THE NATIONAL UNIVERSITY
of SINGAPORE

School of Computing
Computing 1, 13 Computing Drive, Singapore 117417

TRB5/12

The Price is Right
The Price of Relational and Probabilistic Relational Data

Ruiming Tang, Huayu Wu, Thomas Kister and Stéphane Bressan

May 2012

T e c h n i c a l R e p o r t

Foreword

This technical report contains a research paper, development or tutorial article, which has been submitted for publication in a journal or for consideration by the commissioning organization. The report represents the ideas of its author, and should not be taken as the official views of the School or the University. Any discussion of the content of the report should be sent to the author, at the address shown on the cover.

OOI Beng Chin
Dean of School

The Price Is Right

The Price of Relational and Probabilistic Relational Data

Ruiming Tang
National University of Singapore
tangruiming@nus.edu.sg

Huayu Wu
Institute for Infocomm Research, Singapore
huwu@i2r.a-star.edu.sg

Thomas Kister
National University of Singapore
thomas.kister@nus.edu.sg

Stéphane Bressan
National University of Singapore
steph@nus.edu.sg

ABSTRACT

Data is a modern commodity. Electronic data market places are being designed and deployed. Yet current pricing models either focus on processing or are proprietary, opaque and not conducive of a healthy commodity market dynamics.

In this paper we propose a generic data pricing model that is based on minimal lineages, i.e. sets of tuples contributing to the result of a query. We show that the proposed model fulfils desirable properties such as contribution monotonicity, bounded price and arbitrage-freedom. We extend the model to the case of probabilistic databases in the X-tuple model. We propose algorithms to the compute the price of a query based on our pricing models.

1. INTRODUCTION

Organizational judgement is currently undergoing a fundamental change, from a reliance of a leader's human intuition to a data based analysis [3]. Brynjolfsson et al. in [3] found that among 179 large publicly traded firms, the ones that adopt data-driven decision-making have an output and productivity increase of between 5 and 6% beyond what can be explained by traditional inputs and IT usage.

In the science community, data is taking such a prominent place that Jim Gray argues: "The techniques and technologies for such data-intensive science are so different that it is worth distinguishing data-intensive science from computational science as a new, *fourth paradigm* for scientific exploration." [9]

As argued by Balazinska, Suciu and their co-authors in [2] and [11], data is being bought and sold. Electronic data market places are being designed and deployed. Independent vendors, such as Aggdata [12] and Microsoft's Azure MarketPlace [13], aggregate and organize the distribution of data online.

In current pricing models sellers set prices to views. Buyers are restricted to buy these pre-defined views. Such pricing models are simplistic, inflexible and can create undesirable arbitrage situations [11].

The authors of [11] propose a pricing model that defines the price of an arbitrary query as the minimum sum of the prices of views that can answer the query.

In this paper we revisit the pricing model of [11] from the point of view of data lineage, i.e. sets of tuples contributing to the result of a query. Our model consists in attributing a price to each tuple. We calculate the price of a query result solely on the basis of the necessary tuples for the result, which we call *contributing tuples*. We do not take the computational cost into account. As such, we consider that for a given query, a tuple which is used more than once is only charged once.

We explore alternative ways of computing prices using p-norms. We give a definition of a pricing model that fulfils the following desirable properties:

Contribution monotonicity: the more tuples one query uses, the higher the price is.

Bounded price: the price of a given query is not higher than the price of all the tuples in the relations involved in it.

Arbitrage-freedom: the price of a query Q cannot be higher than the sum of the prices of queries which only need subsets of the contributing tuples needed by Q.

Furthermore, modern applications need to process uncertain data coming from diverse and autonomous sources. This requirement compels new data models able to manage uncertainty. Probabilistic data models are candidates for such solutions. Thus we further extend the pricing model to the case of probabilistic databases in the X-tuple model. We consider an X-tuple to be a contributing tuple, which means that using any alternative tuple of a given X-tuple means the whole X-tuple is charged. However we still consider that there is a price for each alternative tuple and discuss how to compute the price of an X-tuple.

We provide two algorithms to compute the price of a query in relational data model and in probabilistic relation data model, based on our proposed pricing functions.

This paper is organized as follows: we first provide in section 2 a complete background information about the notion of lineage in relational databases which is the basis of our paper. Section 3 is dedicated to our devised pricing model on relational databases, and section 4 extends our work on probabilistic relational databases. Finally section 5 concludes our work and reviews the different problems and questions that are raised in this paper and could be addressed by future works.

2. BACKGROUND AND RELATED WORK

2.1 Relational Data Lineage

Data provenance, sometimes called "lineage", is the description of the source of a piece of data ([5]). As a development of the increased usage of curated databases, more and more data from different sources are integrated into one database. Users may want to know the source of output results when they query the curated database. Data provenance has attracted a lot of research interests.

The authors of [5] state that there are three most common forms of "lineage" describing the relationship between data in the input and in the output. The first one is "why-provenance", showing the input tuples that explain why the output tuple is produced. [7] and [4] introduce two kinds of "why-provenance". The second one is "how-provenance" [10], showing how the output tuple is produced by the input tuples. The third one is "where-provenance" [4], showing where the attributes of the output tuple come from the attributes of input tuples. We use Example 2.1 to illustrate these forms of lineage.

Relation S

tid	Name	Code
t_1	JSH	J37
t_2	UOB	U11
t_3	DBS	C09
t_4	UOB	GL8

Relation T (on 20/01/2012)

tid	Code	Timestamp	Price ($)
t_5	J37	01:22 pm	31.120
t_6	J37	02:40 pm	30.956
t_7	U11	11:34 am	5.954
t_8	GL8	12:15 am	10.097

Figure 1: Database $\mathcal{D}$

EXAMPLE 2.1. *$\mathcal{D}$ is a database described by Figure 1. Relation S records the codes of stocks, bonds and other securities together with the name of the issuing company. For instance, the company JSH issues the stock J37. Relation T records the historical prices of securities on the 20th of January 2012. For instance, the stock GL8 was sold at $10.097 at 12:15 am.*

If we are interested in the names of the different companies whose securities were traded on 20th of January 2012, the query is:

$$Q = \{< n > |\exists c, \exists t, \exists p(S(n, c) \wedge T(c, t, p))\}$$

The query result is:

$$Q(\mathcal{D}) = \{\texttt{<JSH>}, \texttt{<UOB>}\}$$

Name	Why provenance [7]	Minimal why provenance [4]
JSH	$\{t_1, t_5, t_6\}$	$\{t_1, t_5\}, \{t_1, t_6\}$
UOB	$\{t_2, t_4, t_7, t_8\}$	$\{t_2, t_7\}, \{t_4, t_8\}$
Name	How provenance [10]	Where provenance [4]
JSH	$t_1 \cdot t_5 + t_1 \cdot t_6$	$\{t_1.Name\}$
UOB	$t_2 \cdot t_7 + t_4 \cdot t_8$	$\{t_2.Name\}, \{t_4.Name\}$

Table 1: Lineage

Table 1 records different kinds of provenances of each output tuple.

2.1.1 Why-provenance

The why-provenance is the set of input tuples contributing to an output tuple. Different definitions of "contributing" bring different versions of why-provenance. The simplest approach is to define the set of input tuples contributing to an output tuple to be a subset of the input database that is sufficient to ensure that the output tuple is produced. There may be a large number of such sets because many tuples are "irrelevant" to the presence of the output tuple.

To avoid this problem, the authors of [7] try to filter the "irrelevant" tuples. The why-provenance they introduced is the maximal subset of tuples producing the output tuple, of which each tuple helps to produce the output tuple. In Example 2.1, the why-provenance of the output tuple <JSH> is $\{t_1, t_5, t_6\}$. It cannot be $\{t_1, t_5\}$ because that it is not the maximal set. It cannot be $\{t_1, t_5, t_6, t_8\}$ neither because t_8 does not help to produce <JSH>. One issue about the why-provenance computed by [7] is that it may contain tuples that are unnecessary for the output tuple. For example, in Example 2.1, the why-provenance of <JSH>, which is $\{t_1, t_5, t_6\}$, contains unnecessary tuples for producing <JSH>. $\{t_1, t_5\}$ (or $\{t_1, t_6\}$) is enough to produce <JSH>, thus $\{t_6\}$ (or $\{t_5\}$) is unnecessary.

The authors of [7] construct another query (or queries) to get the why-provenance. In our example, to compute the why-provenance of the output tuple <JSH>, the process is:

1. The value of the joined attribute "J37" can be obtained by:
$$\{< c > |S(n, c) \wedge n = JSH\}$$
2. The rewritten queries are:
$$Q_1^* = \{< n, c > |S(n, c) \wedge (c = J37)\}$$
$$Q_2^* = \{< c, t, p > |T(c, t, p) \wedge (c = J37)\}$$
3. The why-provenance is $Q_1^*(\mathcal{D}) \cup Q_2^*(\mathcal{D})$.

To overcome the problem of including unnecessary tuples in the why-provenance, [4] tries to compute smaller sets of why-provenances, and even the minimal ones. The authors construct another query (or queries) to separate the set of contributing tuples into several different why-provenances. In our example, to compute the why-provenance of the output tuple <JSH>, the last step of the process is instead:

3. The why-provenance is the pair-wise union of $Q_1^*(\mathcal{D})$ and $Q_2^*(\mathcal{D})$, i.e. $\{s \cup t | s \in Q_1^*(\mathcal{D}), t \in Q_2^*(\mathcal{D})\}$, that is:
$$\{\{t_1, t_5\}, \{t_1, t_6\}\}$$

Among the set of why-provenances, they define "minimal" ones. A why-provenance is minimal if and only if none of its proper subsets can produce the output tuple. There may exist several minimal why-provenances for an output tuple. In Example 2.1, there are two minimal why-provenances of <JSH>: $\{t_1, t_5\}$ and $\{t_1, t_6\}$. $\{t_1, t_5\}$ is a minimal why-provenance of <JSH> because it can produce <JSH> and none of its proper subsets can produce <JSH>.

Both [7] and [4] try to filter the "irrelevant" tuples from why-provenance, and both of the two why-provenances are sufficient to produce the output tuple. The differences between them are as follows. Firstly, the why-provenance introduced by [7] is the maximal set of tuples being able to produce the output tuple, without "irrelevant" tuples, while the why-provenance introduced by [4] is the minimal set of tuples being able to produce the output tuple. Secondly, each tuple in the why-provenance introduced by [7] may not be necessary to produce the output tuple, while each tuple in the why-provenance introduced by [4] is necessary to produce the output tuple. Lastly, the why-provenance introduced by [7] is unique, while the number of why-provenances introduced by [4] may be larger than one.

2.1.2 How-provenance

Why-provenance describes the source tuples which contribute to an output tuple in the query result. However, it does not include the information of how an output tuple is derived from the source tuples. How-provenance, defined in [10], describes how the input tuples contribute to an output tuple. First, an annotation is assigned to every input tuple (t_i in Example 2.1). Then different binary operators are defined for different relational algebra operators. The product is represented by the $\cdot$ operator. The annotation of doing a product between two tuples t_i and t_j is $t_i \cdot t_j$. Union and projection are represented by the $+$ operator. The annotation of doing a projection from several tuples $t_1, t_2, ..., t_x$ is $t_1 + t_2 + ... + t_x$. In this way, the how-provenance of each output tuple is represented as a polynomial. In Example 2.1, the how-provenance of <JSH> is $t_1 \cdot t_5 + t_1 \cdot t_6$. It means that there are two different ways to produce <JSH>: the first way is joining t_1 and t_5, the second way is joining t_1 and t_6. By knowing this polynomial expression, we understand how these input tuples contributing to the output tuple <JSH>.

2.1.3 Where-provenance

Why-provenance describes why an output tuple is in the result, and how-provenance concerns how an output tuple is produced. However, one might ask where does each cell of an output tuple come from. Based on this, [4] introduces where-provenance. Intuitively, where-provenance describes where a piece of data is copied from. It describes the relationship between input and output cells. The where-provenance of a value in a cell in an output tuple consists of cells of input tuples from where the value is copied from. In Example 2.1, the where-provenance of <JSH> is $\{t_1.Name\}$. It means that the value <JSH> comes from the "Name" attribute of t_1.

2.1.4 Usage

The differences among these three definitions of provenance are stated in Table 2.

	Why	Where	How
tuple-level	√		√
attribute-level		√	
declarative	√	√	
procedural			√

Table 2: Differences among different definitions of provenance

The authors of [5] summarize different approaches of computing provenance. They categorize the approaches into two clusters, which are eager approach and lazy approach. In the eager approach, the query is re-engineered so that the provenance information is carried along the query processing. In the lazy approach, the provenance is computed after we get the query result by examining the source database, the query result and also the query itself.

These two approaches can be used in different scenarios due to their advantages and disadvantages. The advantage of the eager approach is that it can provide the provenance as soon as the query result is generated, and it never needs to use the source database again. However, since the provenance information is carried along query processing, it will introduce overhead. Furthermore, the eager approach requires to re-engineer the query, thus it needs to change the current systems. The eager approach is suitable in the case that the current systems can be modified and we need the provenance information as soon as we get the query result. The lazy approach can avoid the overhead and avoid to change the current systems. However, in the lazy approach it is usually very complicated to derive the provenance after the query processing. Thus, the lazy approach is suitable when we cannot modify the current query processing systems and when we can reach the data source after querying.

There are some works computing the why-provenance by the lazy approach ([7], [4]). There are some works computing the how-provenance by the eager ([8]) or the lazy approach ([6]). There are some works computing the where-provenance by the eager approach.

All the proposed lineages (or provenances) work for individual tuples in the query result. The lineage in our paper works for the query result as a whole.

2.2 Data Pricing

[2] shows that the existing vendors selling data online mainly use two pricing models. In the first model, data consumers pay monthly subscriptions that enable a maximum number of queries per month. In the other model, a user can pay for downloading the entire data set. [2] also shows that these two models suffer from several weaknesses, such as assuming all tuples are of equal value, leading to arbitrage situations, etc. The authors of [2] present some research challenges. One of the challenge is defining a proper pricing model. One possible pricing model they suggest is captured in the framework of provenance semirings ([10]): the pricing semiring is $(R^+, min, +, \infty, 0)$.

The authors of [11] propose a framework for pricing data that allows the seller to set explicit prices for only a few views, yet allows the buyers to buy any query: the price of the query is derived automatically from the explicit prices of the views. They identify two important properties that the pricing function has to satisfy, called arbitrage-free and discount-free. They prove there must be such a unique pricing function, which is both arbitrage-free and discount-free, and also agrees with the seller's explicit prices. Informally, the pricing function they propose is computing the price of the cheapest set of views which can determine the query. They show that when the views are selections, computing the price of certain conjunctive queries is NP-hard in the size of the input database. Thus they study the problem of computing the price of "Generalized Chain Query". They prove that this problem can be reduced to the classic "MIN-CUT" problem, which can be solved in P-TIME.

In this paper, we propose a framework for pricing relational data, and extend it for pricing probabilistic relational data. Our work differentiates itself from [11] in the following aspects. Firstly, our work considers probabilistic data pricing, while [11] does not. Secondly, we revisit the pricing model of [11] from the point of view of data lineage, i.e. sets of tuples contributing to the result of a query. We explore alternative ways of computing prices using p-norms. Thirdly, [11] does not include algorithms computing their pricing function in general cases, while we propose an algorithm to compute the price using our pricing function.

3. PRICING RELATIONAL DATA

In this section, we propose a generic pricing model which consists in a price setting function attributing a price to each tuple and a pricing function defining the price of a query in Section 3.1. We define the minimal lineages of the result of a query, and prove that they are invariant to query rewriting in Section 3.2. We use p-norm to define the price of a set of tuples in Section 3.3. We also propose a pricing function which calculates the price of a query result solely on the basis of its *contributing tuples*, i.e. the necessary and sufficient tuples to produce the result, and prove it satisfies three properties in Section 3.4. We propose an algorithm to compute the price in Section 3.5.

3.1 The Pricing Model

In our pricing model, it consists in two functions.

DEFINITION 3.1. *(Price setting function). Let $\mathcal{D}$ be a database. A* price setting function *is a function $s_{\mathcal{D}} : \mathbb{T} \to \mathbb{R}^+$, where $\mathbb{T}$ is the set of tuples in $\mathcal{D}$, and $\mathbb{R}^+$ is the set of non-negative real numbers.*

DEFINITION 3.2. *(Pricing function). Let $\mathcal{D}$ be a database. A* pricing function *is a function $pr_{\mathcal{D}} : \mathbb{Q} \to \mathbb{R}^+$, where $\mathbb{Q}$ is the set of queries, and $\mathbb{R}^+$ is the set of non-negative real numbers.*

If the user asks for a query Q, then he has to pay the price $pr_{\mathcal{D}}(Q)$, where $\mathcal{D}$ is the current database. The data seller does not specify the pricing function, but only gives prices for individual tuples in $\mathcal{D}$ (as in Definition 3.1). The system computes the pricing function automatically.

3.2 Minimal Lineage

We only consider the necessary tuples, in order not to charge for the entire database for any arbitrary query. However we have a set of minimal solutions rather than a minimum solution. We define the lineage for the query result as a whole, without defining the lineage for the individual tuples in the query result. In our pricing model, we charge for a tuple at most once, regardless of how many times this tuple contributes to the query result. This is because we aim to charge by data content, rather than computation.

DEFINITION 3.3. *(A lineage of a set of tuples). Let Q be a query. Let $\mathcal{D}$ be a database. Let $Q(\mathcal{D})$ be the result of the query Q on the database $\mathcal{D}$. The* lineage *of $Q(\mathcal{D})$ is a set of tuples L ($L \subseteq \mathcal{D}$) such that:*

$$Q(\mathcal{D}) \subseteq Q(L)$$

DEFINITION 3.4. *(A minimal lineage of a set of tuples). Let Q be a query. Let $\mathcal{D}$ be a database. Let $Q(\mathcal{D})$ be the result of the query Q on the database $\mathcal{D}$. The* minimal lineage *of $Q(\mathcal{D})$ is a lineage L of $Q(\mathcal{D})$ such that:*

$$\forall L', L' \subseteq L \Rightarrow L' = L$$

where L' is a lineage L of $Q(\mathcal{D})$.

EXAMPLE 3.1. *Following Example 2.1, there are four minimal lineages of the query result $Q(\mathcal{D})$: $L_1 = \{t_1, t_2, t_5, t_7\}$, $L_2 = \{t_1, t_4, t_5, t_8\}$, $L_3 = \{t_1, t_2, t_6, t_7\}$, $L_4 = \{t_1, t_4, t_6, t_8\}$.*

From Example 3.1, we can see that the query result $Q(\mathcal{D})$ may have several minimal lineages. We use $M(Q, \mathcal{D})$ to represent the set of minimal lineages of $Q(\mathcal{D})$.

We prove that, the set of minimal lineages remains invariant to query rewriting.

THEOREM 3.1. *Two equivalent queries have the same set of minimal lineages for any database.*

The proof of Theorem 3.1 is in the Appendix.

3.3 p-norm

In this section, we use p-norm to define the price of a set of tuples. The p-norm [14] is a function defining a norm of a vector.

DEFINITION 3.5. *(p-norm of a vector). Let X be a vector $(x_1, x_2, ..., x_n)$, for a real number $p \geq 1$, the p-norm of X is defined by*

$$||X||_p = (\sum_{i=1}^{n} |x_i|^p)^{\frac{1}{p}}$$

For $p = 1$, $||X||_1 = \sum_{i=1}^{n} |x_i|$. $||X||_1$ is the Manhattan norm of X.

For $p = 2$, $||X||_2 = (\sum_{i=1}^{n} x_i^2)^{\frac{1}{2}}$. $||X||_2$ is the Euclidean norm of X.

For $p = \infty$, $||X||_\infty = max\{|x_1|, |x_2|, ..., |x_n|\}$.

[14] states several properties of p-norm. X, Y are vectors of n dimensions and a is a complex number.

1. positivity. If $X \neq \mathbf{0}$, then $||X||_p > 0$.
2. positive scalability. $||a \times X||_p = |a| \times ||X||_p$.
3. subadditivity. $||X + Y||_p \leq ||X||_p + ||Y||_p$.
4. separate point. If $||X||_p = 0$, then $X = \mathbf{0}$.

In this paper, we use the notion of p-norm to define the price of a set of tuples.

DEFINITION 3.6. *(The price of a set of tuples). Let $\mathcal{D}$ be a database. Let X be a set of tuples, and $X \subseteq \mathcal{D}$. Let $s_{\mathcal{D}}$ be a pricing setting function. The price of the set of tuples X is defined by*

$$||X||_p = (\sum_{i=1}^{n} s_{\mathcal{D}}(t_i)^p)^{\frac{1}{p}}$$

where $t_i \in X$.

This definitions allows us to define the price of a set of tuples as a function that ranges from a sum of the prices of the individual tuples when p is equal to 1, to a maximum function when p tends to infinity. In between, the value of the prices decreases as p increases.

PROPOSITION 3.1. *If $p \geq 1$ and $a \geq 0$, then $||X||_{p+a} \leq ||X||_p$.*[1]

The proof of Proposition 3.1 is in the Appendix.

PROPOSITION 3.2. *Assume there are two sets X_1 and X_2. if $X_1 \subseteq X_2$, then $||X_1||_p \leq ||X_2||_p$.* [2]

The proof of Proposition 3.2 is in the Appendix.

3.4 The Pricing Function

DEFINITION 3.7. *(Contributing tuples). Let Q be a query. Let $\mathcal{D}$ be a database. Let $Q(\mathcal{D})$ be the result of the query Q on the database $\mathcal{D}$. The set of* contributing tuples *of $Q(\mathcal{D})$ is the set of minimal lineages of $Q(\mathcal{D})$, i.e. $M(Q, \mathcal{D})$.*

Based on the definition of contributing tuples and Theorem 3.1, we can infer that two equivalent queries have the same set of contributing tuples for any database.

DEFINITION 3.8. *(Contribution containment and contribution equivalence). Let $\mathcal{D}$ be a database. Let Q_1 and Q_2 be two queries. $M(Q_1, \mathcal{D})$, $M(Q_2, \mathcal{D})$ are the sets of contributing tuples of $Q_1(\mathcal{D})$, $Q_2(\mathcal{D})$, respectively. Q_1 is* contribution contained *in Q_2, namely $Q_1 \subseteq_{C(\mathcal{D})} Q_2$, if and only if:*

$$\forall L'(L' \in M(Q_2, \mathcal{D}) \Rightarrow \exists L(L \in M(Q_1, \mathcal{D}) \wedge L \subseteq L'))$$

Q_1 and Q_2 are contribution equivalent*, namely $Q_1 \equiv_{C(\mathcal{D})} Q_2$, if and only if $Q_1 \subseteq_{C(\mathcal{D})} Q_2$ and $Q_2 \subseteq_{C(\mathcal{D})} Q_1$.*

$Q_1 \subseteq_C Q_2$ if and only if for any database $\mathcal{D}$, $Q_1 \subseteq_{C(\mathcal{D})} Q_2$.

$Q_1 \equiv_C Q_2$ if and only if $Q_1 \subseteq_C Q_2$ and $Q_2 \subseteq_C Q_1$.

[1] It works for the general case of p-norm.

[2] It can be easily adapted from set to vector. It works for the general case of p-norm.

EXAMPLE 3.2. *Let $\mathcal{D}$ be a database. Let Q_1, Q_2 and Q_3 be three queries. $M(Q_1, \mathcal{D})$, $M(Q_2, \mathcal{D})$ and $M(Q_3, \mathcal{D})$ are the sets of contributing tuples of $Q_1(\mathcal{D})$, $Q_2(\mathcal{D})$ and $Q_3(\mathcal{D})$. Assume $M(Q_1, \mathcal{D}) = \{l_1^1 = \{t_1, t_2\}, l_1^2 = \{t_2, t_3\}\}$, $M(Q_2, \mathcal{D}) = \{l_2^1 = \{t_1, t_2, t_3\}\}$, $M(Q_3, \mathcal{D}) = \{l_3^1 = \{t_1, t_2\}, l_3^2 = \{t_1, t_3\}\}$.*

$Q_1 \subseteq_{C(\mathcal{D})} Q_2$ because for l_2^1, there exists l_1^1 such that $l_1^1 \subseteq l_2^1$. However, $Q_1 \nsubseteq_{C(\mathcal{D})} Q_3$ because for l_3^2, $l_1^1 \nsubseteq l_3^2$ and $l_1^2 \nsubseteq l_3^2$.

DEFINITION 3.9. *(contribution monotonicity). A pricing function is said to be* contribution monotonic *if given a database $\mathcal{D}$, whenever two queries Q_1, Q_2 satisfy $Q_1 \subseteq_{C(\mathcal{D})} Q_2$, then their prices satisfy $pr_{\mathcal{D}}(Q_1) \leq pr_{\mathcal{D}}(Q_2)$.*

The contribution monotonicity tells that if a query Q_1 needs less source tuples to produce the query result than Q_2, then the price of Q_1 is lower than the price of Q_2. However, in general, for each query, there may exist more than one way to produce the query result. We require that the price of Q_1 is lower than the price of Q_2 if for every way to produce the query result of Q_2, we can find a cheaper way to produce the query result of Q_1.

DEFINITION 3.10. *(bounded price). A pricing function is said to be* bounded price *if for any database $\mathcal{D}$, the price of a query is always not higher than the p-norm of the prices of the tuples in the relations which are involved in the query, i.e.*

$$pr_{\mathcal{D}}(Q) \leq ||S||_p$$

where $S = \{t \in \mathcal{R} | \mathcal{R} \propto Q\}$ and $\mathcal{R} \propto Q$ means that the relation $\mathcal{R}$ is involved in the query Q.

The bounded price tells that the price of any query is not higher than the p-norm of the price of buying the entire database, or more precisely, it is not higher than the p-norm of the price of the relations involved in the query. Below we show that bounded price is a product of contribution monotonicity.

COROLLARY 3.1. *If a pricing function is contribution monotonic, then it is bounded price.*

PROOF. Let $\bigcup_{\mathcal{R} \propto Q} \mathcal{R}$ be the set of relations involved in the query Q. Let Q^* be a special query as: `select * from` $\bigcup_{\mathcal{R} \propto Q} \mathcal{R}$, which means select all the tuples in the relations involved in the query.

It is true that for any query Q, $Q \subseteq_C Q^*$. The reason is as following. For any database $\mathcal{D}$, $M(Q^*, \mathcal{D}) = \{\{t | t \in \bigcup_{\mathcal{R} \propto Q} \mathcal{R}\}\}$. For each $L \in M(Q, \mathcal{D})$, we can have $L \subseteq \{t | t \in \bigcup_{\mathcal{R} \propto Q} \mathcal{R}\}$. Thus $Q \subseteq_{C(\mathcal{D})} Q^*$.

Thus for any database $\mathcal{D}$,

$$pr_{\mathcal{D}}(Q) \leq pr_{\mathcal{D}}(Q^*) = ||S||_p$$

□

DEFINITION 3.11. *(Arbitrage-freedom). Given a database $\mathcal{D}$, a query Q, and a set of queries $Q_i (1 \leq i \leq m)$. $M(Q, \mathcal{D})$ is the set of contributing tuples of query Q, and $M(Q_i, \mathcal{D})$ is the set of contributing tuples of query Q_i. A pricing function is* arbitrage-free *if*

$$\forall S (S = \bigcup_{1 \leq i \leq m} L_{k_i}^i (L_{k_i}^i \in M(Q_i, \mathcal{D})) \Rightarrow \exists L (L \in M(Q, \mathcal{D}) \wedge L \subseteq S))$$

$||\{pr_{\mathcal{D}}(Q_1), ..., pr_{\mathcal{D}}(Q_m)\}||_p \geq pr_{\mathcal{D}}(Q)$ holds.

EXAMPLE 3.3. *Let $\mathcal{D}$ be a database. Let Q_1, Q_2 and Q be three queries. $M(Q_1, \mathcal{D})$, $M(Q_2, \mathcal{D})$ and $M(Q, \mathcal{D})$ are the sets of contributing tuples of $Q_1(\mathcal{D})$, $Q_2(\mathcal{D})$ and $Q(\mathcal{D})$. Assume $M(Q_1, \mathcal{D}) = \{l_1^1 = \{t_1, t_2\}, l_1^2 = \{t_2, t_3\}\}$, $M(Q_2, \mathcal{D}) = \{l_2^1 = \{t_2, t_4\}, l_2^2 = \{t_1, t_4\}\}$, $M(Q, \mathcal{D}) = \{l^1 = \{t_1, t_2\}, l^2 = \{t_2, t_3\}, l^3 = \{t_1, t_4\}\}$.*

For $l_1^1 \cup l_2^1 = \{t_1, t_2, t_4\}$, there exists $l^1 \subseteq l_1^1 \cup l_2^1$.
For $l_1^2 \cup l_2^1 = \{t_2, t_3, t_4\}$, there exists $l^2 \subseteq l_1^2 \cup l_2^1$.
For $l_1^1 \cup l_2^2 = \{t_1, t_2, t_4\}$, there exists $l^1 \subseteq l_1^1 \cup l_2^2$.
For $l_1^2 \cup l_2^2 = \{t_1, t_2, t_3, t_4\}$, there exists $l^1 \subseteq l_1^2 \cup l_2^2$.

If we have $||\{pr_{\mathcal{D}}(Q_1), pr_{\mathcal{D}}(Q_2)\}||_p \geq pr_{\mathcal{D}}(Q)$, then we say this pricing function is arbitrage-free.

Given a query Q and a batch of queries $Q_i (i = 1, 2, ..., n)$, arbitrage-freedom tells that the price of Q is lower than the p-norm of the prices of Q_i if for every way to produce the query result of all $Q_i (i = 1, 2, ..., n)$, we can find a cheaper way to produce the query result of Q. Arbitrage-freedom is more general than contribution monotonicity, in the sense that contribution monotonicity describes price relationship between a query and another query, while arbitrage-freedom describes price relationship between a query and a batch of queries.

DEFINITION 3.12. *(Price of a query). The price of a query Q in a database $\mathcal{D}$ is defined as the price of cheapest minimal lineage of $Q(\mathcal{D})$:*

$$pr_{\mathcal{D}}(Q) = min_{L \in M(Q, \mathcal{D})} ||L||_p$$

THEOREM 3.2. *This pricing function is contribution monotonic.*

PROOF. Assuming Q_1, Q_2 satisfy $Q_1 \subseteq_{C(\mathcal{D})} Q_2$.

We use L'_{min} to represent the cheapest minimal lineage of $Q_2(\mathcal{D})$, thus $pr_{\mathcal{D}}(Q_2) = ||L'_{min}||_p$.

Since $Q_1 \subseteq_{C(\mathcal{D})} Q_2$, by definition we know $\exists L \in M(Q_1, \mathcal{D})$ such that $L \subseteq L'_{min}$.

$$\begin{aligned} pr_{\mathcal{D}}(Q_2) &= ||L'_{min}||_p \geq ||L||_p \\ &\geq min_{L \in M(Q_1, \mathcal{D})} ||L||_p = pr_{\mathcal{D}}(Q_1) \end{aligned}$$

□

COROLLARY 3.2. *This pricing function is bounded price.*

THEOREM 3.3. *This pricing function is arbitrage-free.*

PROOF. Assuming Q and $Q_i (1 \leq i \leq m)$ satisfy

$$\forall S (S = \bigcup_{1 \leq i \leq m} L_{k_i}^i (L_{k_i}^i \in M(Q_i, \mathcal{D})) \Rightarrow \exists L (L \in M(Q, \mathcal{D}) \wedge L \subseteq S))$$

Assume L_{min}^i is the cheapest minimal lineage of $M(Q_i, \mathcal{D})$, thus $\exists L (L \in M(Q, \mathcal{D}) \wedge L \subseteq \bigcup_i L_{min}^i)$

$$\begin{aligned} pr_{\mathcal{D}}(Q) &\leq ||L||_p \leq ||\bigcup_i L_{min}^i||_p \\ &\leq ||\{||L_{min}^i||_p, ..., ||L_{min}^m||_p\}||_p \\ &= ||\{pr_{\mathcal{D}}(Q_1), ..., pr_{\mathcal{D}}(Q_m)\}||_p \end{aligned}$$

□

3.5 Computing Price

In this section, we present a naive algorithm to compute the price of a query using our proposed pricing function. However, as we show, the complexity of the algorithm is intractable. Therefore,

in future, we need to look for heuristics to approximate the price of data at lower complexity and for special favorable classes of queries.

Our proposed minimal lineage is similar to the minimal why-provenance proposed in [4], in the sense that they are both minimal. However, our proposed minimal lineage differentiates itself from the minimal why-provenance in the sense that it works for a set of tuples, while minimal why-provenance works for individual tuples. We do not consider that a minimal lineage of a set of tuples is the union of the minimal why-provenances of the individual tuples. We give a counter-example.

EXAMPLE 3.4. *Assume $Q(\mathcal{D}) = \{t_1, t_2\}$. The set of minimal why-provenance of t_1 is $\{\{a_1, a_2\}, \{a_1, a_4\}\}$. The set of minimal why-provenance of t_2 is $\{\{a_3, a_4\}\}$. If the argument is correct, then the set of minimal lineage of $Q(\mathcal{D})$ is $\{\{a_1, a_2, a_3, a_4\}, \{a_1, a_3, a_4\}\}$. It is incorrect. $\{a_1, a_2, a_3, a_4\}$ is not a minimal lineage, since its proper subset ($\{a_1, a_3, a_4\}$) is a minimal lineage.*

In order to get the correct minimal lineages, we check for the validation of each candidate in the "union" set. If a proper subset of the checked candidate is also a candidate, then this checked candidate is not a valid minimal lineage.

THEOREM 3.4. *Let Q be a query. Let $\mathcal{D}$ be a database. Let $Q(\mathcal{D})$ be the result of the query Q on the database $\mathcal{D}$. For each tuple $t_i \in Q(\mathcal{D})$, L_i is the set of minimal why-provenances (defined in [4]). We have $M(Q, \mathcal{D})$ to be the set of minimal lineages of $Q(\mathcal{D})$. We denote $U(Q, \mathcal{D}) = \{\bigcup_i l_i | l_i \in L_i\}$. We get $U'(Q, \mathcal{D})$ from $U(Q, \mathcal{D})$ by filtering non-minimal lineages. It holds that $M(Q, \mathcal{D}) = U'(Q, \mathcal{D})$*

The proof of Theorem 3.4 is in the Appendix.

As reviewed in Section 2.1.4, there exist two approaches to compute the lineages of a tuple in the query result, which are eager approach and lazy approach. In this section, we adapt a lazy approach to compute the minimal lineages of the query result, and to compute the price. The advantage of this lazy approach is that we do not need to modify the current database system and can use them directly. An eager approach may also be developed but it requires to modify the database systems.

Algorithm 1 is the algorithm computing the price of a query Q given a database $\mathcal{D}$. Theorem 3.4 guarantees the correctness of Algorithm 1. In Algorithm 1, we use $M(Q, \mathcal{D})$, instead of $U(Q, \mathcal{D})$ (or $U'(Q, \mathcal{D})$) since we can prove the equality of $M(Q, \mathcal{D})$ and $U'(Q, \mathcal{D})$.

Assume we have n tuples in the query result, and each tuple has m why-provenances.

We first determine the why-provenances in line 2 according to the method described in [5] (the why-provenance is defined in [4], and the the way of computing it in the context of relational database is adapted in [5]). It uses an extended relational algebra which can generate an inverse query (or inverse queries). The new operators provided by this algebra aim at recombining the results of their subqueries to provide the correct sets of why-provenances. To deal with selection, we include the result of the inverse query as why-provenance. To deal with projection, we separate each tuple in the result of the inverse query as one why-provenance. To deal with join, we need two inverse queries, and the set of the why-provenances is the pair wise union of the results of the two inverse queries. The example of Section 2.1.1 already details how a join operator is taken into account.

Then we compute the set of minimal why-provenances of each tuple (from line 3 to line 7). The approach is traversing all the

Algorithm 1: Price computation algorithm

Data: a query Q, a database $\mathcal{D}$
Result: price $pr_{\mathcal{D}}(Q)$

```
1  for each tuple t_i in the query result Q(D) do
2      get the set of why-provenances l_i defined in [4];
3      for each why-provenance of t_i do
4          if it is a minimal one then
5              keep it;
6          else
7              filter it from l_i;
8  Construct the set of all the candidates of minimal lineages of
   Q(D), namely M(Q, D), by selecting a minimal
   why-provenance from l_i of each t_i;
9  for each candidate of minimal lineage L ∈ M(Q, D) do
10     if L is a minimal lineage then
11         keep it;
12     else
13         filter it from M(Q, D);
14 for each minimal lineage L ∈ M(Q, D) do
15     compute ||L||_p;
16 pr_D(Q) = min_{L∈M(Q,D)} ||L||_p;
```

why-provenances of this tuple to filter the ones which are superset of another why-provenance. The complexity is $O(nm^2)$. Next, we enumerate all the lineages of these n tuples, by selecting one minimal lineage from each tuple (line 8). The complexity is $O(n^m)$. Third, for each lineage, we need to check whether it is minimal or not, thus we need to traverse all the lineages (from line 9 to line 13). The complexity is $O(n^{2m})$. Last, we compute the prices of all the minimal lineages and choose the smallest one (from line 14 to line 16). The complexity is $O(n^m)$. In total, the complexity of computing the price of this query is $O(n^{2m})$.

However, we can compute the prices of the lineages in step two, without selecting the minimal lineages. Our goal is the price of the cheapest minimal lineage. If a lineage is not minimal, then there is always a minimal lineage whose price is lower than its price. Thus we save the computation of step three and step four. The total complexity of computing the price of this query is $O(n^m)$.

The complexity of our algorithm depends on the computation of enumerating the lineage of $Q(\mathcal{D})$ from the set of minimal lineages of each $t_i \in Q(\mathcal{D})$. Since this enumeration is in non-polynomial time, our algorithm is intractable. In order to get a p-time solution, we have to get the minimum price of the minimal lineages of $Q(\mathcal{D})$ without enumerating every minimal lineage of $Q(\mathcal{D})$. How to achieve this for price computation under our model is still an open problem, and will be our future research direction.

4. PRICING PROBABILISTIC RELATIONAL DATA

Since many modern data are collected or integrated from different sources, uncertainty is unavoidable. Probabilistic database systems have already been developed to manage such uncertain data. In order to support deals involving uncertain data, we further extend our pricing model for relational database to the case of probabilistic relational database.

4.1 Probabilistic Relational Data Model

In this paper, we use the *X-tuple* model (introduced by [1]) for representing probabilistic data.

DEFINITION 4.1. *(X-tuple, probabilistic relation, probabilistic*

database). Each X-tuple *is a set of mutually exclusive tuples. We denote each tuple t_i as (Xid, tid, U). Here Xid is the identifier of the X-Tuple, to which t_i belongs. tid is an unique identifier. For the sake of representation, we use tid to represent the corresponding tuple. U is the set of real-valued attributes.*

A probabilistic relation *in X-tuple model is a relation with a set of X-tuples. We denote the k-th X-tuple as τ_k. We assume that each X-tuple is independent to other X-tuples.*

There exits a probability function *p: $\mathbb{T} \rightarrow [0, 1]$ mapping each tuple to a real number. $p(t_i)$ is the probability that t_i is actual in the real world. The sum of probabilities of tuples in one X-Tuple is less or equal to one, i.e. $\sum_{t_i \in \tau} p(t_i) \leq 1$.*

A probabilistic database *in X-tuple model is a set of probabilistic relations in X-tuple model.*

In general probabilistic databases, there may exist complex correlations between different tuples. However, in probabilistic databases in X-tuple model, there are only two correlations between tuples, which are independent and mutually exclusive. Two tuples with different XID values are independent, and two tuples with the same XID value but different tid values are mutually exclusive.

DEFINITION 4.2. *Given a probabilistic database $\mathcal{X}$, and a query Q,* the query result *$Q(\mathcal{X})$ is a set of tuples with their qualification probability: $\{(t_1, e_1), (t_2, e_2), ..., (t_m, e_m)\}$, where e_i is the probability that t_i satisfies Q.*

EXAMPLE 4.1. *Given a database $\mathcal{X}$ as shown in Table 3 and Table 4. There is a probability function p: $p(t_1) = 0.4, p(t_2) = 0.6, p(t_3) = 0.5, p(t_4) = 0.5, p(t_5) = 0.7, p(t_6) = 0.3, p(t_7) = 0.2, p(t_8) = 0.8$. Table 3 records the weather of a country in some date. For instance, t_1 and t_2 belong to the same X-tuple τ_1, and they present the information that on 2012/5/20, the weather in Singapore is thunder storm with the probability 0.4, and is sunny with the probability 0.6. Table 4 records the choices of sports on different weathers. For instance, t_5 and t_6 belong to the same X-tuple τ_3, and they present the information that if the weather is sunny, he plays football with the probability 0.7, and plays basketball with the probability 0.3.*

If we are interested in the sports he can do, the query is:

$$Q = \{< s > | \exists c, \exists w, \exists d(Weather(c, w, d) \wedge Sports(w, s))\}$$

The query result $Q(\mathcal{X}) = \{$(football, 0.56), (basketball, 0.24)$\}$.

XID	tid	Country	Date	Weather
τ_1	t_1	Singapore	2012/5/20	thunder storm
τ_1	t_2	Singapore	2012/5/20	sunny
τ_2	t_3	Singapore	2012/5/21	sunny
τ_2	t_4	Singapore	2012/5/21	rainy

Table 3: Relation Weather

XID	tid	Weather	Sports
τ_3	t_5	sunny	football
τ_3	t_6	sunny	basketball
τ_4	t_7	cloudy	tennis
τ_4	t_8	cloudy	badminton

Table 4: Relation Sports

4.2 The Pricing Model

DEFINITION 4.3. *Let $\mathcal{X}$ be a probabilistic database in X-tuple model. A* price setting function *is a function $s_{\mathcal{X}} : \mathbb{T} \rightarrow \mathbb{R}^+$, where $\mathbb{T}$ is the tuples in $\mathcal{X}$, and $\mathbb{R}^+$ is the set of non-negative real numbers.*

For a tuple t_i in $\mathcal{X}$, the price $s_{\mathcal{X}}(t_i)$ represents the value of t_i if it is actual.

DEFINITION 4.4. *Let $\mathcal{X}$ be a probabilistic database in X-tuple model. A* pricing function *is a function $pr_{\mathcal{X}} : \mathbb{Q} \rightarrow \mathbb{R}^+$, where $\mathbb{Q}$ is the set of queries, and $\mathbb{R}^+$ is the set of non-negative real numbers.*

4.3 Price of an X-tuple

In this section, we define the price of an X-tuple. Assume an X-tuple τ has n different tuples: $t_1, t_2, ..., t_n$, with corresponding probabilities $p_1, p_2, ..., p_n$ ($\sum_1^n p_i = 1$), and corresponding prices $s_{\mathcal{X}}(t_1), s_{\mathcal{X}}(t_2), ..., s_{\mathcal{X}}(t_n)$. An X-tuple may have different prices for individual tuples. Some tuple may be more outstanding or more surprisal than other tuples, thus it may get a higher price than others.

For an X-tuple τ, if $\sum_1^n p_i < 1$, none of the alternatives of τ is actual, with the probability $1 - \sum_1^n p_i$. To be consistent, we construct a dummy tuple with the probability $1 - \sum_1^n p_i$. A price is set to such a constructed dummy tuple. If $\sum_1^n p_i = 1$ holds, there is no need to construct such a dummy tuple.

Let A be the price of an X-tuple τ. For the X-tuple τ, there is only one tuple can be actual. When t_i is actual, the price of the X-tuple τ is the price of t_i, which is $s_{\mathcal{X}}(t_i)$. Thus the sample space for A is $\{s_{\mathcal{X}}(t_1), s_{\mathcal{X}}(t_2), ..., s_{\mathcal{X}}(t_n)\}$. The probability of the sample point $s_{\mathcal{X}}(t_i)$ is p_i. We define the price of τ to be the expected value of the random variable A, as $E(A)$.

DEFINITION 4.5. *(The price of an X-tuple) In a probabilistic database $\mathcal{X}$, given the probability vector $\mathcal{P}$ and the price vector $\mathcal{R}$, the price of an X-tuple τ is*

$$r_{\mathcal{X}}(\tau) = \sum_{i=1}^{n} (s_{\mathcal{X}}(t_i) \times p_i)$$

For each tuple t_i in the X-tuple τ, if there is no uncertainty in t_i (i.e. $p_i = 1$) then we pay the full price for t_i; if this tuple is completely unrealistic (i.e. $p_i = 0$), we pay nothing for it; if this tuple is actual with probability p_i and $0 < p_i < 1$, we pay a discount price, which is penalized with the coefficient being the probability, for it.

4.4 The Pricing Function

When defining the lineage of a probabilistic database, we not only consider the content of the tuples, but also consider the probabilities of the tuples. When we query on the lineage, we get the exactly the same set of tuples, and the same corresponding probabilities as well.

We consider a lineage consists of X-tuples, since we want to charge the whole X-tuple if any alternative tuple is used.

DEFINITION 4.6. *(A lineage of a set of probabilistic tuples). Let Q be a query. Let $\mathcal{X}$ be a probabilistic database. Let $Q(\mathcal{X}) = \{(t_1, e_1), ..., (t_m, e_m)\}$ be the result of the query Q on the probabilistic database $\mathcal{X}$. The* lineage *of a set of$Q(\mathcal{X})$ is a set of X-tuples L ($L \subseteq \mathcal{X}$) such that:*

$$\forall i, ((t_i, e_i) \in Q(\mathcal{X}))(t_i, e_i) \in Q(L)$$

where e_i is the qualification probability of t_i.

DEFINITION 4.7. *(A minimal lineage of a set of probabilistic tuples). Let Q be a query. Let $\mathcal{X}$ be a probabilistic database. Let $Q(\mathcal{X})$ be the result of the query Q on the database $\mathcal{X}$. The* minimal lineage *of $Q(\mathcal{X})$ is a lineage L of $Q(\mathcal{X})$ such that:*

$$\forall L', L' \subseteq L \Rightarrow L' = L$$

where L' is a lineage L of $Q(\mathcal{X})$.

EXAMPLE 4.2. *Following Example 4.1, we compute the minimal lineages of $Q(\mathcal{X}) = \{$(football, 0.56), (basketball, 0.24)$\}$. If we only consider the content of the tuples, the minimal lineages would be $\{\tau_1, \tau_3\}$ and $\{\tau_2, \tau_3\}$. However, if we process Q on $\{\tau_1, \tau_3\}$, we will get the answer $\{(football, 0.42), (basketball, 0.18)\}$, whose probabilities are different from $Q(\mathcal{X})$, thus $\{\tau_1, \tau_3\}$ is not a lineage of $Q(\mathcal{X})$. If we process Q on $\{\tau_2, \tau_3\}$, we will get the answer $\{(football, 0.35), (basketball, 0.15)\}$, whose probabilities are different from $Q(\mathcal{X})$, thus $\{\tau_2, \tau_3\}$ is not a lineage of $Q(\mathcal{X})$.*

We can get $Q(\mathcal{X})$ only by process Q on $\{\tau_1, \tau_2, \tau_3\}$. Thus $\{\tau_1, \tau_2, \tau_3\}$ is a minimal lineage of $Q(\mathcal{X}) = \{$(football, 0.56), (basketball, 0.24)$\}$.

For a given query Q and a given probabilistic database $\mathcal{X}$, the query result $Q(\mathcal{X})$ may have several minimal lineages. We use $M(Q, \mathcal{X})$ to represent the set of minimal lineages of $Q(\mathcal{X})$.

DEFINITION 4.8. *(Contributing tuples). Let Q be a query. Let $\mathcal{X}$ be a probabilistic database. Let $Q(\mathcal{X})$ be the result of the query Q on the probabilistic database $\mathcal{X}$. The* contributing tuples *of $Q(\mathcal{X})$ is the set of minimal lineages of $Q(\mathcal{X})$, i.e. $M(Q, \mathcal{X})$*

We use p-norm to define the price of a set of X-tuples.

DEFINITION 4.9. *(The price of a set of X-tuples). Let $\mathcal{X}$ be a probabilistic database. Let X be a set of X-tuples, and $X \subseteq \mathcal{X}$. Let $s_{\mathcal{X}}$ be a pricing setting function. The p-norm of the set of X-tuples X is defined by*

$$||X||_p = (\sum_{i=1}^{n} s_{\mathcal{X}}(\tau_i)^p)^{\frac{1}{p}}$$

where $\tau_i \in X$.

DEFINITION 4.10. *(Price of a query). The price of a query Q in a probabilistic database $\mathcal{X}$ is defined as the price of the cheapest minimal lineage of $Q(\mathcal{X})$:*

$$pr_{\mathcal{X}}(Q) = min_{L \in M(Q,\mathcal{X})} ||L||_p$$

4.5 Computing price

In this section, we present a naive algorithm to compute the price of a query using our proposed pricing function in probabilistic databases in X-tuple model. However, the same as the algorithm of computing price in relational databases, this proposed algorithm is also intractable.

We compute the minimal lineage for each individual tuple (t_i, e_i) in the query result $Q(\mathcal{X})$ first. For each individual tuple (t_i, e_i), there may exist several different ways to produce t_i. Recall the pricing model in relational databases, if there exist several ways to produce an output tuple, each possible way is one minimal why-provenance of the output tuple. However, in probabilistic databases, if $e_i \neq 1$, every possible way to produce t_i contributes to the final probability e_i, thus all of them need to be included in the minimal lineage of (t_i, e_i), otherwise we cannot get the correct probability. If $e_i = 1$, not all the possible way to produce t_i contribute to the final probability e_i, thus we do not need to include all of them as the minimal why-provenance (see the example below in Case 2). When $e_i = 1$, t_i has no uncertainty. In this case, we need at least one way with probability 1 to produce t_i (the reason is that if the probabilities of all the independent ways are not equal to 1, the final probability e_i cannot be 1). Each minimal why-provenance is necessary to include a set of X-tuples being able to produce t_i with probability 1, regardless of other possible ways to produce t_i with probability not being 1.

We discuss these two cases separately as below.

Case 1: $e_i \neq 1$**.** In this case, we need to include all the X-tuples contributing to (t_i, e_i) in the why-provenance. In Example 4.2, to produce the output tuple $(football, 0.56)$, we need τ_1, τ_2, τ_3 to get "football" and also the correct probability 0.56. If we only have τ_1, τ_3 (or τ_2, τ_3), we can produce "football" but with an incorrect probability. Thus we have to adapt the why-provenance defined in [7], instead of the one defined in [4].

Algorithm 2: minimal lineage computation for (t_i, e_i) when $e_i \neq 1$

Data: a query Q, a probabilistic database $\mathcal{X}$, a tuple $(t_i, e_i) \in Q(\mathcal{X})$
Result: minimal lineage l_i for (t_i, e_i)

1 get the why-provenance l_i defined in [7];
2 **for** *each* $t_k \in l_i$ **do**
3 | replace t_k by the X-tuple to which t_k belongs;

Algorithm 2 is the algorithm computing the minimal lineage for (t_i, e_i) when $e_i \neq 1$. We adapt the why-provenance defined in [7], by replacing the tuples by the X-tuples to which they belong. For instance, in Example 4.2, the why-provenance for "football" is $\{t_2, t_3, t_5\}$. Thus replacing the tuples by the X-tuples to which the tuples belong, we have the minimal lineage of $(football, 0.56)$ is $\{\tau_1, \tau_2, \tau_3\}$.

Case 2: $e_i = 1$**.** In this case, we do not need to include all the X-tuples contributing to the probability since $e_i = 1$. For instance, there is a probabilistic database as shown in Table 5. $p(t_1) = 0.8, p(t_2) = 0.2, p(t_3) = 0.7, p(t_4) = 0.3, p(t_5) = 0.5, p(t_6) = 0.5$. We have query $Q = \{< a > | \exists b(R(a, b))\}$. The query result $Q(\mathcal{X}) = \{(a, 1), (b, 0.5)\}$. If we use Algorithm 2 to compute the minimal lineage of $(a, 1)$, then the minimal lineage would be $\{\tau_1, \tau_2, \tau_3\}$. Actually, only τ_1 or only τ_2 can produce the result $(a, 1)$. τ_1, τ_2, τ_3 all contribute to the probability 1, however, not all of them are necessary. However, in Case 1, all the X-tuples computed by Algorithm 2 are necessary. This is the *key difference* between Case 1 and Case 2. In our example, there are two minimal lineages of $(a, 1)$: $\{\tau_1\}$ and $\{\tau_2\}$.

Algorithm 3 is the algorithm computing the minimal lineage for (t_i, e_i) when $e_i = 1$. We first compute the why-provenance (from line 1 to line 3), as the same as Algorithm 2. We have one more step in Algorithm 3, to separate the minimal lineage into several sets, due to the reason we state in the previous example.

Algorithm 4 is the algorithm computing the price of a query Q given a database $\mathcal{X}$. First we compute the minimal lineage or the set of minimal lineages for each tuple in the query result (from line

XID	tid	A	B
τ_1	t_1	a	c
τ_1	t_2	a	d
τ_2	t_3	a	e
τ_2	t_4	a	f
τ_3	t_5	a	g
τ_3	t_6	b	h

Table 5: Relation R

Algorithm 3: minimal lineage computation for (t_i, e_i) when $e_i = 1$

Data: a query Q, a probabilistic database $\mathcal{X}$, a tuple $(t_i, e_i) \in Q(\mathcal{X})$
Result: a set of minimal lineages c_i for (t_i, e_i)

1 get the why-provenance l_i defined in [7];
2 **for** *each* $t_k \in l_i$ **do**
3 | replace t_k by the X-tuple to which t_k belongs;
4 separate l_i into a set of minimal lineages c_i, such that for each minimal lineage $m \in c_i$, m is minimal to produce $(t_i, 1)$;

Algorithm 4: Price computation algorithm

Data: a query Q, a probabilistic database $\mathcal{X}$
Result: price $pr_{\mathcal{X}}(Q)$

1 **for** *each tuple* (t_i, e_i) *in the query result* $Q(\mathcal{X})$ **do**
2 | **if** $e_i \neq 1$ **then**
3 | | compute the minimal lineage of t_i using Algorithm 2;
4 | **else**
5 | | compute the set of minimal lineages of t_i using Algorithm 3
6 Construct the set of all the candidates of minimal lineages of $Q(\mathcal{X})$, namely $M(Q, \mathcal{X})$, by selecting a minimal lineage from each (t_i, e_i);
7 **for** *each minimal lineage* $L \in M(Q, \mathcal{X})$ **do**
8 | compute $||L||_p$;
9 $pr_{\mathcal{X}}(Q) = min_{L \in M(Q,\mathcal{X})} ||L||_p$;

1 to line 5). Then we enumerate all the lineages of the query result (line 6). However, we do not check the minimal since we can have the optimization mentioned in Section 3.5. Last, we compute the prices of all the minimal lineages and choose the smallest one (from line 7 to line 9).

Assume we have n tuples in the query result and assume we have the why-provenance defined in [7] beforehand. First, we analyze the best case: all the tuples in the query result are in Case 1. Thus there is a unique minimal lineage for each resulting tuple, and there is only one minimal lineage for the query result. The complexity is $O(1)$. Next, we analyze the worst case: all the tuples in the query result are in Case 2. Assume there are m minimal lineages for each tuple. The complexity is $O(n^m)$ (the same as the one in Section 3.5).

5. CONCLUSION AND FUTURE WORK

This work sets a framework for future studies on data pricing based on provenance. In this paper we propose a generic data pricing model that is based on minimal lineages, i.e. sets of tuples contributing to the result of a query. We show that the proposed model fulfils desirable properties such as contribution monotonicity, bounded price and arbitrage-freedom. We propose a naive algorithm to computet the price of a query using the proposed pricing function. We extend the model to the case of probabilistic databases in the X-tuple model and also propose a naive algorithm to compute the price of a query in probabilistic databases.

We are now looking for heuristics to approximate the price of data at lower complexity. We are also looking for special favorable classes of queries of which the prices can be computed efficiently.

6. REFERENCES

[1] P. Agrawal, O. Benjelloun, A. D. Sarma, C. Hayworth, S. Nabar, T. Sugihara, and J. Widom. Trio: A system for data, uncertainty, and lineage. In *VLDB*, 2006.

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

[3] E. Brynjolfsson, L. M. Hitt, and H. Kim. Strength in numbers: How does data-driven decision-making affect firm performance? In *ICIS*, 2011.

[4] P. Buneman, S. Khanna, and W.-C. Tan. Why and Where: A Characterization of Data Provenance. In *ICDT*, pages 316–330, 2001.

[5] J. Cheney, L. Chiticariu, and W. C. Tan. Provenance in Databases: Why, How, and Where. *Foundations and Trends in Databases*, 1(4):379–474, 2009.

[6] L. Chiticariu and W. C. Tan. Debugging schema mappings with routes. In *VLDB*, pages 79–90, 2006.

[7] Y. Cui and J. Widom. Practical Lineage Tracing in Data Warehouses. In *ICDE*, pages 367–378, 2000.

[8] T. J. Green, G. Karvounarakis, Z. G. Ives, and V. Tannen. Update exchange with mappings and provenance. In *VLDB*, Vienna, Austria, September 2007.

[9] T. Hey, S. Tansley, and K. M. Tolle, editors. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, 2009.

[10] T. J.Green, G. Karvounarakis, and V. Tannen. Provenance Semirings. In *PODS*, 2007.

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

[12] A. LLC. AggData. `http://www.aggdata.com/`.

[13] Microsoft. Windows Azure Marketplace. `https://datamarket.azure.com/`.

[14] E. Prugovečki. *Quantum Mechanics in Hilbert Space: Second Edition*. Dover Books on Physics Series. Dover Publications, 2006.

APPENDIX

A. ADDITIONAL PROOFS

A.1 Proof of Theorem 3.1

PROOF. Assume Q_1, Q_2 are two equivalent queries: $Q_1 \equiv Q_2$. For any database $\mathcal{D}$, $Q_1(\mathcal{D}) = Q_2(\mathcal{D})$.

Assume $\exists \mathcal{D}'$ such that $M(Q_1, \mathcal{D}') \neq M(Q_2, \mathcal{D}')$. Then at least one of the two sets contains some element that does not exist in the other set. Assume

$$\exists L^* \in M(Q_1, \mathcal{D}') \wedge (\forall L_2 \in M(Q_2, \mathcal{D}') \Rightarrow L^* \neq L_2)$$

We have the following cases:

1. $\exists L' \in M(Q_2, \mathcal{D}')$ such that $L^* \subset L'$. Construct a database $D^- = L^*$. Since L^* is a minimal lineage of $Q_1(\mathcal{D}')$,

$$Q_1(D^-) = Q_1(L^*) = Q_1(\mathcal{D}')$$

On the other hand, L^* cannot be a lineage of $Q_2(\mathcal{D}')$. Otherwise by definition of minimal lineage, L' is not a minimal lineage.

$$Q_2(D^-) = Q_2(L^*) \not\supseteq Q_2(\mathcal{D}')$$

Thus

$$Q_2(D^-) \neq Q_2(\mathcal{D}')$$

Since $Q_1 \equiv Q_2$, $Q_1(\mathcal{D}') = Q_2(\mathcal{D}')$. Then $Q_1(D^-) \neq Q_2(D^-)$. It violates $Q_1 \equiv Q_2$.

2. $\exists L' \in M(Q_2, \mathcal{D}')$ such that $L' \subset L^*$. Construct a database $D^- = L'$. Since L' is a minimal lineage of $Q_2(\mathcal{D}')$,

$$Q_2(D^-) = Q_2(L') = Q_2(\mathcal{D}')$$

On the other hand, L' cannot be a lineage of $Q_1(\mathcal{D}')$. Otherwise by definition of minimal lineage, L^* is not a minimal lineage.

$$Q_1(D^-) = Q_1(L') \not\supseteq Q_1(\mathcal{D}')$$

Thus

$$Q_1(D^-) \neq Q_1(\mathcal{D}')$$

Since $Q_1 \equiv Q_2$, $Q_1(\mathcal{D}') = Q_2(\mathcal{D}')$. Then $Q_1(D^-) \neq Q_2(D^-)$. It violates $Q_1 \equiv Q_2$.

3. $\nexists L' \in M(Q_2, \mathcal{D}')$ such that $L' \subset L^*$ or $L^* \subset L'$. Construct a database $D^- = L^*$. Since L^* is a minimal lineage of $Q_1(\mathcal{D}')$,

$$Q_1(D^-) = Q_1(L^*) = Q_1(\mathcal{D}')$$

On the other hand, L^* cannot be a minimal lineage of $Q_2(\mathcal{D}')$ since $\forall L' \in M(Q_2, \mathcal{D}') \Rightarrow L^* \neq L'$. Also, L^* cannot be a lineage of $Q_2(\mathcal{D}')$ since $\nexists L' \in M(Q_2, \mathcal{D}')$ such that $L' \subset L^*$.

$$Q_2(D^-) = Q_2(L^*) \not\supseteq Q_2(\mathcal{D}')$$

Thus

$$Q_2(D^-) \neq Q_2(\mathcal{D}')$$

Since $Q_1 \equiv Q_2$, $Q_1(\mathcal{D}') = Q_2(\mathcal{D}')$. Then $Q_1(D^-) \neq Q_2(D^-)$. It violates $Q_1 \equiv Q_2$.

We can conclude that $\forall \mathcal{D}', M(Q_1, \mathcal{D}') = M(Q_2, \mathcal{D}')$. □

A.2 Proof of Proposition 3.1

PROOF. Let $f(p) = (\sum_{i=1}^{n} |x_i|^p)^{\frac{1}{p}}$.

$f(p) = e^{\frac{1}{p} ln(x_1^p + ... x_n^p)}$

$f'(p) = e^{\frac{1}{p} ln(x_1^p + ... x_n^p)} \times (\frac{1}{p} ln(x_1^p + ... x_n^p))'$

$= e^{\frac{1}{p} ln(x_1^p + ... x_n^p)} \times ((\frac{1}{p})' \times ln(x_1^p + ... x_n^p) + \frac{1}{p} \times (ln(x_1^p + ... x_n^p))')$

$= e^{\frac{1}{p} ln(x_1^p + ... x_n^p)} \times (-\frac{1}{p^2} \times ln(x_1^p + ... x_n^p) + \frac{1}{p} \times \frac{1}{x_1^p + ... x_n^p} \times$
$(x_1^p ln x_1 + ... + x_n^p ln x_n))$

$= e^{\frac{1}{p} ln(x_1^p + ... x_n^p)} \times \frac{1}{p} \times (\frac{1}{x_1^p + ... x_n^p} \times (x_1^p ln x_1 + ... + x_n^p ln x_n) -$
$\frac{1}{p} \times ln(x_1^p + ... x_n^p))$

$= e^{\frac{1}{p} ln(x_1^p + ... x_n^p)} \times \frac{1}{p} \times$
$(\frac{p \times (x_1^p ln x_1 + ... + x_n^p ln x_n) - (x_1^p + ... x_n^p) \times ln(x_1^p + ... x_n^p)}{p \times (x_1^p + ... x_n^p)})$

$= e^{\frac{1}{p} ln(x_1^p + ... x_n^p)} \times \frac{1}{p} \times$
$(\frac{(x_1^p ln x_1^p + ... + x_n^p ln x_n^p) - (x_1^p + ... x_n^p) \times ln(x_1^p + ... x_n^p)}{p \times (x_1^p + ... x_n^p)})$

$= e^{\frac{1}{p} ln(x_1^p + ... x_n^p)} \times \frac{1}{p} \times \frac{1}{p \times (x_1^p + ... x_n^p)} \times$
$(x_1^p ln \frac{x_1^p}{x_1^p + ... x_n^p} + ... + x_n^p ln \frac{x_n^p}{x_1^p + ... x_n^p})$

$since\ e^{\frac{1}{p} ln(x_1^p + ... x_n^p)} \times \frac{1}{p} \times \frac{1}{p \times (x_1^p + ... x_n^p)} > 0$

$since\ x_1^p ln \frac{x_1^p}{x_1^p + ... x_n^p} + ... + x_n^p ln \frac{x_n^p}{x_1^p + ... x_n^p} < 0$

$so\ f'(p) < 0$

$so\ ||X||_{p+a} \leq ||X||_p (a \geq 0)$

□

A.3 Proof of Proposition 3.2

PROOF. Assume $X_1 = \{x_1, ..., x_m\}, X_2 = \{x_1, ..., x_m, ..., x_n\}$ $(n \geq m)$.

By definition, $||X_2||_p = (|x_1|^p + ... + |x_m|^p + ... + |x_n|^p)^{\frac{1}{p}}$.
$\because |x_i| \geq 0 \therefore |x_i|^p \geq 0$.
Thus $|x_1|^p + ... + |x_m|^p + ... + |x_n|^p \geq |x_1|^p + ... + |x_m|^p$.
$f(z) = z^{\frac{1}{k}}$ is increasing as z increases ($z \geq 0, k \geq 1$), the reason is as following:
$f'(z) = \frac{1}{k} \times z^{\frac{1}{k} - 1} \geq 0$ when $z \geq 0, k \geq 1$.
We can have $(|x_1|^p + ... + |x_m|^p + ... + |x_n|^p)^{\frac{1}{p}} \geq (|x_1|^p + ... + |x_m|^p)^{\frac{1}{p}}$, which means $||X_2||_p \geq ||X_1||_p$.

□

A.4 Proof of Theorem 3.4

PROOF. First, we prove that $U'(Q, \mathcal{D})$ includes all the minimal lineages. Prove by contradiction.

Assume there exist a minimal lineage of $Q(\mathcal{D})$, namely L^*, which is not included in $U'(Q, \mathcal{D})$. It means that

$$\nexists u \in U'(Q, \mathcal{D})\ such\ that\ u = L^*$$

We have the following cases:

1. $\exists u \in U'(Q, \mathcal{D})$ such that $u \subset L^*$. In this case, L^* is not a minimal lineage of $Q(\mathcal{D})$. The reason is that its proper subset u is a lineage of $Q(\mathcal{D})$. There is a contradiction.
2. $\exists u \in U'(Q, \mathcal{D})$ such that $L^* \subset u$. Without loss of generality, assume that $L^* = u - \{b\}$. Let us assume that $b \in u$ due to the fact that b belongs to a minimal why-provenance l of a tuple $t \in Q(\mathcal{D})$. If L^* is a minimal lineage, it means that b is unnecessary to produce $Q(\mathcal{D})$. Moreover, b is unnecessary to produce t, which means l is not a minimal why-provenance of t. There is a contradiction.
3. $\forall u \in U'(Q, \mathcal{D})$, u and L^* are not comparable. Without loss of generality, assume that $L^* = u - \{b\} + \{c\}$. Let us assume that $b \in u$ due to the fact that b belongs to a minimal why-provenance l of a tuple $t \in Q(\mathcal{D})$. If L^* is a minimal lineage, it means that $l - \{b\} + \{c\}$ is a minimal why-provenance. Since $L^* \notin U'(Q, \mathcal{D})$, $l - \{b\} + \{c\}$ cannot be a minimal why-provenance, otherwise L^* can be computed by Algorithm 1. There is a contradiction.

Then, we prove that every element in $U'(Q, \mathcal{D})$ is a minimal lineage. It is easy to see that every element in $U'(Q, \mathcal{D})$ is a lineage of $Q(\mathcal{D})$. After filtering the non-minimal lineages, we have only minimal lineages in $U'(Q, \mathcal{D})$. □