, \omega_1) = \mathbb{C}(q_B, \omega_2) = \mathbb{C}(q_C, \omega_3)$. Thus price is invariant under rescaling the probability vector by a rational number. We extend this to real numbers using the continuity of the pricing function together with the finite dimensionality of the vector space ($\vec{P}[q(\cdot) = \omega]$ is a finite dimensional vector with dimensionality $|\mathcal{I}|$). □

### 4.1.4 Almost-certain arbitrage

One of the prerequisites for avoiding almost-certain arbitrage is continuity, so that a small change in the probabilistic behavior of a query bundle results in small changes to the price. Since price is determined by the vector $\vec{P}[q(\cdot) = \omega)]$, a necessary condition is that the pricing function should be continuous with respect to each component of the vector. Once continuity is established, it is up to the data owner to decide what is the maximal allowable change in price due to a small change in probabilities (for example, by requiring an upper bound on the derivative of the pricing function).

### 4.1.5 Serendipitous Arbitrage

In the case of serendipitous arbitrage, the consumer issues a query bundle $q^* = [q_1, \dots, q_k]$ and receives the (possibly noisy) bundled answers $\omega^* = [\omega_1, \dots, \omega_k]$. We need to ensure that any fact the consumer can deduce from these answers has a price that is at most $\mathbb{C}(q^*, \omega^*)$. We can formalize this idea by considering whether a query $q$ with answer $\omega$ provides any additional information about the true database beyond what we know from query bundle $q^*$ with output $\omega^*$. This concept can be conveniently expressed using Bayesian inference. We need the following definition:

Definition 4.2. *A prior distribution $\theta$ over databases $D \in \mathcal{I}$ is **non-exclusionary** if $P_\theta(D) > 0$ for all $D \in \mathcal{I}$.*

Given a non-exclusionary prior $\theta$, let $P_\theta(\cdot \mid (q_1, \omega_1))$ be the posterior distribution over databases conditioned on the fact that query bundle $q_1$ returned $\omega_1$. Then a new query bundle $q_2$ with output $\omega_2$ provides no new information if it does not change the posterior distribution.[3] When the answer to $q_2$ provides no new information then, one can argue, it should not have a higher price. This condition for avoiding serendipitous arbitrage is formalized as:

$$\begin{aligned}
\text{If } & P_\theta(\cdot \mid (q_1, \omega_1)) = P_\theta(\cdot \mid (q_1, \omega_1) \wedge (q_2, \omega_2))\\
\text{then } & \quad \mathbb{C}(q_2, \omega_2) \leq \mathbb{C}(q_1, \omega_1)
\end{aligned} \tag{6a}$$

for some non-exclusionary prior $\theta$. Which one should be used? It turns out that this does not matter, as they all will produce the same conditions:

Theorem 4.3. *The condition in Equation 6a is unaffected by the choice of the non-exclusionary prior $\theta$. More specifically the condition in Equation 6a is equivalent to the condition $\mathbb{C}(q_a, \omega_a) \leq \mathbb{C}(q_b, \omega_b)$ where:*

- *$P(q_a(D) = \omega_a) = c$ (for some fixed constant $c$) for all databases $D$ where $P(q_b(D) = \omega_b) > 0$, and*
- *$P(q_a(D) = \omega_a)$ can be arbitrary for all databases $D$ where $P(q_b(D) = \omega_b) = 0$*

[3] Non-exclusionary priors rule out pathological situations where a query bundle $q$ returns an answer $\omega$ that can only be derived from databases that have $P_\theta(D) = 0$.

PROOF. Intuitively, these two conditions mean that $q_a$ behaves identically on all databases for which query $q_b$ might provide the answer $\omega_b$, therefore when $q_a$ returns $\omega_a$ it does not tell us which of the possible input databases is more likely.

Let $\theta$ be a non-exclusionary prior. The condition in Equation 6a can be restated as

$$\frac{P_\theta(D_a \mid (q_1,\omega_1))}{P_\theta(D_b \mid (q_1,\omega_1))} = \frac{P_\theta(D_a \mid (q_1,\omega_1),(q_2,\omega_2))}{P_\theta(D_b \mid (q_1,\omega_1),(q_2,\omega_2))}$$

for every $D_a, D_b \in \mathcal{I}$ where $P(q_1(D_a) = \omega_1) > 0$ and $P(q_1(D_b) = \omega_1) > 0$, since no change in the posterior distribution means that the ratio of the posterior probabilities of possible input databases is unchanged. Equivalently, this means:

$$\frac{P_\theta(D_a)P(q_1(D_a)=\omega_1)}{P_\theta(D_b)P(q_1(D_b)=\omega_1)}$$
$$= \frac{P_\theta(D_a)P(q_1(D_a)=\omega_1)P(q_2(D_a)=\omega_2)}{P_\theta(D_b)P(q_1(D_b)=\omega_1)P(q_2(D_b)=\omega_2)}$$

and, since $P_\theta(D) > 0$ for all $D \in \mathcal{I}$ ($\theta$ is non-exclusionary),

$$\frac{P(q_1(D_a)=\omega_1)}{P(q_1(D_b)=\omega_1)} = \frac{P(q_1(D_a)=\omega_1)P(q_2(D_a)=\omega_2)}{P(q_1(D_b)=\omega_1)P(q_2(D_b)=\omega_2)}$$

for $D_a, D_b \in \mathcal{I}$ where $P(q_1(D_a) = \omega_1) > 0$ and $P(q_1(D_b) = \omega_1) > 0$. It easily follows that $P(q_2(D) = \omega_2) = c$ (for some fixed constant $c$) whenever $P(q_1(D) = \omega_1) > 0$ (and the behavior of $q_2$ can be arbitrary on databases for which $P(q_1(D) = \omega_1) = 0$). Note that this is true regardless of which non-exclusionary $\theta$ was used.

The proof in the other direction is obvious. □

Consider the following example relating Theorem 4.3 to Equation 6a.

EXAMPLE 4.4. *Consider the query $q_1$=*`SELECT name FROM T WHERE grade='A'` *with the answer $\omega_1 = \{Bob, Alice\}$. Let $q_2$ be the query that always returns 1 if table $T$ contains a tuple with name="Bob" and grade="A," and for all other tables it returns a random integer from the Poisson distribution. Clearly, for every database where $q_1$ returns $\omega_1$, $q_2$ behaves in exactly the same way (it returns 1 with probability 1). By Theorem 4.3, if we know that $q_1$ returned $\omega_1$ then we learn nothing if we are told $q_2$ returns 1. Indeed, if we already know $q_1$ returns $\omega_1$, we can already predict that $q_2$ will return 1. Hence, this answer to $q_2$ should not cost more than that answer to $q_1$ – by the condition in Equation 6a, $\mathbb{C}(q_2, 1) \leq \mathbb{C}(q_1, \omega_1)$.*

While this condition is appealing, it turns out that it is too strong – a continuous delayed pricing function $\mathbb{C}(\cdot)$ that satisfies Equation 6a will assign the same price to all non-trivial outputs of all deterministic query bundles (e.g. retrieving one tuple or the whole database will cost the same).

THEOREM 4.5. *Let $\mathcal{I}$ be finite with $|\mathcal{I}| \geq 2$. If $\mathbb{C}(\cdot)$ is continuous and satisfies Equation 6a then there exists a constant c such that $\mathbb{C}(q, \omega) = c$ for all deterministic query bundles q and nontrivial*[4] *$\omega \in$ range$(q)$.*

PROOF. For any proper set $S \subset \mathcal{I}$, define $q_S^\epsilon$ to be the query such that $q_S^\epsilon(D) = 1$ for $D \in S$ and for $D \notin S$, $q_S^\epsilon(D) = 1$ with probability $\epsilon$ and 0 with probability $1 - \epsilon$ (thus $q_S^\epsilon$ is a probabilistic approximation of the indicator function of $S$). If $S_A$ and $S_B$ are disjoint, then according to Theorem 4.3 and Equation 6a, $\mathbb{C}(q_{S_A}^\epsilon, 0) \leq \mathbb{C}(q_{S_B}^0, 1)$. By continuity $\mathbb{C}(q_{S_A}^0, 0) \leq \mathbb{C}(q_{S_B}^0, 1)$. Similarly, $\mathbb{C}(q_{S_A}^\epsilon, 1) \leq \mathbb{C}(q_{S_B}^0, 1)$ and by continuity $\mathbb{C}(q_{S_A}^0, 1) \leq \mathbb{C}(q_{S_B}^0, 1)$. Reversing the roles of $S_A$ and $S_B$ (and later interchanging 0 and 1), we see that if $S_A$ and $S_B$ are disjoint, then $\mathbb{C}(q_{S_A}^0, \omega_1) = \mathbb{C}(q_{S_B}^0, \omega_2)$ for all $\omega_1, \omega_2 \in \{0, 1\}$

Now let $S_A$ and $S_B$ be proper subsets of $\mathcal{I}$ that are not necessarily disjoint. Choose a $D_A \in \mathcal{I} \backslash S_A$ and $D_B \in \mathcal{I} \backslash S_B$. Our results show that

$$\mathbb{C}(q_{S_A}^0, \omega_1) = \mathbb{C}(q_{\{D_A\}}^0, \omega_2) = \mathbb{C}(q_{\{D_B\}}^0, \omega_3) = \mathbb{C}(q_{S_B}^0, \omega_4)$$

In other words, all answers to Boolean queries have the same price.

Now let $q$ be a deterministic query bundle with nontrivial output $\omega$. Let $S = \{D : q(D) = \omega\}$. Then $\vec{P}[q(\cdot) = \omega] = \vec{P}[q_S^0(\cdot) = 1]$ and hence $\mathbb{C}(q, \omega) = \mathbb{C}(q_S^0, 1)$. Since $q_S^0$ is a Boolean query, this means that all nontrivial answers to deterministic queries have the same price. □

The key to Theorem 4.5 is that we must assign prices for certain randomized queries $q_S$ (in the proof they were variations of randomized response [12] but other queries can also be used). Continuity (for avoiding almost-certain arbitrage) and Equation 6a created relations between the prices of answers to $q_S$ and the prices of deterministic query bundles. It turned out these relations had undesirable consequences. This is a typical occurrence in the negative results we demonstrate in Section 4.2.

In light of this result, we can slightly weaken our requirements for avoiding serendipitous arbitrage by insisting that $q_2$ be deterministic:

$$\text{If } P_\theta(\cdot \mid (q_1,\omega_1)) = P_\theta(\cdot \mid (q_1,\omega_1) \wedge (q_2,\omega_2)) \text{ and } q_2 \text{ is}$$
$$\text{deterministic then} \quad \mathbb{C}(q_2,\omega_2) \leq \mathbb{C}(q_1,\omega_1) \qquad \text{(6b)}$$

for some non-exclusionary prior $\theta$. The motivation for this relaxed condition is that if $q_2$ is deterministic, then a query answer $\omega_2$ tells us which databases are possible (i.e. all $D$ where $q_2(D) = \omega_2$). We can obtain the same type of information from a randomized query $q_1$ with output $\omega_1$ by determining $\{D : P(q_1(D) = \omega) \neq 0\}$. So if the query/answer pair $(q_2, \omega_{,2})$ does not rule out any additional possible worlds, then its price should not be more than that of the query/answer pair $(q_1, \omega_1)$. Note that Equation 6b covers standard deterministic notions about logical inference about queries: if, from the query/answer pair $(q_1, \omega_1)$ we can deduce that $q_2$ will return $\omega_2$, the $\mathbb{C}(q_2, \omega_2) \leq \mathbb{C}(q_1, \omega_1)$.

## 4.2 Pricing Functions & Negative Results

We first provide a positive result – a nontrivial class of arbitrage-free delayed-pricing functions that have an intuitive motivation.

DEFINITION 4.6. *For each $D \in \mathcal{I}$, let $w_D$ be a nonnegative weight. For any query bundle q and possible output $\omega \in$ range$(q)$, define:*

$$\mathbb{C}_{max}(q,\omega) = \sum_{D \in \mathcal{I}} w_D \left(1 - \frac{P(q(D) = \omega)}{\max\limits_{D' \in \mathcal{I}} P(q(D') = \omega)}\right)$$

[4] We say that $\omega$ is trivial if $P(q(D) = \omega)$ is the same for all $D$; since a trivial $\omega$ provides no information about the true dataset, it should have 0 cost according to Equation 2.

*If $\mathcal{I}$ is countably infinite, the sum of the $w_D$ must be finite. In case $\mathcal{I}$ has a continuous domain, then the summations are replaced with integrals.*

A pricing function of this form assigns a value $w_D$ to the piece of knowledge that the true database is not $D$. If $q$ is a deterministic query bundle and $\omega$ is one of its outputs, then the price $\mathbb{C}_{\max}(q, \omega)$ is equal to the sum of the weights of databases $D$ that have been ruled out (i.e. for which $q(D) \neq \omega$). In the case of a randomized query bundle $q^*$, which tells us which databases are more likely than others, the price $\mathbb{C}_{\max}(q^*, \omega)$ depends on the degree to which a database becomes less likely to produce $\omega$.

Theorem 4.7. *The delayed pricing function $\mathbb{C}_{max}(\cdot)$ satisfies Equations 1, 2, 3, 4, 5, and 6b.*

Proof. It is clear that $\mathbb{C}_{\max}(\cdot)$ satisfies Equations 1, 2, and 5. To prove it satisfies Equation 3, let $q^*$ be the query bundle $[q_1, q_2]$. Then $P(q^*(D) = [\omega_1, \omega_2]) = P(q_1(D) = \omega_1)P(q_2(D) = \omega_2)$ and so $\mathbb{C}_{\max}(q^*, [\omega_1, \omega_2])$ equals:

$$
\begin{aligned}
&\sum_{D\in\mathcal{I}} w_D \left(1 - \frac{P(q_1(D) = \omega_1)P(q_2(D) = \omega_2)}{\max\limits_{D'\in\mathcal{I}} P(q_1(D') = \omega_1)P(q_2(D') = \omega_2)}\right)\\
&\leq \sum_{D\in\mathcal{I}} w_D \left(1 - \frac{P(q_1(D) = \omega_1)P(q_2(D) = \omega_2)}{\max\limits_{D_1\in\mathcal{I}} P(q_1(D_1) = \omega_1) \max\limits_{D_2\in\mathcal{I}} P(q_2(D_2) = \omega_2)}\right)\\
&\leq \sum_{D\in\mathcal{I}} w_D \left(2 - \frac{P(q_1(D) = \omega_1)}{\max\limits_{D_1\in\mathcal{I}} P(q_1(D_1) = \omega_1)} - \frac{P(q_2(D) = \omega_2)}{\max\limits_{D_2\in\mathcal{I}} P(q_2(D_2) = \omega_2)}\right)\\
&= \mathbb{C}_{\max}(q_1, \omega_1) + \mathbb{C}_{\max}(q_2, \omega_2)
\end{aligned}
$$

where the last inequality comes from the fact $1 - ab \leq 2 - a - b$ when $a, b \in [0, 1]$. For Equation 4, suppose that $q, q_A, q_B$ and $\omega$ are such that $P(q(D) = \omega) = P(q_A(D) = \omega) + P(q_B(D) = \omega)$ for all $D \in \mathcal{I}$. Then $\mathbb{C}_{\max}(q, \omega)$ equals:

$$
\begin{aligned}
&\sum_{D\in\mathcal{I}} w_D \left(1 - \frac{P(q_A(D) = \omega) + P(q_B(D) = \omega)}{\max\limits_{D'\in\mathcal{I}} [P(q_A(D') = \omega) + P(q_B(D') = \omega)]}\right)\\
&\leq \sum_{D\in\mathcal{I}} w_D \left(1 - \frac{P(q_A(D) = \omega) + P(q_B(D) = \omega)}{\max\limits_{D_1\in\mathcal{I}} P(q_A(D_1) = \omega) + \max\limits_{D_2\in\mathcal{I}} P(q_B(D_2) = \omega)}\right)\\
&= (\sum_{D\in\mathcal{I}} w_D) - \frac{\sum\limits_{D\in\mathcal{I}} w_D P(q_A(D) = \omega) + \sum\limits_{D\in\mathcal{I}} w_D P(q_B(D) = \omega)}{\max\limits_{D_1\in\mathcal{I}} P(q_A(D_1) = \omega) + \max\limits_{D_2\in\mathcal{I}} P(q_B(D_2) = \omega)}\\
&\leq \sum_{D\in\mathcal{I}} w_D\\
&\qquad - \min\left\{\frac{\sum\limits_{D\in\mathcal{I}} w_D P(q_A(D) = \omega)}{\max\limits_{D_1\in\mathcal{I}} P(q_A(D_1) = \omega)}, \frac{\sum\limits_{D\in\mathcal{I}} w_D P(q_B(D) = \omega)}{\max\limits_{D_2\in\mathcal{I}} P(q_B(D_2) = \omega)}\right\}\\
&= \max\{\mathbb{C}_{\max}(q_A, \omega), \mathbb{C}_{\max}(q_B, \omega)\}
\end{aligned}
$$

Finally, for Equation 6b, choose a non-exclusionary prior $\theta$, two queries $q_1$ and $q_2$ (where $q_2$ is deterministic) and answers $\omega_1$, $\omega_2$ such that

$$P_\theta(\cdot \mid (q_1, \omega_1)) = P_\theta(\cdot \mid (q_1, \omega_1) \wedge (q_2, \omega_2))$$

It is easy to see that this implies:

$$
\begin{aligned}
\{D \,:\, P(q_2(D) = \omega_2) = 1\} &= \{D \,:\, P(q_2(D) = \omega_2) > 0\}\\
&\supseteq \{D \,:\, P(q_1(D) = \omega_1) > 0\}
\end{aligned}
$$

Since $q_2$ is deterministic, $\mathbb{C}(q_2, \omega_2)$ is the sum of the weights of the databases for which $q_2(D) \neq \omega_2$. Now, $\mathbb{C}(q_1, \omega_1)$ is at least as large because of the form of the pricing function and the fact

$$
\begin{aligned}
&\{D \,:\, q_2(D) = \omega_2\} \supseteq \{D \,:\, P(q_1(D) = \omega_1) > 0\}\\
&\quad\Rightarrow\ \{D \,:\, q_2(D) \neq \omega_2\} \subseteq \{D \,:\, P(q_1(D) = \omega_1) = 0\}
\end{aligned}
$$

□

Computing the price, especially in the presence of database constraints, is computationally challenging and appropriate approximation algorithms are an area of future work.

### 4.2.1 Extending Tuple-based Pricing

A pay-per-tuple pricing scheme [1] is a simple coarse-grained scheme where a user purchases a subset of tuples from a table and pays $c$ units for every tuple returned. In this section, we provide a negative result for fine-grained arbitrage-free pricing functions that extend pay-per-tuple pricing to general queries.

First, we formalize what it means to extend the pay-per-tuple pricing scheme.

Definition 4.8. *Let $\mathcal{TUP}$ be the domain of tuples. For any set $A \subseteq \mathcal{TUP}$, let $Q_A(T)$ be the query that returns "true" if table $T$ contains all tuples from $A$ and "false" otherwise.*

Definition 4.9. *A delayed pricing function $\mathbb{C}(\cdot)$ is a* pay-per-tuple extension *if there exists a constant $c > 0$ such that $\mathbb{C}(Q_A, \text{"true"}) = c|A|$ for all nonempty sets of tuples $A$.*

We feel that these are minimal conditions needed to define pay-per-tuple pricing schemes since they formalize the condition that it costs $ck$ units to learn that $k$ specific tuples are in a database. By examining the probability vector $\vec{P}[q(\cdot) = \omega]$, it is clear that these conditions are satisfied when we charge $c$ units for each tuple returned by a `SELECT` query.

As a motivating example, consider the following method for generating a pay-per-tuple extension $\mathbb{C}(\cdot)$ that can price arbitrary queries: figure out the subset of tuples that are needed to answer the query and charge the consumer $c$ times the size of this subset (effectively forcing the user to buy all of those tuples). For example, an answer to the count query `SELECT COUNT(*) FROM T WHERE T.state='NY'` would cost the same as the answer to the more detailed query `SELECT * FROM T WHERE T.state='NY'` because the set of tuples from New York is the smallest set from which the count query can be answered. By similar reasoning, to obtain the size of the table (`SELECT COUNT(*) FROM T`), a consumer would have to pay $c$ times the size of the table. Under this pricing scheme, aggregate queries are quite expensive and a consumer is better off asking for subsets of the tables instead (more information for the same price).

It turns out that aggregate queries are even more problematic than this example illustrates. A pay-per-tuple extension $\mathbb{C}(\cdot)$ that avoids arbitrage will have to set a high price for any query answer that reveals that a table is empty. As the proof of the following theorem shows, the conclusion follows because of the prices that need to be assigned to certain aggregate queries.

Theorem 4.10. *Let $\mathcal{TUP}$ be a finite tuple domain and let the set of all database instances $\mathcal{I}$ be the collection of all subsets of $\mathcal{TUP}$ (so that each database instance is a table*

*$T$ with distinct rows). Let $\mathbb{C}(\cdot)$ extend pay-per-tuple-pricing with constant $c$ (as in Definition 4.9). Let $N$ be the size of the tuple domain. Then, if $T$ is an empty table, the price of learning that $T$ contains no tuples is at least $c(N-1)$.*

PROOF. Without loss of generality, let $\mathbb{C}(\cdot)$ extend pay-per-tuple pricing with constant $c = 1$. Consider the aggregate queries $q_1$ =`SELECT abs(N/2 - count(*)) FROM T` and $q_2$ =`SELECT count(*) FROM T`. The query/answer pair $(q_2, 0)$ implies the pair $(q_1, N/2)$ (that is, if we know $q_2$ returned 0 then we can determine that $q_1$ will return $N/2$). Similarly the query/answer pair $(q_2, N)$ implies the pair $(q_1, N/2)$. By the condition in Equation 6b, we must have

$$\begin{aligned} \mathbb{C}(q_2, 0) &\geq \mathbb{C}(q_1, N/2) \\ \mathbb{C}(q_2, N) &\geq \mathbb{C}(q_1, N/2) \end{aligned}$$

Furthermore,

$$\mathbb{C}(q_2, N) = N$$

since it is equivalent to a query/answer pair $(Q_{\mathcal{TUP}}, \text{true})$ (see Definition 4.8) which tells us that every tuple is in the table and so must have price $N$.

Now, for any tuple $t^*$, let $Q_{\{t^*\}}$ be the query that returns `true` if table $T$ contains tuple $t^*$ (note that $\mathbb{C}(Q_{\{t^*\}}, \text{true}) = 1$). Now, receiving the answer $N$ for query $q_2$ is equivalent to the query bundle $[q_1, Q_{\{t^*\}}]$ returning the bundled answers $N/2$ and `true`. To avoid separate-account arbitrage (Equation 3), we therefore must have

$$\mathbb{C}(q_2, N) \leq \mathbb{C}(q_1, N/2) + \mathbb{C}(Q_{\{t^*\}}, \text{true})$$

Combined with our previous results, we get:

$$\begin{aligned} N &\leq \mathbb{C}(q_1, N/2) + 1 \\ \mathbb{C}(q_2, 0) &\geq \mathbb{C}(q_1, N/2) \geq N - 1 \end{aligned}$$

Therefore learning that the table is empty ($q_2$ returning 0) has a substantial cost. We note that various additional variations of this result are possible, such as when the database size $n$ is already known. □

### 4.2.2 Bayesian Pricing

Given an answer $\omega$ to a query bundle $q$, we can use probabilistic inference to learn about the original database $D$. Often this inference comes in the form of a posterior distribution $P(D \mid (q, \omega))$. Can we use such a posterior distribution to set prices for query answers?

In this section, we answer the question with a negative result. We show that under general conditions, a delayed pricing scheme $\mathbb{C}(\cdot)$ that can be computed from the posterior distribution will either fail to protect against separate-account arbitrage, or set a large price for queries about a small number of tuples (relative to the price of the entire database). We first formally state the theorem, and then illustrate its consequences.

THEOREM 4.11. *Let the finite set of database instances $\mathcal{I}$ have size $\geq 3$. Let $P_\theta$ be a non-exclusionary prior over $\mathcal{I}$ and let $P_\theta(\cdot \mid (q, \omega))$ denote the posterior distribution over $\mathcal{I}$ (conditioned on $q$ returning $\omega$). Let $\mathbb{C}(\cdot)$ be a continuous delayed pricing function such that $\mathbb{C}(\cdot)$ is a function of the posterior probability – that is, there exists a function $f$ such that $\mathbb{C}(q, \omega) = f(P_\theta(\cdot \mid (q, \omega))$ for all query/answer pairs $(q, \omega)$.[5] If $\mathbb{C}(\cdot)$ prevents separate-account arbitrage (Equation 3), then*

$$\mathbb{C}(q_1, \omega_1) \leq \mathbb{C}(q_2, \omega_2) + \mathbb{C}(q_3, \omega_3)$$

*for every collection of 3 deterministic pairwise incompatible query/answer pairs $(q_1, \omega_1), (q_2, \omega_2), (q_3, \omega_3)$.*[6]

PROOF. Define the sets $\mathcal{I}_1 = \{D : q_1(D) = \omega_1\}$, $\mathcal{I}_2 = \{D : q_2(D) = \omega_2\}$, $\mathcal{I}_3 = \{D : q_3(D) = \omega_3\}$ (and note that they are pairwise disjoint). Define the query/answer pairs $(q_2^\epsilon, \omega_2^\epsilon)$ and $(q_3^\epsilon, \omega_3^\epsilon)$ so that

$$P(q_2^\epsilon(D) = \omega_2^\epsilon) = \begin{cases} \epsilon & \text{if } D \in \mathcal{I}_1 \\ 1 & \text{if } D \in \mathcal{I}_2 \\ 0 & \text{if } D \in \mathcal{I}_3 \end{cases}$$

$$P(q_3^\epsilon(D) = \omega_3^\epsilon) = \begin{cases} \epsilon & \text{if } D \in \mathcal{I}_1 \\ 0 & \text{if } D \in \mathcal{I}_2 \\ 1 & \text{if } D \in \mathcal{I}_3 \end{cases}$$

First, note that $q_1$ is deterministic and also $\mathbb{C}(q_1, \omega_1) = \mathbb{C}([q_2^\epsilon, q_3^\epsilon], [\omega_2^\epsilon, \omega_3^\epsilon])$, because of the equality of the posterior distributions: $P_\theta(\cdot \mid (q_1, \omega_1)) = P_\theta(\cdot \mid ([q_2^\epsilon, q_3^\epsilon], [\omega_2^\epsilon, \omega_3^\epsilon]))$. Second, by continuity of $\mathbb{C}(\cdot)$, both $\mathbb{C}(q_2^\epsilon, \omega_2^\epsilon) \to \mathbb{C}(q_2, \omega_2)$ and $\mathbb{C}(q_3^\epsilon, \omega_3^\epsilon) \to \mathbb{C}(q_3, \omega_3)$ as $\epsilon \to 0$.

Thus, by Equation 3 (for avoiding separate-account arbitrage)

$$\begin{aligned} \mathbb{C}(q_1, \omega_1) &= \mathbb{C}([q_2^\epsilon, q_3^\epsilon], [\omega_2^\epsilon, \omega_3^\epsilon]) \\ &\leq \mathbb{C}(q_2^\epsilon, \omega_2^\epsilon) + \mathbb{C}(q_3^\epsilon, \omega_3^\epsilon) \end{aligned}$$

The result follows by taking the limit as $\epsilon \to 0$. □

Let us explore the consequences of this theorem. Suppose the database contains a table $T$ and let $t_a$, $t_b$ be arbitrary tuples. Let $q_2$ be a query about the truth value of the formula $t_a \in T \wedge t_b \notin T$. Let $q_3$ be a query about the truth value of $t_a \notin T \wedge t_b \in T$. Finally, let $q_1$ be the query about the truth value of $T = D$ (for some specific set $D$ of tuples that contains neither $t_a$ nor $t_b$). The query/answer pairs $(q_1, \texttt{true})$, $(q_2, \texttt{true})$, $(q_3, \texttt{true})$ are all mutually exclusive.

Our first observation is that queries $q_2$ and $q_3$ only affect a small number of tuples. Consequently we may set low prices for answers to these queries, such as $\mathbb{C}(q_2, \texttt{true}) = \mathbb{C}(q_3, \texttt{true}) = 1$ (similarly, we can set $\mathbb{C}(q_2, \texttt{false})$ and $\mathbb{C}(q_3, \texttt{false})$ to be small).

Now, what price should we charge for answers to $q_1$? If $q_1$ returns `true` then an entire table is revealed. Hence, this answer should be expensive. However, if we set prices based on posterior distributions, then modulo its technical conditions, Theorem 4.11 says that the most we can charge is $\mathbb{C}(q_2, \texttt{true}) + \mathbb{C}(q_3, \texttt{true}) = 2$.

Clearly this is an undesirable pricing function, which leaves us the following alternatives:

- Consider many posterior distributions (each one built from a different prior over $\mathcal{I}$), and somehow set the price to be a function of all of them (however, the negative results in the theorem can be generalized to many different classes of priors).

[5] A possible example of such an $f$ can be $f(P(\cdot \mid (q, \omega))) = \max_{D \in \mathcal{I}} P(D \mid (q, \omega))$.

[6] That is, in addition to being deterministic, for all $D \in \mathcal{I}$ and $i \neq j$, if $q_i(D) = \omega_i$ then $q_j(D) \neq \omega_j$ ($q_i$ and $q_j$ can never simultaneously return the answers $\omega_i$ and $\omega_j$ on the same database.)

- Consider priors where only 2 databases are possible (hence, effectively setting $|\mathcal{I}| = 2$).

- Weaken our requirements to sometimes allow various arbitrage situations.

- Use a restricted query language, and only set prices for those queries. Naturally, this language should not include the problematic queries such as $q_2^\epsilon$ from the proof of Theorem 4.11 (it is a variation of the commonly used randomized response [12]) because such can queries can force undesirable relations between other queries.

# 5. INSTANCE-INDEPENDENT PRICING

Recall from Section 2 that in instance-independent pricing, the pricing function $\pounds(\cdot)$ depends on the query bundle but not the true database instance. As such, it must be a function of the probabilities $P(q(D) = \omega)$ (for all $D \in \mathcal{I}$ and $\omega \in \text{range}(q)$). Furthermore, $\pounds(\cdot)$ is publicly available, so that consumers can determine the prices of any query bundles they choose. This means that if $\pounds(\cdot)$ has tunable parameters, they should be chosen independently of the data (to avoid leaking information). For such cases we explain how to select the pricing function parameters in Section 7.

We first formulate conditions needed for $\pounds(\cdot)$ to avoid arbitrage situations described in Section 2.2, and then provide pricing functions satisfying those conditions.

## 5.1 Conditions for Avoiding Arbitrage

Instance-independent pricing has been studied before [8, 7], so many of the conditions (for pricing functions defined over *arbitrary* queries) that appear in this section are direct extensions of previously proposed conditions. However, prior work [8, 7] relaxed those conditions and developed pricing functions for those relaxed conditions. Hence our main contributions for this pricing scheme are pricing functions that satisfy the more stringent conditions for preventing arbitrage.

### *5.1.1 Avoiding price-based arbitrage*

Clearly, instance-independent pricing schemes avoid price-based arbitrage if the following conditions are satisfied for all query bundles $q$:

$$\pounds(q) \geq 0 \tag{7}$$
$$\pounds(q) = 0 \text{ if } q \text{ ignores its input.} \tag{8}$$

### *5.1.2 Avoiding separate-account arbitrage*

The condition for avoiding separate account arbitrage is straightforward:

$$\pounds([q_1, q_2]) \leq \pounds(q_1) + \pounds(q_2) \tag{9}$$

### *5.1.3 Avoiding post-processing arbitrage*

We now consider post-processing arbitrage. Let $q_1$ and $q_2$ be query bundles such that $q_2$ operates on the output of $q_1$. Since the output of the composite query bundle $q \equiv q_2 \circ q_1$ is $q_2(q_1(D))$, this output can be computed from $q_1(D)$. A consumer who wants the answer to $q$ should not pay less than $\pounds(q)$. Thus, avoiding post-processing arbitrage requires:

$$\pounds(q_2 \circ q_1) \leq \pounds(q_1) \tag{10}$$

Note that this requirement is also known as *answerability* [8]. However, the pricing scheme in [8] uses a weaker version of Equation 10 (where $q_1$ is a linear function of the data and $q_2$ is linear in the output of $q_1$).

### *5.1.4 Almost-certain arbitrage*

As with delayed pricing, we require that instance-independent pricing functions be continuous with respect to the quantity $P(q(D) = \omega)$ for each $D$ and $\omega$. Again, it is up to the data owner to decide what is the maximal allowable change in price due to a small change in probabilities.

### *5.1.5 Serendipitous Arbitrage*

Avoiding serendipitous arbitrage for instance-independent query pricing is a tricky matter: the price is unrelated to the actual database instance $D$, but (as discussed in Section 2.2) conditions for the arbitrage opportunity are related to $D$ and the answers returned by queries. Thus it is challenging to formally express the appropriate conditions for $\pounds(\cdot)$ using instance-independent language. Because of these notational difficulties, we propose an indirect route where protections are "inherited" from some delayed-pricing function $\mathbb{C}(\cdot)$ by defining $\pounds^{\mathbb{C}}(q) = \max_{\omega \in \text{range}(q)} \mathbb{C}(q, \omega)$, the maximum delayed-price cost of any output of $q$. First, we show that previously mentioned arbitrage protections are also inherited.

Theorem 5.1. *Let $\pounds^{\mathbb{C}}(q) = \max_{\omega \in \text{range}(q)} \mathbb{C}(q, \omega)$. If $\mathbb{C}(\cdot)$ prevents price-based arbitrage (Equations 1 and 2), separate-account arbitrage (Equation 3), and post-processing arbitrage (Equations 4 and 5), then the instance-independent pricing function $\pounds^{\mathbb{C}}(\cdot)$ will satisfy Equations 7, 8, 9 and 10.*

Proof. Equations 7 and 8 clearly follow from Equations 1 and 2, respectively. To verify Equation 9, let $q = [q_1, q_2]$ and use the condition that $\mathbb{C}(\cdot)$ satisfies Equation 3:

$$\begin{aligned}
\pounds^{\mathbb{C}}(q) &= \max_{\omega \in \text{range}(q)} \mathbb{C}(q, \omega) = \max_{\substack{\omega_1 \in \text{range}(q_1) \\ \omega_2 \in \text{range}(q_2)}} \mathbb{C}([q_1, q_2], [\omega_1, \omega_2]) \\
&\leq \max_{\substack{\omega_1 \in \text{range}(q_1) \\ \omega_2 \in \text{range}(q_2)}} (\mathbb{C}(q_1, \omega_1) + \mathbb{C}(q_2, \omega_2)) \\
&= \max_{\omega_1 \in \text{range}(q_1)} \mathbb{C}(q_1, \omega_1) + \max_{\omega_2 \in \text{range}(q_2)} \mathbb{C}(q_2, \omega_2) \\
&= \pounds^{\mathbb{C}}(q_1) + \pounds^{\mathbb{C}}(q_2)
\end{aligned}$$

To verify Equation 10, let $q = q_2 \circ q_1$. Now, $P(q(D) = \omega) = \sum_{\eta \in \text{range}(q_1)} P(q_1(D) = \eta) P(q_2(\eta) = \omega)$. Therefore $\vec{P}[q(\cdot) = \omega] = \sum_{\eta \in \text{range}(q_1)} \vec{P}[q_1(\cdot) = \eta] P(q_2(\eta) = \omega)$. Since $\mathbb{C}(\cdot)$ satisfies Equations 4 and 5, we can conclude $\mathbb{C}(q, \omega) \leq \max_{\eta \in \text{range}(q_1)} \mathbb{C}(q_1, \eta)$. Thus

$$\pounds^{\mathbb{C}}(q_2 \circ q_1) = \max_{\omega \in \text{range}(q)} \mathbb{C}(q) \leq \max_{\eta \in \text{range}(q_1)} \mathbb{C}(q_1, \eta) = \pounds^{\mathbb{C}}(q_1)$$

□

Thus, properties that can be notationally expressed in terms of $\pounds^{\mathbb{C}}(\cdot)$ are indeed inherited from $\mathbb{C}(\cdot)$. Any additional protections (such as protections against serendipitous arbitrage) that are provided by $\mathbb{C}(\cdot)$ also extend in the worst-case to $\pounds^{\mathbb{C}}(\cdot)$ because of the use of max in the definition.

On the other hand, if the data owner is unconcerned with serendipitous arbitrage (e.g., by deeming it unlikely), then this construction would not be needed, and the owner would only expect $\pounds(\cdot)$ to satisfy Equations 7, 8, 9 and 10. We give examples of such pricing functions in Section 5.2.

Example 5.2. *To illustrate the difference in the two approaches, consider the queries $q_1$ =SELECT COUNT(*) FROM T, $q_2$ =SELECT * FROM T, and the query $q_3$ which returns the result of $q_1$ with probability* 0.99 *and returns the result of $q_2$ with probability* 0.01. *The pricing approach suggested by Theorem 5.1 would minimize the seller's downside risk – query $q_3$ would be priced the same as $q_2$ because of the possibility (however remote) that $q_3$ would return the entire table. The consumer, of course, will generally not be happy to pay a large price for a query that usually just returns the size of the table.*

*On the other hand, if the seller uses a pricing function from Section 5.2, the price of $q_3$ will be a weighted average of the prices of $q_1$ and $q_2$. In fact, it will be closer to the price of $q_1$ because most of the time, $q_3$ will act like a simple count query. The consumer may be happier because sometimes a large amount of information (the entire table) would be returned for a small price.*

The tradeoffs in the previous example are another illustration of the theme in this paper that flexible/fine-grained query pricing is a double-edged sword – on the one hand consumers should only pay for the information they need (specified by the queries they choose), but letting them ask arbitrary queries will create unreasonable prices (if we wish to eliminate arbitrage). As a result, we believe that a middle ground can be achieved if the data seller wisely restricts the possible queries that can be issued. Such a general, but not too general, set of queries is a direction for further research.

## 5.2 Generating Pricing Functions

There is a very strong connection between data pricing and information theory. In particular, it turns out that mutual information [2] can serve as an instance-independent pricing function satisfying Equations 7, 8, 9 and 10. If we view the database instance as a random variable and a query answer as a random variable, then after some mathematical simplifications, their mutual information is equal to:

Definition 5.3. *For each database instance $D \in \mathcal{I}$, let $w_D$ be a nonnegative weight such that $\sum_{D\in\mathcal{I}} w_D = 1$. For any query $q$, define:*

$$\pounds_{MI}(q) = \sum_{D\in\mathcal{I}} \sum_{\omega\in\text{range}(q)} w_D P(q(D)=\omega) \log \frac{P(q(D)=\omega)}{\sum\limits_{D'\in\mathcal{I}} w_{D'} P(q(D')=\omega)}$$

Note that for query pricing applications, the weights $w_D$ should *not* be interpreted as the estimated probabilities of database instances. The reason is that probabilities are often estimated based on the actual database instance $D$. Including these estimated probabilities in a publicly-available pricing function leaks information about $D$ and can lead to arbitrage. In Section 7 we discuss how to set the weights $w_D$ based on anticipated query workloads.

Theorem 5.4. *For any choice of nonnegative $w_D$ (for each $D \in \mathcal{I}$) that sum up to 1, $\pounds_{MI}(\cdot)$ is continuous and satisfies Equations 7, 8, 9 and 10.*

Proof. Continuity, Equations 7 and 8 are well-known properties of mutual information. Equation 10 follows directly from the data-processing inequality for mutual information [2]. To prove Equation 9, let $X$ be a random variable such that $P(X = D) = w_D$ for $D \in \mathcal{I}$. Let $Y$ be the random variable corresponding to the output of $q_1$ (i.e. $P(Y = \omega) = \sum_{D\in\mathcal{I}} P(X = D)P(q_1(D) = \omega)$) and similarly let $Z$ be the random variable corresponding to the output of $q_2$. Finally let $V$ be a random variable corresponding to the output of the bundle $[q_1, q_2]$, which is the joint distribution of $Y$ and $Z$ (i.e. $P(Z = [\omega_1, \omega_2]) = \sum_{D\in\mathcal{I}} P(X = D)P(q_1(D) = \omega_1)P(q_2(D) = \omega_2)$). Letting $H$ represent entropy and $I$ represent mutual information and using standard facts about them [2],

$$\begin{aligned}
\pounds_{\text{MI}}([q_1, q_2]) &= I(X;V) = H(V) - H(V|X) \\
&= H(Y,Z) - H(Y,Z|X) \\
&= H(Y,Z) - H(Y|X) - H(Z|X) \qquad (11) \\
&\le H(Y) + H(Z) - H(Y|X) - H(Z|X) \\
&= I(X;Y) + I(X;Z) = \pounds_{\text{MI}}(q_1) + \pounds_{\text{MI}}(q_2)
\end{aligned}$$

The equality in Equation 11 is true because the joint distribution $P(X, Y, Z) = P(X)P(Y|X)P(Z|X)$ means that $Y$ and $Z$ are conditionally independent given $X$. □

Another class of parametrized pricing functions is:

Definition 5.5. *For every pair of database instances $D, D'$, let $w_D^{D'}$ be a nonnegative weight (possibly equal to 0). Define:*

$$\pounds_{abs}(q) = \sum_{\omega\in\text{range}(q)} \sum_{D,D'\in\mathcal{I}} w_D^{D'} \left|P(q(D)=\omega) - P(q(D')=\omega)\right|$$

Theorem 5.6. *For any choice of nonnegative weights $w_D^{D'}$ (for all database instances $D, D'$), the function $\pounds_{abs}(\cdot)$ is continuous and satisfies Equations 7, 8, 9 and 10.*

Proof. Continuity is obvious and clearly Equations 7 and 8 hold. For Equation 9, note that if $q = [q_1, q_2]$ then $P(q(D) = [\omega_1, \omega_2]) = P(q_1(D) = \omega_1)P(q_2(D) = \omega_2)$. Thus

$$\begin{aligned}
\pounds_{\text{abs}}(q) =& \sum_{\substack{(\omega_1,\omega_2)\in\\ \text{range}(q_1\times q_2)}} \sum_{D,D'\in\mathcal{I}} w_D^{D'} \left| \begin{matrix} P(q_1(D)=\omega_1)P(q_2(D)=\omega_2) \\ -P(q_1(D')=\omega_1)P(q_2(D')=\omega_2) \end{matrix} \right| \\
&\le \sum_{(\omega_1,\omega_2)} \sum_{D,D'\in\mathcal{I}} w_D^{D'} P(q_1(D)=\omega_1) \left| \begin{matrix} P(q_2(D)=\omega_2) \\ -P(q_2(D')=\omega_2) \end{matrix} \right| \\
&\quad + \sum_{(\omega_1,\omega_2)} \sum_{D,D'\in\mathcal{I}} w_D^{D'} P(q_2(D')=\omega_2) \left| \begin{matrix} P(q_1(D)=\omega_1) \\ -P(q_1(D')=\omega_1) \end{matrix} \right| \\
&= \sum_{\omega_2\in\text{range}(q_2)} \sum_{D,D'\in\mathcal{I}} w_D^{D'} \left| \begin{matrix} P(q_2(D)=\omega_2) \\ -P(q_2(D')=\omega_2) \end{matrix} \right| \\
&\quad + \sum_{\omega_1\in\text{range}(q_1)} \sum_{D,D'\in\mathcal{I}} w_D^{D'} \left| \begin{matrix} P(q_1(D)=\omega_1) \\ -P(q_1(D')=\omega_1) \end{matrix} \right|,
\end{aligned}$$

which equals $\pounds_{\text{abs}}(q_1) + \pounds_{\text{abs}}(q_2)$. To prove Equation 10, let $q = q_2 \circ q_1$ and note that, by definition, $\text{range}(q) \subseteq \text{range}(q_2)$.

$$\pounds_{\text{abs}}(q) = \sum_{\substack{\omega\in\\ \text{range}(q_2)}} \sum_{D,D'\in\mathcal{I}} w_D^{D'} \left|P(q(D)=\omega) - P(q(D')=\omega)\right|$$

$$= \sum_{\omega} \sum_{D,D'\in\mathcal{I}} w_D^{D'} \left| \sum_{\omega'\in\text{range}(q_1)} \Big( P(q_1(D)=\omega')P(q_2(\omega')=\omega) - P(q_1(D')=\omega')P(q_2(\omega')=\omega) \Big) \right|$$

$$\leq \sum_{\omega} \sum_{D,D' \in \mathcal{I}} w_D^{D'} \sum_{\omega' \in \text{range}(q_1)} \Big| P(q_1(D) = \omega') P(q_2(\omega') = \omega)$$

$$- P(q_1(D') = \omega') P(q_2(\omega') = \omega) \Big|$$

(pulling out the $P(q_2(\omega') = \omega)$ term and summing over $\omega$)

$$= \sum_{\omega'} \sum_{D,D' \in \mathcal{I}} w_D^{D'} \Big| P(q_1(D) = \omega') - P(q_1(D') = \omega') \Big|$$

$$= \pounds_{\text{abs}}(q_1)$$

□

## 6. DYNAMIC PRICING

In arbitrage free pricing, the price of a query bundle $q = [q_1, \dots, q_k]$ is often strictly less than the sum of the prices of the individual queries $q_i$ in the bundle [5, 8]. Often, a consumer may purchase the answer to $q_1$ and then realize that they also need the answer to $q_2$. At this point, the consumer may feel regret [8, 6] because it would have been cheaper to purchase $q_1$ and $q_2$ as a bundle instead of separately. As a favor to the consumer, the seller may keep a history of the consumer's purchases and dynamically adjust prices to avoid this regret. The standard solution also applies to the pricing functions we have been investigating. Suppose a user previously purchased query $q_1$ and obtained answer $\omega$. In the case of instance-independent pricing, the user has already paid $\pounds(q_1)$, and so the dynamically set price for $q_2$ would be $\pounds([q_1, q_2]) - \pounds(q_1)$. Similarly, for delayed-pricing, the price would be $\mathbb{C}([q_1, q_2], [\omega_1, \omega_2]) - \mathbb{C}(q_1, \omega_1)$. In both cases the cost of purchasing $q_1$ and then $q_2$ at some later time is the same as the cost of purchasing them together.

## 7. ELICITING PRICING FUNCTIONS

The delayed and instance-independent pricing functions we discussed in Sections 4.2 and 5.2 all had tunable parameters. For example, the pricing function $\pounds_{\text{MI}}(\cdot)$, based on mutual information, had parameters $w_D$ that could be interpreted as prior probabilities over database instances $D$. However, we shouldn't use the true database instance $D^*$ to estimate the data-generating distribution (via statistical inference) and use it to assign probabilities $w_D$ to all possible database instances $D$. The reason is that the pricing function and hence each $w_D$ parameter is public knowledge (therefore these parameters could leak information about the true database instance $D^*$ from which they were derived). We provide an alternative.

Given an estimated workload of queries $q_1, \dots, q_k$, the value $\sum_{i=1}^{k} \pounds(q_i)$ is the amount the seller will earn from this workload. The seller may wish to choose parameters to maximize this sum. In the case of mutual information pricing $\pounds_{\text{MI}}(\cdot)$, this sum is a concave function of the parameters $w_D$ and hence convex optimization algorithmic frameworks can be used to derive algorithms for setting the parameters.

This strategy will not work for tuning delayed-pricing functions $\mathbb{C}(\cdot)$ because, in addition to the query workload, we would need to use query answers.[7] However, there are other possibilities. Recall that a consumer may choose to minimize worst-case cost by issuing queries that limit their output (such as restricting the output to at most 20 tuples). If the seller can specify, for each $q_i$ in the workload, the answers that should have similar prices (e.g., they return the same number of tuples), one can set up an optimization problem of choosing the parameters that minimize the dissimilarity in price. We plan to explore this option in future work.

## 8. CONCLUSIONS

In this paper, we studied the security implications of query pricing – how to set prices for data queries while protecting the seller's revenue (preventing arbitrage). We investigated a variety of pricing schemes, proposed arbitrage-free pricing functions, and proved negative results concerning pricing flexibility, reasonableness, and arbitrage-prevention. The negative results stem from the fact that certain queries, when assigned a price, caused undesirable interactions between the prices of other queries. These results point to the need for future research on how to carefully choose a subset of queries that should be priced and made available to consumers.

*Acknowledgments.* This work is supported by NSF grants CNS-1228669 and CCF-1317560 and a gift from Google.

## 9. REFERENCES

[1] M. Balazinska, B. Howe, and D. Suciu. Data markets in the cloud: An opportunity for the database community. In *PVLDB*, 2011.

[2] Thomas M. Cover and Joy A. Thomas. *Elements of information theory*. Wiley-Interscience, New York, NY, USA, 1991.

[3] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, 2006.

[4] A. Ghosh and A. Roth. Selling privacy at auction. In *EC*, 2011.

[5] P. Koutris, P. Upadhyaya, M. Balazinska, B. Howe, and D. Suciu. Query-based data pricing. In *PODS*, 2012.

[6] P. Koutris, P. Upadhyaya, M. Balazinska, B. Howe, and D. Suciu. Toward practical query pricing with querymarket. In *SIGMOD*, 2013.

[7] C. Li, D. Y. Li, G. Miklau, and D. Suciu. A theory of pricing private data. In *ICDT*, 2013.

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

[9] F. McSherry. Privacy integrated queries: An extensible platform for privacy-preserving data analysis. In *SIGMOD*, 2009.

[10] R. Tang, D. Shao, S. Bressan, and P. Valduriez. What you pay for is what you get. In *DEXA*, 2013.

[11] R. Tang, H. Wu, Z. Bao, S. Bressan, and P. Valduriez. The price is right - models and algorithms for pricing data. In *DEXA*, 2013.

[12] S. L. Warner. Randomized response: A survey technique for eliminating evasive answer bias. *JASA*, 1965.

[13] Windows azure marketplace. `http://datamarket.azure.com/`.

[7] Obviously we cannot use the answers that would be generated on the true database instance, since that would leak information about the database.