Valuating Queries for Data Trading in Modern Cities

Ruiming Tang*, Huayu Wu†, Xiuqiang He*, Stéphane Bressan‡

*Huawei Noah’s Ark Lab, China  
†Institute for Infocomm Research, A*STAR, Singapore  
‡School of Computing, National University of Singapore

{tangruiming, hexiuqiang}@huawei.com  
huwu@i2r.a-star.edu.sg  
steph@nus.edu.sg

Abstract—The availability of data trading mechanisms and platforms is a paramount prerequisite to the development of effective smart city services. In order for data to become a commodity ready for consumption, transformation and exploitation by smart services, it must be made available and tradable on data market places. For such data market places to be viable there is a compelling need for a sound data pricing model that is conducive of the healthiness of the market. In this paper, we discuss the definition of a pricing model in which views are priced and queries are valuated using views. We define the price of a query as the cheapest combination of the prices of a set of views that can answer the query. We discuss the devising of effective and efficient algorithms of the computation of the price of a query. We show that the problem of computing the price is similar but not identical to the problem of answering queries using views. We therefore adapt the MiniCon algorithm, which was designed to answer queries using views, to the task at hand. We finally discuss further challenges created by the definition of a framework for valuating queries using views.

I. INTRODUCTION

More and more urban data are being collected by sensors, probe vehicles, public and private organizations or simply made available by citizens on the Internet and other communication networks. Meanwhile, service providers, decision makers and city planners are increasingly aware of the importance of such data for business intelligence [1], insight gain of the city [2] and service improvement [3]. In many cities, a lot of urban data are published by the government for innovations and application development, to make the city smarter1.

On the other hand, in order for data to become a commodity ready for consumption, transformation and exploitation smart services such as data mining and analytics services, data must be tradable on data market places. There already exists such data market places as Aggdata [4] and Microsoft’s Azure MarketPlace [5] that provide a space for data owners to sell their data and data consumers to buy data. Further, some government agencies and private companies have started looking into ways to make part of their data tradable for profit. For example, the National Environment Agency (NEA) of Singapore sells the Singapore’s climate data to interested industry partners2.

For the data markets to be healthy there is a compelling need for sound pricing models. While some data pricing models have been proposed, there remain, as remarked by the authors of [6], several issues to be addressed for the realization of a practical trading environment. In this paper, we are considering data pricing mechanisms based on queries and views as they are the best suited for usage based-pricing. The authors of [6] propose in [7] a framework for pricing data in which sellers set the price of pre-defined views, and buyers pay for their query according to the prices of views. Namely, the price of a query is the price of the cheapest set of views that determine the query in the current database instance. However, the computation of the price of a query for most practical query classes is intractable. No exact or approximate practical algorithm has been proposed. The authors of [7] argue in favor of instance-based determinacy against rewritability. We argue that determinacy is not sufficient. We define the price of a query as the price of the cheapest set of views that answer the query. The problem of finding an equivalent rewriting, or even a maximally contained one, for many practical query classes, is still intractable [8], [9]. However, thanks to results on the bounded size of the rewriting [8], several practical algorithms have been devised for answering queries using views [10], [11], [12]. The most efficient of them is Pottinger’s MiniCon [12].

We show that the naïve application of the MiniCon algorithm for the computation of the price of a query using views leads to overcharging. In this paper, we propose an extension that overcomes this shortcoming, by adding a post-processing phase, the MiniCon algorithm for the computation of the price of a query using views. We compute the price of conjunctive queries and views with comparison operators (involving one variable and one constant).

1For example, http://www.data.gov.sg/  
II. BACKGROUND

A. Instance-based determinacy, Determinacy and Rewritability

In this section, we discuss the relationship and difference among instance-based determinacy, determinacy and answerability. We also state the reason why we use rewritability instead of instance-based determinacy (as [7] does).

In [7], the authors use instance-based determinacy to connect the input query to a set of views.

**Definition 2.1**: (Instance-based determinacy) Let \( \mathcal{V} \) be a set of views, \( Q \) be a query, \( \mathcal{D} \) be a database, \( \mathcal{V}(\mathcal{D}) \) and \( Q(\mathcal{D}) \) are the results of issuing \( \mathcal{V} \) and \( Q \) on \( \mathcal{D} \). \( \mathcal{V} \) determines \( Q \) given \( \mathcal{D} \), denoted \( \mathcal{D} \vdash \mathcal{V} \rightarrow Q \), if for any \( \mathcal{D}' \), \( \mathcal{V}(\mathcal{D}) = \mathcal{V}(\mathcal{D}') \rightarrow Q(\mathcal{D}) = Q(\mathcal{D}') \).

As proved in [7], deciding instance-based determinacy is co-NP-complete when the queries and views are unison of conjunctive queries or even conjunctive queries. Determinacy [13] is different from instance-based determinacy.

**Definition 2.2**: (Determinacy) Let \( \mathcal{V} \) be a set of views, \( Q \) be a query, \( \mathcal{D} \) be a database, \( \mathcal{V}(\mathcal{D}) \) and \( Q(\mathcal{D}) \) are the results of issuing \( \mathcal{V} \) and \( Q \) on \( \mathcal{D} \). \( \mathcal{V} \) determines \( Q \) given \( \mathcal{D} \), denoted \( \mathcal{D} \vdash \mathcal{V} \rightarrow Q \), if for any database \( \mathcal{D}_1, \mathcal{D}_2 \), \( \mathcal{V}(\mathcal{D}_1) = \mathcal{V}(\mathcal{D}_2) \rightarrow Q(\mathcal{D}_1) = Q(\mathcal{D}_2) \).

Determinacy is undecidable when the queries and views are union of conjunctive queries, and its status is unknown when the queries and views are conjunctive queries. In the example below, we show the difference between instance-based determinacy and determinacy.

**Example 2.1**: Let \( Q(x, y, z) : \neg R(x, y), S(y, z); Q_1(x, y) : \neg R(x, y), Q_1 \rightarrow Q \) is not true. However, let \( \mathcal{D} \) be a database instance such that \( Q_1(\mathcal{D}) = \emptyset \). Then \( \mathcal{D} \vdash Q_1 \rightarrow Q \), because \( Q(\mathcal{D}) = \emptyset \).

[14], [13] study the relationship between determinacy and rewritability.

**Definition 2.3**: (Rewritability) A query \( Q \) can be rewritten using a set of views \( \mathcal{V} \) specific to a query language \( \mathcal{R} \) if for any database \( \mathcal{D} \), \( Q(\mathcal{D}) = R(\mathcal{V}(\mathcal{D})) \), where \( R \in \mathcal{R} \) is a query.

Rewritability is always specific to a query language \( \mathcal{R} \). As argued in [13], if \( Q \) can be rewritten in terms of \( \mathcal{V} \) using the language \( \mathcal{R} \), then \( \mathcal{V} \rightarrow Q \). The converse is generally not true. The existence of such a rewriting depends on the languages of the query and of the views. We show such an example below. Through this paper, we illustrate our ideas with an consistent example in the context of Mass Rapid Transit System. More specifically, we consider two relations: \( \textit{Passenger}(\text{NRIC}, \text{name}, \text{gender}, \text{age}) \) and \( \textit{At\_station}(\text{NRIC}, \text{station}, \text{day}, \text{time}) \). The relation “\( \textit{Passenger} \)” stores personal information such as identifier (NRIC), name, gender and age, while relation “\( \textit{At\_station} \)” stores stations of passengers at specific time points.

**Example 2.2**: Let us consider a query and two views.

\( Q(\text{name}) : \neg \textit{Passenger}(\text{NRIC}, \text{name}, \text{‘male’}, \text{age}) \)

\( V_1(\text{name}) : \neg \textit{Passenger}(\text{NRIC}, \text{name}, \text{gender}, \text{age}) \)

\( V_2(\text{name}, \text{gender}) : \neg \textit{Passenger}(\text{NRIC}, \text{name}, \text{gender}, \text{age}) \)

It is true that \( V_1 \rightarrow Q \) and \( V_2 \rightarrow Q \). However, the query \( Q \) is not rewritable using \( V_1 \), but only rewritable using \( V_2 \), if we restrict the rewriting language to be union of conjunctive query.

In practice, during query pricing with pre-defined views, we need to consider not only determinacy, but also answerability. If we only follow determinacy for query pricing, it is possible that the views charged for a query cannot answer the query. For the above example, since both \( V_1 \) and \( V_2 \) can determine \( Q \), in some pricing model that uses determinacy (e.g., [7]), probably the cheapest view \( V_1 \) (it contains less information than \( V_2 \), thus is cheaper) will be used to price the query, but it actually cannot answer \( Q \). Thus, besides determinacy, we need also consider answerability of views during query pricing. As stated, for any query language, rewritability implies determinacy, and it is naturally used for query answering, thus in this paper we use rewritability to connect the input query to a set of views.

B. Query Containment

**Definition 2.4**: (Query containment and equivalence) A query \( Q_1 \) is said to be contained in a query \( Q_2 \), denoted by \( Q_1 \subseteq Q_2 \), if for all database instance \( \mathcal{D} \), the query result of \( Q_1 \) is a subset of the query result of \( Q_2 \), i.e. \( Q_1(\mathcal{D}) \subseteq Q_2(\mathcal{D}) \). The two queries are said to be equivalent if \( Q_1 \subseteq Q_2 \) and \( Q_2 \subseteq Q_1 \).

The complexity of the problem of query containment is dependent on the expressiveness of the query language. The query containment problem of conjunctive queries with comparison operators is NP-complete ([15]). It is proved in [16] that the query containment problem of conjunctive queries with comparison operators is \( \Pi^P_2 \)-complete. [17] shows some special cases that the complexity of query containment problem is lower than the general case. The query containment of union of conjunctive queries is \( \Pi^P_2 \)-complete ([18]).

C. Answering queries using views

In [8], the author distinguishes between two types of query rewritings: \textit{equivalent rewritings} and \textit{maximally-contained rewritings}. Informally, an equivalent rewriting gets the same set of tuples as the original query. However, sometimes we cannot find an equivalent rewriting because of the views’ limited coverage. Instead, we have to search for a maximally-contained rewriting, which provides the best answer possible, given the view definitions.

**Definition 2.5**: (Equivalent rewritings [8]) Let \( Q \) be a query and \( \mathcal{V} = \{ V_1, \ldots, V_m \} \) be a set of view definitions. The query \( Q' \) is an equivalent rewriting of \( Q \) using \( \mathcal{V} \) if \( Q' \) and \( Q \) are equivalent.

**Definition 2.6**: (Maximally-contained rewritings [8]) Let \( Q \) be a query, \( \mathcal{V} = \{ V_1, \ldots, V_m \} \) be a set of view definitions and \( L \) be a query language. The query \( Q' \) is a maximally-contained rewriting of \( Q \) using \( \mathcal{V} \) with respect to \( L \) if:

- for any database \( \mathcal{D} \) and extensions \( v_1, \ldots, v_m \) such that \( v_i \subseteq V_i(\mathcal{D}) \), then \( Q'(v_1, \ldots, v_m) \subseteq Q(\mathcal{D}) \).
- there is no rewriting \( Q_1 \in L \), such that for every database \( \mathcal{D} \) and extensions \( v_1, \ldots, v_m \) \( Q'(v_1, \ldots, v_m) \subseteq Q_1(v_1, \ldots, v_m) \) and \( 2 \) \( Q_1(v_1, \ldots, v_m) \subseteq Q(\mathcal{D}) \), and there
exists at least one database for which (1) is a strict set inclusion.

In order to categorize different works, we use two different dimensions, which are open v.s. closed world assumption and different query languages. Although there exist many different dimensions (mentioned in [8]) to distinguish different works, we believe these two dimensions are most relevant to our problem.

1) Closed world assumption: In the closed world assumption, all views are complete. One application of work under the closed world assumption is query optimization.

In query optimization, answering queries using views can improve the efficiency of the query processing since using materialized views saves computation. The algorithms in this context return the equivalent rewriting with the lowest cost in form of a query plan. Note that some views existing in the returned query plan do not contribute in semantics, but only to reduce the cost.

In this context, each work differentiates itself from others based on the languages of views and queries. The languages of queries and views can be conjunctive queries ([19], [20]), grouping and aggregation ([21], [22]), and bag semantics and multi-block queries ([23]).

2) Open world assumption: In the open world assumption, all views are incomplete. One application of work under the open world assumption is data integration.

In data integration, we have a global mediated schema, and multiple data sources. Queries are in terms of the mediate schema, but only the data in data sources can be used. The task becomes to answer the given query in terms of the mediated schema by using the data in data sources. A query transformation is needed. Views in this context refer to source descriptions from the data source schemas to the mediate schema. Views are incomplete since we may have a large number (or maybe infinite number) of data sources. We need query rewritings instead of query plans in this context. Due to the incompleteness of views, equivalent rewritings are impossible to get. The algorithms in this context return a maximally-contained rewriting.

Similarly, the work in this category focus on different query languages, such as conjunctive queries ([10], [11], [12]), conjunctive queries in the presence of functional dependencies ([24], [25]), conjunctive queries in the presence of inclusion dependencies ([26]), in the presence of access pattern limitations ([25], [27]), and union in the views ([28]).

III. THE PRICE OF A QUERY

In this section, we define the price of a query, based on the set of views which can answer the input query.

Let \( V = \{ V_1, ..., V_m \} \) be a set of pre-defined views. Each pre-defined view has its price, which is set up by the data seller, i.e. the price of \( V_i \) is \( p_i \). Given a query \( Q \), we aim to define the price of \( Q \). We assume that the views are complete (under the closed world assumption), since we assume we work on a single database. We only consider conjunctive queries.

Definition 3.1: (The price of a query) Let \( V \) be a set of pre-defined views. Let \( p_i \) be the price of \( V_i \) \( (V_i \in V) \). For a rewriting \( R \), \( V(R) \) denotes the set of views in the rewriting \( R \). We denote the price of \( R \) as \( p(R) = \sum_{V_i \in V(R)} p_i \). Let \( MaxR(Q) \) denote the set of maximally-contained rewritings of \( Q \).

The price of a query \( Q \) is

\[
\text{price}^V(Q) = \min_{R \in MaxR(Q)} p(R)
\]

This price represents the price of the cheapest maximally-contained rewriting.

IV. ADAPTING THE MINICON ALGORITHM

The MiniCon algorithm considers conjunctive queries and views. It returns a set of conjunctive rewritings, and it works under the open world assumption. We choose MiniCon algorithm to adapt, due to the following reasons. First, we need rewritings of the input query to valuate the query. Second, we are interested in conjunctive queries and views.

A. Description of MiniCon

MiniCon [12] is an algorithm that uses materialized views to rewrite queries for the purpose of data integration. The MiniCon algorithm returns the union of a set of conjunctive queries as the maximally-contained rewriting of the input query. The MiniCon algorithm guarantees to be sound and complete [12] considering conjunctive queries.

The algorithm is basically divided into two major steps. The first step tries to find a minimum set of subgoals for each view if there exists at least one subgoal of the query which can be mapped to the view. Some more subgoals may have to be added to the set in order to cover the join predicates based on the proposed properties. This set of subgoals is called a MiniCon Description (MCD). The second step creates rewritings of the query by combining MCDs obtained in the first step. At last, all the generated conjunctive rewritings are returned, the union of which is the maximally-contained rewriting.

Example 4.1: Let us consider a query and three views.

\( Q(name) : \neg \text{Passenger}(NRIC, \text{name}, \text{gender}, \text{age}) \), \( \text{At}_\text{station}(NRIC, \text{relationship}, \text{day}, \text{time}) \)

\( V_1(NRIC, \text{name}) : \neg \text{Passenger}(NRIC, \text{name}, \text{gender}, \text{age}) \)

\( V_2(NRIC, \text{station}) : \neg \text{At}_\text{station}(NRIC, \text{relationship}, \text{day}, \text{time}) \)

\( V_3(NRIC, \text{station}) : \neg \text{At}_\text{station}(NRIC, \text{’Clementi’}, \text{day}, \text{time}) \)

The conjunctive rewritings returned by the MiniCon algorithm are:

\( Q_1(name) : \neg V_1(NRIC, \text{name}), V_2(NRIC, \text{station}) \)

\( Q_2(name) : \neg V_1(NRIC, \text{name}), V_3(NRIC, \text{station}) \)

Note that the maximally-contained rewriting is the union of the two rewritings, namely

\( R(name) : \neg Q_1(name) \cup Q_2(name) \)
It describes two ways of obtaining answer to the query from the available sources. In the closed world assumption, all the views are completed. The first rewriting \( Q_1 \) is equivalent to the original query. However, in the context of data integration, open world assumption is made. The sources are not guaranteed to have all the tuples in the definition of the view. The definition of \( V_2 \) covers the definition of \( V_3 \), however, because of source incompleteness, \( V_3 \) may contain some tuples which not in \( V_2 \). Thus the result should be the union of the two rewritings.

B. Applying MiniCon Algorithm to Pricing

The MiniCon algorithm can be directly adapted for our pricing problem, by summing up the prices of all the views in the returned set of conjunctive rewritings. We use Algorithm 1 to illustrate the price computation by this adoption.

**Algorithm 1: Price of a query**

**Data:** \( Q(Q, V) = \{Q_1, ..., Q_n\} \), and price \( p_i \) for \( V_i \), \( i \in [1, m] \)

**Result:** price of the query \( \text{price}^V(Q) \)

1. for \( Q_i \in Q(Q, V) \) do
2. \( V_{\text{maxr}}(Q_i) = V\text{(maxr}(Q)) \cup V(Q_i) \);
3. \( \text{price}^V(Q) = \sum_{i \in V_{\text{maxr}}(Q)} p_i; \)

We explain Algorithm 1 in detail below.
Assume we have a query \( Q \) and a set of views \( V = \{V_1, ..., V_m\} \), and the MiniCon algorithm returns the set of conjunctive rewritings as \( Q(Q, V) = \{Q_1, ..., Q_n\} \), where \( Q_i \) represents a conjunctive rewriting.

First, we get the set of views in the returned set of conjunctive rewritings, denoted as \( V_{\text{maxr}}(Q) \) (line 1 to line 2). Then we sum up the prices of all the views in \( V_{\text{maxr}}(Q) \) (line 3).

We find that if there is no containment relationship between any two sets of the conjunctive queries returned by the MiniCon algorithm, Algorithm 1 computes the price correctly. Below we show such two cases.

**Case 1: there is only one rewriting returned.**

**Example 4.2:**
Let us consider a query and two views.
\( Q\text{(name):} \neg \text{Passenger}(\text{NRIC}, \text{name}, \text{gender}, \text{age}), \)
\( \text{At}_\text{station}(\text{NRIC}, \text{station}, \text{day}, \text{time}) \)
\( V_1(\text{NRIC}, \text{name}): \neg \text{Passenger}(\text{NRIC}, \text{name}, \text{gender}, \text{age}) \)
\( V_2(\text{NRIC}, \text{station}): \neg \text{At}_\text{station}(\text{NRIC}, \text{station}, \text{day}, \text{time}) \)
with the pre-set price \( p_1 = 15, p_2 = 5 \).
The only rewriting returned by the MiniCon algorithm is
\( Q_1\text{(name):} \neg V_1(\text{NRIC}, \text{name}), V_2(\text{NRIC}, \text{station}) \)
The correct price of \( Q \) as defined in Definition 3.1 is
\[ \text{price}^V(Q) = p_1 + p_2 = 15 + 5 = 20 \]

The price of \( Q \) computed by Algorithm 1 is
\[ \text{price}^V(Q)' = p_1 + p_2 = 15 + 5 = 20 \]
\[ \text{price}^V(Q) = \text{price}^V(Q)' \]

**Example 4.3:**
Let us consider a query and two views.
\( Q\text{(name):} \neg \text{Passenger}(\text{NRIC}, \text{name}, \text{gender}, \text{age}), \)
\( \text{At}_\text{station}(\text{NRIC}, \text{station}, \text{day}, \text{time}) \)
\( V_1(\text{NRIC}, \text{name}): \neg \text{Passenger}(\text{NRIC}, \text{name}, \text{gender}, \text{age}) \)
\( V_2(\text{NRIC}, \text{station}): \neg \text{At}_\text{station}(\text{NRIC}, \text{station}, \text{day}, \text{time}) \)
with the pre-set price \( p_1 = 15, p_2 = 12 \).
The only rewriting returned rewriting by the MiniCon algorithm is
\( Q_1\text{(name):} \neg V_1(\text{NRIC}, \text{name}), V_2(\text{NRIC}, \text{station}) \)
The correct price of \( Q \) is
\[ \text{price}^V(Q) = p_1 + p_2 = 15 + 12 = 27 \]

The price of \( Q \) computed by Algorithm 1 is
\[ \text{price}^V(Q)' = p_1 + p_2 = 15 + 12 = 27 \]
\[ \text{price}^V(Q) = \text{price}^V(Q)' \]

**Case 2: more than one rewriting returned and there is no containment relationship.**

**Example 4.4:**
Let us consider a query and three views.
\( Q\text{(name):} \neg \text{Passenger}(\text{NRIC}, \text{name}, \text{gender}, \text{age}), \)
\( \text{At}_\text{station}(\text{NRIC}, \text{station}, \text{day}, \text{time}) \)
\( V_1(\text{NRIC}, \text{name}): \neg \text{Passenger}(\text{NRIC}, \text{name}, \text{gender}, \text{age}), \quad 0 < \text{age} < 25 \)
\( V_2(\text{NRIC}, \text{name}): \neg \text{Passenger}(\text{NRIC}, \text{name}, \text{gender}, \text{age}), \quad 18 < \text{age} < 30 \)
\( V_3(\text{NRIC}, \text{station}): \neg \text{At}_\text{station}(\text{NRIC}, \text{station}, \text{day}, \text{time}) \)
with the pre-set price \( p_1 = 6, p_2 = 7, p_3 = 12 \).
The two conjunctive rewritings returned by the MiniCon algorithm are:
\( Q_1\text{(name):} \neg V_1(\text{NRIC}, \text{name}), V_2(\text{NRIC}, \text{station}) \)
\( Q_2\text{(name):} \neg V_2(\text{NRIC}, \text{name}), V_3(\text{NRIC}, \text{station}) \)
The correct price of \( Q \) is
\[ \text{price}^V(Q) = p_1 + p_2 + p_3 = 6 + 7 + 12 = 25 \]

The price of \( Q \) computed by Algorithm 1 is
\[ \text{price}^V(Q)' = p_1 + p_2 + p_3 = 6 + 7 + 12 = 25 \]
\[ \text{price}^V(Q) = \text{price}^V(Q)' \]

Example 4.4 shows that there are two non-equivalent rewritings \( Q_1, Q_2 \) returned by MiniCon algorithm. There is no containment relationship between \( Q_1 \) and \( Q_2 \). In this case,
all the views used in \( Q_1, Q_2 \) will be charged. Although the common tuples in the results of \( Q_1 \) and \( Q_2 \) are charged twice, the consumer has to accept it since the prices are set to views.

Because our pricing problem is under the closed world assumption, while the MiniCon algorithm works under the open world assumption, Algorithm 1 overcharges in some cases. In the next section, we will look into such cases.

C. Limitations

The MiniCon algorithm works under the open world assumption. Under this assumption, there is no equivalent rewriting. All the rewritings has to be returned. The MiniCon algorithm generates conjunctive rewritings, the union of which is a maximally-contained rewriting.

However, in our problem, the views are complete. There possibly exist equivalent rewritings of the input query. If we use Algorithm 1 to compute the price of a query, we may overcharge in some cases.

We find that if there is some containment relationship between any two sets of the conjunctive rewritings returned by the MiniCon algorithm, Algorithm 1 overcharges. Below we present some cases in which Algorithm 1 does not work correctly.

**Case 1: there exists some containment relationship between the rewritings returned.**

**Example 4.5:**

Let us consider a query and three views.

\[ Q(name) : \neg \text{Passenger}(NRIC, name, gender, age), \quad At_{station}(NRIC, station, day, time) \]

\[ V_1(NRIC, name) : \neg \text{Passenger}(NRIC, name, gender, age), \quad At_{station}(NRIC, station, day, time) \]

\[ V_2(NRIC, name) : \neg \text{Passenger}(NRIC, name, gender, age) \]

\[ V_3(NRIC, station) : \neg At_{station}(NRIC, ‘Clementi’, day, time) \]

with the pre-set price \( p_1 = 25, p_2 = 15, p_3 = 12 \).

The two conjunctive rewritings returned by the MiniCon algorithm are:

\[ Q_1(name) : \neg V_1(NRIC, name) \]

\[ Q_2(name) : \neg V_2(NRIC, name), V_3(NRIC, station) \]

The correct price \( Q \) is

\[ price^V(Q) = \min\{p_1, p_2 + p_3\} = \min\{25, 15 + 12\} = 25 \]

The price of \( Q \) computed by Algorithm 1 is

\[ price^V(Q) = p_1 + p_2 + p_3 = 25 + 15 + 12 = 52 \]

\[ price^V(Q) < price^V(Q)', \] which means Algorithm 1 overcharges.

Example 4.5 shows the case that there are two equivalent rewritings \( Q_1, Q_2 \) returned by the MiniCon algorithm. There exist containment relationships between these two rewritings, i.e. \( Q_1 \subseteq Q_2, Q_2 \subseteq Q_1 \). By applying Algorithm 1, all the views used in \( Q_1, Q_2 \) will be charged. In fact, we need only \( Q_1 \) or \( Q_2 \), since they are equivalent rewritings.

**Example 4.6:**

Let us consider a query and three views.

\[ Q(name) : \neg \text{Passenger}(NRIC, name, gender, age), \quad At_{station}(NRIC, station, day, time) \]

\[ V_1(NRIC, name) : \neg \text{Passenger}(NRIC, name, gender, age), \quad At_{station}(NRIC, station, day, time) \]

\[ V_2(NRIC, name) : \neg \text{Passenger}(NRIC, name, gender, age) \]

\[ V_3(NRIC, station) : \neg At_{station}(NRIC, ‘Clementi’, day, time) \]

with the pre-set price \( p_1 = 25, p_2 = 15, p_3 = 5 \).

The two conjunctive rewritings returned by the MiniCon algorithm are:

\[ Q_1(name) : \neg V_1(NRIC, name) \]

\[ Q_2(name) : \neg V_2(NRIC, name), V_3(NRIC, station) \]

The correct price \( Q \) is

\[ price^V(Q) = p_1 = 25 \]

The price of \( Q \) computed by Algorithm 1 is

\[ price^V(Q) = p_1 + p_2 + p_3 = 25 + 15 + 5 = 45 \]

\[ price^V(Q) < price^V(Q)', \] which means Algorithm 1 overcharges.

**Case 2: there exist some containment relationships between two sets of the rewritings returned.**

**Example 4.7:**

Let us consider a query and four views.

\[ Q(name) : \neg \text{Passenger}(NRIC, name, gender, age), \quad At_{station}(NRIC, station, day, time) \]

\[ V_1(NRIC, name) : \neg \text{Passenger}(NRIC, name, gender, age), \quad 0 < age < 25 \]

\[ V_2(NRIC, name) : \neg \text{Passenger}(NRIC, name, gender, age), \quad 18 < age \]

\[ V_3(NRIC, name) : \neg \text{Passenger}(NRIC, name, gender, age), \quad 15 < age < 23 \]

\[ V_4(NRIC, station) : \neg At_{station}(NRIC, station, day, time) \]

with the pre-set price \( p_1 = 6, p_2 = 10, p_3 = 8, p_4 = 12 \).

The three conjunctive rewritings returned by the MiniCon algorithm are:

\[ Q_1(name) : \neg V_1(NRIC, name), V_4(NRIC, station) \]

\[ Q_2(name) : \neg V_2(NRIC, name), V_4(NRIC, station) \]

\[ Q_3(name) : \neg V_3(NRIC, name), V_4(NRIC, station) \]

The price \( Q \) in Definition 3.1 is

\[ price^V(Q) = p_1 + p_2 + p_3 + p_4 = 28 \]

The price of \( Q \) computed by Algorithm 1 is

\[ price^V(Q) = p_1 + p_2 + p_3 + p_4 = 6 + 10 + 8 + 12 = 36 \]

\[ price^V(Q) < price^V(Q)', \] which means Algorithm 1 overcharges in this case.

Example 4.7 shows that there are three non-equivalent rewritings \( Q_1, Q_2, Q_3 \) returned by the MiniCon algorithm.
$Q_1 \cup Q_2$ is equivalent to $Q$. There is no containment relationship among $Q_1, Q_2, Q_3$. However, there exists a containment relationship between two sets of rewritings, i.e. $Q_3 \subseteq Q_1 \cup Q_2$. By applying Algorithm 1, all the views used in $Q_1, Q_2, Q_3$ will be charged. In fact, we need only $Q_1, Q_2$ since $Q_3 \subseteq Q_1 \cup Q_2$.

V. MINICON WITH POST-PROCESSING

In this section, we devise an algorithm as a post-processing of the MiniCon algorithm to compute the price of a query in Definition 3.1.

Assume we have a query $Q$ and a set of views $V = \{V_1, ..., V_m\}$, and the MiniCon algorithm returns the set of rewritings as $Q(Q, V) = \{Q_1, ..., Q_n\}$, where $Q_i$ represents a conjunctive rewriting.

A. Challenges

According to Definition 3.1, the price of a given query is the price of the cheapest maximally-contained rewriting.

If there exists an equivalent rewriting, we get the cheapest one, whose price is the price of the query.

The first challenge is finding all the equivalent rewritings from $Q(Q, V)$. There are two cases of equivalent rewritings.

1) An equivalent rewriting can be a rewriting in $Q(Q, V)$, e.g. $Q_1, Q_2$ in Example 4.5 are two equivalent rewritings.

2) An equivalent rewriting can be the union of several rewritings in $Q(Q, V)$, e.g. $Q_1 \cup Q_2$ in Example 4.7 is an equivalent rewriting.

If there is no equivalent rewriting, the union of all the rewritings is a maximally-contained rewriting. However, there may exist redundant rewritings which are contained in another rewriting or contained in a set of rewritings.

The second challenge is removing all the unnecessary rewritings from the union of all the rewritings since we aim to find the cheapest maximally-contained rewriting. For example, in Example 4.7 we modify the $V_2$ to be

$V_2(NRIC, name) = \neg \text{Passenger} (NRIC, name, gender, age) \quad 18 < age < 30$

The result returned by the MiniCon algorithm is the same as that in Example 4.7. However, in this case, there is no equivalent rewriting. $Q_1 \cup Q_2 \cup Q_3$ is a maximally-contained rewriting, however it is not the cheapest one. The reason is that $Q_3$ is unnecessary due to $Q_3 \subseteq Q_1 \cup Q_2$.

B. Algorithm

To tackle the above challenges, we are facing the query containment and query equivalence problems which are NP-hard problems. In this section, we propose an algorithm to compute the price of a given query according to Definition 3.1. We discuss the complexity issues in Section 6.1.

1) Equivalent rewritings exist: To get the equivalent rewritings from the first case (line 2 to line 6), we have to check the query equivalence between the input query $Q$ and each rewriting $Q_i$. We get the set of equivalent rewritings $E = \{E_1, ..., E_p\}$ (line 4), and the set of non-equivalent rewritings $N = \{N_1, ..., N_q\}, p + q = n$ (line 6). In total, we test the query equivalence between two conjunctive queries $n$ times.

To get the equivalent rewritings from the second case (line 8 to line 11), we have to check the query equivalence between the input query $Q$ and a subset of $N$. We get the set of equivalent rewritings $U = \{U_1, ..., U_k\}$ (line 11). In total, we test query equivalence between one conjunctive query and one union of conjunctive query $2^n$ times.

If $E \cup U \neq \emptyset$, we choose the price of the cheapest equivalent rewriting as the price of $Q$, i.e.

$$price^V(Q) = \min_{X_i \in E \cup U} \sum_{V \in V(X_i)} p_i$$

where $V(X_i)$ represents the set of views in the rewriting (or the union of rewritings) $X_i$ (line 14).

2) No Equivalent rewritings: If $E \cup U = \emptyset$, there is no equivalent rewriting. In other words, all the returned rewritings are non-equivalent ($N = Q$ and $q = n$). We charge the views in all the rewritings, excluding the ones which are contained in some other rewritings (line 16 to line 21). For each rewriting $N_i \in N$, we check whether $N_i \subseteq \bigcup \{N(N_i \in N - \{N_i\})\}$ (line 18). If the answer is yes, remove $N_i$ from $N$ (line 19 to line 20). For instance, in the modified Example 4.7 in Section 5.1, $N = \{Q_1, Q_2, Q_3\}$. $Q_3$ is removed, since $Q_3 \subseteq Q_1 \cup Q_2$.

In total, we test the query containment between a conjunctive query and a union of conjunctive query $n$ times.
The price of $Q$ is the sum of the prices of the views used in the rewritings in $N$, i.e.

$$\text{price}^v(Q) = \sum_{V_k \in \bigcup_i V(N_i), N_i \in N} p_k$$

where $V(N_i)$ represents the set of views in the rewriting $N_i$ (line 21).

VI. CONCLUSION AND FURTHER ISSUES

A. Conclusion

We have proposed a novel data pricing model. The model is based on views. It uses rewritability instead of determinacy to connect the input query to a set of views. We have argued that determinacy is not sufficient since the views that determine the input query may not always be able to answer the query.

We have shown that the naïve application of the MiniCon algorithm for the computation of the price of a query using views leads to overcharging. Therefore we have circumvented this shortcoming and adapted the MiniCon algorithm for the computation of the price of a query using views by devising a post-processing phase.

B. Further Issues

Several further research issues worth pursuing concern the complexity of the algorithm, the selection of a view set and the revision of the price of views. We briefly outline the challenges ahead.

1) Complexity: After getting the set of rewritings from MiniCon algorithm, query containment is repeatedly tested. We transfer the query equivalence problem to a query containment problem since $Q_1 \equiv Q_2 \iff (Q_1 \subseteq Q_2 \land Q_2 \subseteq Q_1)$. In total, we have to test query containment between two conjunctive queries $2n$ times, and test query containment between a conjunctive query and a union of conjunctive queries $2^{n+1}+n$ times. The query containment problem of two conjunctive queries with arithmetic comparison is $\Pi_2^p$-complete. The query containment problem of union of conjunctive queries is also $\Pi_2^p$-complete. The complexity of Algorithm 2 is high when the number of returned conjunctive rewritings is large. One open question is to determine whether there exist some special and practical classes of queries for which the price computation can be done efficiently. Another question is whether, for general queries, there are any heuristics to approximate the price, and how effective and efficient such heuristics can be.

2) Choosing a view set: Using pre-defined views to answer queries and pricing queries with view prices faces a critical problem, which is how to choose a proper view set to (1) cover as many queries as possible, in terms of precise result returned and, (2) produce reasonable price for arbitrary queries. Generally, the more views defined, the more queries can be covered. However, when the number of views increases, especially when a group of views closely related to each other (e.g., in containment relationship or naturally joinable), the price set to each view must be theoretically verified to avoid potential problems, such as non-monotonicity. In particular, for any two queries in which one’s result contains the other’s, a good pricing model should charge higher price to the first query, even though the cheapest views used to answer the two queries are different.

3) Price revision for views: The view-based pricing model allows to prepare the processing for popular queries. However, when the database is updated, e.g., more data are added, the result for relevant views should also be updated. For example, it is not reasonable to set a fixed price to a view that returns tuples of 3 months and 6 months in two versions of database. To reflect the value of data, the price of a view needs to be revised when the database is updated. How to revise the price of a view based on data returned is not trivial. It is co-related to how to set price to a view, how to value each data record, etc.

REFERENCES