Queries and Materialized Views on Probabilistic Databases

Nilesh Dalvi[a], Christopher Re[b], Dan Suciu[c*]

[a]Yahoo Reserach

[b]University of Washington

[c]University of Washington

We review in this paper some recent yet fundamental results on evaluating queries over probabilistic databases. While one can see this problem as a special instance of general purpose probabilistic inference, we describe in this paper two key database specific techniques that significantly reduce the complexity of query evaluation on probabilistic databases. The first is the separation of the query and the data: we show here that by doing so, one can identify queries whose data complexity is #P-hard, and queries whose data complexity is in PTIME. The second is the aggressive use of previously computed query results (materialized views): in particular, by rewriting a query in terms of views, one can reduce its complexity from #P-complete to PTIME. We describe a notion of a partial representation for views, and show that, once computed and stored, this partial representation can be used to answer subsequent queries on the probabilistic databases. evaluation.

1. Introduction

Probabilistic databases are databases where the presence of a tuple, or the value of an attribute is a probabilistic event. The major difficulty in probabilistic database is query evaluation: the result of a SQL query over a probabilistic database is a set of tuples together with the probability that those tuples belong to the output, and those probabilities turn out to be hard to compute. In fact, computing those output probabilities is a special instance of probabilistic inference, which is a problem that has been studied extensively by the Knowledge Representation community. Unlike general purpose probabilistic inference, in query evaluation we have a few specific techniques that we can deploy to speed up the evaluation considerably.

The first is the separation between the query and the data: the query is small, the data is large. Following Vardi [31], define the *data complexity* to be the complexity of query evaluation where the query is fixed, and the complexity is measured only in the size of the database. A number of results in query processing over probabilistic databases have shown that for some queries the data complexity is PTIME, while for others it is #P-hard: we review those results in Sec. 3, following mostly [11].

*This work was partially supported by NSF Grants IIS-0454425, IIS-0513877, IIS-0713576, and a Gift from Microsoft.

The second technique is the use of materialized views. A materialized view is simply the result of a query that has been precomputed and stored. In traditional databases, materialized views are used widely to speed up query processing [28,3,18]. Examples of traditional materialized views include standard indexes, join-indexes, and aggregates in data cubes. We describe in this paper an approach by which materialized views can be used to speedup query processing in probabilistic databases. As in traditional databases, we compute and store the answer to a view over a probabilistic database, then rewrite a query to use the view(s) whenever possible. In our approach, the view looks like any other probabilistic relation: it consists of a set of tuples, and their marginal probabilities. The main difficulty is how to represent the correlations between these tuples. One possibility would be to store their lineage, as done by the Trio system [7]: the lineage represents how a tuple was derived, and thus can be used during query processing to account for correlations. There are two costs in evaluating a query over a probabilistic database: the cost of relational processing (selections, joins, etc), and the cost of the probabilistic inference. Views with explicit lineage expressions can reduce the cost of relational processing, but not that of the probabilistic inference, and for that reason we favor a different approach to represent correlations in the view. We compute a *partial representation* of the probabilistic view, which allows us to determine for every two tuples in the view whether they are independent, disjoint, or have a more complex correlation that is not captured in the partial representation. We say that a query is *well defined* on the view if its output probability depends only on the marginal tuple probabilities in the view, assuming that the independence and disjointness relationships specified in the partial representation hold; in other words, the query is well defined if it does not depend on the unknown correlations between tuples in the view. We prove several results about partial representations and well-definedness. First, we show that every probabilistic database represented by a c-table admits a "best" partial representation that captures all independence relationships; this representation is instance specific, i.e. best for the given instance. Next, we show that every probabilistic view defined by a conjunctive query admits a "best" partial representation; this representation is instance-independent, i.e. it depends on the expression defining the view, but is independent on the input probabilistic database instance. Finally, we give a necessary and sufficient condition for a query to be well-defined given a view. The results on materialized views and well-definedness are described in Section 4. Some of the results in this section have been announced previously in [26].

2. Definition: the Possible Worlds Data Model

We review here the definition of a probabilistic database based on possible worlds, and of a disjoint-independent database. We restrict our discussion to relational data over an infinite discrete domain: extensions to continuous domains [14] and to XML [19,1,30] have also been considered.

We fix a relational schema $\mathcal{R} = (R_1, \ldots, R_k)$, where R_i is a relation name, has a set of attributes $Attr(R_i)$, and a key $Key(R_i) \subseteq Attr(R_i)$. Denote D a finite domain of atomic values: D is the active domain or a superset thereof, and is considered part of the input. Let Tup be the set of all typed tuples of the form $t = R_i(a_1, \ldots, a_m)$, for some $i = 1, k$ and $a_1, \ldots, a_m \in D$. We denote $Key(t)$ the tuple consisting of the key attributes in t

(hence its arity is $|Key(R_i)|$). A database instance is any subset $I \subseteq Tup$ that satisfies all key constraints.

In a probabilistic database the state of the database, *i.e.* the instance I is not known. Instead the database can be in any one of a finite number of possible states $I_1, I_2, \ldots$, called *possible worlds*, each with some probability.

Definition 2.1 *A* probabilistic database *is a probability space* $PDB = (W, \mathbf{P})$ *where the set of outcomes is a set of* possible worlds $W = \{I_1, \ldots, I_n\}$*, and* $\mathbf{P}$ *is a function* $\mathbf{P} : W \to (0, 1]$ *s.t.* $\sum_{I \in W} \mathbf{P}(I) = 1$.

Fig. 1 illustrates three possible worlds of a probabilistic database. The probabilistic database has more worlds, and the probabilities of all worlds must sum up to 1; the figure illustrates only three worlds. The intuition is that we have a database with schema $R(\underline{A, B}, C, D)$, but we are not sure about the content of the database: there are several possible contents, each with a probability.

A *possible tuple* for a probabilistic database PDB is a tuple that occurs in at least one possible world; we typically denote T the set of possible tuples.

2.1. Query Semantics

Consider a query q of output arity k, expressed over the relational schema $\mathcal{R}$. Recall that, when evaluated over a standard database instance I, a query returns a relation of arity k, $q(I) \subseteq D^k$. If $k = 0$, then we call the query a *Boolean* query.

Definition 2.2 *Let q be a query of arity k and $PDB = (W, \mathbf{P})$ a probabilistic database. Then $q(PDB)$ is the following probability distribution on the query's outputs: $q(PDB) = (W', \mathbf{P}')$ where:*

$$\begin{aligned} W' &= \{q(I) \mid I \in W\} \\ \mathbf{P}'(J) &= \sum_{I \in W : q(I) = J} \mathbf{P}(I) \end{aligned}$$

That is, when applied to a probabilistic database PDB the query returns another probabilistic database obtained by applying the query separately on each world. The probability space $q(PDB)$ is called an *image probability space* in [16].

A particular case of great importance to us is when q is a Boolean query. Then q defines the event $\{I \mid I \models q\}$ over a probabilistic database, and its marginal probability is $\mathbf{P}(q) = \sum_{I \in W | I \models q} \mathbf{P}(I)$: note that the image probability space in this case has only two possible worlds: $q(PDB) = (\{I_0, I_1\}, \mathbf{P}')$, where $I_0 = \emptyset$, $I_1 = \{()\}$, and $\mathbf{P}'(I_0) = 1 - \mathbf{P}(q)$, $\mathbf{P}'(I_1) = \mathbf{P}(q)$. Thus, for all practical purposes $q(PDB)$ and $\mathbf{P}(q)$ are the same, and we will refer only to $\mathbf{P}(q)$ when the query q is Boolean. A special case of a Boolean query is a single tuple t, and its marginal probability is $\mathbf{P}(t) = \sum_{I \in W | t \in I} \mathbf{P}(I)$. Note that $t \neq t'$ and $Key(t) = Key(t')$ implies $\mathbf{P}(t, t') = 0$, *i.e.* t, t' are disjoint events.

2.2. Block-Independent-Disjoint Databases

In order to study query complexity on probabilistic databases we need to choose a way to represent the input database. We could enumerate all possible worlds I_1, I_2, $\ldots$, together with their probabilities p_1, p_2, $\ldots$, assumed to be rational numbers. But such an enumeration is clearly infeasible in practice because it is too verbose. A number of researchers have searched for compact representations of probabilistic databases [12,7,16, 6,5], and Green and Tannen [16] observed a strong connection between representation systems for probabilistic databases and for incomplete databases.

In our study we choose a representation of probabilistic databases where tuples are either disjoint probabilistic events, or independent probabilistic events.

Definition 2.3 *A probabilistic database PDB is* block-independent-disjoint*, or BID, if* $\forall t_1, \ldots, t_n \in T$, $Key(t_i) \neq Key(t_j)$ *for* $i \neq j$ *implies* $\mathbf{P}(t_1, \ldots, t_n) = \mathbf{P}(t_1) \cdots \mathbf{P}(t_n)$.

This justifies the I in BID: the D is justified by the fact that tuples with the same values of the key attributes are disjoint (this holds in any probabilistic database, not only in BIDs).

A *BID specification* is $(T, \mathbf{P})$, where $T \subseteq Tup$ is a set of tuples, called *possible tuples*, and $\mathbf{P} : T \to [0,1]$ is such that, denoting $K = \{Key(t) \mid t \in T\}$ the set of key values, $\forall k \in K$, $\sum_{t \in T : Key(t) = k} \mathbf{P}(t) \leq 1$.

Theorem 2.4 *Let $(T, \mathbf{P}_0)$ be a BID specification. Then there exists a unique BID probabilistic database $PDB = (W, \mathbf{P})$ s.t. its set of possible tuples is T and for all $t \in T$ its marginal probability $\mathbf{P}(t)$ is equal to $\mathbf{P}_0(t)$.*

Proof: (Sketch) Let $PDB = (W, \mathbf{P})$ be a BID probabilistic database whose marginal tuple probabilities are $\mathbf{P}_0$, and let $I \in W$. Obviously $I \subseteq T$, and we will show that $\mathbf{P}(I)$ is uniquely defined by $(T, \mathbf{P}_0)$ and the independence assumption. For any key $k \in K$, let $p_k = \mathbf{P}_0(t)$, if there exists $t \in I$ s.t. $Key(t) = k$, and $p_k = 1 - \sum_{t \in T : Key(t) = k} \mathbf{P}_0(t)$ otherwise. Then $\mathbf{P}(I) = \prod_{k \in K} p_k$. Conversely, define the $PDB = (Inst(T), \mathbf{P})$, where $Inst(T)$ denotes the set of instances over the tuples T, and $\mathbf{P}$ is defined as above: it is easy to check that this is a probability space ($\sum_I \mathbf{P}(I) = 1$), that it is BID, and that its marginal tuple probabilities are given by $\mathbf{P}_0$. $\square$

The *size* of a BID specification $(T, \mathbf{P})$ is $|T|$; we always assume the probabilities to be rational numbers. Fig. 2 illustrates a BID, which has 16 possible worlds, three of which are shown in Fig. 1. There are seven possible tuples, each with some probability and it is convenient to group the possible tuples by their keys, A, B, to emphasize that at most one can be chosen in each group.

We call the database *independent* if $Key(R_i) = Attr(R_i)$ for all relation symbols R_i, *i.e.* there are no disjoint tuples.

2.3. pc-Tables and Lineage

BID's are known to be an incomplete representation system[2]. Several, essentially equivalent, complete representation systems have been discussed in the literature [12,16,7].

[2]For example, consider three possible tuples, $T = \{t_1, t_2, t_3\}$, and a probabilistic database with three possible worlds, $\{t_1, t_2\}$, $\{t_1, t_2\}$, $\{t_2, t_3\}$, each with probability 1/3. Then the tuples t_1, t_2 are neither disjoint, nor independent.

Here we follow the representation system described by Green and Tannen [16] and called pc-tables, which extends the c-tables of [20].

Fix a set of variables $\bar{X} = \{X_1, \ldots, X_m\}$, and for each variable X_i fix a finite domain $Dom(X_j) = \{0, 1, \ldots, d_j\}$. We consider Boolean formulas φ consisting of Boolean combinations of atomic predicates of the form $X_j = v$, where $v \in Dom(X_j)$. Define a *constant c-table* to be conventional relation R, where each tuple t_i is annotated with a Boolean formula φ_i, called the *lineage* of t. Note that our definition is a restriction of the standard definition of c-tables [20] in that no variables are allowed in the tuples, hence the term "constant": we will drop this term and refer to a constant c-table simply as a c-table in the rest of this paper.

A valuation θ assigns each variable X_j to a value $\theta(X_j) \in Dom(X_j)$, and we write $\theta(R) = \{t_i \mid \theta(\varphi_i) = \texttt{true}\}$.

A pc-table [16] PR is a pair $(R, \mathbf{P})$, consisting of a c-table R and a set of probability spaces $(Dom(X_j), \mathbf{P}_j)$, one for each $j = 1, \ldots, m$. We denote $\mathbf{P}$ the product space, i.e. where the variables X_j are independent: $\mathbf{P}(\theta) = \prod_j \mathbf{P}_j(\theta(X_j))$, for every valuation θ.

A BID database $PDB = (T, \mathbf{P})$ can be expressed as pc-tables as follows. Denote $K = \{k_1, k_2, \ldots, k_m\}$ the set of all key values in T, and define a set of variables $\mathcal{X} = \{X_1, \ldots, X_m\}$ (one variable for each key value). Suppose that the key value k_j occurs in d_j distinct tuples in T, call them $t_{j1}, t_{j2}, \ldots, t_{jd_j}$: then define $Dom(X_j) = \{0, 1, \ldots, d_j\}$ and annotate the tuple t_{ji} with the Boolean expression $X_j = i$. Finally, for each j define the probability space $(Dom(X_j), \mathbf{P}_j)$ by setting $\mathbf{P}_j(i) = \mathbf{P}(t_{ji})$ for $i > 0$ and $\mathbf{P}_j(0) = 1 - \sum_i \mathbf{P}(t_{ji})$.

A fundamental result of c-tables [20] is that they are closed under relational queries. Given a database consisting of a set of c-tables over variables $\bar{X}$ and a relational query q of output arity k, the query's output can also be represented as a c-table, of arity k, where the lineage of each output tuple $t \in D^k$, is some Boolean expressions $\varphi_q(t)$ using the same variables $\bar{X}$. We illustrate this query lineage with a simple example.

Example 2.5 Consider the schema $R(\underline{A}, B)$, $S(\underline{B})$, and consider seven possible tuples, four in R and three in S:

R:

$\underline{A}$	B	
a_1	b_1	$X_1 = 1$
a_1	b_2	$X_1 = 2$
a_2	b_1	$X_2 = 1$
a_2	b_3	$X_2 = 2$

S:

$\underline{B}$	
b_1	$Y_1 = 1$
b_2	$Y_2 = 1$
b_3	$Y_3 = 1$

Assume $Dom(X_1) = Dom(X_2) = \{0, 1, 2\}$ and $Dom(Y_1) = Dom(Y_2) = Dom(Y_3) = \{0, 1\}$. There are $3 \cdot 3 \cdot 8 = 72$ possible worlds. Another way to look at this is that the 72 possible worlds are precisely those subsets of the seven possible tuples where A is a key in R. Consider the Boolean query $h_2 \equiv R(\underline{x}, y), S(\underline{y})$. Then φ_{h_2} is:

$$\begin{aligned} \varphi_{h_2} \quad = \quad & (X_1 = 1) \wedge (Y_1 = 1) \vee \\ & (X_1 = 2) \wedge (Y_2 = 1) \vee \\ & (X_2 = 1) \wedge (Y_1 = 1) \vee \\ & (X_2 = 2) \wedge (Y_3 = 1) \end{aligned}$$

I_1

A	B	C	D
a_1	b_1	c_1	d_1
a_2	b_1	c_3	d_1
a_2	b_2	c_4	d_2

$\mathbf{P}(I_1) = 0.06$
$(= p_1 p_3 p_6)$

I_2

A	B	C	D
a_1	b_1	c_2	d_2
a_2	b_1	c_2	d_1
a_2	b_2	c_4	d_2

$\mathbf{P}(I_2) = 0.12$
$(= p_2 p_5 p_6)$

I_3

A	B	C	D
a_1	b_1	c_1	d_1
a_2	b_2	c_4	d_2

$\mathbf{P}(I_3) = 0.04$
$(= p_1(1\text{-}p_3\text{-}p_4\text{-}p_5)p_6)$

Figure 1. A probabilistic database $PDB = (\{I_1, I_2, I_3, \ldots\}, \mathbf{P})$ with schema $R(\underline{A, B}, C, D)$; we show only three possible worlds.

A	B	C	D	**P**
a_1	b_1	c_1	d_1	$p_1 = 0.25$
		c_2	d_2	$p_2 = 0.75$
a_2	b_1	c_3	d_1	$p_3 = 0.3$
		c_1	d_3	$p_4 = 0.3$
		c_2	d_1	$p_5 = 0.2$
a_2	b_2	c_4	d_2	$p_6 = 0.8$
		c_5	d_2	$p_7 = 0.2$

Figure 2. Representation of a BID. The seven possible tuples are grouped by their keys, for readability. There are 16 possible worlds; three are shown in Fig. 1.

3. Query Evaluation on Probabilistic Databases

In this section we summarize the results on query evaluation from [11].

We study here the following problem. Given a Boolean query q and a BID $PDB = (T, \mathbf{P})$, compute $q(PDB)$. In general the query is expressed in FO, and we will study extensively the case when q is a conjunctive query. We are interested in the data complexity [31]: fix q, and study the complexity of $\mathbf{P}(q)$ as a function of the input PDB. The probabilities are assumed to be rational numbers.

Since c-tables are closed under relational queries, given a set of possible tuples T, any query q expressed in FO has a well defined lineage φ_q, which is a Boolean formula whose size is polynomial in the size of T. Moreover, if q is a conjunctive query, then φ_q is a DNF formula of polynomial size in T, therefore, any upper bounds for computing the probability of a Boolean formula $\mathbf{P}(\varphi_q)$ become upper bounds for computing a query probability $\mathbf{P}(q)$. Thus:

Theorem 3.1 *(1) Computing $\mathbf{P}(\varphi)$ for a Boolean expression φ is in #P [29]. It follows* *that for any query q in FO, the problem "given a BID, compute $\mathbf{P}(q)$" is in #P. (2) Computing $\mathbf{P}(\varphi)$ for a DNF formula φ has a FPTRAS* [3] *[21]. It follows that for any conjunctive query q the problem "given a BID, compute $\mathbf{P}(q)$" has an FPTRAS.*

The complexity class #P consists of problems of the following form: given an NP machine, compute the number of accepting computations [23]. For a Boolean expression φ, let $\#\varphi$ denote the number of satisfying assignments for φ. Valiant [29] has shown that the problem: given φ, compute $\#\varphi$, is #P-complete. The statement above "computing $\mathbf{P}(\varphi)$ is in #P" means the following: there exists a function F over the input probabilities $\mathbf{P}(X_1), \ldots, \mathbf{P}(X_n)$ (which are rational numbers) s.t. (a) F can be computed in PTIME in n, and (b) the problem "compute $F \cdot \mathbf{P}(\varphi)$" is in #P. For example, in the case of a uniform distribution where $\mathbf{P}(X_i) = 1/2$ and all variables are independent, then we take $F = 2^n$, and $2^n\mathbf{P}(\varphi) = \#\varphi$, hence computing $F \cdot \mathbf{P}(\varphi)$ is in #P. To be more accurate, computing $\mathbf{P}(\varphi)$ is in the class FP with an oracle to #P, in fact with a single call to the #P oracle.

A Dichotomy for Queries without Self-joins

We now establish the following dichotomy for conjunctive queries without self-joins: computing $\mathbf{P}(q)$ is either #P-hard or is in PTIME in the size of the database $PDB = (T, \mathbf{P})$. A query q is said to be without self-joins if each relational symbol occurs at most once in the query body [9,8]. For example $R(x, y), R(y, z)$ has self-joins, $R(x, y), S(y, z)$ has not.

Theorem 3.2 *For each of the queries below (where $k, m \geq 1$), computing $\mathbf{P}(q)$ is #P-hard in the size of the database:*

$$\begin{aligned}
h_1 &= R(\underline{x}), S(\underline{x, y}), T(\underline{y}) \\
h_2^+ &= R_1(\underline{x}, y), \ldots, R_k(\underline{x}, y), S(\underline{y}) \\
h_3^+ &= R_1(\underline{x}, y), \ldots, R_k(\underline{x}, y), S_1(x, \underline{y}), \ldots, S_m(x, \underline{y})
\end{aligned}$$

The underlined positions represent the key attributes (see Sec. 2), thus, in h_1 the database is tuple independent, while in h_2^+, h_3^+ it is a BID. When $k = m = 1$ then we omit the $+$ superscript and write:

$$\begin{aligned}
h_2 &= R(\underline{x}, y), S(\underline{y}) \\
h_3 &= R(\underline{x}, y), S(x, \underline{y})
\end{aligned}$$

The significance of these three (classes of) queries is that the hardness of any other conjunctive query without self-joins follows from a simple reduction from one of these

[3]FPTRAS stands for *fully poly-time randomized approximation scheme.* More precisely: there exists a randomized algorithm A with inputs φ, ε, δ, which runs in polynomial time in $|\varphi|$, $1/\varepsilon$, and $1/\delta$, and returns a value $\tilde{p}$ s.t. $\mathbf{P}_A(|\tilde{p}/p - 1| > \varepsilon) < \delta$. Here $\mathbf{P}_A$ denotes the probability over the random choices of the algorithm. Grädel et al. [15] show how to extend this FPTRAS from the case when all probabilities are 1/2 to arbitrary (but independent) probabilities; we showed in [11] how to extend it to disjoint-independent probabilities.

three (Lemma 3.4). By contrast, the hardness of these three queries is shown directly (by reducing Positive Partitioned 2DNF [24] to h_1, and PERMANENT [29] to h_2^+, h_3^+) and these proofs are more involved.

Previously, the complexity has been studied only for independent probabilistic databases. De Rougemont [13] claimed that it is is in PTIME. Grädel at al. [13,15] corrected this and proved that the query $R(\underline{x}), R(\underline{y}), S_1(\underline{x,z}), S_2(\underline{y,z})$ is #P-hard, by reduction from regular (non-partitioned) 2DNF: note that this query has a self-join (R occurs twice); h_1 does not have a self-join, and was first shown to be #P-hard in [9]; h_2^+ and h_3^+ were first shown to be #P-hard in [11].

A PTIME Algorithm We describe here an algorithm that evaluates $\mathbf{P}(q)$ in polynomial time in the size of the database, which works for some queries, and fails for others. We need some notations. $Vars(q)$ and $Sg(q)$ are the set of variables, and the set of subgoals respectively. If $g \in Sg(q)$ then $Vars(g)$ and $KVars(g)$ denote all variables in g, and all variables in the key positions in g: *e.g.* for $g = R(\underline{x, a, y}, x, z)$, $Vars(g) = \{x, y, z\}$, $KVars(g) = \{x, y\}$. For $x \in Vars(q)$, let $sg(x) = \overline{\{g \mid g \in Sg(q), x \in KVars(g)\}}$. Technically $sg(x)$ depends both on the variable x and the query q, but we will use the notation $sg(x)$ rather than $sg(x, q)$, whenever the query q is clear from the context. Given a database $PDB = (T, \mathbf{P})$, D is its active domain.

Algorithm 3.1 computes $\mathbf{P}(q)$ by recursion on the structure of q. If q consists of connected components q_1, q_2, then it returns $\mathbf{P}(q_1)\mathbf{P}(q_2)$: this is correct since q has no self-joins, e.g $\mathbf{P}(R(\underline{x}), S(\underline{y}, z), T(\underline{y})) = \mathbf{P}(R(\underline{x}))\mathbf{P}(S(\underline{y}, z), T(\underline{y}))$. (We will comment below on the significance of the assumption that q has no self-joins.) If some variable x occurs in a key position in all subgoals, then it applies the independent-project rule: *e.g.* $\mathbf{P}(R(\underline{x})) = 1 - \prod_{a \in D}(1 - \mathbf{P}(R(\underline{a})))$ is the probability that R is nonempty. For another example, we apply an independent project on x in $q = R(\underline{x}, y), S(\underline{x, y})$: this is correct because $q[a_1/x], q[a_2/x], \ldots$ are independent events whenever the constants $a_1, a_2, \ldots$ are distinct. If there exists a subgoal g whose key positions are constants, then it applies a disjoint project on any variable in g: *e.g.* x is such a variable in $q = R(\underline{x}, y), S(\underline{c, d}, x)$, and any two events $q[a/x]$, $q[b/x]$ are disjoint because of the S subgoal.

We illustrate the algorithm on the query below, where a is a constant, and x, y, u are variables:

$$\begin{aligned}
q &= R(\underline{x}), S(\underline{x, y}), T(\underline{y}), U(\underline{u}, y), V(\underline{a}, u)\\
\mathbf{P}(q) &= \sum_{b \in D} \mathbf{P}(R(\underline{x}), S(\underline{x, y}), T(\underline{y}), U(\underline{b}, y), V(\underline{a}, b))\\
&= \sum_{b \in D} \mathbf{P}(R(\underline{x}), S(\underline{x, y}), T(\underline{y}), U(\underline{b}, y))\mathbf{P}(V(\underline{a}, b))\\
&= \sum_{b \in D} \sum_{c \in D} \mathbf{P}(R(\underline{x}), S(\underline{x, c}), T(\underline{c}), U(\underline{b}, c))\mathbf{P}(V(\underline{a}, b))\\
&= \sum_{b \in D} \sum_{c \in D} \mathbf{P}(R(\underline{x}), S(\underline{x, c}))\mathbf{P}(T(\underline{c}))\mathbf{P}(U(\underline{b}, c))\mathbf{P}(V(\underline{a}, b))\\
&= \sum_{b \in D} \sum_{c \in D} (1 - \prod_{d \in D}(1 - \mathbf{P}(R(\underline{d}))\mathbf{P}(S(\underline{d, c})))) \cdot \mathbf{P}(T(\underline{c}))\mathbf{P}(U(\underline{b}, c))\mathbf{P}(V(\underline{a}, b))
\end{aligned}$$

Algorithm 3.1 `Safe-Eval`
Input: query q and database $PDB = (T, \mathbf{P})$
Output: $\mathbf{P}(q)$

```
1: Base Case: if $q = R(\bar{a})$
   return if $R(\bar{a}) \in T$ then $\mathbf{P}(R(\bar{a}))$ else 0
2: Join: if $q = q_1, q_2$ and $Vars(q_1) \cap Vars(q_2) = \emptyset$
   return $\mathbf{P}(q_1)\mathbf{P}(q_2)$
3: Independent project: if $sg(x) = Sg(q)$
   return $1 - \prod_{a \in D}(1 - \mathbf{P}(q[a/x]))$
4: Disjoint project: if $\exists g(x \in Vars(g), KVars(g) = \emptyset)$
   return $\sum_{a \in D} \mathbf{P}(q[a/x])$
5: Otherwise: FAIL
```

We call a query *safe* if algorithm `Safe-Eval` terminates successfully; otherwise we call it *unsafe*. Safety is a property that depends only on the query q, not on the database PDB, and it can be checked in PTIME in the size of q by simply running the algorithm over an active domain of size 1, $D = \{a\}$. Based on our previous discussion, if the query is safe then the algorithm computes the probability correctly:

Proposition 3.3 *For any safe query q, the algorithm computes correctly $\mathbf{P}(q)$ and runs in time $O(|q| \cdot |D|^{|Vars(q)|})$.*

We first described `Safe-Eval` in [8], in a format more suitable for an implementation, by translating q into an algebra plan using joins, independent projects, and disjoint projects, and stated without proof the dichotomy property. Andritsos et al. [4] describe a query evaluation algorithm for a more restricted class of queries.

The Dichotomy Property We define below a rewrite rule $q \Rightarrow q'$ between two queries. Here q is a conjunctive query without self-joins over a schema $\mathcal{R}$, while q' is a conjunctive query without self-joins over a possibly different schema $\mathcal{R}'$. The symbols g, g' denote subgoals below:

$$\begin{array}{rcll}
q & \Rightarrow & q[a/x] & \text{if } x \in Vars(q), a \in D \\
q & \Rightarrow & q_1 & \text{if } q = q_1, q_2, Vars(q_1) \cap Vars(q_2) = \emptyset \\
q & \Rightarrow & q[y/x] & \text{if } \exists g \in Sg(q), x, y \in Vars(g) \\
q, g & \Rightarrow & q & \text{if } KVars(g) = Vars(g) \\
q, g & \Rightarrow & q, g' & \text{if } KVars(g') = KVars(g), \\
 & & & Vars(g') = Vars(g), arity(g') < arity(g)
\end{array}$$

The intuition is that if $q \Rightarrow q'$ then evaluating $\mathbf{P}(q')$ can be reduced in polynomial time to evaluating $\mathbf{P}(q)$. The reduction is quite easy to prove in each case. For example consider an instance of the first reduction: if $q[a/x]$ is a hard query, then obviously q (which has no self-joins) is hard too: otherwise, we can compute $q[a/x]$ on a BID instance by simply removing all possible tuples that do not have an a in the positions where x occurs. All other cases can be checked similarly. This implies:

Lemma 3.4 *If $q \Rightarrow^* q'$ and q' is #P-hard, then q is #P-hard.*

Thus, $\Rightarrow$ gives us a convenient tool for checking if a query is hard, by trying to rewrite it to one of the known hard queries. For example, consider the queries q and q' below: `Safe-Eval` fails immediately on both queries, *i.e.* none of its cases apply. We show that both are hard by rewriting them to h_1 and h_3^+ respectively. By abuse of notations we reuse the same relation name during the rewriting. Strictly speaking, the relation schema in the third line should contain new relation symbols S', T', different from those in the second line, but we reuse the same symbols for readability:

$$\begin{array}{rll}
q & = & R(\underline{x}), R'(\underline{x}), S(\underline{x, y}, y), T(\underline{y, z, b}) \\
 & \Rightarrow & R(\underline{x}), S(\underline{x, y}, y), T(\underline{y, z, b}) \\
 & \Rightarrow^* & R(\underline{x}), S(\underline{x, y}), T(\underline{y}) \qquad\qquad = h_1 \\
q' & = & R(\underline{x}, y), S(\underline{y}, z), T(\underline{z}, x), U(\underline{y}, x) \\
 & \Rightarrow & R(\underline{x}, y), S(\underline{y}, x), T(\underline{x}, x), U(\underline{y}, x) \\
 & \Rightarrow^* & R(\underline{x}, y), S(\underline{y}, x), U(\underline{y}, x) \qquad\quad = h_3^+
\end{array}$$

Call a query q *final* if it is unsafe, and $\forall q'$, if $q \Rightarrow q'$ then q' is safe. Clearly every unsafe query rewrites to a final query: simply apply $\Rightarrow$ repeatedly until all rewritings are to safe queries. We prove in [11]:

Lemma 3.5 *h_1, h_2^+, h_3^+ are the only final queries.*

This implies immediately the dichotomy property:

Theorem 3.6 *Let q be a query without self-joins. Then one of the following holds:*

- *q is unsafe and q rewrites to one of h_1, h_2^+, h_3^+. In particular, q is #P-hard.*
- *q is safe. In particular, it is in PTIME.*

Proof: The rewrite rule $\Rightarrow$ is strongly terminating: each rule results in a query that has either fewer variables, or fewer subgoals, or subgoals with smaller arities. Let $l(q)$ denote the length of the longest rewriting $q \Rightarrow q_1 \Rightarrow \ldots \Rightarrow q_{l(q)}$. We prove that every unsafe query q rewrites to one of h_1, h_2^+, or h_3^+. There are two cases. First, q is final: then, by Lemma 3.5, q is one of h_1, h_2^+, h_3^+. Second, q is not final: then, by definition, there exists an unsafe query q' such that $q \Rightarrow q'$. Since $l(q') < l(q)$, we can apply induction to q' and show that it rewrites to one of h_1, h_2^+, h_3^+: hence, q also rewrites to one of h_1, h_2^+, h_3^+. $\Box$

How restrictive is the assumption that the query has no self-joins ? It is used both in `Join` and in `Independent project`. We illustrate on $q = R(\underline{x, y}), R(\underline{y, z})$ how, by dropping the assumption, independent projects become incorrect. Although y occurs in all subgoals, we cannot apply an independent project because the two queries $q[a/y] = R(\underline{x, a}), R(\underline{a, z})$ and $q[b/y] = R(\underline{x, b}), R(\underline{b, z})$ are not independent: both $\varphi_{q[a/y]}$ and $\varphi_{q[b/y]}$ depend on the tuple $R(a, b)$ (and also on $R(b, a)$). In fact q is #P-hard [10]. The restriction to queries without self-joins is thus significant. We have extended the dichotomy property to unrestricted conjunctive queries, but only over independent probabilistic databases [10];

the complexity of unrestricted conjunctive queries over BID probabilistic databases is open.

The Complexity of the Complexity We complete our analysis by studying the following problem: given a relational schema $\mathcal{R}$ and conjunctive query q without self-joins over $\mathcal{R}$, decide whether q is safe[4]. We have seen that this problem is in PTIME (simply run the algorithm on a PDB with one tuple per relation and see if it gets stuck); here we establish tighter bounds.

In the case of *independent databases*, the key in each relation R consists of all the attributes, $Key(R) = Attr(R)$, hence $sg(x)$ becomes: $sg(x) = \{g \mid x \in Vars(g)\}$.

Definition 3.7 *A conjunctive query is* hierarchical *if for any two variables x, y, either $sg(x) \cap sg(y) = \emptyset$, or $sg(x) \subseteq sg(y)$, or $sg(y) \subseteq sg(x)$.*

As an example, the query[5] $q = R(x), S(x, y)$ is hierarchical because $sg(x) = \{R, S\}$, $sg(y) = \{S\}$, while $h_1 = R(x), S(x, y), T(y)$ is not hierarchical because $sg(x) = \{R, S\}$ and $sg(y) = \{S, T\}$. `SAFE-EVAL` works as follows on independent databases. When the hierarchy $\{sg(x) \mid x \in Vars(q)\}$ has a root variable x, then it applies an independent project on x; when it has multiple connected components, then it applies joins. One can check easily that a query is unsafe iff it contains a sub-pattern:

$$R(x, \ldots), S(x, y, \ldots), T(y, \ldots)$$

Proposition 3.8 *Let SG be a binary relation name. We represent a pair $\mathcal{R}$, q, where $\mathcal{R}$ is a relational schema for an independent database and q a conjunctive query without self-joins, as an instance over SG, as follows[6]. The constants are $\mathcal{R} \cup Vars(q)$, and for each subgoal R of q and each variable $x \in Vars(R)$, there is a tuple $SG(R, x)$. Then the property "given $\mathcal{R}$, q, q is unsafe" can be expressed in FO over the vocabulary SG.*

In fact, it is expressed by the following conjunctive query with negations, with variables R, S, T, x, y:

$$SG(R, x), \neg SG(R, y), SG(S, x), SG(S, y), SG(T, y), \neg SG(T, x)$$

In the case of BIDs, checking safety is PTIME complete. Recall the Alternating Graph Accessibility Problem (AGAP): given a directed graph where the nodes are partitioned into two sets called AND-nodes and OR-nodes, decide if all nodes are accessible. An AND-node is accessible if all its parents are; an OR node is accessible if at least one of its parents is. AGAP is PTIME-complete [17]. We prove:

Proposition 3.9 *AGAP is reducible in LOGSPACE to the following problem: given a schema $\mathcal{R}$ and a query q without self-joins, check if q is safe. In particular, the latter is PTIME-hard.*

[4]For a fixed $\mathcal{R}$ there are only finitely many queries without self-joins: this is the reason why $\mathcal{R}$ is part of the input.

[5]Since all attributes are keys we don't underline them.

[6]This representation is lossy, because it ignores both the positions where the variables occur in the subgoals in q, and it also ignores all constants in q.

Proof: Let G be an AND/OR graph, we construct the following relational schema $\mathcal{R}$ and query q. There will be one variable in q for every node in G, plus one extra variable c; the invariant is: a node is accessible in the graph iff the corresponding variable can be made constant by `SAFE-EVAL`. For every AND node x with parents $y, z, \ldots$ we introduce a new relational symbol R and a subgoal $R(\underline{y, z, \ldots}, x)$ in q (a disjoint project applies to x iff all its parents are constants). For every OR node x with parents $y, z, \ldots$ we introduce a new subgoal for each parent $S_y(\underline{y}, z), \ldots$ (a disjoint project applies to z iff one of its parents is constant). Finally, for each node $x, y, z, \ldots$ we create a new subgoal $T_x(\underline{c}, x)$, $T_y(\underline{c}, y)$: these prevent an independent-project until all variables became constants. □

4. Materialized Views on Probabilistic Databases

We have shown that there exists a sharp separation between *safe queries*, which can be evaluated efficiently, and *unsafe queries*, which are provably hard. Since a probabilistic database system needs to support arbitrary queries, not just safe queries, a key research problem is how one can improve the performance of unsafe queries. The approach that we describe here uses materialized views.

Materialized views are widely used today to speed up query evaluation in traditional databases. Early query optimizers used materialized views that were restricted to indexes (which are simple projections on the attributes being indexed) and join indexes [28]; modern query optimizers can use arbitrary materialized views [3].

When used in probabilistic databases, materialized views can make dramatic impact. Suppose we need to evaluate a Boolean query q on a BID probabilistic database, and assume q is unsafe. In this case, one has to use some general-purpose probabilistic inference method, for example Luby and Karp's FPTRAS, and its performance in practice is much worse than that of safe plans: in one experimental study [25] we have observed two orders of magnitudes difference in performance. However, by rewriting q in terms of a view it may be possible to transform it into a safe query, which can be evaluated very efficiently. There is no magic here: we simply pay the #P cost when we materialize the view, then evaluate the query in PTIME at runtime.

Example 4.1 Consider the schema $R(\underline{C, A})$, $S(\underline{C, A, B})$, $T(\underline{C, B})$, and the view:

$$v(z) \ :\!- \ R(z,x), S(z,x,y), T(z,y)$$

Denote $V(Z)$ the schema of the materialized view. Then, all tuples in the materialized view V are independent. For the intuition behind this statement, notice that for two different constants $a \neq b$, the Boolean queries $v(a)$ and $v(b)$ depend on disjoint sets of tuples in the input BID probabilistic database: the first depends on inputs of the form $R(a, \ldots)$, $S(a, \ldots)$, $T(a, \ldots)$, while the second on inputs of the form $R(b, \ldots)$, $S(b, \ldots)$, $T(b, \ldots)$. Thus, $v(a)$ and $v(b)$ are independent probabilistic events, and we say that the tuples a and b in the view are independent. In general, any set of tuples $a, b, c, \ldots$ in the view are independent. Suppose we compute and store the view, meaning that we will determine all its tuples $a, b, c, \ldots$ and compute their probabilities. This will be expensive, because, for each constant a, the Boolean query $v(a)$ rewrites to h_1 (see Theorem 3.2), hence it is a #P-hard. Nevertheless, we will pay this cost, and materialize the view. Later, we will use V to answer queries. For example consider the Boolean query q:-$R(z,x), S(z,x,y), T(z,y), U(z,v)$, where $U(\underline{C,D})$ is another relation. Then q is #P-hard, but after rewriting it as q:-$V(z), U(z,v)$ it becomes a safe query, and can be computed by a safe plan. Thus, by using V to evaluate q we obtain a dramatic reduction in complexity.

The major challenge in using probabilistic views for query processing is how to find, represent, and use independence relationships between the tuples in the view. In general, the tuples in the view may be correlated in complex ways. One possibility is to store the lineage for each tuple t (this is the approach in Trio [27]), but this makes query evaluation on the view no more efficient than expanding the view definition in the query.

We propose an alternative approach:

- When we materialize the view we store only the set of possible tuples and their marginal probabilities. We do not store their lineage.
- We compute a *partial representation* for the view. This is a schema level information, consisting of a set of attributes such that tuples in the view that have distinct values of these attributes are guaranteed to be independent; the representation can be further refined with a second set of attributes, such that any distinct tuples that agree on these attributes are disjoint. Thus, a partial representation describes independence or disjointness relationships between some tuples, while leaving other correlations unspecified.
- To evaluate a query q, one first checks if q can be rewritten in terms of the view (using standard techniques [18]). If that is the case, then an additional test needs to be performed to see if the probability of q depends only on sets of tuples in the view whose correlations are known, that is, if q inspects only sets of independent tuples or sets that contain at least one pair of disjoint tuples. In the latter case, q can be evaluated by using the data and marginal probabilities in the view v. Otherwise, the view cannot be used and the computation needs to evaluate q on the base tables.

Some of the results in this section have been announced in [26].

4.1. Partial Representation of a Probabilistic Table

Consider a probabilistic database $(W, \mathbf{P})$, as in Definition 2.1, over a schema consisting of a single relation name V. In this section we will no longer assume that V is a block disjoint-independent (BID) table, but rather allow it to be an arbitrary probability space over the set of possible instances for V. The intuition is that V is a view computed by evaluating a query over some BID database: the output of the view is, in general, not a BID table, but some arbitrary probabilistic database $(W, \mathbf{P})$.

Recall that $\mathbf{P}(t)$ is the probability of the event "a randomly chosen instance contains the tuple t", and $\mathbf{P}(q)$ is the probability of the event "a randomly chosen instance satisfies q". A set of events $\{e_1, \ldots, e_n\}$ is called *independent* if $\mathbf{P}(e_1 \wedge \cdots \wedge e_n) = \mathbf{P}(e_1) \cdots \mathbf{P}(e_n)$. Two events e_1, e_2 are *disjoint* if $\mathbf{P}(e_1 \wedge e_2) = 0$. We abbreviate the event $e_1 \wedge e_2$ with e_1, e_2.

When the view V is a BID table, then, denoting $K = Key(V)$, the following two properties hold: for any set of tuples, if any two tuples in the set differ on at least one of the K attributes, then the set of tuples is independent, and any two tuples that agree on the K attributes are disjoint. If the view V is not a BID table, but an arbitrary probabilistic relation, then we may still be able to find two sets of attributes, L and K, that satisfy these two properties separately. Formally:

Definition 4.2 *Let $(W, \mathbf{P})$ be a probabilistic database over a single relation V.*

- *We say that $(W, \mathbf{P})$ is L-block independent, where $L \subseteq Attr(V)$, if any set of tuples $\{t_1, \ldots, t_n\}$ s.t. $t_i.L \neq t_j.L$, $1 \leq i < j \leq n$, is independent.*
- *We say that $(W, \mathbf{P})$ is K-block disjoint, where $K \subseteq Attr(V)$, if any two tuples t, t' s.t. $t.K = t'.K$ are disjoint. Equivalently, K is a key in each possible world of V.*

A *partial representation* is a pair of sets of attributes (L, K) such that V is L-block independent and K-block disjoint. A partial representation for a view allows us to answer queries by using the view. For example, if the partial representation (L, K) is such that $K = L$, then V is a BID table with $Key(V) = K$, and therefore, once we have computed the marginal probabilities for each tuple in V, we can use it to compute other queries by using the view as a regular base tables. Even if $L \neq K$ we can often still leverage the view to answer some queries from the view. For a trivial example suppose the query is q:-$V(\bar{a}), V(\bar{b})$, where $\bar{a}$ and $\bar{b}$ are two ground tuples that differ on at least one attribute in L. Then, these two tuples are independent, hence we can answer the query q as the product of the marginal probabilities of $\bar{a}$ and $\bar{b}$.

A probabilistic table V may admit more than one partial representation. For example, every probabilistic table V admits the trivial partial representation $L = \emptyset$ and $K = Attr(V)$, but this representation is not useful to answer queries, except the most trivial queries that check for the presence of a single ground tuple. Intuitively, we want a "large" L and a "small" K. It is easy to check that we can always relax the representation in the other way: if (L, K) is a partial representation, and L', K' are such that $L' \subseteq L$ and $K \subseteq K'$, then (L', K') is also a partial representation. Of course, we want to go in the opposite direction: increase L, decrease K. We will give several results in the following sections showing that a unique maximal L exists, and will explain how to compute it. On the other hand, no minimal K exists in general; we will show that the space of possible choices for K can be described using standard functional dependencies theory.

It is not obvious at all that a maximal L exists, and in fact it fails in the most general case, as shown by the following example.

Example 4.3 Consider a probabilistic table $V(A, B, C)$ with three possible tuples:

T :

A	B	C	
a	b	c	t_1
a	b'	c	t_2
a	b'	c'	t_3

and four possible worlds: $I_1 = \emptyset$, $I_1 = \{t_1, t_2\}$, $I_2 = \{t_2, t_3\}$, $I_3 = \{t_1, t_3\}$, each with probability 1/4. Any two tuples are independent: indeed $\mathbf{P}(t_1) = \mathbf{P}(t_2) = \mathbf{P}(t_3) = 1/2$ and $\mathbf{P}(t_1t_2) = \mathbf{P}(t_1t_3) = \mathbf{P}(t_2t_3) = 1/4$. V is AB-block independent: this is because the only sets of tuples that differ on AB are $\{t_1, t_2\}$ and $\{t_1, t_3\}$, and they are independent.

Similarly, V is also AC-block independent. But V is not ABC-block independent, because any two tuples in set $\{t_1, t_2, t_3\}$ differ on ABC, yet the entire set is not independent: $\mathbf{P}(t_1t_2t_3) = 0$. This shows that there is no largest set L: both AB and AC are maximal.

A weaker result holds. For a set $L \subseteq \mathit{Attr}(V)$ we say that V is L-block 2-independent if any two tuples t_1, t_2 s.t. $t_1.L \neq t_2.L$ are independent.

Lemma 4.4 *Let V be a probabilistic table. Then there exists a maximal set L s.t. V is L-block 2-independent.*

Proof: We prove the following: if $L_1, L_2 \subseteq \mathit{Attr}(V)$ are such that V is L_i-block 2-independent for each $i = 1, 2$, then V is also L-block 2-independent, where $L = L_1 \cup L_2$. Indeed, let t_1, t_2 be two tuples s.t. $t_1.L \neq t_2.L$. Then either $t_1.L_1 \neq t_2.L_1$ or $t_1.L_2 \neq t_2.L_2$, hence t_1, t_2 are independent tuples. The largest set L claimed by the lemma is then the union of all sets L' s.t. V is L'-block 2-independent. □

Continuing Example 4.3 we note that V is ABC-block 2-independent, since any two of the tuples t_1, t_2, t_3 are independent.

4.2. Partial Representation of a pc-Table

If V is a view defined by an FO query over a BID database, then V can be expressed as a pc-table. In this section we study the partial representation of a pc-table. Recall that a pc-table consists of two parts: $V = (CV, \mathbf{P})$, where CV is a c-table and $\mathbf{P}$ a product probability space on the set of variables $\bar{X}$ that are used in the Boolean expressions of the c-table. Our main result in this section is the following: given the c-table CV there exists a maximal set of attributes L such that for any product probability space $\mathbf{P}$, the pc-table $(CV, \mathbf{P})$ is L-block independent. Thus, if V is a view defined by an FO query over a BID database, then this result shows us how to compute a good partial representation for the view.

Let $\bar{X} = \{X_1, \ldots, X_m\}$ be the variables used in the Boolean expressions in the c-table CV, and let $\mathit{Dom}(X_j)$ be the finite domain of values for the variable X_j, $j = 1, m$. The c-table CV consists of n tuples $t_1, \ldots, t_n$, each annotated with a Boolean expression $\varphi_1, \ldots, \varphi_n$ obtained from atomic formulas of the form $X_j = v$, where $v \in \mathit{Dom}(X_j)$, and the connectives $\wedge$, $\vee$, and $\neg$. A *valuation* θ is a function $\theta : \bar{X} \to \bigcup_i \mathit{Dom}(X_j)$ s.t., $\theta(X_j) \in \mathit{Dom}(X_j)$, for $j = 1, m$.

The following definition is adapted from [22].

Definition 4.5 *Let φ be a Boolean expression over variables $\bar{X}$. A variable X_j is called a* critical variable *for φ if there exists a valuation θ for the variables $\bar{X} - \{X_j\}$ and two values $v', v'' \in \mathit{Dom}(X_j)$ s.t. $\varphi[\theta \cup \{(X_j, v')\}] \neq \varphi[\theta \cup \{(X_j, v'')\}]$.*

For a simple illustration, suppose $\mathit{Dom}(X_1) = \mathit{Dom}(X_2) = \mathit{Dom}(X_3) = \{0, 1, 2\}$, and consider the Boolean expression

$$\varphi \equiv (X_1 = 0) \vee (X_1 = 1) \vee ((X_3 = 1) \wedge \neg(X_1 = 2))$$

Then X_2 is not a critical variable for φ, because it is not mentioned in the expression of φ. X_3 is also not a critical variable, because φ simplifies to $(X_1 = 0) \vee (X_1 = 1)$ (since

$Dom(X_1) = \{0, 1, 2\}$). On the other hand, X_1 is a critical variable: by changing X_1 from 0 to 2 we change φ from *true* to *false*. In notation, if θ is the valuation $\{(X_2, 0), (X_3, 0)\}$, then $\varphi[\theta \cup \{(X_1, 0)\}] = \mathit{true}$, $\varphi[\theta \cup \{(X_1, 2)\}] = \mathit{false}$.

To obtain a pc-table from a c-table CV we need to define a product probability space $\mathbf{P}$. This consists of n independent probabilty spaces $\mathbf{P}_i$ over $Dom(X_i)$, $\mathbf{P}_i : Dom(X_i) \to [0, 1]$ s.t. $\sum_{v \in Dom(X_i)} \mathbf{P}_i(v) = 1$, and is defined as $\mathbf{P}(X_1 = v_1, \ldots, X_n = v_n) = \prod_i \mathbf{P}_i(v_i)$. Any Boolean expression φ is a probabilistic event, and we denote $\mathbf{P}(\varphi)$ its probability. The main result in this section is based on the following technical lemma, which is a generalization of [22].

Lemma 4.6 *Let φ, ψ be two Boolean expressions. Then the following two statements are equivalent:*

- *For every product probability space $\mathbf{P}$ on the set of variables $\bar{X}$, φ and ψ are independent events.*
- *φ and ψ have no common critical variables.*

Proof: The "if" direction is straightforward: if φ, ψ use disjoint sets of variables, then $\mathbf{P}(\varphi \wedge \psi) = \mathbf{P}(\varphi)\mathbf{P}(\psi)$, hence they are independent for any choice of $\mathbf{P}$.

The "only if" direction was shown in [22] for the case when all variables X_j are Boolean, i.e. $|Dom(X_j)| = 2$. We briefly review the proof here, then show how to extend it to non-boolean variables. Given a probability spaces $(Dom(X_j), \mathbf{P}_j)$, denote $x_j = \mathbf{P}_j(X_j = 1)$, hence $\mathbf{P}_j(X_j = 0) = 1 - x_j$. Then $\mathbf{P}(\varphi)$ is a polynomial in the variables $x_1, \ldots, x_m$ where each variable has degree ≤ 1. (For example, if $\varphi = \neg(X_1 \otimes X_2 \otimes X_3)$ (exclusive or) then $\mathbf{P}(\varphi) = x_1 x_2 (1 - x_3) + x_1 (1 - x_2) x_3 + (1 - x_1) x_2 x_3 + (1 - x_1)(1 - x_2)(1 - x_3)$, which is a polynomial of degree 1 in x_1, x_2, x_3.) One can check that if X_j is a critical variable for φ then the degree of x_j in the polynomial $\mathbf{P}(\varphi)$ is 1; on the other hand, if X_j is not a critical variable, then the degree of x_j in the polynomial $\mathbf{P}(\varphi)$ is 0 (in other words the polynomial does not depend on x_j). The identity $\mathbf{P}(\varphi)\mathbf{P}(\psi) = \mathbf{P}(\varphi \wedge \psi)$ must hold for any values of $x_1, \ldots, x_m$, because φ, ψ are independent for any $\mathbf{P}$. If X_j were a common critical variable for both φ and ψ then the degree of x_j in the left hand side polynomial is 2, while the right hand side has degree at most 1, which is a contradiction.

We now extend this proof to non-Boolean domains. In this case a variable X_j may take values $0, 1, \ldots, d_j$, for $d_j \geq 1$. Define the variables x_{ij} to be $x_{ij} = \mathbf{P}(X_j = i)$, for $i = 1, \ldots, d_j$, thus $\mathbf{P}(X_j = 0) = 1 - x_{1j} - x_{2j} - \cdots - x_{d_j j}$. As before $\mathbf{P}(\varphi)$ is a polynomial of degree 1 in the variables x_{ij} with the additional property that if $i_1 \neq i_2$ then $x_{i_1 j}$ and $x_{i_2 j}$ cannot appear in the same monomial. We still have the identity $\mathbf{P}(\varphi\psi) = \mathbf{P}(\varphi)\mathbf{P}(\psi)$, for all values of the variables x_{ij} (since the identity holds on the set $x_{ij} \geq 0$ for all i, j, and $\sum_i x_{ij} \leq 1$, for all j, and this set has a non-empty interior). If X_j is a critical variable for φ then $\mathbf{P}(\varphi)$ must have a monomial containing some $x_{i_1 j}$; if it is also critical for ψ, then $\mathbf{P}(\psi)$ has a monomial containing $x_{i_2 j}$. Hence their product contains $x_{i_1 j} \cdot x_{i_2 j}$, contradiction. $\square$

We will use the lemma to prove the main result in this section:

Theorem 4.7 *Let CV be a c-table. Then there exists a unique maximal set of attributes L such that, for any product probability space $\mathbf{P}$, the pc-table $(CV, \mathbf{P})$ is L-block independent.*

Proof: Denote $t_1, \ldots, t_n$ the tuples of the c-table CV, and $\varphi_1, \ldots, \varphi_n$ their Boolean expressions annotations. Let L be a set of attributes. We prove the following:

Lemma 4.8 *The following three statements are equivalent:*

1. *For any* $\mathbf{P}$, *the pc-table* $(CV, \mathbf{P})$ *is* L*-block independent.*
2. *For any* $\mathbf{P}$, *the pc-table* $(CV, \mathbf{P})$ *is* L*-block 2-independent.*
3. *For any two tuples* t_i, t_j, *if* $t_i.L \neq t_j.L$ *then the Boolean expressions* φ_i *and* φ_j *do not have any common critical variables.*

Proof: Obviously (1) implies (2), and from the Lemma 4.6 it follows that (2) implies (3) (since $\mathbf{P}(t_i) = \mathbf{P}(\varphi_i)$ and $\mathbf{P}(t_i, t_j) = \mathbf{P}(\varphi_i \wedge \varphi_j)$). For the last implication, let $\mathbf{P}$ be any product probability space, and consider some m tuples $t_{i_1}, \ldots, t_{i_m}$ that have distinct values for their L attributes. Then, the Boolean expressions $\varphi_{i_1}, \ldots, \varphi_{i_m}$ depend on disjoint sets of Boolean variables, hence $\mathbf{P}(t_1, \ldots, t_m) = \mathbf{P}(\varphi_{i_1}, \ldots, \varphi_{i_m}) = \mathbf{P}(\varphi_{i_1}) \cdots \mathbf{P}(\varphi_{i_m}) = \mathbf{P}(t_1) \cdots \mathbf{P}(t_m)$, proving that they are independent. Thus, the three statements above are equivalent. $\square$

Continuing the proof of Theorem 4.7, consider two sets of attributes L_1, L_2 such that each satisfies condition (3) in Lemma 4.8. Then their union, $L_1 \cup L_2$, also satisfies condition (3): indeed, if t_i and t_j are two tuples such that $t_i.(L_1 \cup L_2) \neq t_j.(L_1 \cup L_2)$, then either $t_i.L_1 \neq t_j.L_1$ or $t_i.L_2 \neq t_j.L_2$, and in either case φ_i and φ_j do not have any common critical tuples. It follows that there exists a maximal set of attributes L that satisfies condition (3), which proves the theorem. $\square$

As an application of this result, we show how to apply it to a view V defined by an FO expression over a BID database. Let $\mathcal{R}$ be a relational schema, and let $PDB = (T, \mathbf{P})$ be a BID database, where T is a set of possible tuples.

Corollary 4.9 *For every FO view definition* v *over the relational schema* $\mathcal{R}$ *and for every set of possible tuples* T *there exists a unique maximal set of attributes* L *such that: for any BID database* $PDB = (T, \mathbf{P})$ *the probabilistic view* $v(T, \mathbf{P})$ *is* L*-block independent.*

The proof follows immediately from Theorem 4.7 and the observation that the view $v(PDB)$ is a pc-table.

We end this section by noting that, in general, no unique minimal K exists. For example the c-table below has two minimal keys, $\{A\}$ and $\{B\}$:

A	B	
a_1	b_1	$X = 1 \wedge Y = 1$
a_1	b_2	$X = 1 \wedge Y = 2$
a_2	b_1	$X = 2 \wedge Y = 1$
a_2	b_2	$X = 2 \wedge Y = 2$

Therefore, any pc-table defined over this table is both A-block disjoint, and B-block disjoint, but it is not $\emptyset$-block disjoint.

4.3. Partial Representation of a Conjunctive View

In the previous section we have shown how to compute a partial representation for a given view (expressed in FO) and a given input BID database. We now study how to compute a partial representation given only the view expression, and not the input database. In this case we seek a partial representation (L, K) that is satisfied by the the view for *any* input BID database. This partial representation depends only on the view expression, not the data instance, and is computed through static analysis on the view expression only. Throughout this section we will restrict the view expression to be a conjunctive query: we will allow self-joins, unless otherwise stated.

Fix the schema $\mathcal{R}$ of a BID database, and let V be defined by a conjunctive query v over $\mathcal{R}$. Given a BID input, PDB, we denote $V = v(PDB)$ the probabilistic table obtained by evaluating the view on the BID input. We prove in this section two results.

Theorem 4.10 *Fix a relational schema $\mathcal{R}$.*

1. *For any conjunctive query v there exists a unique maximal set of attributes $L \subseteq Attrs(v)$ such that for any BID database PDB over the schema $\mathcal{R}$, $v(PDB)$ is L-block independent.*

2. *The problem* "given a conjunctive query v and $L \subseteq Attrs(v)$ check whether for any BID database PDB, $v(PDB)$ is L-block independent"*, is in Π_2^p. Moreover, there exists a schema $\mathcal{R}$ for which this problem is Π_2^p-hard.*

The theorem, in essence, says that there exists a unique maximal set of attributes L, and computing such a set is Π_2^p-complete in the size of the conjunctive query v. To obtain a good partial representation (L, K) for the view v, we also need to compute K: finding K is equivalent to inferring functional dependencies on the output of a conjunctive view, from the key dependencies in the input schema $\mathcal{R}$. This problem is well studied [2], and we will not discuss it further, but note that, in general, there may not be a unique minimal set K.

Before proving Theorem 4.10 we give some examples of partial representations for conjunctive views.

Example 4.11 1. Consider the schema $R(\underline{A})$, $S(\underline{A, B}, C, D)$, and the view:

$$v(x, y, z) \quad \text{:-} \quad R(\underline{x}), S(\underline{x, y}, z, u)$$

Denote $V(X, Y, Z)$ the schema of the materialized view. A partial representation for V is (X, XY). To see that V is X-block independent, notice that if two tuples in V differ on their X attribute, then the lineage of the two tuples depends on disjoint sets of input tuples in R and S. To see that V is XY-block disjoint, it suffices to see that the functional dependency $XY \to Z$ holds in V (because it holds in S). Thus, (X, XY) is a partial representation for V, and one can see that it is the best possible (that is, we cannot increase X nor decrease XY).

2. Consider the schema $R(\underline{A, B}, C)$, $S(\underline{A, C}, B)$ and the view:

   $$v(x, y, z) \ \text{:-}\ R(\underline{x, y}, z), S(\underline{x, z}, y)$$

   Here V is X-block independent. In addition, V is both XY-block disjoint and XZ-block disjoint: but it is not X-block disjoint. There are two "best" partial representations: (X, XY) and (X, XZ).

In the remainder of this section we will prove Theorem 4.10, and start with Part 1. Fix a BID probabilistic database PDB. Then $V = v(PDB)$ is a pc-table: by Theorem 4.7 there exists a unique maximal set of attributes L_{PDB} such that V is L-block independent (for any choice of the probability function in PDB). Then the set of attributes $\bigcap_{PDB} L_{PDB}$ is the unique, maximal set of attributes claimed by the theorem.

Before proving part 2 of Theorem 4.10, we need to review the notion of a critical tuple for a Boolean query q.

Definition 4.12 *A ground tuple t is called* critical *for a Boolean query q if there exists a (conventional) database instance I s.t. $q(I) \neq q(I \cup \{t\})$.*

A tuple is critical if it makes a difference for the query. For a simple illustration, consider the Boolean query q:-$R(x, x), S(a, x, y)$, where a is a constant. Then $R(b, b)$ (for some constant b) is a critical tuple because q is false on the instance $I = \{S(a, b, c)\}$ but true on the instance $\{R(b, b), S(a, b, c)\}$. On the other hand $R(b, c)$ is not a critical tuple. In general, if the query q is a conjunctive query, then any critical tuple must be the ground instantiation of a subgoal. The converse is not true as the following example from [22] shows: q:-$R(x, y, z, z, u), R(x, x, x, y, y)$. The tuple $t = R(a, a, b, b, c)$, which is a ground instantiation of the first subgoal, is not a critical tuple. Indeed, if q is true on $I \cup \{t\}$, then only the first subgoal can be mapped to t, and therefore the second subgoal is mapped to the ground tuple $R(a, a, a, a, a)$, which must be in I: but then q is also true on I, hence t is not critical. We review here the main results from [22]. While these results were shown for a relational schema $\mathcal{R}$ where the key of each relation is the set of all its attributes (in other words, any BID database over schema $\mathcal{R}$ is a tuple-independent probabilistic database), they extend immediately to BID probabilistic databases (the main step of the extension consists of Lemma 4.6).

Theorem 4.13 *[22] Fix a relational schema $\mathcal{R}$.*

1. *Let q, q' be two Boolean conjunctive queries over the schema $\mathcal{R}$. Then the following two statements are equivalent: (a) q and q' have no common critical tuples, (b) for any BID probabilistic database over the schema $\mathcal{R}$, q and q' are independent events.*

2. *The problem* "given two Boolean queries q, q', check whether they have no common critical tuples" *is in* Π_2^p.

3. *There exists a schema $\mathcal{R}$ such that the problem* "given a Boolean query q and a ground tuple t, check whether t is not a critical tuple for q", *is* Π_2^p*-hard.*

We now prove part 2 of Theorem 4.10. Given a set of attributes L, the following two conditions are equivalent: (a) for any input PDB, $v(PDB)$ is L-block independent, and (b) for any two distinct grounded $|L|$-tuples $\bar{a}$, $\bar{b}$, the two boolean queries $v(\bar{a})$ and $v(\bar{b})$ have no common critical tuples. (Here $v(\bar{a})$ denotes the Boolean query where the head variables corresponding to the L attributes are subtituted with $\bar{a}$, while the rest of the head variables are existentially quantified). The equivalence between (a) and (b) follows from Lemma 4.8 (2) and Theorem 4.13 (1). Membership in Π_2^2 follows now from property (b) and Theorem 4.13 (2).

To prove hardness for Π_2^p, we use Theorem 4.13 (3): we reduce the problem "given a query q and a tuple t check whether t is not a critical tuple for q" to the problem (b) above. Let $\mathcal{R}$ be the vocabulary for q and t, and let the ground tuple t be $T(a_1, \ldots, a_k)$. Construct a new vocabulary $\mathcal{R}'$ obtained from $\mathcal{R}$ by adding two new attributes to each relation name: that is, if $R(A_1, \ldots, A_m)$ is a relation name in $\mathcal{R}$, then $R'(U, V, A_1, \ldots, A_m)$ is a relation name in $\mathcal{R}'$. Let U, V be two variables. Denote $q'(U, V)$ the query obtained from q by replacing every subgoal $R(\ldots)$ in q with $R'(U, V, \ldots)$ (thus, the variables U, V will occur in all subgoals of $q'(U, V)$), and define the following view:

$$v(U, V) \quad :\!- \quad q'(U, V), T(V, U, a_1, \ldots, a_k)$$

We show that v is UV-block independent iff t is not a critical tuple for q. For that, we consider two distinct constant tuples (u_1, v_1) and (u_2, v_2) and examine whether the Boolean queries $v(u_1, v_1)$ and $v(u_2, v_2)$ are independent, or, equivalently, have no common critical tuples. All critical tuples of $v(u_1, v_1)$ must have the form $R(u_1, v_1, \ldots)$, or $T(v_1, u_1, \ldots)$; in other words, they must start with the constants u_1, v_1 or with v_1, u_1. Similarly for $v(u_2, v_2)$; hence, if $\{u_1, v_1\} \neq \{u_2, v_2\}$ then the queries $v(u_1, v_1)$ and $v(u_2, v_2)$ have no common critical tuples. The only case when they could have common critical tuples is when $u_1 = v_2$ and $u_2 = v_1$ (since $(u_1, v_1) \neq (u_2, v_2)$), and in that case they have a common critical tuple iff $T(u_1, v_1, a_1, \ldots, a_k)$ is a critical tuple for $q'(u_1, v_1)$, and this happens iff $T(a_1, \ldots, a_k)$ is a critical tuple for q. This completes the hardness proof.

4.4. Querying Partially Represented Views

We have shown how to compute a "best" partial representation (L, K) for a materialized view V: tuples with distinct values for L are independent, while tuples that agree on K are disjoint. All other pairs of tuples, which we call *intertwined*, have unspecified correlations. In this section we study the problem of deciding whether a query q can be answered by using only the marginal probabilities in V: we say that q is *well-defined* in terms of the view and its partial representation. Intuitively, q does not look at pairs of intertwined tuples. This problem is complementary to the query answering using views problem [18]: there, we are given a query q over a conventional database and a set of views, and we want to check if q can be rewritten into an equivalent query q' that uses the views. We assume that the rewriting has already been done, thus q already mentions the view(s).

We illustrate with an example:

Example 4.14 Let $V(A, B, C)$ have the following partial representation: $L = A$, $K = AB$. Consider the following queries:

$$\begin{aligned} q_1 &\text{ :- } V(a, y, z) \\ q_2 &\text{ :- } V(x, b, y) \\ q_3 &\text{ :- } V(x, y, c) \end{aligned}$$

Here x, y, z are variables, and a, b, c are constants. For example, the view could be that from Example 4.11 (1), $v(x, y, z)$:-$R(\underline{x}), S(\underline{x, y}, z, u)$, and the query q_1 could be q_1:-$R(\underline{a}), S(\underline{a, y}, z, u)$: after rewriting q_1 in terms of the view, we obtain the equivalent expression q_1:-$V(a, y, z)$.

We argue that q_2 is well-defined, while q_1, q_3 are not. Consider first q_2. Its value depends only on the tuples of the form (a_i, b, c_j), where the constants a_i and c_j range over the active domain, and b is the fixed constant occurring in the query. Partition these tuples by a_i. For each $i = 1, 2, \ldots$, any two tuples in the set defined by a_i are disjoint (because V satisfies the partial representation (A, AB), and any two tuples in the same group agree on both A and B): thus, the Boolean query $\exists z.V(a_i, b, z)$ is a disjunction of the exclusive events $V(a_i, b, c_j)$, and therefore its probability is the sum of the probabilities $\mathbf{P}(V(a_i, b, c_j))$. Moreover, the set of events $\{\exists z.V(a_1, b, z), \exists z.V(a_2, b, z), \ldots\}$, is independent, which allows us to compute the probability of the query q_2, since it is the disjunction of these independent events: $q_2 \equiv \exists x.\exists z.V(x, b, z)$. Thus, assuming that the view V satisfies the partial representation (A, AB), the probability of q_2 depends only on the marginal tuples probabilities in the view V.

In contrast, neither q_1 nor q_2 are well-defined. To see this, suppose that the view has exactly two tuples, $t_1 = (a, b_1, c)$ and $t_2 = (a, b_2, c)$. These tuples are intertwined, *i.e.* the probability $\mathbf{P}(t_1, t_2)$ is unknown. Further, $\mathbf{P}(q_1) = \mathbf{P}(q_3) = \mathbf{P}(t_1 \vee t_2) = \mathbf{P}(t_1) + \mathbf{P}(t_2) - \mathbf{P}(t_1 t_2)$: $\mathbf{P}(t_1)$ and $\mathbf{P}(t_2)$ are well defined in the view, but $\mathbf{P}(t_1, t_2)$ is unknown. So, neither q_1 nor q_3 is well-defined.

In this section we consider Boolean, monotone queries q, which includes conjunctive queries. We assume that q is over a single view V, and mentions no other relations. This is not too restrictive, since q may have self-joins over V, or unions (since we allow arbitrary monotone queries). It is straightforward to extend our results to a query expressed over multiple views $V_1, V_2, \ldots$, each with its own partial representation, assuming that all views are independent.

Definition 4.15 *Let V be a view with a partial representation (L, K), and let q be a monotone Boolean query over the single relation name V. We say that q is well-defined given the partial representation (L, K), if for any two probabilistic relations $PV = (W, \mathbf{P})$ and $PV' = (W', \mathbf{P}')$ that satisfy the partial representation (L, K), and that have identical tuple probabilities*[7]*, the following holds: $\mathbf{P}(q) = \mathbf{P}'(q)$.*

Thus, q is well defined iff $\mathbf{P}(q)$ depends only on the marginal tuple probabilities $\mathbf{P}(t)$ (which we know), and not on the entire distribution $\mathbf{P}$ (which we don't know). We will give now a necessary and sufficient condition for q to be well defined, but first we need to introduce two notions: intertwined tuples, and a set of critical tuples.

[7] $\forall t\ \mathbf{P}(t) = \mathbf{P}'(t)$

Definition 4.16 *Let (L, K) be a partial representation of a view V. Let t, t' be two ground tuples of the same arity as V. We call t,t'* intertwined *if $t.L = t'.L$ and $t.K \neq t'.K$.*

Next, we generalize a critical tuple (see Definition 4.12) to a set of critical tuples. Let $Inst = \mathcal{P}(Tup)$ be the set of (conventional) database instances over the set of ground tuples Tup. To each Boolean query q we associate the numerical function $f_q : Inst \rightarrow \mathsf{R}$:

$$f_q(I) \quad = \quad \begin{cases} 1 & \text{if } q(I) \text{ is true} \\ 0 & \text{if } q(I) \text{ is false} \end{cases}$$

Definition 4.17 *Let $f : Inst \rightarrow \mathsf{R}$ be a numerical function on instances. The* differential *of f w.r.t. a set of tuples $S \subseteq Tup$ is the numerical function $\Delta_S f : Inst \rightarrow \mathsf{R}$ defined as follows:*

$$\begin{aligned} \Delta_\emptyset f(I) &= f(I) \\ \Delta_{\{t\} \cup S} f(I) &= \Delta_S f(I) - \Delta_S(f(I - \{t\})) \quad \textit{if } t \notin S \end{aligned}$$

Definition 4.18 *A set of tuples $C \subseteq Tup$ is critical for f if there exists an instance I s.t. $\Delta_C f(I) \neq 0$. A set of tuples C is critical for a Boolean query q if it is critical for f_q.*

We can now state the main result in this section:

Theorem 4.19 *Let V be a view with a partial representation (L, K).*

1. *A monotone Boolean query q over V is well defined iff for any two intertwined tuples t, t' the set $\{t, t'\}$ is not critical for q.*
2. *The problem* "given a Boolean conjunctive query q over the view V, decide whether q is well defined" *is in Π_2^p. Moreover, there exists a view V and partial representation (L, K) for which this problem is Π_2^p hard.*

Thus, in order to evaluate a query q using a view V with partial representation (L, K) one proceeds as follows. First, we check if q is well defined, by checking if it has no pair of intertwined, critical tuples: this is a Π_2^p-complete problem in the size of the query q. Second, if this holds, then we evaluate q over V by assuming V is a BID table with key attributes L; or, alternatively, we may assume a BID table with key attributes K. The well-definedness condition ensures that we obtain the same answer over any of these two BID tables as over the view V.

In the rest of the section we prove Theorem 4.19, and for that we give a definition, and three lemmas.

Definition 4.20 *Let $T \subseteq Tup$ be a set of tuples. The* restriction *of a numerical function f to T is: $f^T(I) = f(I \cap T)$. Similarly, the restriction of a Boolean query q to T is: $q^T(I) = q(I \cap T)$.*

The first lemma establishes some simple properties of critical sets of tuples. Note that a set of tuples C is not critical for f iff $\Delta_C f = 0$, meaning $\forall I : \Delta_C f(I) = 0$.

Lemma 4.21 *Let C be a set of tuples and suppose $\Delta_C f = 0$. Then:*

1. *For any set of tuples $D \supseteq C$, $\Delta_D f = 0$.*
2. *For any set of tuples S, $\Delta_C \Delta_S f = 0$.*
3. *For any set of tuples T, $\Delta_C f^T = 0$.*

Proof: (1): we show that $\Delta_D f(I) = 0$ by induction on the size of D. If $D = C$ then it follows from the assumption that $\Delta_C f = 0$. We show this for $D \cup \{t\}$, where $t \notin D$: $\Delta_{D\cup\{t\}}(I) = \Delta_D(I) - \Delta_D(I - \{t\}) = 0 - 0 = 0$. (2): $\Delta_C \Delta_S f(I) = \Delta_{S\cup C} f(I)$, and the latter is 0 by the previous claim. (3): $\Delta_C f^T(I) = \Delta_C f(I \cap T) = 0$, because $\Delta_C f = 0$. □

The second lemma gives a series expansion for any numerical function $f : \mathit{Inst} \to \mathsf{R}$, in terms of its differentials.

Lemma 4.22 *For any set of tuples $T \subseteq Tup$:*

$$f = \sum_{S \subseteq T} \Delta_S f^{Tup-(T-S)} \tag{1}$$

As a consequence:

$$f = \sum_{S \subseteq Tup} \Delta_S f^S \tag{2}$$

Equation (1) can be written equivalently as $f(I) = \sum_{S\subseteq T} \Delta_S f(I - (T - S))$. For example, by setting $T = \{t\}$ or $T = \{t_1, t_2\}$ in Eq.(1) we obtain:

$$\begin{aligned} f(I) &= f(I - \{t\}) + \Delta_t f(I) \\ f(I) &= f(I - \{t_1, t_2\}) + \Delta_{t_1} f(I - \{t_2\}) + \Delta_{t_2} f(I - \{t_1\}) + \Delta_{t_1,t_2} f(I) \end{aligned}$$

Proof: (Sketch) We prove (1) by induction on the size of the set T. The first example above shows the base case. Assuming $s \notin T$, we can split the sum over $S \subseteq T \cup \{t\}$ into to sums: one iterating over S where $S \subseteq T$ and the other iterating over $S \cup \{t\}$ where $S \subseteq T$:

$$\begin{aligned} &\sum_{S\subseteq T\cup\{t\}} \Delta_S f^{Tup-(T\cup\{t\}-S)}(I) = \\ &= \sum_{S\subseteq T} \Delta_S f^{Tup-(T\cup\{t\}-S)}(I) + \sum_{S\subseteq T} \Delta_{S\cup\{t\}} f^{Tup-(T-S)}(I) \\ &= \sum_{S\subseteq T} \Delta_S f^{Tup-(T\cup\{t\}-S)}(I) + \sum_{S\subseteq T} \Delta_S f^{Tup-(T-S)}(I) - \sum_{S\subseteq T} \Delta_S f^{Tup-(T-S)}(I - \{t\}) \\ &= \sum_{S\subseteq T} \Delta_S f^{Tup-(T-S)}(I) = f(I) \end{aligned}$$

The last identity holds because $f^{Tup-(T\cup\{t\}-S)}(I) = f^{Tup-(T-S)}(I - \{t\})$. This completes the proof of (1). To prove (2) we set $T = Tup$ in (1). □

For the third lemma, we fix the partial representation (L, K) of the view.

Definition 4.23 *A set of ground tuples T is non-intertwined, or NIT, if $\forall t, t' \in T$, t and t' are not intertwined. In other words: $\forall t, t' \in T$, either $t.L \neq t'.L$ or $t.K = t'.K$.*

Lemma 4.24 *Let (L, K) be a partial representation of a view V, and let q be a monotone Boolean query over V. Assume q has no critical pairs of intertwined tuples, and let T be a NIT set of tuples. Then q^T is well defined given the partial representation (L, K) of the view V.*

Proof: A *minterm* for q^T is a minimal instance J s.t. $q^T(J)$ is true; that is, J is a set of tuples s.t. $q^T(J)$ is true and forall $J' \subseteq J$, if $q^T(J')$ is true then $J = J'$. Denote M the set of all minterms for q^T. Obviously, each minterm for q^T is a subset of T. Since q^T is monotone (because q is monotone), it is uniquely determined by M:

$$q^T(I) \;=\; \bigvee_{J \in \mathsf{M}} (J \subseteq I)$$

In other words, q^T is true on an instance I iff the set of tuples I contains a minterm J. Denote r^J the Boolean query $r^J(I) = (J \subseteq I)$, we apply the inclusion-exclusion formula to derive:

$$\mathbf{P}(q^T) \;=\; \mathbf{P}(\bigvee_{J \in \mathsf{M}} r^J) = \sum_{N \subseteq \mathsf{M}, N \neq \emptyset} (-1)^{|N|} \mathbf{P}(r^{\bigcup N})$$

Finally, we observe that for each $N \subseteq \mathsf{M}$, the expression $\mathbf{P}(r^{\bigcup N})$ is well defined. Indeed, the set $J = \bigcup N$ is the union of minterms in N, thus it is a subset of T, hence it is a NIT set. If $J = \{t_1, t_2, \ldots\}$, the query r^J simply checks for the presence of all tuples $t_1, t_2, \ldots$; in more familiar notation $\mathbf{P}(r^J) = \mathbf{P}(t_1 t_2 \cdots)$. If the set J contains two disjoint tuples ($t_i.K = t_j.K$) then $\mathbf{P}(t_1 t_2 \cdots) = 0$. Otherwise, it contains only independent tuples ($t_i.L \neq t_j.L$), hence $\mathbf{P}(t_1 t_2 \cdots) = \mathbf{P}(t_1)\mathbf{P}(t_2) \cdots$ In either cases it is well-defined and, hence, so is $\mathbf{P}(q^T)$. $\square$

We now prove Theorem 4.19

Proof: Part 1: We start with the "only if" direction. Let q be well defined, and assume it has a pair t, t' of intertwined, critical tuples. By definition there exists an instance I s.t. $f_q(I) - f_q(I - \{t\}) - f_q(I - \{t'\}) + f_q(I - \{t, t'\}) \neq 0$. Since q is monotone it follows that f_q is monotone, hence $q(I) = \textit{true}$, $q(I - \{t, t'\}) = \textit{false}$, and either $q(I - \{t\}) = q(I - \{t'\}) = \textit{false}$ or $q(I - \{t\}) = q(I - \{t'\}) = \textit{true}$. Without loss of generality, assume that $q(I - \{t\}) = q(I - \{t'\}) = \textit{false}$. Then we define two probabilistic databases $PV = (W, \mathbf{P})$ and $PV' = (W, \mathbf{P}')$ as follows. Each has four possible worlds: $I, I - \{t\}, I - \{t'\}, I - \{t, t'\}$. In PV these worlds are assigned probability $\mathbf{P} = (0.5, 0, 0, 0.5)$, respectively; here, t_1 and t_2 are positively correlated. In PV', all worlds are assigned probability 0.25 *i.e.* the two tuples are independent. Observe that in both cases, the marginal probability of any tuple is the same, $\mathbf{P}(t) = \mathbf{P}(t') = 0.5$ and all other tuples have probability 1. Then we have $\mathbf{P}(q) = 0.5$ and $\mathbf{P}'(q) = 0.25$, contradicting the assumption that q is well-defined.

Next we prove the "if" part, so assume q has no pair of intertwined, critical tuples. The basic plan is this. Suppose an instance I contains two intertwined tuples t, t' (hence we don't know their correlations). Write $q(I) = q(I - \{t, t'\}) + \Delta_t q(I - \{t'\}) + \Delta_{t'} q(I - \{t\})$ (because $\Delta_{t,t'} q = 0$). Thus, we can "remove" t or t' or both from I and get a definition of q on a smaller instance, and by repeating this process we can eliminate all intertwined tuples from I, then we apply Lemma 4.24

Formally, let q be a monotone Boolean query that has no critical pair of intertwined tuples for (L, K). Let $PV = (W, \mathbf{P})$ be a probabilistic database that satisfies the partial representation (L, K), and let Tup be the set of possible tuples in PV. We expand f_q using Lemma 4.22:

$$\begin{aligned} f_q &= \sum_{T \subseteq Tup} \Delta_T f_q^T \\ &= \sum_{T \subseteq Tup : T \text{ is NIT}} \Delta_T f_q^T \\ &= \sum_{T \subseteq Tup : T \text{ is NIT}} \sum_{S \subseteq T} (-1)^{|S|} f_q^{T-S} \end{aligned} \tag{3}$$

The first line is Eq.(2) in Lemma 4.22. To show the second line, we start from the fact that $\Delta_{\{t,t'\}} f_q = 0$ when t, t' are intertwined tuples, because we assumed that q has no critical pair of intertwined tuples. Then, every set T that is not NIT can be written as $T = \{t, t'\} \cup T'$ where t, t' are two intertwined tuples. We apply Lemma 4.21 twice: $\Delta_{\{t,t'\}} f_q = 0$ implies $\Delta_{\{t,t'\}} f_q^T = 0$, which further implies $\Delta_T f_q^T = 0$. Thus, the only terms in the first line that are non-zero are those that correspond to sets T that are NIT: this is what the second line says. Finally, the last line is the direct definition of Δ_T.

Next we apply the expectation on both sides of Eq.(3), and use the linearity of expectation plus $\mathbf{P}(q) = E[f_q]$:

$$\begin{aligned} \mathbf{P}(q) = E[f_q] &= \sum_{T \subseteq Tup : T \text{ is NIT}} \sum_{S \subseteq T} (-1)^{|S|} E[f_q^{T-S}] \\ &= \sum_{T \subseteq Tup : T \text{ is NIT}} \sum_{S \subseteq T} (-1)^{|S|} \mathbf{P}(q^{T-S}) \end{aligned}$$

The claim of the theorem follows from that fact that by Lemma 4.24 each expression $\mathbf{P}(q^{T-S})$ is well defined.

We now prove Part 2, by providing a reduction from the problem *"given a conjunctive query q and a tuple t check whether t is critical for q"*. Assume w.l.o.g. that the query and the tuple are over a vocabulary with a single relation symbol (namely V). If not, we rename relation symbols by padding them so that they have the same arities, then adding constants: for example if the vocabulary is $R_1(A, B), R_2(C, D, E, F), R_3(G, H, K)$, then we will rewrite both the query and the tuple using a single relation symbol V of arity 5 (the largest of the three arities plus one), and replace $R_1(x, y)$ with $V(x, y, a, a)$, replace

$R_2(x, y, z, u)$ with $V(x, y, z, u, b)$, and replace $R_3(x, y, z)$ with $V(x, y, z, c, c)$, where a, b, c are fixed constants.

Thus, we have a query q and a ground tuple t, over a single relation symbol V of arity k, in particular $t = V(c_1, \ldots, c_k) = V(\bar{c})$. We want to check whether t is a critical tuple for q. We reduce this problem to the problem of checking whether some query q' is well defined over some view V'; the new view will have arity $k+1$. Fix two distinct constants a, b. The new query q' is obtained from q by replacing every subgoal $V(x, y, \ldots)$ with $V'(a, x, y, \ldots)$, and by adding the constant subgoal $V(b, c_1, \ldots, c_k)$. Thus, queries q and q' look like this:

$$\begin{aligned} q &= V(\bar{x}_1), V(\bar{x}_2), \ldots, V(\bar{x}_m) \\ q' &= V'(a, \bar{x}_1), V'(a, \bar{x}_2), \ldots, V'(a, \bar{x}_m), V'(b, \bar{c}) \end{aligned}$$

Consider the partial representation (L, K) for V', where $L = \mathit{Attr}(V)$ and $K = \mathit{Attr}(V')$. Recall that q' is well defined over this partial representation, iff it has no pairs of intertwined, critical tuples. Thus, to prove the hardness claim it suffices to show that the following two statements are equivalent:

1. There exists two intertwined, critical tuples for q'.

2. t is a critical tuple for q.

We start by showing 1 implies 2. Two tuples t_1, t_2 are intertwined iff they agree on the L attributes, $t_1.L = t_2.L$, and disagree on the K attributes, hence $t_1.A \neq t_2.A$, where $A = \mathit{Attrs}(V') - L$ is the extra attribute that was added to V'. On the other hand, if the set $\{t_1, t_2\}$ is also critical for q', then $t_1.A = a$ and $t_2.A = b$ (since q' only inspects tuples that have $A = a$ or $A = b$), and, moreover, $t_1.L = t_2.L = \bar{c}$ (since the only tuple with $A = b$ that is critical for q' is $V'(b, \bar{c})$). Let I' be an instance that witnesses the fact that $\{t_1, t_2\}$ is doubly critical:

$$\begin{aligned} 0 \neq \Delta_{t_1,t_2} f_{q'}(I') &= f_{q'}(I') - f_{q'}(I' - \{t_1\}) - f_{q'}(I' - \{t_2\}) + f_{q'}(I' - \{t_1, t_2\}) \\ &= 1 - f_{q'}(I' - \{t_1\}) - 0 + 0 \end{aligned}$$

We used the fact that $f_{q'}(I') = 1$ (otherwise, if $f_{q'}(I') = 0$ then $\Delta_{t_1,t_2}(f_{q'}(I')) = 0$), which implies that $t_2 = V'(b, \bar{c}) \in I'$. Thus, we have $q'(I') = \mathit{true}$ and $q'(I' - \{t_1\}) = \mathit{false}$. We construct from here an instance I such that $q(I) = \mathit{true}$ and $q(I - \{t\}) = \mathit{false}$: indeed, take $I = \{V(\bar{d}) \mid V(a, \bar{d}) \in I'\}$, it obviously satisfies this property. Thus, I is a witness for t being a critical tuple for q.

To prove 2 implies 1 we use the same argument, in reverse. We start with an instance I such that $q(I) = \mathit{true}$, $q(I - \{t\}) = \mathit{false}$, and define $I' = \{V(a, \bar{d}) \mid V(\bar{d}) \in I\} \cup \{V(b, \bar{c})\}$. It is easy to check that $\Delta_{t_1,t_2} f_{q'}(I') \neq 0$. This completes the proof. □

5. Conclusions

At a superficial look, query evaluation on probabilistic databases seems just a special instance of probabilistic inference, e.g. in probabilistic networks. However, there are specific concepts and techniques that have been used on conventional databases for many years, and that can be deployed to probabilistic databases as well, to scale up query processing to large data instances. We have presented two such techniques in this paper. The first is the separation of the query and the data: we have shown here that by doing so, one can identify queries whose data complexity is #P-hard, and queries whose data complexity is in PTIME. The second is the aggressive use of materialized views (or any previously computed query results): we have shown that by using a materialized view the query complexity can decrease from #P-hard to PTIME, and have described static analysis techniques to derive a partial representation for the view, and to further use it in query evaluation.

REFERENCES

1. S. Abiteboul and P. Senellart. Querying and updating probabilistic information in XML. In *EDBT*, pages 1059–1068, 2006.
2. Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison Wesley Publishing Co, 1995.
3. Sanjay Agrawal, Surajit Chaudhuri, and Vivek R. Narasayya. Automated selection of materialized views and indexes in sql databases. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 496–505. Morgan Kaufmann, 2000.
4. P. Andritsos, A. Fuxman, and R. J. Miller. Clean answers over dirty databases. In *ICDE*, 2006.
5. L. Antova, C. Koch, and D. Olteanu. 10^(10^6) worlds and beyond: Efficient representation and processing of incomplete information. In *ICDE*, 2007.
6. L. Antova, C. Koch, and D. Olteanu. World-set decompositions: Expressiveness and efficient algorithms. In *ICDT*, pages 194–208, 2007.
7. O. Benjelloun, A. Das Sarma, A. Halevy, and J. Widom. ULDBs: Databases with uncertainty and lineage. In *VLDB*, pages 953–964, 2006.
8. N. Dalvi, Chris Re, and D. Suciu. Query evaluation on probabilistic databases. *IEEE Data Engineering Bulletin*, 29(1):25–31, 2006.
9. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, Toronto, Canada, 2004.
10. N. Dalvi and D. Suciu. The dichotomy of conjunctive queries on probabilistic structures. In *PODS*, pages 293–302, 2007.
11. N. Dalvi and D. Suciu. Management of probabilistic data: Foundations and challenges. In *PODS*, pages 1–12, Beijing, China, 2007. (invited talk).
12. A. Das Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working models for uncertain data. In *ICDE*, 2006.
13. Michel de Rougemont. The reliability of queries. In *PODS*, pages 286–291, 1995.
14. A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, pages 588–599, 2004.
15. E. Grädel, Y. Gurevich, and C. Hirsch. The complexity of query reliability. In *PODS*, pages 227–234, 1998.
16. T. Green and V. Tannen. Models for incomplete and probabilistic information. *IEEE Data Engineering Bulletin*, 29(1):17–24, March 2006.
17. R. Greenlaw, J. Hoover, and W. Ruzzo. *Limits to Parallel Computation. P-*

Completeness Theory. Oxford University Press, New York, Oxford, 1995.

18. Alon Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4):270–294, 2001.
19. E. Hung, L. Getoor, and V.S. Subrahmanian. PXML: A probabilistic semistructured data model and algebra. In *ICDE*, 2003.
20. T. Imielinski and W. Lipski. Incomplete information in relational databases. *Journal of the ACM*, 31:761–791, October 1984.
21. R. Karp and M. Luby. Monte-Carlo algorithms for enumeration and reliability problems. In *Proceedings of the annual ACM symposium on Theory of computing*, 1983.
22. G. Miklau and D. Suciu. A formal analysis of information disclosure in data exchange. *J. Comput. System Sci.*, 73(3):507–534, 2007.
23. Christos Papadimitriou. *Computational Complexity.* Addison Wesley Publishing Company, 1994.
24. J. S. Provan and M. O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. Comput.*, 12(4):777–788, 1983.
25. C. Re, N. Dalvi, and D. Suciu. Efficient Top-k query evaluation on probabilistic data. In *ICDE*, 2007.
26. C. Re and D.Suciu. Materialized views in probabilistic databases for information exchange and query optimization. In *Proceedings of VLDB*, 2007.
27. A. Das Sarma, M. Theobald, and J. Widom. Exploiting lineage for confidence computation in uncertain and probabilistic databases. In *ICDE*, pages 1023–1032, 2008.
28. Patrick Valduriez. Join indices. *ACM Transactions on Database Systems*, 12(2):218–246, 1987.
29. L. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8:410–421, 1979.
30. M. van Keulen, A. de Keijzer, and W. Alink. A probabilistic XML approach to data integration. In *ICDE*, pages 459–470, 2005.
31. M. Y. Vardi. The complexity of relational query languages. In *Proceedings of 14th ACM SIGACT Symposium on the Theory of Computing*, pages 137–146, San Francisco, California, 1982.