Query-Based Data Pricing

Paraschos Koutris, Prasang Upadhyaya,
Magdalena Balazinska, Bill Howe, and Dan Suciu
University of Washington, Seattle, WA
{pkoutris,prasang,magda,billhowe,suciu}@cs.washington.edu

ABSTRACT

Data is increasingly being bought and sold online, and Web-based marketplace services have emerged to facilitate these activities. However, current mechanisms for pricing data are very simple: buyers can choose only from a set of explicit views, each with a specific price. In this paper, we propose a framework for pricing data on the Internet that, given the price of a few views, allows the price of any query to be derived automatically. We call this capability "query-based pricing." We first identify two important properties that the pricing function must satisfy, called *arbitrage-free* and *discount-free*. Then, we prove that there exists a unique function that satisfies these properties and extends the seller's explicit prices to all queries. When both the views and the query are Unions of Conjunctive Queries, the complexity of computing the price is high. To ensure tractability, we restrict the explicit prices to be defined only on selection views (which is the common practice today). We give an algorithm with polynomial time data complexity for computing the price of any chain query by reducing the problem to network flow. Furthermore, we completely characterize the class of Conjunctive Queries without self-joins that have PTIME data complexity (this class is slightly larger than chain queries), and prove that pricing all other queries is NP-complete, thus establishing a dichotomy on the complexity of the pricing problem when all views are selection queries.

Categories and Subject Descriptors

H.2.4 [**Systems**]: Relational Databases

General Terms

Algorithms, Economics, Theory

Keywords

Data Pricing, Arbitrage, Query Determinacy

PODS '12, May 21–23, 2012, Scottsdale, Arizona, USA.

1. INTRODUCTION

Whether for market research, targeted product advertisement, or other business decisions, companies commonly purchase data. Increasingly, such data is being bought and sold online. For example, Xignite [30] sells financial data, Gnip [1] provides data from social media, PatientsLikeMe [2] sells anonymized, self-reported patient statistics to pharmaceutical companies, and AggData [6] aggregates various types of data available on the Web. To support and facilitate this online *data market*, Web-based marketplace services have recently emerged: the Windows Azure Marketplace [9] offers over 100 data sources from 42 publishers in 16 categories, and Infochimps [17] offers over 15,000 data sets also from multiple vendors.

Current marketplace services do not support complex ad hoc queries, in part because it is not clear how to assign a price to the result. Instead, sellers are asked to define a fixed set of (possibly parameterized) views and assign each a specific price. This simplistic approach not only forces the seller to try and anticipate every view in which a buyer might be interested, but also forces the buyer to browse a large catalog of views (with possibly unknown redundancies and relationships), then often purchase some superset of the data they actually need. Worse, this pricing model can expose non-obvious arbitrage situations that can allow a cunning buyer to obtain data for less than the advertised price. A better approach, which we explore in this paper, is to allow the seller to assign prices to a manageable number of views, then automatically derive the correct price for *any query*.

Consider an example. CustomLists [13] sells the American Business Database for $399; a customer can also buy the subset of companies that have an e-mail address for $299 or only information about businesses in Washington State for $199. A customer interested in only a set of specific counties in various states may not be willing to pay $399 for data she does not need, and so refuses to buy. In response, the seller might provide a view for each county in every state. However, the relationship between state-based pricing and county-based pricing is difficult for either the seller or the buyer to reason about, and inconsistencies or arbitrage situations may result. For example, if the database does not contain any business information for some fraction of counties in a state, then purchasing the data for the remaining counties could be cheaper, yet could yield the same information content as purchasing the data for the entire state.

Query-based Pricing. To address the above challenge, in this paper, we propose a framework for pricing data on the Internet that allows the seller to assign explicit prices

to only a few views (or sets of views), yet allows the buyer to issue and purchase any query. The price of the query is derived *automatically* from the explicit prices of the views. Thus, buyers have full freedom to choose which query to buy, without requiring the seller to explicitly set prices on an exhaustive catalog of all possible queries. We call this pricing mechanism *query-based pricing*. Our mechanism is based on the economic theory of pricing information products using *versions* [26] (reviewed in Section 5), in the sense that each query corresponds to a version of the original data. Since every query (in a given query language) is a version, our framework allows a large number of versions, and, consequently, appeals to large variety of buyers with a large variety of needs.

Formally, query-based pricing consists of a *pricing function*, which takes as input a database instance and a query (or set of queries) and returns a non-negative real number representing the price. We argue that a reasonable pricing function should satisfy two axioms.

First, the pricing function should be *arbitrage-free*. Consider the USA business dataset: if p is the price for the entire dataset and $p_1, \ldots p_{50}$ the prices for the data in each of the 50 states, then a rational seller would ensure that $p_1 + \ldots + p_{50} \geq p$. Otherwise, no buyer would pay for the entire dataset, but would instead buy all 50 states separately. In general, we say that a pricing function is arbitrage-free if whenever a query q is "determined" by the queries $q_1, \ldots, q_n$, then their prices satisfy $p \leq p_1 + \ldots + p_n$. Determinacy means that the first query can be answered from the latter queries. It should not be confused with query containment: if q is contained in q', it does not mean that the price of q' is related to that of q. For example, q' may return the list of all Fortune 500 companies, while q returns a small subset of companies whose stocks have a "strong buy" recommendation. The seller computes q by semi-joining with a proprietary database, and the buyer cannot answer it only from q'. In this example, while q is contained in q', we expect the price of q to be much higher than the price of q', even though q returns less data.

Second, the pricing function should be *discount-free*. This axiom concerns the way the pricing function is derived from the explicit views and prices set by the seller. When she specifies an explicit price p_i for view V_i, the sellers' intent is to sell the view at a discount over the entire data set: the latter is normally sold at a premium price, $p \gg p_i$. The discount-free axiom requires that the pricing function will not introduce any new discounts additional to those explicitly defined by the seller.

In addition to these two axioms, we argue that the pricing function should also be *monotone* with respect to database updates: when new data items are inserted into the database, the price of a query should not decrease. We show that, in general, the pricing function is not necessarily monotone, but give sufficient conditions under which it is.

In today's data markets, the price of the data reflects only its information value, and does not include any computational costs associated with user queries. We make the same assumption in this paper. We also ignore the computational cost in the definition of arbitrage, assuming the worst-case scenario that a buyer has the capability to determine any query given apropriate views, assuming the task is computationally feasible.

Contributions. In this paper, we present several results on query-based pricing.

Our first result is a simple but fundamental formula for computing an arbitrage-free, discount-free pricing function that agrees with the seller's explicit price points, and for testing whether one exists; if it exists, we call the set of price points *consistent*. To check consistency, it suffices to check that no arbitrage is possible between the explicit price points defined by the seller: there are only finitely many arbitrage combinations, as opposed to the infinitely many arbitrage combinations on all possible queries; hence, consistency is decidable. When the set is consistent, the pricing function is unique, and is given by the *arbitrage-price* formula (Equation 2). This implies an explicit, yet inefficient method for computing the price, which is presented in Section 2.

Second, we turn to the tractability question. We show that even when the seller's explicit price points are restricted to selection queries (which is the common case for data sold online today), computing the price of certain conjunctive queries is NP-hard in the size of the input database. For this reason, we propose a restriction of conjunctive queries, which we call *Generalized Chain Queries*, or GCHQ. These are full conjunctive queries whose atoms can be ordered in a sequence such that for any partition into a prefix and a suffix, the two sets of atoms share at most one variable. GCHQ includes all path joins, like $R(x,y), S(y,z), T(z,u), P(u,v)$, star joins, like $R(x,y), S(x,z,u), T(x,v), P(x,w)$, and combinations. We prove that, when all explicit price points are selection queries, one can compute the price of every GCHQ query in PTIME data complexity. This is the main result of our paper, and provides a practical framework for query-based pricing. The algorithm is based on a non-trivial reduction to the MIN-CUT problem in weighted graphs, which is the dual of the MAX-FLOW problem [12], Subsection 3.1.

Third, we study the complexity of all conjunctive queries without self-joins. We prove that cycle queries (which are *not* generalized chain queries) can also be computed in polynomial time: this is the most difficult result in our paper, and the algorithm is quite different from the algorithm for GCHQ. With this result, we can prove a dichotomy of the data complexity of all conjunctive queries without self-joins, in PTIME or NP-complete, Subsection 3.2.

Our pricing framework is based on a notion of query determinacy. Informally, we say that a set of views V *determines* some query Q if we can compute the answer of Q only from the answers of the views without having access to the underlying database. *Information-theoretic* determinacy, denoted $V \twoheadrightarrow Q$, is discussed by Segoufin and Vianu [25] and by Nash, Segoufin, and Vianu [22, 23], and is a notion that is independent of the database instance; their motivation comes from *local-as-view* data integration and *semantic caching*, where an instance independent rewriting is needed. For query-based pricing, however, the database instance cannot be ignored when checking determinacy, since the price normally depends on the state of the database. For example, consider a query Q_1 that asks for the businesses that are located in both Oregon and Washington State and a query Q_2 that asks for the restaurants located in Oregon, Washington and Idaho. In general, we cannot answer Q_2 if we know the answer of Q_1. But suppose we examine the answer for Q_1 and note that it includes no restaurants: then we can safely determine that Q_2 is empty. We define *instance-based determinacy*, $D \vdash V \twoheadrightarrow Q$, to mean that, for all D' if $V(D') = V(D)$,

then $Q(D) = Q(D')$. Information-theoretic determinacy is equivalent to instance-based determinacy *for every instance* D. We prove several results on the complexity of checking instance-based determinacy: for unions of conjunctive queries, it is Π_2^p, and the data complexity (when V, Q are fixed and the input is only D) is co-NP complete (Theorem 2.3). When the views are restricted to selection queries (which is a case of special interest in query-based pricing), then for any monotone query Q, instance-based determinacy has polynomial time data complexity, assuming Q itself has PTIME data complexity (Theorem 3.3).

The paper is organized as follows. We introduce the query-based pricing framework and give the fundamental formula for checking consistency and computing the pricing function in in Section 2. We turn to the tractability questions in Section 3: we describe our main result consisting of the PTIME algorithm for Generalized Chain Queries Subsection 3.1, and give the dichotomy theorem in Subsection 3.2. We discuss some loose ends in Section 2 and related work in Section 5, then conclude in Section 6.

2. THE QUERY PRICING FRAMEWORK

2.1 Notations

Fix a relational schema $\mathbf{R} = (R_1, \ldots, R_k)$; we denote a database instance with $D = (R_1^D, \ldots, R_k^D)$, and the set of all database instances with $Inst_{\mathbf{R}}$ [19]. In this paper we only consider monotone queries, and we denote $\mathcal{L}$ a fixed query language; in particular, CQ, UCQ are Conjunctive Queries, and Unions of Conjunctive Queries respectively. $Q(D)$ denotes the answer of a query Q on a database D. A *query bundle* is a finite set of queries; we use the term "bundle" rather than "set" to avoid confusion between a set of queries and a set of answers. We denote by $B(\mathcal{L})$ the set of query bundles over $\mathcal{L}$, and write a bundle as $\mathbf{Q} = (Q_1, \ldots, Q_m)$. The *output schema* of a query bundle is $\mathbf{R_Q} = (R_{Q_1}, \ldots, R_{Q_m})$, and consists of one relation name for each query. Thus, a bundle defines a function $\mathbf{Q} : Inst_{\mathbf{R}} \to Inst_{\mathbf{R_Q}}$.

The *identity bundle*, $\mathbf{ID}$, is the bundle that returns the entire dataset, $\mathbf{ID}(D) = (R_1^D, \ldots, R_k^D)$. The *empty bundle* is denoted $()$: it is the empty set of queries, not to be confused with the emptyset query. Given two bundles, $\mathbf{Q}_1$ and $\mathbf{Q}_2$, we denote their union as $\mathbf{Q} = \mathbf{Q}_1, \mathbf{Q}_2$: this is the query bundle consisting of all queries in $\mathbf{Q}_1$ and $\mathbf{Q}_2$, not to be confused with the union $Q_1 \cup Q_2$ of two queries of the same arity.

2.2 The Pricing Function

Definition 2.1 (Pricing Function). *Fix a database instance $D \in Inst_{\mathbf{R}}$. A* static pricing function *is a function $p_D : B(\mathcal{L}) \to \mathbb{R}$.*

A dynamic pricing function *is a partial function $p : Inst_{\mathbf{R}} \to (B(\mathcal{L}) \to \mathbb{R})$, s.t. for each D where p is defined, $p(D)$ is a static pricing function. We write p_D for $p(D)$.*

For now, we allow prices to be negative, but show in Proposition 2.8 that they are always non-negative under reasonable assumptions. The intuition is as follows. If the user asks for the bundle $\mathbf{Q}$, then she has to pay the price $p_D(\mathbf{Q})$, where D is the current database instance. The static pricing function is defined only for the current state of the database D. A dynamic pricing function p allows the database to be updated, and associates a different pricing function p_D to each database; notice that it need not be defined for all instances $D \in Inst_{\mathbf{R}}$. We start with static pricing in this section, and call a static pricing function simply a pricing function; we discuss dynamic pricing in Subsection 2.7.

The price is for an entire query bundle, not just for one query. For example, if a user needs to compute queries Q_1, Q_2, and Q_3, then she could issue them separately, and pay $p_D(Q_1)+p_D(Q_2)+p_D(Q_3)$, but she also has the option of issuing them together, as a bundle, and pay $p_D(Q_1, Q_2, Q_3)$. We will show that, in general, the pricing function is subadditive: the price of a bundle is always lower than the sum of the individual prices.

In the query pricing framework, the seller does not specify the pricing function directly, but gives only a finite set of explicit price points. The system then computes the pricing function on all queries; this function must, furthermore, satisfy two axioms, arbitrage-free and discount-free. In the rest of this section we discuss the details of this framework.

2.3 Axiom 1: Arbitrage-Free

The first axiom that a pricing function must satisfy is defined in terms of a notion of determinacy. Intuitively, a bundle $\mathbf{V}$ determines a bundle $\mathbf{Q}$ given a database D, denoted $D \vdash \mathbf{V} \twoheadrightarrow \mathbf{Q}$, if one can answer $\mathbf{Q}$ from the answer of $\mathbf{V}$ by applying a function f such that $\mathbf{Q}(D) = f(\mathbf{V}(D))$. The impact on pricing is that if the user needs to answer $\mathbf{Q}$, she also has the option of purchasing $\mathbf{V}$, and then applying f. The arbitrage-free axiom requires that $p_D(\mathbf{Q}) \leq p_D(\mathbf{V})$, meaning that the user never has the incentive to compute $\mathbf{Q}$ indirectly by purchasing $\mathbf{V}$. Thus, the notion of arbitrage depends on the notion of determinacy, which we define here:

Definition 2.2 (Instance-based Determinacy). *We say* $\mathbf{V}$ determines $\mathbf{Q}$ *given a database D, denoted $D \vdash \mathbf{V} \twoheadrightarrow \mathbf{Q}$, if for any D', $\mathbf{V}(D) = \mathbf{V}(D')$ implies $\mathbf{Q}(D) = \mathbf{Q}(D')$.*

The connection to answerability is the following. Let $f : Inst_{\mathbf{R_V}} \to Inst_{\mathbf{R_Q}}$ be Q composed with any left inverse of $\mathbf{V}$: that is, for every $E \in Inst_{\mathbf{R_V}}$, if there exists D s.t. $\mathbf{V}(D) = E$, then choose any such D and define $f(E) = \mathbf{Q}(D)$; otherwise, $f(E)$ is undefined. One can check that $D \vdash \mathbf{V} \twoheadrightarrow \mathbf{Q}$ iff $\forall D'.\mathbf{V}(D) = \mathbf{V}(D') \Rightarrow f(\mathbf{V}(D')) = \mathbf{Q}(D')$. Thus, if the user knows $\mathbf{V}(D)$ and $D \vdash \mathbf{V} \twoheadrightarrow \mathbf{Q}$ holds, then she can compute $\mathbf{Q}(D)$ as $f(\mathbf{V}(D))$. The proof of the following theorem is in the full version of the paper [**?**].

Theorem 2.3. *The combined complexity of instance-based determinacy, $D \vdash \mathbf{V} \twoheadrightarrow \mathbf{Q}$, when $\mathbf{V}, \mathbf{Q}$ are in $B(UCQ)$ is in Π_2^p; the data complexity (where $\mathbf{V}, \mathbf{Q}$ are fixed) is co-NP-complete, and remains co-NP complete even for $B(CQ)$.*

We leave open the question whether the bound on the combined complexity is tight. Instance-based determinacy is different from *information-theoretic determinacy*, defined in [22] as follows: $\mathbf{V} \twoheadrightarrow \mathbf{Q}$ if $\forall D : D \vdash \mathbf{V} \twoheadrightarrow \mathbf{Q}$. Information-theoretic determinacy, $\mathbf{V} \twoheadrightarrow \mathbf{Q}$, is undecidable for $B(UCQ)$ and its status is unknown for $B(CQ)$ [22].

Example 2.4. *Let $Q_1(x, y, z) = R(x, y), S(y, z)$, $Q_2(y, z, u) = S(y, z), T(z, u)$ and $Q(x, y, z, u) = R(x, y), S(y, z), T(z, u)$. Then $(Q_1, Q_2) \twoheadrightarrow Q$, since it suffices to define f as the function that joins $Q_1(D)$ and $Q_2(D)$. Then, we have that $Q(D) = f(Q_1(D), Q_2(D))$ for all D. On the other hand, $Q_1 \not\twoheadrightarrow Q$. However, let D be a database instance s.t. $Q_1(D) =$*

$\emptyset$. *Then* $D \vdash Q_1 \twoheadrightarrow Q$, *because we know that* $Q(D) = \emptyset$. *For example, let* f *always return the emptyset: then, for any* D' *s.t.* $Q_1(D) = Q_1(D')(= \emptyset)$ *we have* $Q(D') = f(Q_1(D'))$.

In this paper we use instance-based determinacy to study pricing. However, other options are possible: for example one may use information-theoretic determinacy, or one may use its restriction that we discuss in Subsection 2.7. To keep the framework general, we base our discussion on an abstract notion of determinacy, defined below. Our results in this section apply to any determinacy relation that satisfies this definition, except for complexity results, which are specific to instance-based determinacy. Our results in the next section are specific to instance-based determinacy.

DEFINITION 2.5. *A* determinacy relation *is a ternary relation* $D \vdash \mathbf{V} \twoheadrightarrow \mathbf{Q}$ *that satisfies the following properties:*

Reflexivity: $D \vdash \mathbf{V}_1, \mathbf{V}_2 \twoheadrightarrow \mathbf{V}_1$.
Transitivity: *if* $D \vdash \mathbf{V}_1 \twoheadrightarrow \mathbf{V}_2$ *and* $D \vdash \mathbf{V}_2 \twoheadrightarrow \mathbf{V}_3$, *then* $D \vdash \mathbf{V}_1 \twoheadrightarrow \mathbf{V}_3$.
Augmentation: *if* $D \vdash \mathbf{V}_1 \twoheadrightarrow \mathbf{V}_2$, *then* $D \vdash \mathbf{V}_1, \mathbf{V}' \twoheadrightarrow \mathbf{V}_2, \mathbf{V}'$.
Boundedness: $D \vdash \mathbf{ID} \twoheadrightarrow \mathbf{V}$

We prove in [**?**] that both instance-based and information-theoretic determinacy satisfy this definition. We also have:

LEMMA 2.6. *If* $\twoheadrightarrow$ *is a determinacy relation, then (a)* $D \vdash \mathbf{V} \twoheadrightarrow ()$ *for every bundle* $\mathbf{V}$, *and (b) if* $D \vdash \mathbf{V} \twoheadrightarrow \mathbf{V}_1$ *and* $D \vdash \mathbf{V} \twoheadrightarrow \mathbf{V}_2$, *then* $D \vdash \mathbf{V} \twoheadrightarrow \mathbf{V}_1, \mathbf{V}_2$.

PROOF. The reflexivity axiom $D \vdash \mathbf{V}, () \twoheadrightarrow ()$ proves the first claim, since $\mathbf{V}, () = \mathbf{V}$. For the second, we apply augmentation to $D \vdash \mathbf{V} \twoheadrightarrow \mathbf{V}_1$ and obtain $D \vdash \mathbf{V}, \mathbf{V} \twoheadrightarrow \mathbf{V}, \mathbf{V}_1$; next apply augmentation to $D \vdash \mathbf{V} \twoheadrightarrow \mathbf{V}_2$ and obtain $D \vdash \mathbf{V}, \mathbf{V}_1 \twoheadrightarrow \mathbf{V}_1, \mathbf{V}_2$; transitivity gives us $D \vdash \mathbf{V}, \mathbf{V} \twoheadrightarrow \mathbf{V}_1, \mathbf{V}_2$, which proves the claim because $\mathbf{V}, \mathbf{V} = \mathbf{V}$. □

The Arbitrage-Free Axiom. We can now state the first axiom that a pricing function must satisfy:

DEFINITION 2.7 (ARBITRAGE-FREE). *A pricing function* p_D *is* arbitrage-free *if, whenever* $D \vdash \mathbf{Q}_1, \ldots, \mathbf{Q}_k \twoheadrightarrow \mathbf{Q}$, *then* $p_D(\mathbf{Q}) \leq \sum_i p_D(\mathbf{Q}_i)$.

Of course, even if $\mathbf{Q}_1, \ldots, \mathbf{Q}_k$ determine $\mathbf{Q}$, it may be non-trivial for the buyer to compute the answer of $\mathbf{Q}$ from the answers of $\mathbf{Q}_1, \ldots, \mathbf{Q}_k$, for two reasons: she first needs to find the function f for which $f(\mathbf{Q}_1(D), \ldots, \mathbf{Q}_k(D)) = \mathbf{Q}(D)$, and, second, it may be computationally expensive to evaluate f. In this paper, however, we do not address the economic cost of the computation, focusing only on the information-theoretic aspect; i.e. we assume that the only cost that matters is that of the data itself. Thus, if an arbitrage condition exists, then the buyer will exploit it, by avoiding to pay $p_D(\mathbf{Q})$ and purchasing $\mathbf{Q}_1, \ldots, \mathbf{Q}_k$ instead, then computing $\mathbf{Q}$ (at no extra cost).

Arbitrage-free pricing functions exists: the trivial function $p_D(\mathbf{Q}) = 0$, for all $\mathbf{Q}$, is arbitrage-free; we will show non-trivial functions below. First, we prove some properties.

PROPOSITION 2.8. *Any arbitrage-free pricing function* p_D *has the following properties:*

1. *Subadditive:* $p_D(\mathbf{Q}_1, \mathbf{Q}_2) \leq p_D(\mathbf{Q}_1) + p_D(\mathbf{Q}_2)$.
2. *Non-negative:* $p_D(\mathbf{Q}) \geq 0$.
3. *Not asking*[1] *is free:* $p_D() = 0$.
4. *Upper-bounded:* $p_D(\mathbf{Q}) \leq p_D(\mathbf{ID})$.

PROOF. We apply arbitrage-freeness to two instances of the reflexivity property. First to $D \vdash \mathbf{Q}_1, \mathbf{Q}_2 \twoheadrightarrow \mathbf{Q}_1, \mathbf{Q}_2$, and derive $p_D(\mathbf{Q}_1, \mathbf{Q}_2) \leq p_D(\mathbf{Q}_1) + p_D(\mathbf{Q}_2)$, which proves item 1. Next to $D \vdash \mathbf{Q}, \mathbf{Q}' \twoheadrightarrow \mathbf{Q}'$, and derive $p_D(\mathbf{Q}') \leq p_D(\mathbf{Q}) + p_D(\mathbf{Q}')$, which implies $p_D(\mathbf{Q}) \geq 0$, proving item 2. For item 3, take $\mathbf{Q} = ()$ and $k = 0$ in Definition 2.7: then $D \vdash \mathbf{Q}_1, \ldots, \mathbf{Q}_k \twoheadrightarrow \mathbf{Q}$ holds by reflexivity ($D \vdash () \twoheadrightarrow ()$) and $p_D(\mathbf{Q}) \leq \sum_i p_D(\mathbf{Q}_i)$ implies $p_D() \leq 0$. Also, arbitrage-freeness applied to the boundedness axiom $D \vdash \mathbf{ID} \twoheadrightarrow \mathbf{Q}$ proves item 4. □

2.4 Explicit Price Points

It is difficult to specify a non-trivial arbitrage-free pricing function, and we do not expect the seller to define such a function herself. Instead, the data seller specifies a set of explicit price points, and the system extrapolates them to a pricing function on all query bundles. A *price point* is a pair consisting of a *view* (query bundle) and a *price* (positive real number).

DEFINITION 2.9 (PRICE POINTS). *A* price point *is a pair* $(\mathbf{V}, p)$, *where* $\mathbf{V} \in B(\mathcal{L})$ *and* $p \in \mathbb{R}^+$. *We denote a finite set of price points* $\mathcal{S}$ *as* $\{(\mathbf{V}_1, p_1), \ldots, (\mathbf{V}_m, p_m)\}$.

We will assume that $D \vdash (\mathbf{V}_1, \ldots, \mathbf{V}_m) \twoheadrightarrow \mathbf{ID}$; i.e., the seller is always willing to sell the entire dataset, perhaps indirectly through other views. This is a reasonable assumption: if the seller does not wish to sell certain parts of the data, we can simply not model those parts by removing relation names from the schema or removing tuples from the instance. To simplify the discussion, in this section we assume that $(\mathbf{ID}, B) \in \mathcal{S}$; i.e., $\mathbf{ID}$ is sold explicitly at a (high) premium price B. We relax this assumption in Section 3.

DEFINITION 2.10 (VALIDITY). *A pricing function* p_D *is* valid *w.r.t. a set* $\mathcal{S}$ *of price points if:*

1. p_D *is arbitrage-free.*
2. $\forall (\mathbf{V}_i, p_i) \in \mathcal{S}$, $p_D(\mathbf{V}_i) = p_i$.

Our goal is to compute a valid pricing function for a set $\mathcal{S}$. In general, such a function may not exist; if it exists, then we call $\mathcal{S}$ consistent.

DEFINITION 2.11 (CONSISTENCY). *A set of price points* $\mathcal{S}$ *is* consistent *if it admits a valid pricing function.*

2.5 Axiom 2: Discount-Free

To see the intuition behind the second axiom, recall that B is the price set by the seller for the entire dataset. Any arbitrage-free pricing function will be $\leq B$, by Proposition 2.8 (item 4). The explicit price points in $\mathcal{S}$ can be viewed as *discounts* offered by the seller relative to the price that would be normally charged if that price point were not included in $\mathcal{S}$. The second axiom requires that a pricing function makes no additional implicit discounts.

DEFINITION 2.12 (DISCOUNT-FREE). *A valid pricing function* p_D *for* $\mathcal{S}$ *is called* discount-free *if for every valid pricing function* p'_D *for* $\mathcal{S}$ *we have:* $\forall \mathbf{Q} \in B(\mathcal{L})$, $p'_D(\mathbf{Q}) \leq p_D(\mathbf{Q})$.

[1] $p_D()$ means $p_D(())$, the price of the empty bundle.

A discount-free pricing function is unique, because if both p_D and p'_D are discount-free, then we have both $p_D \leq p'_D$ and $p'_D \leq p_D$, hence $p_D = p'_D$. We will show that, if $\mathcal{S}$ is consistent, then it admits a discount-free pricing function.

2.6 The Fundamental Query Pricing Formula

The fundamental formula gives an explicit means for checking consistency and for computing the discount-free price. The formula associates with any $\mathcal{S}$ (not necessarily consistent) a pricing function, called *arbitrage-price*; if $\mathcal{S}$ is consistent, then the arbitrage-price is the unique valid, discount-free pricing function, and if $\mathcal{S}$ is inconsistent, we can use arbitrage-price to detect it.

If $\mathbf{Q}_i$, $i = 1, 2 \ldots, k$, are query bundles, then denote their union as $\bigodot_i \mathbf{Q}_i = \mathbf{Q}_1, \ldots, \mathbf{Q}_k$. If $\mathcal{C} \subseteq \mathcal{S}$ is a set of price points, then we denote its total price as $p(\mathcal{C}) = \sum_{(\mathbf{V}_i, p_i) \in \mathcal{C}} p_i$.

Fix a price points set $\mathcal{S}$ and an instance D. The *support* of a query bundle $\mathbf{Q}$ is:

$$supp_D^{\mathcal{S}}(\mathbf{Q}) = \{\mathcal{C} \subseteq \mathcal{S} \mid D \vdash \bigodot_{(\mathbf{V},p) \in \mathcal{C}} \mathbf{V} \twoheadrightarrow \mathbf{Q}\} \quad (1)$$

The support is non-empty, because we assumed that $\mathcal{S}$ contains **ID**. This allows us to define:

Definition 2.13 (Arbitrage-price). *The* arbitrage-price *of a query bundle* $\mathbf{Q}$ *is:*

$$p_D^{\mathcal{S}}(\mathbf{Q}) = \min_{\mathcal{C} \in supp_D^{\mathcal{S}}(\mathbf{Q})} p(\mathcal{C}) \quad (2)$$

The arbitrage-price represents the strategy of a savvy buyer: to purchase $\mathbf{Q}$, buy the cheapest support $\mathcal{C}$ for $\mathbf{Q}$, meaning the cheapest set of views that determine $\mathbf{Q}$. We prove:

Lemma 2.14. *(a) For all* $(\mathbf{V}_i, p_i) \in \mathcal{S}$, $p_D^{\mathcal{S}}(\mathbf{V}_i) \leq p_i$. *In other words, the arbitrage-price is never larger than the explicit price. (b) The arbitrage-price* $p_D^{\mathcal{S}}$ *is arbitrage-free.*

Proof. The first claim follows from the fact that $\{(\mathbf{V}_i, p_i)\} \in supp_D^{\mathcal{S}}(\mathbf{V}_i)$, because of the reflexivity axiom $D \vdash \mathbf{V}_i \twoheadrightarrow \mathbf{V}_i$. For the second claim, consider $D \vdash \mathbf{Q}_1, \ldots, \mathbf{Q}_k \twoheadrightarrow \mathbf{Q}$; we will prove that $p_D^{\mathcal{S}}(\mathbf{Q}) \leq \sum_i p_D^{\mathcal{S}}(\mathbf{Q}_i)$. For $i = 1, \ldots, k$, let $\mathcal{C}_i^m = \arg\min_{\mathcal{C} \in supp_D^{\mathcal{S}}(\mathbf{Q}_i)} p(\mathcal{C})$. By definition, $D \vdash \bigodot_{(\mathbf{V}_j, p_j) \in \mathcal{C}_i^m} \mathbf{V}_j \twoheadrightarrow \mathbf{Q}_i$ and $p_D^{\mathcal{S}}(\mathbf{Q}_i) = p(\mathcal{C}_i^m)$. Let $\mathcal{C} = \bigcup_i \mathcal{C}_i^m \subseteq \mathcal{S}$. Let $\mathbf{V}^m = \bigodot_{(\mathbf{V}_j, p_j) \in \mathcal{C}} \mathbf{V}_j$. Since $\mathcal{C}_i^m \in supp_D^{\mathcal{S}}(\mathbf{Q}_i)$, it follows that $\mathcal{C} \in supp_D^{\mathcal{S}}(\mathbf{Q}_i)$ because the set $supp_D^{\mathcal{S}}(\mathbf{Q}_i)$ is upwards closed[2]. It follows that $D \vdash \mathbf{V}^m \twoheadrightarrow \mathbf{Q}_i$, for every $i = 1, k$. By inductively applying Lemma 2.6 (b), we derive $D \vdash \mathbf{V}^m \twoheadrightarrow \mathbf{Q}_1, \ldots, \mathbf{Q}_k$ and, by transitivity, we further derive $D \vdash \mathbf{V}^m \twoheadrightarrow \mathbf{Q}$. This implies $\mathcal{C} \in supp_D^{\mathcal{S}}(\mathbf{Q})$, and therefore:

$$p_D^{\mathcal{S}}(\mathbf{Q}) \leq p(\mathcal{C}) = \sum_{(V_j, p_j) \in \mathcal{C}} p_j \leq \sum_i \sum_{(\mathbf{V}_j, p_j) \in \mathcal{C}_i^m} p_j = \sum_i p_D^{\mathcal{S}}(\mathbf{Q}_i)$$

The second inequality holds because the p_i's are non-negative (Proposition 2.8). This proves that $p_D^{\mathcal{S}}$ is arbitrage-free. □

The arbitrage-price is our fundamental formula because it allows us to check consistency, and, in that case, it gives the discount-free price.

Theorem 2.15. *Consider a set of price points* $\mathcal{S}$. *Let* $p_D^{\mathcal{S}}$ *denote the arbitrage-price function (Equation 2). Then:*

1. $\mathcal{S}$ *is consistent iff* $\forall (\mathbf{V}_i, p_i) \in \mathcal{S}$, $p_i \leq p_D^{\mathcal{S}}(\mathbf{V}_i)$.
2. *If* $\mathcal{S}$ *is consistent, then* $p_D^{\mathcal{S}}$ *is the unique discount-free pricing function for* $\mathcal{S}$.

Proof. We claim that, for any pricing function p_D valid for $\mathcal{S}$ and every query bundle $\mathbf{Q}$, we have that $p_D(\mathbf{Q}) \leq p_D^{\mathcal{S}}(\mathbf{Q})$. The claim proves the theorem. Indeed, the "if" direction of item 1 follows from two facts. First, $p_D^{\mathcal{S}}$ is arbitrage-free by Lemma 2.14(b). Second, if $p_i \leq p_D^{\mathcal{S}}(\mathbf{V}_i)$ holds for all price points $(\mathbf{V}_i, p_i) \in \mathcal{S}$, then by Lemma 2.14(a) $p_D^{\mathcal{S}}(\mathbf{V}_i) = p_i$. Hence, $p_D^{\mathcal{S}}$ is valid, proving that $\mathcal{S}$ is consistent. The "only if" direction follows from the claim: if p_D is any valid pricing function for $\mathcal{S}$ then $p_i = p_D(\mathbf{V}_i) \leq p_D^{\mathcal{S}}(\mathbf{V}_i)$. The claim also implies item 2 immediately.

To prove the claim, let p_D be a valid pricing function (thus $p_D(\mathbf{V}_i) = p_i$ for all $(\mathbf{V}_i, p_i) \in \mathcal{S}$), and let $\mathbf{Q}$ be a bundle. Let $\mathcal{C} \in supp_D^{\mathcal{S}}(\mathbf{Q})$, and $\mathbf{V} = \bigodot_{(\mathbf{V}_i, p_i) \in \mathcal{C}} \mathbf{V}_i$. By definition we have $D \vdash \mathbf{V} \twoheadrightarrow \mathbf{Q}$. Since p_D is arbitrage-free, we have:

$$p_D(\mathbf{Q}) \leq \sum_{(\mathbf{V}_i, p_i) \in \mathcal{C}} p_D(\mathbf{V}_i) = \sum_{(\mathbf{V}_i, p_i) \in \mathcal{C}} p_i = p(\mathcal{C})$$

It follows that $p_D(\mathbf{Q}) \leq \min_{\mathcal{C} \in supp_D^{\mathcal{S}}(\mathbf{Q})} p(\mathcal{C}) = p_D^{\mathcal{S}}(\mathbf{Q})$ □

The theorem says that, in order to check consistency, it suffices to rule out arbitrage situations among the views in $\mathcal{S}$. There are infinitely many possible arbitrage situations in Definition 2.7, but the theorem reduces this to a finite set.

Next, we examine the complexity of checking consistency and computing the price. For this discussion, we will assume that $\twoheadrightarrow$ is the instance-based determinacy given by Definition 2.2. Denote by Price$(\mathcal{S}, \mathbf{Q})$ the decision version of the price computation problem: "given a database D and k, is the price $p_D^{\mathcal{S}}(\mathbf{Q})$ less than or equal to k"? Let us also denote by Price$(\mathbf{Q})$ the decision version of the same problem, but where the set of price points $\mathcal{S}$ is now part of the input.

Corollary 2.16. *Suppose* $\mathcal{S}, \mathbf{Q}$ *consist of UCQs. Then, (a) the complexity of* Price$(\mathbf{Q})$ *is in* Σ_2^p *and (b) the complexity of* Price$(\mathcal{S}, \mathbf{Q})$ *is coNP-complete.*

Proof. For (a), to check whether $p_D^{\mathcal{S}}(\mathbf{Q}) \leq k$, guess a subset of price points $(\mathbf{V}_1, p_1), \ldots, (\mathbf{V}_m, p_m)$ in $\mathcal{S}$, then check that both $D \vdash \mathbf{V}_1, \ldots, \mathbf{V}_m \twoheadrightarrow \mathbf{Q}$ (this is in coNP by Theorem 2.3) and that $\sum_i p_i \leq k$. For (b), instead of guessing, we can iterate over all subset of price points, since there is only a fixed number of them. □

Thus, computing the price is expensive. This expense is unacceptable in practice, since prices are computed as frequently as queries, perhaps even more frequently (for example, users may just inquire about the price, then decide not to buy). We have an extensive discussion of tractability in Section 3, and will describe an important restriction under which pricing is tractable. For now, we restrict our discussion of the complexity to showing that pricing is at least as complex as computing the determinacy relation.

Let Price-Consistency$(\mathcal{S})$ be the problem of deciding whether a set of price points $\mathcal{S}$ is consistent for a database D, and Determinacy$(\mathbf{V}, Q)$ the problem of checking determinacy $D \vdash \mathbf{V} \twoheadrightarrow Q$. The proof of Corollary 2.16 shows that the former problem is no more than exponentially worse than the latter. We prove in the full paper [**?**] a weak converse:

[2]For any query bundle $\mathbf{Q}$, if $\mathcal{C}_1 \in supp_D^{\mathcal{S}}(\mathbf{Q})$ and $\mathcal{C}_1 \subseteq \mathcal{C}_2$ then $\mathcal{C}_2 \in supp_D^{\mathcal{S}}(\mathbf{Q})$, by the reflexivity axiom.

PROPOSITION 2.17. *There is a polynomial time reduction from* DETERMINACY$(\mathbf{V}, Q)$ *to*[3] PRICE-CONSISTENCY$(\mathcal{S})$.

We end this section with a brief discussion of the case when **ID** is not determined by $\mathcal{S} = \{(\mathbf{V}_1, p_1), \ldots, (\mathbf{V}_k, p_k)\}$, that is, $D \vdash (\mathbf{V}_1, \ldots, \mathbf{V}_k) \not\twoheadrightarrow \mathbf{ID}$; in other words, the seller does not sell the entire dataset. In this case, $\mathcal{S}$ admits no discount-free pricing function. Indeed, consider any B such that $B \geq \sum_i p_i$, and denote $\mathcal{S}+B = \mathcal{S} \cup \{(\mathbf{ID}, B)\}$. One can check that, if $\mathcal{S}$ is consistent, then so is $\mathcal{S}+B$, and that $p_D^{\mathcal{S}+B}$ is a valid pricing function for $\mathcal{S}$. If p_D were any discount-free pricing function for $\mathcal{S}$, we fix some database instance D and choose $B > p_D(\mathbf{ID})$. Then, $p_D^{\mathcal{S}+B}(\mathbf{ID}) = B > p_D(\mathbf{ID})$, contradicting the fact that p is discount-free. In the rest of the paper, we will always assume that **ID** is included in the set of price points.

2.7 Dynamic Pricing

So far we assumed that the database instance was static. We now consider the pricing function in a dynamic setting, i.e. when the database D is updated; in this paper, we consider only insertions. Note that the set of price points $\mathcal{S}$ remains unchanged, even when the database gets updated. For example, the seller has decided to sell the entire dataset for the price B, $(\mathbf{ID}, B) \in \mathcal{S}$, and this price remains unchanged even when new items are inserted in the database. This is the most common case encountered today: explicit prices remain fixed over long periods of time, even when the underlying data set is updated.

When the database is updated, we can simply recompute the pricing function on the new data instance. However, we face two issues. The first is that the price points $\mathcal{S}$ may become inconsistent: $\mathcal{S}$ was consistent at D_1, but after inserting some items, $\mathcal{S}$ becomes inconsistent at D_2. This must be avoided in practice. Second, as more data items are added, in most cases the seller does not want any price to drop. As an example, adding more businesses to the USA business database should not remove value from any query. However, there are cases where more data adds noise and hence decreases the value of the dataset: we do not explore this scenario in this paper.

EXAMPLE 2.18. *Let $V(x,y) = R(x), S(x,y)$ and $Q() = R(x)$ (a boolean query checking whether R is non-empty). Let $D_1 = \emptyset$, $D_2 = \{R(a), S(a,b)\}$. Then $D_1 \vdash V \not\twoheadrightarrow Q$ and $D_2 \vdash V \twoheadrightarrow Q$. The second claim is obvious: since $V(D_2) = \{(a,b)\}$ we know for certain that $R^{D_2} \neq \emptyset$. To see the first claim consider $D_1' = \{R(a)\}$. Then $V(D_1) = V(D_1') = \emptyset$ but $Q(D_1) = false \neq Q(D_1') = true$, proving that $D_1 \vdash V \not\twoheadrightarrow Q$.*

This example implies two undesired consequences. First, let $\mathcal{S}_1 = \{(V, \$1), (Q, \$10), (\mathbf{ID}, \$100)\}$: the entire dataset costs \$100, the query Q \$10, and the view V \$1. $\mathcal{S}_1$ is consistent when the database instance is D_1, but when tuples are inserted and the database instance becomes D_2, then $\mathcal{S}_1$ is no longer consistent (because a buyer can avoid paying \$10 for Q by asking V instead, for just \$1). Alternatively, consider the set of price points $\mathcal{S}_2 = \{(V, \$1), (\mathbf{ID}, \$100)\}$. The reader may check that $\mathcal{S}_2$ is consistent for any database instance D. However, the price of Q decreases when the database is updated: $p_{D_1}^{\mathcal{S}_2}(Q) = \$100$, while $p_{D_2}^{\mathcal{S}_2}(Q) = \$1$.

Next, we describe two ways to fix both issues.

[3] $\mathcal{S}$ has one price point for each $V \in \mathbf{V}$ and one for Q; the database instance D is part of the input in both cases.

DEFINITION 2.19. *Fix the bundles $\mathbf{V}, \mathbf{Q}$. We say that a determinacy relation $\twoheadrightarrow$ is* monotone *for $\mathbf{V}, \mathbf{Q}$ if, whenever $D_1 \subseteq D_2$ and $D_2 \vdash \mathbf{V} \twoheadrightarrow \mathbf{Q}$, then $D_1 \vdash \mathbf{V} \twoheadrightarrow \mathbf{Q}$.*

Information-theoretic determinacy is vacuously monotone, since it does not depend on the instance. But, as we saw in Example 2.18, instance-based determinacy is not monotone in general. We prove in [**?**]:

PROPOSITION 2.20. *If $\mathbf{V}$ is a bundle consisting only of selection queries and $\mathbf{Q}$ is a bundle of full conjunctive queries, then instance-based determinacy is monotone for $\mathbf{V}, \mathbf{Q}$.*

A conjunctive query is full if it has no projections; in particular, a selection query is full. We show in the full version [**?**] that the property fails for CQs with projections.

As we mentioned earlier, we would like that a dynamic pricing function be *monotone*: when data is added to the database, the price should never decrease:

DEFINITION 2.21 (MONOTONICITY). *Let p be a totally defined, dynamic pricing function. We say that p is* monotone *on $\mathbf{Q}$ if, for any $D_1 \subseteq D_2$, $p_{D_1}(\mathbf{Q}) \leq p_{D_2}(\mathbf{Q})$.*

Fix a set of price points $\mathcal{S}$. The arbitrage-price given by Equation 2 is a totally defined function $p^{\mathcal{S}}$, since $p_D^{\mathcal{S}}$ is well defined for every database instance D. We prove:

PROPOSITION 2.22. *Fix $\mathcal{S}$ and $\mathbf{Q}$, and suppose that $\twoheadrightarrow$ is monotone for every subset $\mathbf{V}_1, \ldots, \mathbf{V}_m$ of $\mathcal{S}$, and $\mathbf{Q}$. Then, the dynamic arbitrage-price $p^{\mathcal{S}}$ is monotone on $\mathbf{Q}$.*

PROOF. By Equation 1: $supp_{D_1}^{\mathcal{S}}(\mathbf{Q}) \supseteq supp_{D_2}^{\mathcal{S}}(\mathbf{Q})$. By Equation 2: $p_{D_1}^{\mathcal{S}}(\mathbf{Q}) \leq p_{D_2}^{\mathcal{S}}(\mathbf{Q})$. □

PROPOSITION 2.23. *If $p^{\mathcal{S}}$ is monotone on every $\mathbf{V}_i$, $\mathcal{S}$ is consistent on D_1, and $D_1 \subseteq D_2$, then $\mathcal{S}$ is consistent on D_2.*

PROOF. To check consistency on D_2 it suffices to check that $p_i \leq p_{D_2}^{\mathcal{S}}(\mathbf{V}_i)$, for all $i = 1, \ldots, m$. We have $p_i \leq p_{D_1}^{\mathcal{S}}(\mathbf{V}_i)$ since $\mathcal{S}$ is consistent on D_1, and $p_{D_1}^{\mathcal{S}}(\mathbf{V}_i) \leq p_{D_2}^{\mathcal{S}}(\mathbf{V}_i)$ because $p^{\mathcal{S}}$ is monotone. □

The goal in the dynamic setting is to ensure that $p^{\mathcal{S}}$ is monotone on every query (Definition 2.21). There are two ways to achieve this. One is to restrict all views to selection queries and all queries to full conjunctive queries: we pursue this in Section 3. However, if one needs more general views and queries, then we propose a second alternative: to consider a different determinacy relation along with monotone views. Let $\twoheadrightarrow$ be any determinacy relation (Definition 2.5). Its *restriction* $D \vdash \mathbf{V} \twoheadrightarrow^* \mathbf{Q}$ is: $\forall D_0, \mathbf{V}(D_0) \subseteq \mathbf{V}(D)$, $D_0 \vdash \mathbf{V} \twoheadrightarrow \mathbf{Q}$. We prove in detail in [**?**]:

PROPOSITION 2.24. *(a) $\twoheadrightarrow^*$ is a determinacy relation (Definition 2.5), (b) $\twoheadrightarrow^*$ is monotone (Definition 2.19) for any monotone $\mathbf{V}$ and any $\mathbf{Q}$, (c) if $p_D^{\mathcal{S}}$ and $q_D^{\mathcal{S}}$ are the arbitrage-prices for $\twoheadrightarrow$ and $\twoheadrightarrow^*$, respectively, then $p_D^{\mathcal{S}}(\mathbf{Q}) \leq q_D^{\mathcal{S}}(\mathbf{Q})$ for all $\mathbf{Q}$, and (d) if $\twoheadrightarrow$ is instance-based determinacy, then the data complexity of $\twoheadrightarrow^*$ is in coNP.*

Thus, by replacing instance-based determinacy $\twoheadrightarrow$ with its restriction $\twoheadrightarrow^*$, we obtain a monotone pricing function $q^{\mathcal{S}}$. In particular, if $\mathcal{S}$ is consistent in a database D, then it will remain consistent after insertions. To illustrate, recall that in Example 2.18 $\mathcal{S}_1$ became inconsistent when D_1 was updated to D_2: this is because $D_1 \vdash V \not\twoheadrightarrow Q$ and $D_2 \vdash V \twoheadrightarrow Q$. Now we have both $D_1 \vdash V \not\twoheadrightarrow^* Q$ and $D_2 \vdash V \not\twoheadrightarrow^* Q$, hence $\mathcal{S}_1$ is consistent in both states of the database.

3. TRACTABLE QUERY-BASED PRICING

The combined complexity for computing the price when the views and queries are UCQs is high: it is coNP-hard and in Σ_2^P. This is unacceptable in practice. In this section, we restrict both the views on which the seller can set explicit prices and the queries that the buyer can ask, and present a polynomial time algorithm for computing the price. This is the main result in the paper, since it represents a quite practical framework for query-based pricing. For the case of conjunctive queries without self-joins, we prove a dichotomy of their complexity into polynomial time and NP-complete, which is our most technically difficult result.

The Views. We restrict the views to *selection queries*. We denote a selection query by $\sigma_{R.X=a}$, where R is a relation name, X an attribute, and a a constant. For example, given a ternary relation $R(X, Y, Z)$, the selection query $\sigma_{R.X=a}$ is $Q(x, y, z) = R(x, y, z) \wedge x = a$. Throughout this section, the seller can set explicit prices only on selection views. We argue that this restriction is quite reasonable in practice. Many concrete instances of online data pricing that we have encountered set prices only on selection queries[4]. For example, CustomLists [13] sells the set of all businesses in any given state for $199, thus it sells 50 selection views. Infochimps [17] sells the following selection queries, in the form of API calls. The Domains API: *given IP address, retrieve the domain, company name and NAICS Code*. The MLB Baseball API: *given an MLB team name, retrieve the wins, losses, current team colors, seasons played, final regular season standings, home stadium, and team_ids*. The Team API: *given the team_ids, get the team statistics, records, and game_ids*. And, the Game API: *given game_id, get the attendance, box scores, and statistics*. Thus, we argue, restricting the explicit price points to selection queries is quite reasonable for practical purposes.

An important assumption made by sellers today is that the set of values on which to select is known. For example, the set of valid MLB team names is known to the buyers, or can be obtained for free from somewhere else. In general, for each attribute $R.X$ we assume a finite set $Col_{R.X} = \{a_1, \ldots, a_n\}$, called the *column*. This set is known both to the seller and the buyer. Furthermore, the database D satisfies the inclusion constraint $R^D.X \subseteq Col_{R.X}$. The input to the pricing algorithm consists of both the database instance D, and all the columns $Col_{R.X}$: thus, the latter are part of the input in data complexity. A column should not be confused with a domain: while a domain may be infinite, a column has finitely many values. It should not be confused with the active domain either, since the database need not have all values in a column. We also assume that columns always remain fixed when the database is updated.

We call the set of all selections on column $R.X$, $\Sigma_{R.X} = \{\sigma_{R.X=a} \mid a \in Col_{R.X}\}$, the *full cover* of $R.X$. Note that $D \vdash \Sigma_{R.X} \twoheadrightarrow R$. We denote Σ the set of all selections on all columns. Given $\mathbf{V} \subseteq \Sigma$, we say that it *fully covers* $R.X$ if $\Sigma_{R.X} \subseteq \mathbf{V}$. Thus, the explicit price points $\mathcal{S} = \{(V_1, p_1), (V_2, p_2), \ldots\}$ are such that each $V_i \in \Sigma$. We denote $p : \Sigma \to \mathbb{R}^+$ the partial function defined as: $p(V_i) = p_i$ if $(V_i, p_i) \in \mathcal{S}$.

Recall that $\textsc{Price}(\mathcal{S}, \mathbf{Q})$ denotes the data complexity of the pricing problem in Section 2. Since now $\mathcal{S}$ can be as large as Σ, we treat it as part of the input. Thus, we denote the pricing problem as $\textsc{Price}(\mathbf{Q})$, where the input consists of the database instance D, all columns $Col_{R.X}$, and the function p. We start with a lemma (which we prove in [**?**]):

LEMMA 3.1. *Let $\mathbf{V} \subseteq \Sigma$. Then $D \vdash \mathbf{V} \twoheadrightarrow \sigma_{R.X=a}$ iff (a) it is trivial (i.e. $\sigma_{R.X=a} \in \mathbf{V}$), or (b) $\mathbf{V}$ fully covers some attribute Y of R.*

The lemma has two consequences. First, recall that in Subsection 2.4 we required that the views in $\mathcal{S}$ determine **ID**. By the lemma, this requirement becomes equivalent to requiring that, for any relation R, $\mathcal{S}$ fully covers some attribute X. Second, the lemma gives us a simple criterion for checking whether $\mathcal{S}$ is consistent. By Theorem 2.15, this holds iff there is no arbitrage between the views in $\mathcal{S}$. The lemma implies that the only risk of arbitrage is between a full cover $\Sigma_{R.Y}$ and a selection view $\sigma_{R.X=a}$, hence:

PROPOSITION 3.2. *$\mathcal{S}$ is consistent iff for every relation R, any two attributes X, Y of R and any constant $a \in Col_{R.X}$:*

$$p(\sigma_{R.X=a}) \leq \sum_{b \in Col_{R.Y}} p(\sigma_{R.Y=b})$$

Note that now consistency is independent of the database instance; this is unlike Subsection 2.7, where we showed that consistency may change with the database.

The Queries. We would like to support a rich query language that buyers can use, while ensuring tractability for the price computation. We start with an upper bound on the data complexity of pricing. We say that a query Q has PTIME data complexity if $Q(D)$ can be computed in polynomial time in the size of D. UCQ queries, datalog queries, and extensions of datalog with negation and inequalities have PTIME data complexity[4].

THEOREM 3.3. *Assume $\mathbf{V} \subseteq \Sigma$. Let Q be any monotone query that has PTIME data complexity. Then, $D \vdash \mathbf{V} \twoheadrightarrow Q$ for $\mathbf{V} \subseteq \Sigma$ can be decided in PTIME data complexity.*

We give the proof in the full version of this paper [**?**].

COROLLARY 3.4. *Let $\mathbf{Q}$ be a bundle of monotone queries that have polynomial time data complexity. Then $\textsc{Price}(\mathbf{Q})$ is in NP.*

PROOF. To check if $p_D^{\mathcal{S}}(\mathbf{Q}) \leq k$, guess a subset of selection views $\mathbf{V} \subseteq \Sigma$, then check that both $D \vdash \mathbf{V} \twoheadrightarrow \mathbf{Q}$ (which is equivalent to $D \vdash \mathbf{V} \twoheadrightarrow Q_i$, for all $Q_i \in \mathbf{Q}$, by Lemma 2.6) and that $\sum_{V \in \mathbf{V}} p(V) \leq k$. □

Thus, the restriction to selection queries has lowered the complexity of price computation from Σ_2^p (Corollary 2.16) to NP. However, for some Conjunctive Queries, computing the price is still NP-hard (the detailed proof is in [**?**]):

THEOREM 3.5 (NP-COMPLETE QUERIES). *$\textsc{Price}(Q)$ is NP-complete (data complexity) when Q is any of the following queries:*

$$H_1(x, y, z) = R(x, y, z), S(x), T(y), U(z) \quad (3)$$

$$H_2(x, y) = R(x), S(x, y), T(x, y) \quad (4)$$

$$H_3(x, y) = R(x), S(x, y), R(y) \quad (5)$$

$$H_4(x) = R(x, y) \quad (6)$$

[4]The only exception are sites that sell data by the number of tuples; for example, Azure allows the seller to set a price on a "transaction", which means any 100 tuples.

If Q is one of H_1, H_2, H_3 then the pricing complexity remains NP-complete even when the database instance D is restricted s.t. $Q(D) = \emptyset$.

Thus, we cannot afford to price every conjunctive query. In Subsection 3.1, we introduce a class of conjunctive queries whose prices can be computed in PTIME. In Subsection 3.2, we study the complexity of *all* conjunctive queries without self-joins, and establish a dichotomy for pricing into PTIME and NP-complete.

3.1 A PTIME Algorithm

We define a class of conjunctive queries, called *Generalized Chain Queries*, denoted GCHQ, and we provide a non-trivial algorithm that computes their prices in polynomial time.

We consider conjunctive queries with interpreted unary predicates $C(x)$ that can be computed in PTIME: that is, we allow predicates like $x > 10$ or USER-DEFINED-PREDICATE(x), but not $x < y$. A conjunctive query is *without self-joins* if each relation R_i occurs at most once in Q; e.g. query H_3 in Theorem 3.5 has a self-join (since R occurs twice), the other three queries are without self-joins. A conjunctive query is *full* if all variables in the body appear in the head; e.g. queries H_1, H_2, H_3 are full, while H_4 is not. We restrict our discussion to full, conjunctive queries without self-joins. We abbreviate such a query with $Q = R_0, R_1, \ldots, R_k, C_1, \ldots, C_p$, where each R_i is an atomic relational predicate, and each C_j is an interpreted unary predicate; we assume the order $R_1, \ldots, R_k$ to be fixed. For $0 \le i \le j \le k$, we denote $Q_{[i,j]}$ the full conjunctive $Q_{[i,j]} = R_i, R_{i+1}, \ldots, R_j$ (ignoring the unary predicates). For example, if $Q(x, y, z) = R(x), S(x, y), T(y), U(y, z), V(z)$, then $Q_{[1:2]}(x, y) = S(x, y), T(y)$ If k is the index of the last relational predicate, then we abbreviate $Q_{[j,k]}$ with $Q_{[j:*]}$. Denote $Var(Q)$ the set of variables in Q.

DEFINITION 3.6. *A generalized chain query,* GCHQ*, is a full conjunctive query without self-joins, Q, such that, for all i, $|Var(Q_{[0:i-1]}) \cap Var(Q_{[i:*]})| = 1$. We denote x_i the unique variable shared by $Q_{[0:i-1]}$ and $Q_{[i:*]}$. The (not necessarily distinct) variables $x_1, \ldots, x_k$ are called* join variables*. All other variables are called* hanging variables*.*

In other words, a GCHQ query is one in which every join consists of only one shared variable. Note that the definition ignores the interpreted unary predicates occurring in Q. The following are some examples of GCHQ queries:

$$Q_1(x, y) = R(x), S(x, y), T(y)$$
$$Q_2(x, y, z, w) = R(x, y), S(y, z), T(z), U(z), V(z, w)$$
$$Q_3(x, y, z, u, v, w) = R(x, y), S(y, u, v, z), T(z, w), U(w)$$

On the other hand, none of the queries in Theorem 3.5 are GCHQ: the atoms in queries H_1 and H_2 cannot be ordered to satisfy Definition 3.6, H_3 has a self-join, and H_4 is not a full query.

DEFINITION 3.7. *A* GCHQ *query bundle is a set* $\mathbf{Q}$ *of* GCHQ *queries, such that, any two queries $Q, Q' \in \mathbf{Q}$ only share in common a prefix and/or a suffix: $\exists i, j, m : Q_{[0:i-1]} = Q'_{[0:i-1]}, Q_{[j:*]} = Q'_{[m:*]}$, and $Q_{[i:j-1]}, Q'_{[i:m-1]}$ have no common relation names.*

For an example, $\{Q_1 = S(x, y), R(y, z), U(z), Q_2 = S(x, y), T(y, z), U(z), Q_3 = S(x, y), T(y, z), U(z)\}$ forms a GCHQ bundle. We can now state our main result in this paper:

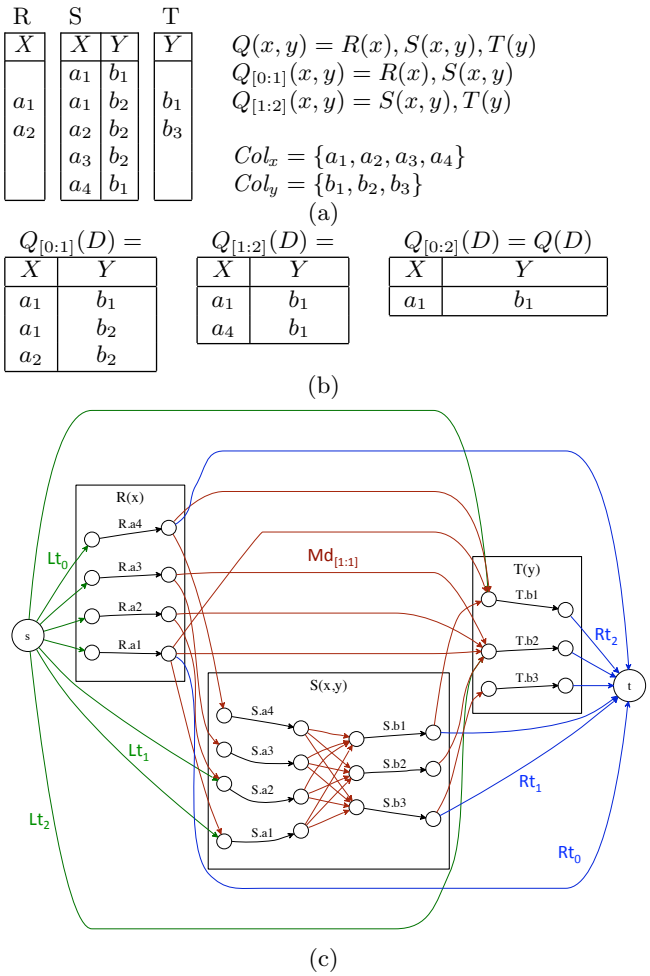

Figure 1: (a) The database D and query Q for Example 3.9. (b) The answers to the partial queries $Q_{[0:1]}, Q_{[1:2]}, Q_{[0:2]}$. (c) The flow graph describing the example (see Theorem 3.13).

THEOREM 3.8 (MAIN THEOREM). *Assume that all explicit price points in $\mathcal{S}$ are selection queries. Then, for any* GCHQ *query, one can compute its price in PTIME (data complexity).*

Before we give the algorithm, we illustrate pricing with an example.

EXAMPLE 3.9. *Consider $Q = R(x), S(x, y), T(y)$ over the database D in Figure 1(a). We have $Q(D) = \{(a_1, b_1)\}$. There are 14 possible selection queries that are priced: $\Sigma_{R.X} = \{\sigma_{R.X=a_1}, \sigma_{R.X=a_2}, \sigma_{R.X=a_3}, \sigma_{R.X=a_4}\}$, as well as $\Sigma_{S.X} = \{\sigma_{S.X=a_1}, \sigma_{S.X=a_2}, \sigma_{S.X=a_3}, \sigma_{S.X=a_4}\}$, and similarly for $S.Y$ and $T.Y$. Suppose $\mathcal{S}$ assigns the price \$1 to each selection query.*

To compute the price of Q, we need to find the smallest set $\mathbf{V} \subseteq \Sigma$ that "determines" Q: that is, forall D' s.t. $\mathbf{V}(D) = \mathbf{V}(D')$, the query must return the same answer $\{(a_1, b_1)\}$ on D', as on D. First, $\mathbf{V}$ must guarantee that (a_1, b_1) is an answer, and for that it must ensure that all three tuples $R(a_1), S(a_1, b_1), T(b_1)$ are in D'; for example, it suffices to include in $\mathbf{V}$ the views $\mathbf{V}_0 = \{\sigma_{R.X=a_1}, \sigma_{S.Y=b_1}, \sigma_{T.Y=b_1}\}$

(we could have chosen $\sigma_{S.X=a_1}$ instead of $\sigma_{S.Y=b_1}$). Second, $\mathbf{V}$ must also ensure that none of the other 11 tuples (a_i, b_j) are in the answer to Q. $\mathbf{V}_0$ is not sufficient yet. For example, consider the tuple (a_3, b_2), which is not in the answer. Let $D' = D \cup \{R(a_3), T(b_2)\}$; then $\mathbf{V}_0(D) = \mathbf{V}_0(D')$, since $\mathbf{V}_0$ does not inquire about either $R(a_3)$ or $T(b_2)$, yet $Q(D')$ contains (a_3, b_2). Thus, $\mathbf{V}$ must ensure that either $R(a_3)$ is not in D', or that $T(b_2)$ is not in D'. Continuing this reasoning, leads us to the following set of views $\mathbf{V} = \{\sigma_{R.X=a_1}, \sigma_{R.X=a_4}, \sigma_{S.Y=b_1}, \sigma_{S.Y=b_3}, \sigma_{T.Y=b_1}, \sigma_{T.Y=b_2}\}$. The reader may check that this is a minimal set that determines Q, hence the price of Q is $p_D^{\mathcal{S}}(Q) = 6$.

We now describe the algorithm, which consists of the four steps below. To simplify the presentation, we discuss only single queries, and defer query bundles to the full paper.

STEP 1: Remove Atomic Predicates. Suppose Q has a variable x with an atomic predicate $C(x)$: here we simply shrink the column[5] of x to $Col'_x = \{a \in Col_x \mid C(a) = true\}$, thus removing all constants that do not satisfy C. Let $\mathcal{S}' \subseteq \mathcal{S}$ be obtained by removing all selection views that refer to these constants, and similarly $D' \subseteq D$ be the database obtained by filtering on the predicate C. Finally, let Q' be the query obtained from Q by removing the predicate $C(x)$. We prove in [**?**] that $p_{D'}^{\mathcal{S}'}(Q') = p_D^{\mathcal{S}}(Q)$.

To illustrate this step, consider the query $Q(y, w, z) = R(y), S(y, w, z), T(z), w = a_1$ and $Col_w = \{a_1, a_2, a_3\}$. Then, we restrict the column of w to $\{a_1\}$, next remove the views $\sigma_{S.W=a_2}, \sigma_{S.W=a_3}$ from $\mathcal{S}$ to obtain $\mathcal{S}'$, filter D on $w = a_1$ to obtain D', and then compute the price of $Q'(x, y, z) = R(y), S(y, w, z), T(z)$.

STEP 2: Remove Multiple Variable Occurrences from Each Atom. We only sketch this step, and defer the details to the full version [**?**]. Suppose a variable x occurs twice in the atom $R(x, x, z)$, where R has schema $R(X, Y, Z)$. Let $R'(X, Z)$ be a new relation name s.t. $Col_{R'.X} = Col_x$, and let us set the prices on $R'.X$ as follows: $p(\sigma_{R'.X=a}) = \min\{p(\sigma_{R.X=a}), p(\sigma_{R.Y=a})\}$. We prove that the price of the new query (obtained by replacing the atom $R(x, x, z)$ with $R'(x, z)$) is the same as the price of Q.

STEP 3: Removing Hanging Variables. Recall that a *hanging* variable is one that occurs in only one atom of Q; by the previous step, it only occurs in one position $R.X$. We prove in the full paper the following:

LEMMA 3.10. *Let x be a hanging variable in Q, occurring in the attribute position $R.X$. Let $\mathbf{V} \subseteq \Sigma$. If $D \vdash \mathbf{V} \twoheadrightarrow Q$ then either (a) $\mathbf{V}$ fully covers $R.X$ or (b) $D \vdash (\mathbf{V} \setminus \Sigma_{R.X}) \twoheadrightarrow \mathbf{Q}$ (in other words, every view in $\mathbf{V}$ referring to $R.X$ is redundant).*

Thus, when computing the price of Q, for each hanging variable we need to consider two cases: either fully cover it, or not cover it at all. We claim that each of these cases becomes another price computation problem, namely for the query Q', obtained from Q by replacing R with R' (obtained from R by removing the attribute $R.X$), on the database D' obtained from D by projecting out $R.X$:

LEMMA 3.11. *Let $R.X$ be an attribute containing a hanging variable in Q, $\mathbf{V} \subseteq \Sigma$, and $\mathbf{V}' = \mathbf{V} \setminus \Sigma_{R.X}$.*

- *If $\mathbf{V}$ fully covers $R.X$, let Y be any attribute $Y \neq X$ of R. Then $D \vdash \mathbf{V} \twoheadrightarrow Q$ iff $D' \vdash \mathbf{V}', \Sigma_{R'.Y} \twoheadrightarrow Q'$.*
- *If $\mathbf{V}$ does not fully cover $R.X$, then $D \vdash \mathbf{V} \twoheadrightarrow Q$ iff $D' \vdash \mathbf{V}' \twoheadrightarrow Q'$.*

We prove this as part of a more general lemma in the full version [**?**]. The lemma gives us an algorithm for removing hanging variables: compute two prices for Q', and take the minimum. The first price corresponds to the case when $R.X$ is fully covered: in that case, we give out R' for free (by setting all prices $\sigma_{R'.Y=a}$ to 0, for some other attribute Y) and compute the price of Q': then, add to that the true cost of the full cover $\Sigma_{R.X}$, i.e. $\sum_a p(\sigma_{R.X=a})$. The second price corresponds to the case when $R.X$ is not covered at all, and is equal to the price of Q'. For a simple example, if $Q(x, y, z) = R(x, y), S(y, z), T(z)$, then $Q'(y, z) = R'(y), S(y, z), T(z)$. Let p_1 be the price of Q' where we set all prices of $\sigma_{R'.Y=b}$ to 0; let p_2 be the regular price of Q' (where all prices are unchanged, but the views $\sigma_{R.X=a}$ are removed); return $\min(p_1 + p(\Sigma_{R.X}), p_2)$. In general, we need to repeat this process once for every hanging variable; thus, we end up computing 2^k prices, if there are k attributes with hanging variables.

STEP 4: Reduction to Maximum Flow. Finally, we have reached the core of the algorithm. At this point, the query is a *Chain Query*:

DEFINITION 3.12. *A* Chain Query *is a full conjunctive query without self-joins, $Q = R_0, R_1, \ldots, R_k$ s.t.: (a) every atom R_i is either binary or unary, (b) any two consecutive atoms R_i, R_{i+1} share exactly one variable, denoted x_i, (c) the first and the last atoms are unary, $R_0(x_0), R_k(x_k)$. Denote* ChQ *the set of chain queries.*

We show that pricing a chain query can be reduced to the Min-Cut problem, which is the dual of the Max-Flow graph problem and can be solved in polynomial time [12].

Given a chain query Q, denote x_i, x_{i+1} the variables occurring in R_i: if R_i is unary, then $x_i = x_{i+1}$. In particular, $x_0 = x_1$ and $x_k = x_{k+1}$, since the first and last atoms are unary. Thus, each query $Q_{[i:j]} = R_i, \ldots, R_j$ has variables $x_i, \ldots, x_{j+1}$. Also, let us define $Q_{[i:i-1]} = Col_{x_i} = Col_{R_{i-1}.Y} \cap Col_{R_i.X}$. Define the *left-, middle-, and right-partial-answers*:

$$
\begin{aligned}
Lt_i &= \Pi_{x_i}(Q_{[0:i-1]}(D)), && 0 \le i \le k \\
Md_{[i:j]} &= \Pi_{x_i, x_{j+1}}(Q_{[i:j]}(D)), && 1 \le i \le k, i-1 \le j \le k-1 \\
Rt_j &= \Pi_{x_{j+1}}(Q_{[j+1:k]}(D)), && 0 \le j \le k
\end{aligned}
$$

We construct the following graph G. The graph has a source node s and a target (sink) node t. Moreover, for each attribute $R.X$ and constant $a \in Col_{R.X}$, we introduce two nodes: $v_{R.X=a}$ and $w_{R.X=a}$. The edges of G are:

View edges: For each attribute $R.X$ and constant $a \in Col_{R.X}$ we create the edge: $v_{R.X=a} \xrightarrow{view} w_{R.X=a}$, where the capacity equals the price[6] $p(\sigma_{R.X=a})$ in $\mathcal{S}$.

Tuple edges: For each binary atom $R(X, Y)$ and constants $a \in Col_{R.X}$, $b \in Col_{R.Y}$, we create the following edge: $w_{R.X=a} \xrightarrow{tuple} v_{R.Y=b}$, where capacity $= \infty$.

[5] If x occurs on several attribute positions $R.X$, $S.Y$, etc, then we may assume w.l.o.g. that $Col_{R.X} = Col_{S.Y} = \ldots$ and denote it with Col_x.

[6] If the query has no explicit price in $\mathcal{S}$ then capacity $= \infty$.

Skip edges: For all partial answers we create the edges:

$$s \xrightarrow{skip} v_{R_i.X=a} \qquad \text{if } a \in Lt_i$$
$$w_{R_{j-1}.Y=b} \xrightarrow{skip} v_{R_{i+1}.X=a} \qquad \text{if } (b,a) \in Md_{[j:i]}$$
$$w_{R_j.Y=b} \xrightarrow{skip} t \qquad \text{if } a \in Rt_j$$

In all cases, capacity $= \infty$.

In particular, since $Lt_0 = Col_{x_0}$, $Md_{[i:i-1]} = Col_{x_i}$, $Rt_k = Col_{x_k}$ we also have the following skip edges:

$$s \xrightarrow{skip} v_{R_0.X=a}, \quad w_{R_{i-1}.Y=a} \xrightarrow{skip} v_{R_i.X=a}, \quad w_{R_k.Y=a} \xrightarrow{skip} t$$

We explain now the intuition behind the graph construction, and will also refer to Figure 1 (b) and (c), which illustrates the graph for Example 3.9. Notice that the edges of finite capacity in G are in one-to-one correspondence with the views in $\mathcal{S}$. The main invariant (which we prove in the full paper) is: *for every set of edges C of finite capacity, C is a "cut" (it separates s and t) if and only if the corresponding set of views $\mathbf{V}$ determines the query.* Before justifying this invariant, note that the core of the graph consists of sequences of three edges:

$$v_{R_i.X=a} \xrightarrow{view} w_{R_i.X=a} \xrightarrow{tuple} w_{R_i.Y=b} \xrightarrow{view} w_{R_i.Y=b}$$

for all binary relations $R_i(X,Y)$ and constants $a \in Col_{R_i.X}, b \in Col_{R_i.Y}$ (unary relations have just one *view* edge.) Consider a possible answer to Q, $t = (u_1, u_2, \ldots, u_k)$, where $u_1 \in Col_{x_1}, \ldots, u_k \in Col_{x_2}$. If $D \vdash \mathbf{V} \twoheadrightarrow Q$, then, for all D' s.t. $\mathbf{V}(D) = \mathbf{V}(D')$, $\mathbf{V}$ must ensure two things: if $t \in Q(D)$ then it must ensure that $t \in Q(D')$, and if $t \notin Q(D)$ then it must ensure that $t \notin Q(D')$. Take the first case, $t \in Q(D)$. For each $i = 0, \ldots, k$, denoting $a = u_i$ and[7] $b = u_{i+1}$, we have: $a \in Lt_i$ (is a left partial answer), $R_i(a,b) \in D$, and $b \in Rt_i$ (is a right partial answer). Hence there are two skip edges:

$$s \xrightarrow{skip} v_{R_i.X=a} \qquad\qquad w_{R_i.Y=b} \xrightarrow{skip} t$$

Combined with the three edges above, they form an $s-t$ path: thus, any cut of finite capacity must include one of the two *view* edges, hence, the corresponding set of views $\mathbf{V}$ includes either $\sigma_{R_i.X=a}$ or $\sigma_{R_i.Y=b}$, ensuring $R_i(a,b) \in D'$. Since this holds for any i, it follows that D' has all the tuples needed to ensure $t \in Q(D')$. For example, in Figure 1 the answer $(a_1, b_1) \in Q(D)$ leads to three $s-t$ paths:

$$s \xrightarrow{skip} v_{R.X=a_1} \xrightarrow{view} w_{R.X=a_1} \xrightarrow{skip} t$$
$$s \xrightarrow{skip} v_{S.X=a_1} \xrightarrow{view} w_{S.X=a_1} \xrightarrow{tuple} v_{S.Y=b_1} \xrightarrow{view} w_{S.Y=a_1} \xrightarrow{skip} t$$
$$s \xrightarrow{skip} v_{T.Y=b_1} \xrightarrow{view} w_{T.Y=b_1} \xrightarrow{skip} t$$

Any cut ensures that $R(a_1), S(a_1,b_1), T(b_1)$ are present.

Take the second case, $t \notin Q(D)$. Then some of the tuples $R_i(u_i, u_{i+1})$ are missing from D, and $\mathbf{V}$ must ensure that *at least one* is missing. The sequence $u_1, \ldots, u_k$ consists of partial answers, alternating with missing tuples. We are interested only in the latter and the skip edges help by skipping over the partial answers. Thus the missing tuples are on a path from s to t. For an illustration, assume that exactly two tuples are missing, $R_i(u_i, u_{i+1})$ and $R_j(u_j, u_{j+1})$; denoting $a = u_i, b = u_{i+1}$, $c = u_j, d = u_{j+1}$ we have:

$a \in Lt_i, (a,b) \notin Md_{[i:i]}, (b,c) \in Md_{[i+1:j-1]}, (c,d) \notin Md_{[j:j]}, d \in Rt_j$

leading to the following $s-t$ path:

$$s \xrightarrow{skip} v_{R_i.X=a} \xrightarrow{view} w_{R_i.X=a} \xrightarrow{tuple} w_{R_i.Y=b} \xrightarrow{view} w_{R_i.Y=b}$$
$$\xrightarrow{skip} v_{R_j.X=c} \xrightarrow{view} w_{R_j.X=c} \xrightarrow{tuple} w_{R_j.Y=d} \xrightarrow{view} w_{R_j.Y=d} \xrightarrow{skip} t$$

To summarize, we prove the following in the full paper [**?**]:

THEOREM 3.13. *The cost of the minimum cut in G is equal to the price of Q. Therefore, the price of Q can be computed in polynomial time, by reduction to* MIN-CUT.

3.2 A Dichotomy Theorem

Are there any other queries besides GCHQ whose data complexity is in PTIME? The answer is *yes*. We study them here, and give a full characterization of the complexity of all conjunctive queries without self-joins, showing that for each query its complexity is either PTIME or NP-complete. Note that our characterization applies to *all* queries without self-joins, not just full queries. However, it only applies to single queries, not to query bundles: we leave open whether query bundles admit a similar dichotomy as single queries.

We start by characterizing the PTIME class. Clearly, all GCHQ queries are in PTIME. By definition, every GCHQ query is connected: it is easy to check that PTIME queries are closed under cartesian products:

PROPOSITION 3.14. *Assume that Q is disconnected, and partitioned into $Q(\bar{x}_1, \bar{x}_2) : -Q_1(\bar{x}_1), Q_2(\bar{x}_2)$, where $\bar{x}_1, \bar{x}_2$ are disjoint sets of variables. Then,*

$$p_D^{\mathcal{S}}(Q) = \begin{cases} \min\{p_D^{\mathcal{S}}(Q_1), p_D^{\mathcal{S}}(Q_2)\} & \text{if } Q_1(D) = Q_2(D) = \emptyset, \\ p_D^{\mathcal{S}}(Q_1) & \text{if } Q_1(D) = \emptyset,\ Q_2(D) \neq \emptyset, \\ p_D^{\mathcal{S}}(Q_2) & \text{if } Q_2(D) = \emptyset,\ Q_1(D) \neq \emptyset, \\ p_D^{\mathcal{S}}(Q_1) + p_D^{\mathcal{S}}(Q_2) & \text{else} \end{cases}$$

We prove this proposition, along with the converse reduction, in [**?**]. As a consequence, the complexity of any disconnected query is no larger than that of any of its connected components.

A more surprising class of queries that admits a PTIME algorithm is the the class of *cycle queries*:

THEOREM 3.15. *For any integer k,* PRICE(C_k) *is in PTIME, where $C_k(x_1, \ldots, x_k) = R_1(x_1, x_2), \ldots, R_k(x_k, x_1)$.*

The algorithm for computing C_k is described in the full version. It is technically the most difficult result in this paper, and is quite different from the reduction to MIN-CUT that we used for GCHQ, suggesting that these two classes cannot be unified in a natural way. The class of queries C_k is also much more brittle than GCHQ: adding a single unary predicate makes the query NP-hard. For example, see the query H_2 in Theorem 3.5: it is obtained by adding one unary predicate to C_2, and is NP-hard. By contrast, we can add freely unary predicates to GCHQ.

We conclude our analysis with the following theorem, whose proof can be found in the full version of this paper [**?**]:

THEOREM 3.16 (DICHOTOMY THEOREM). *Let $\mathcal{S}$ contain only selection views (in Σ) and Q be a CQ w/o self-joins. The data complexity for* PRICE(Q) *is the following:*

[7] When $i = k$ then $u_i = u_{i+1}$, hence $a = b$.

- *If Q has connected components $Q_1, \ldots, Q_k$, then: if all components Q_i are in PTIME, it is in PTIME, and if one component Q_i is NP-complete, Q is NP-complete.*
- *Else if Q is neither full nor boolean, it is NP-complete.*
- *Else if Q is a boolean query, then let Q^f be the corresponding full query (add all variables to the head); then the complexity of Q is the same as that of Q^f.*
- *Else if Q is a full CQ, let Q' be obtained from Q by removing all hanging variables, constants and multiple occurrences of a variable in the same atom: (a) if Q' is a* GCHQ *then it is PTIME, (b) if $Q' = C_k$ for some k, then it is also PTIME, (c) otherwise, Q is NP-complete.*

4. DISCUSSION

We end this this paper with a brief discussion on loose ends and design choices.

Step v.s. smooth pricing function. The pricing function $p^{\mathcal{S}}$ can take only finitely many values, because the arbitrage-price is always the sum of a subset of prices from $\mathcal{S}$. For some applications, this may be too limiting. One such example is selling private data, where the price should be proportional to the degree of privacy breached by the query: since privacy mechanisms add a noise that can be tuned continuously (e.g. the ε parameter in differential privacy [15]), one expects the pricing function to also vary continuously. Studying such "smooth" pricing functions is part of future work.

Pricing and query containment. The price should *not* be required to be monotone w.r.t. query containment. Recall that two queries (of the same arity) are said to be contained if $Q_1(D) \subseteq Q_2(D)$ for any database D. If Q_2 always returns at least as much data as Q_1, one might insist that $p_D(Q_1) \leq p_D(Q_2)$. We argue against this.

EXAMPLE 4.1. *Let $Q_1(x,y) = R(x), S(x,y)$ and $Q_2(x,y) = S(x,y)$. Then, $Q_1 \subseteq Q_2$, but the information in Q_1 may be more valuable than that in Q_2. For example, $S(x,y)$ may be the list of the top 500 companies and their stock price, while $R(x)$ may be an analyst's confidential list of 5 companies with very high potential for growth. Clearly, the seller wants to set $p_D(Q_1) \gg p_D(Q_2)$.*

There is also a theoretical argument: if p_D is arbitrage-free *and* monotone w.r.t. query containment, then all Boolean queries have the same price! Indeed, let T be the Boolean query that is always true, i.e. $T(D) = true$ for any database D, and let Q be any Boolean query. We have $Q \subseteq T$, hence $p_D(Q) \leq p_D(T)$; on the other hand, $D \vdash Q \twoheadrightarrow T$, which implies $p_D(T) \leq p_D(Q)$.

Price updates. What happens if the seller adds price points to $\mathcal{S}$? We prove in the full paper [**?**] that, as long the price points remain consistent, the prices never increase; in other words, the seller can only add more discounts, but cannot raise the prices (of course, one can modify $\mathcal{S}$ to raise prices, but prices do not increase through additions to $\mathcal{S}$.)

Selections on Multiple Attributes. The PTIME algorithm in Subsection 3.1 allows explicit prices only on selection queries on single attributes, e.g. $\sigma_{R.X=a}$. A natural question is whether one can extend it to prices for selections on two or more attributes, e.g. $\sigma_{R.X=a,R.Y=b}$. The answer to this question varies. For Chain Queries (Definition 3.12) this is possible: simply modify the flow graph by setting the capacity of the TUPLE-EDGE $(w_{R.X=a}, v_{R.Y=b})$ to $p(\sigma_{R.X=a,R.Y=b})$ instead of ∞. For Generalized Chain Queries, however, this is not possible in general. In fact, even for a very simple query, $Q(x,y,z) = R(x,y,z)$, if $\mathcal{S}$ has prices on all these types of selection queries: $\sigma_{R.X=a}$, $\sigma_{R.Y=b}$, $\sigma_{R.Z=c}$, $\sigma_{R.X=a,R.Y=b,R.Z=c}$, then we prove in the full paper [**?**] that computing the price of Q is NP-hard.

5. RELATED WORK

There exist many independent vendors selling data online [1, 2, 6, 13, 30] and Amazon cloud users can sell their S3 data for a profit [7]. In addition, digital market services for data have recently emerged in the cloud [9, 17, 28], which enable content providers to upload their data and make it available either freely or for a fee, and support some limited forms of views. In the case of Infochimps [17], the seller can set prices on APIs (modeled as selection queries) or entire datasets. The Azure Marketplace [9] uses data subscriptions with query limits on transactions, *i.e.* a group of records returned by a query that can fit on a page (currently 100). WebScaled is a pre-launch startup providing a marketplace for datasets from ongoing Web crawls: social graphs, lists of sites using particular advertising platforms, frequency of specific doctypes and other HTML elements, etc. [28, 29]. Apollo Mapping sells access to satellite imagery [8]. The approach that we develop in this paper extends these pricing methods with the ability to interpolate prices for *arbitrary queries* over a seller's database.

While the interaction between data management and economics has been studied in the database research community before [14, 27], to the best of our knowledge, this paper is the first to study the problem of data pricing, with the exception of a short vision paper that we recently published [10].

There is a rich literature on pricing information products (*e.g.*, [18, 26]). We were mostly influenced by Shapiro and Varian [26], who argue that the price of *information products* is quite different from that of *physical goods*, and propose a new theory for pricing information products, based on the notion of versions. The difference is that information products have very high fixed costs, while the marginal costs are tiny. For example, the cost of conducting a detailed consumer survey in several countries is very high, while the cost of distributing the resulting data tiny (copying a file). As a consequence, the price of information products cannot be determined by traditional means (production costs and competition), but must be linked to the value that the buyers place on the data. Different buyers may use the data in different ways, and should be charged different prices. For example, a retailer may be willing to pay a high price for the entire consumer survey, while a journalist may only be willing to pay a small amount for a few interesting statistics from the consumer survey. In order to leverage these differences in willingness to pay, Shapiro and Varian conclude that information products should be offered in different *versions*, at different prices. Our approach extends version-based pricing to relational data, by associating a version of the product to each query that a user may ask.

The classic notion of determinacy was extensively studied by Nash, Segoufin and Vianu [25, 22, 23], who have investigated both the decidability question, and the subtle relationship between determinacy and rewritability. We have reviewed information-theoretic determinacy earlier ($\mathbf{V} \twoheadrightarrow Q$ if forall D, D', $\mathbf{V}(D) = \mathbf{V}(D')$ implies $Q(D) = Q(D')$).

Rewritability is specific to a query language $\mathcal{L}$: Q can be *rewritten using* $\mathbf{V}$ *in the language* $\mathcal{L}$ if there exists a query $R \in \mathcal{L}$ s.t. $Q(D) = R(\mathbf{V}(D))$ for all D. One goal of this line of research was to establish tight bounds on the language $\mathcal{L}$; a surprising result is an example where both $\mathbf{V}$ and Q are conjunctive queries, yet R is non-monotone, proving that no monotone language is sufficient for CQ to CQ rewriting. In our query pricing framework we do not impose any restriction on the language used for rewriting; in other words, we assume that the buyer has unrestricted computational power, and as a consequence the two notions become equal. A second goal of the research [25, 22, 23] is to study the decision problem for determinacy: it is shown to be undecidable even for Unions of Conjunctive Queries, and its status is open for Conjunctive Queries. However, several classes of CQ queries where determinacy is well-behaved have been found: path queries [5], syntactic restrictions of FO and UCQ (packed FO and UCQ [20]) and monadic views [23]. Determinacy has also been examined in the restricted setting of aggregate queries [16].

A key difference in our paper is that we consider instance-based determinacy, where determinacy is defined with respect to a view extension. While applications like data integration or semantic caching require instance-independent determinacy, in query pricing the current state of the database cannot be ignored. Instance-based determinacy is identical to the notion of *lossless views* [11] under the exact view assumption. The definition is further based on the notion of *certain answers* [3]. We note that instance-specific reasoning also arises in data security and authorization views: in that context, Zhang and Mendelzon study *conditional query containment*, where containment is conditioned on a particular view output [31].

Finally, we should mention that, on the surface, our complexity results for pricing seem related to complexity results for computing *responsibility* [21]. The PTIME algorithm for responsibility is also based on network flow, but the reduction for pricing is harder to establish. Furthermore, even though some queries have the same complexity for both pricing and responsibility, the connection is superficial: the price of H_2 (Theorem 3.5) is NP-complete, while its responsibility is in PTIME; and the price of C_3 (Theorem 3.15) is in PTIME while its responsibility is NP-complete.

6. CONCLUSION

We have presented a framework for pricing relational data based on queries. The seller sets explicit prices on some views, while the buyer may ask arbitrary queries; their prices are determined automatically. We gave several results: an explicit formula for the price, a polynomial time algorithm for pricing Generalized Chain Queries, and a dichotomy theorem for conjunctive queries without self-joins. We also presented several results on instance-based determinacy.

Interesting future work includes considering competition: when a seller sets prices for her data, she needs to consider other data instances on the market that offer "related" data, to avoid arbitrage. This requires reasoning about mappings between the different data sources, and these mappings are often approximate in practice. Another is the interaction between pricing and privacy. Most of the literature on data privacy [15] focuses on restricting access to private information. Privacy, however, has a broader definition, and usually means the ability of the data owner to control how her private information is used [24]. Setting a price for private data is one form of such control that we plan to investigate.

Acknowledgments. This work is supported in part by the NSF and Microsoft through NSF grant CCF-1047815 and also grant IIS-0915054. We also thank the anonymous reviewer for pointing out an inconsistency in the early version of Corollary 2.16.

7. REFERENCES

[1] http://gnip.com.
[2] http://www.patientslikeme.com.
[3] S. Abiteboul and O. M. Duschka. Complexity of answering queries using materialized views. In *PODS*, pages 254–263. ACM Press, 1998.
[4] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
[5] F. N. Afrati. Rewriting conjunctive queries determined by views. In *MFCS*, pages 78–89, 2007.
[6] http://www.aggdata.com/.
[7] Using Amazon S3 Requester Pays with DevPay. http://docs.amazonwebservices.com/AmazonDevPay/latest/DevPayDeveloperGuide/index.html?S3RequesterPays.html.
[8] http://www.apollomapping.com/.
[9] https://datamarket.azure.com/.
[10] M. Balazinska, B. Howe, and D. Suciu. Data markets in the cloud: An opportunity for the database community. *Proc. of the VLDB Endowment*, 4(12), 2011.
[11] D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Y. Vardi. Lossless regular views. In L. Popa, editor, *PODS*, pages 247–258. ACM, 2002.
[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press and McGraw-Hill Book Company, 2001.
[13] http://www.customlists.net/.
[14] D. Dash, V. Kantere, and A. Ailamaki. An economic model for self-tuned cloud caching. In *Proc. of the 25th ICDE Conf.*, pages 1687–1693, 2009.
[15] C. Dwork. A firm foundation for private data analysis. *Commun. ACM*, 54(1):86–95, 2011.
[16] S. Grumbach and L. Tininini. On the content of materialized aggregate views. *J. Comput. Syst. Sci.*, 66(1):133–168, 2003.
[17] http://www.infochimps.com/.
[18] S. Jain and P. K. Kannan. Pricing of information products on online servers: Issues, models, and analysis. *Management Science*, 48(9):1123–1142, 2002.
[19] L. Libkin. *Elements of Finite Model Theory*. Springer, 2004.
[20] M. Marx. Queries determined by views: pack your views. In L. Libkin, editor, *PODS*, pages 23–30. ACM, 2007.
[21] A. Meliou, W. Gatterbauer, K. F. Moore, and D. Suciu. The complexity of causality and responsibility for query answers and non-answers. *PVLDB*, 4(1):34–45, 2010.
[22] A. Nash, L. Segoufin, and V. Vianu. Determinacy and rewriting of conjunctive queries using views: A progress report. In *ICDT*, pages 59–73, 2007.
[23] A. Nash, L. Segoufin, and V. Vianu. Views and queries: Determinacy and rewriting. *ACM Trans. Database Syst.*, 35(3), 2010.
[24] B. Schneier. *Secrets & Lies, Digital Security in a Networked World*. John Wiley & Sons, 2000.
[25] L. Segoufin and V. Vianu. Views and queries: determinacy and rewriting. In C. Li, editor, *PODS*, pages 49–60. ACM, 2005.
[26] C. Shapiro and H. R. Varian. Versioning: The smart way to sell information. *Harvard Business Review*, 76:106–114, November-December 1998.
[27] Stonebraker et al. Mariposa: a wide-area distributed database system. *VLDB Journal*, 5(1):048–063, 1996.
[28] http://webscaled.com/.
[29] Web marketing. Google group forum post, http://groups.google.com/group/webmarketing/msg/c6643da409802f85.
[30] http://www.xignite.com/.
[31] Z. Zhang and A. O. Mendelzon. Authorization views and conditional query containment. In *ICDT*, pages 259–273, 2005.