Queries and Materialized Views on Probabilistic Databases*

Nilesh Dalvi  Christopher Ré  Dan Suciu

September 11, 2008

Abstract

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, show how to validated it based on the view definition, then show how to use it during query evaluation.

1 Introduction

Probabilistic database 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.

*This work was partially supported by NSF Grants IIS-0454425, IIS-0513877, IIS-0713576, and a Gift from Microsoft.
The first is the separation between the query and the data: the query is small, the data is large. Following Vardi [29], 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].

The second is the ability to use materialized views. These are queries that have been previously computed and whose results have been stored. Computing the views could have been hard, and the system has spent considerable resources to materialize them. However, once computed, the views can be used to answer queries, if these queries can be answered in terms of the views [18]. Today’s database systems routinely use materialized views on conventional databases: indexes are a special case of materialized views, and so are join-indexes [26], and today’s system can use arbitrary views during query processing [2]. However, in probabilistic databases, views cannot always be materialized efficiently, because it is difficult to represent the correlations between the tuples in the view. One approach to represent those correlations is to use lineage [6]. However, using a view with explicit lineage for each tuple during query evaluation does not make the probabilistic inference problem any easier than expanding the view definition in the query body, and then evaluating the query. The approach we propose is to store only the marginal probabilities in the materialized view, and compute, by static analysis of the view definition, a partial representation of the probabilistic table defined by the view. This partial representation is at the schema level, not at the data level, and captures sufficient information about the correlations in the view to allow some queries to be answered from the view. We describe this approach in Section 4. The main results in this part are from [25], but the presentation has been changed, and some new results have been added that shed further light on the view representation problem.

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 a finite domain: extensions to continuous domains [14] and to XML [19, 1, 28] 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 $\text{Attr}(R_i)$, and a key $\text{Key}(R_i) \subseteq \text{Attr}(R_i)$. Denote $D$ a finite domain of atomic values. And denote $\text{Tup}$ the set of all typed tuples of the form $t = R_i(a_1, \ldots, a_k)$, for some $i = 1, k$ and $a_1, \ldots, a_k \in D$. We denote $\text{Key}(t)$ the tuple consisting of the key attributes in $t$ (hence its arity is $|\text{Key}(R_i)|$). A database instance is any subset $I \subseteq \text{Tup}$ 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 $PDB = (W, P)$ where the set of outcomes is a set of possible worlds $W = \{I_1, \ldots, I_n\}$, and $P$ is a function $P : W \to (0, 1]$ s.t. $\sum_{I \in W} P(I) = 1$.

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 $R(A, B, C, D)$, 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 $R$. 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 $PDB = (W, P)$ a probabilistic database. Then $q(PDB)$ is the following probability distribution on the query’s outputs: $q(PDB) = (W', P')$ where:

\[
W' = \{ q(I) \mid I \in W \} \\
P'(J) = \sum_{I \in W : q(I) = J} P(I)
\]

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 $P(q) = \sum_{I \in W : I \models q} P(I)$: note that the image probability space in this case has only two possible worlds: $q(PDB) = (\{I_0, I_1\}, P')$, where $I_0 = \emptyset$, $I_1 = \{()\}$, and $P'(I_0) = 1 - P(q)$, $P'(I_1) = P(q)$. Thus, for all practical purposes $q(PDB)$ and $P(q)$ are the same, and we will refer only to $P(q)$ when the query $q$ is boolean. A special case of a boolean query is a single tuple $t$, and its marginal probability is $P(t) = \sum_{I \in W : t \in I} P(I)$. Note that $t \neq t'$ and $Key(t) = Key(t')$ implies $P(t, t') = 0$, 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
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, 6, 16, 5, 4], 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 \( \forall t_1, \ldots, t_n \in T, \text{Key}(t_i) \neq \text{Key}(t_j) \) for \( i \neq j \) implies \( P(t_1, \ldots, t_n) = P(t_1) \cdot \ldots \cdot P(t_n) \).

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

**Theorem 2.1** Let \((T, P_0)\) be a BID specification. Then there exists a unique BID probabilistic database \(PDB = (W, P)\) s.t. its set of possible tuples is \(T\) and for all \( t \in T\) its marginal probability \(P(t)\) is equal to \(P_0(t)\).

**Proof:** (Sketch) Let \(PDB = (W, P)\) be a BID probabilistic database whose marginal tuple probabilities are \(P_0\), and let \(I \in W\). Obviously \(I \subseteq T\), and we will show that \(P(I)\) is uniquely defined by \((T, P_0)\) and the independence assumption. For any key \(k \in K\), let \(p_k = P_0(t)\), if there exists \(t \in I\) s.t. \(\text{Key}(t) = k\), and \(p_k = 1 - \sum_{t \in T : \text{Key}(t) = k} P_0(t)\) otherwise. Then \(P(I) = \prod_{k \in K} p_k\). Conversely, define the \(PDB = (\text{Inst}(T), P)\), where \(\text{Inst}(T)\) denotes the set of instances over the tuples \(T\), and \(P\) is defined as above: it is easy to check that this is a probability space \((\sum_{I} P(I) = 1)\), that it is BID, and that its marginal tuple probabilities are given by \(P_0\).

The size of a BID specification \((T, P)\) 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 \(\text{Key}(R_i) = \text{Attr}(R_i)\) 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\(^1\). Several, essentially equivalent, complete representation systems have been discussed in the literature [12, 16, 6]. 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 \(X = \{X_1, \ldots, X_m\}\), and for each variable \(X_i\) fix a finite domain \(\text{Dom}(X_j) = \{0, 1, \ldots, d_j\}\). We consider Boolean formulas \(\varphi\) consisting of Boolean combinations of atomic predicates of the form \(X_j = v\), where \(v \in \text{Dom}(X_j)\). Define a constant c-table to be conventional relation \(R\), where each tuple \(t_i\) is annotated with a Boolean formula \(\varphi_i\), called the lineage of \(t_i\). 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 \(\theta\) assigns each variable \(X_j\) to a value \(\theta(X_j) \in \text{Dom}(X_j)\), and we write \(\theta(R) = \{t_i \mid \theta(\varphi_i) = \text{true}\}\).

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

A BID database \(PDB = (T, P)\) 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 \(X = \{X_1, \ldots, X_m\}\) (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 \(\text{Dom}(X_j) = \{0, 1, \ldots, d_j\}\) and annotate the tuple \(t_{ji}\) with the Boolean expression \(X_j = i\). Finally, for each \(j\) define the probability space \((\text{Dom}(X_j), P_j)\) by setting \(P_j(i) = P(t_{ji})\) for \(i > 0\) and \(P_j(0) = 1 - \sum_{i} P(t_{ji})\).

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 \(X\) 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 \(X\). We illustrate query lineage this with a simple example.

**Example 2.1** Consider the schema \(R(\bar{A}, B)\), \(S(R)\), and consider seven possible tuples, four in \(R\) and three in \(S\):

<table>
<thead>
<tr>
<th>(\bar{A})</th>
<th>(B)</th>
<th>(T) :</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a_1)</td>
<td>(b_1)</td>
<td>(X_1 = 1)</td>
</tr>
<tr>
<td>(a_1)</td>
<td>(b_2)</td>
<td>(X_1 = 2)</td>
</tr>
<tr>
<td>(a_2)</td>
<td>(b_1)</td>
<td>(X_2 = 1)</td>
</tr>
<tr>
<td>(a_2)</td>
<td>(b_3)</td>
<td>(X_2 = 2)</td>
</tr>
</tbody>
</table>

\(\bar{A}\) : \(a_1 a_2\) \(B\) : \(b_1 b_2 b_3\) \(X\) : \(X_1 X_2\) \(Y\) : \(Y_1 Y_2 Y_3\)

Here there are \(3 \cdot 3 \cdot 8 = 72\) possible worlds (not all \(2^7\) subsets form a world because \(A\) must be a key in \(R\)). Consider the Boolean query \(h_2 \equiv R(x, y), S(y)\).

\(^1\)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.
Then $\phi_q$ is:

$$\phi_q = (X_1 = 1) \vee (Y_1 = 1) \vee (X_1 = 2) \vee (Y_2 = 1) \vee (X_2 = 1) \vee (Y_1 = 1) \vee (X_2 = 2) \vee (Y_3 = 1)$$

3 Query Evaluation on Probabilistic Databases

In this section we summarize the results on query evaluation from [11]. We omit the proofs, since they are already given in [11].

We study here the following problem. Given a Boolean query $q$ and a BID $PDB = (T, P)$, 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 [29]: fix $q$, and study the complexity of $P(q)$ as a function of the input $PDB$. The probabilities are assumed to be rational numbers.

When $q$ is expressed in FO, then its lineage $\phi_q$ is a Boolean formula whose size is polynomial in the size of the set of possible tuples $T$. Moreover, if $q$ is a conjunctive query, then $\phi_q$ is a DNF formula of polynomial size in $T$, therefore,
any upper bounds for computing the probability of a boolean formula \( P(\varphi_q) \) become upper bounds for computing a query probability \( P(q) \). Thus:

**Theorem 3.1** (1) Computing \( P(\varphi) \) for a Boolean expression \( \varphi \) is in \#P \[27\]. It follows that for any query \( q \) in FO, the problem “given a BID, compute \( P(q) \)” is in \#P. (2) Computing \( P(\varphi) \) for a DNF formula \( \varphi \) has a FPTRAS \[21\]. It follows that for any conjunctive query \( q \) the problem “given a BID, compute \( P(q) \)” 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 \( \varphi \), let \( \#\varphi \) denote the number of satisfying assignments for \( \varphi \). Valiant \[27\] has shown that the problem: given \( \varphi \), compute \( \#\varphi \), is \#P-complete.

The statement above “computing \( P(\varphi) \) is in \#P” means the following: there exists a function \( F \) over the input probabilities \( P(X_1), \ldots, P(X_n) \) (which are rational numbers) s.t. (a) \( F \) can be computed in \( \text{PTIME} \) in \( n \), and (b) the problem “compute \( F \cdot P(\varphi) \)” is in \#P. For example, in the case of a uniform distribution where \( P(X_i) = 1/2 \) and all variables are independent, then we take \( F = 2^n \), and \( 2^n P(\varphi) = \#\varphi \), hence computing \( F \cdot P(\varphi) \) is in \#P.

### A Dichotomy for Queries without Self-joins

We now establish the following dichotomy for conjunctive queries without self-joins: computing \( P(q) \) is either \#P-hard or is in \( \text{PTIME} \) in the size of the database \( \text{PDB} = (T, P) \). 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 \( P(q) \) is \#P-hard in the size of the database:

\[
\begin{align*}
h_1 &= R(x), S(x, y), T(y) \\
h_2^+ &= R_1(x, y), \ldots, R_k(x, y), S(y) \\
h_3^+ &= R_1(x, y), \ldots, R_k(x, y), S_1(x, y), \ldots, S_m(x, y)
\end{align*}
\]

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:

\[
\begin{align*}
h_2 &= R(x, y), S(y) \\
h_3 &= R(x, y), S(x, y)
\end{align*}
\]

\(^2\text{FPTRAS stands for fully poly-time randomized approximation scheme. More precisely: there exists a randomized algorithm } A \text{ with inputs } \varphi, \varepsilon, \delta, \text{ which runs in polynomial time in } |\varphi|, 1/\varepsilon, \text{ and } 1/\delta, \text{ and returns a value } \hat{p} \text{ s.t. } P_A(|\hat{p}/p - 1| > \varepsilon) < \delta. \text{ Here } P_A \text{ denotes the probability over the random choices of the algorithm. Grädel et al. show how to extend this to independent probabilities, and we show in the Appendix how to extend it to disjoint-independent probabilities.}\)
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 three (Lemma 3.1). By contrast, the hardness of these three queries is shown directly (by reducing Positive Partitioned 2DNF [24] to $h_1$, and PERMANENT [27] 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 in PTIME. Grädel at al. [13, 15] corrected this and proved that the query $R(x), R(y), S_1(x, z), S_2(y, z)$ 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 $P(q)$ 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 $g = R(x, a, y, x, z)$, $Vars(g) = \{x, y, z\}$, $KVars(g) = \{x, y\}$. For $x \in Vars(q)$, let $sg(x) = \{g \mid g \in Sg(q), x \in KVars(g)\}$. Given a database $PDB = (T, P)$, $D$ is its active domain.

Algorithm 3.1 computes $P(q)$ by recursion on the structure of $q$. If $q$ consists of connected components $q_1, q_2$, then it returns $P(q_1)P(q_2)$: this is correct since $q$ has no self-joins, e.g. $P(R(x), S(y, z), T(y)) = P(R(x))P(S(y, z), T(y))$. If some variable $x$ occurs in a key position in all subgoals, then it applies the independent-project rule: e.g. $P(R(x)) = 1 - \prod_{a \in D} (1 - P(R(a)))$ is the probability that $R$ is nonempty. For another example, we apply an independent project on $x$ in $q = R(x, y), S(x, z, T)$: this is correct because $q[a/x]$ and $q[b/x]$ are independent events whenever $a \neq b$. 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 $q = R(x, y), S(c, d, x)$, 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:
Algorithm 3.1 Safe-Eval

Input: query $q$ and database $PDB = (T, P)$
Output: $P(q)$

1: **Base Case:** if $q = R(\bar{a})$
   return if $R(\bar{a}) \in T$ then $P(R(\bar{a}))$ else 0

2: **Join:** if $q = q_1, q_2$ and $Vars(q_1) \cap Vars(q_2) = \emptyset$
   return $P(q_1)P(q_2)$

3: **Independent project:** if $sg(x) = Sg(q)$
   return $1 - \prod_{a \in D}(1 - P(q[a/x]))$

4: **Disjoint project:** if $\exists g(x \in Vars(g), KVars(g) = \emptyset)$
   return $\sum_{a \in D} P(q[a/x])$

5: **Otherwise:** FAIL

$q = R(\bar{x}), S(x, y), T(y), U(u, y), V(\bar{a}, u)$

$P(q) = \sum_{b \in D} P(R(\bar{x}), S(x, y), T(y), U(\bar{b}, y), V(\bar{a}, b))$

$= \sum_{b \in D} P(R(\bar{x}), S(x, y), T(y), U(\bar{b}, y))P(V(\bar{a}, b))$

$= \sum_{b \in D} \sum_{c \in D} P(R(\bar{x}), S(x, c), T(c), U(\bar{b}, c))P(V(\bar{a}, b))$

$= \sum_{b \in D} \sum_{c \in D} P(R(\bar{x}), S(x, c))P(T(c))P(U(\bar{b}, c))P(V(\bar{a}, b))$

$= \sum_{b \in D} \sum_{c \in D} (1 - \prod_{d \in D}(1 - P(R(d))P(S(d, c)))) \cdot P(T(c))P(U(\bar{b}, c))P(V(\bar{a}, b))$

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 3.1** For any safe query $q$, the algorithm computes correctly $P(q)$ 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. [3] 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:

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

The intuition is that if $q \Rightarrow q'$ then evaluating $P(q')$ can be reduced in polynomial time to evaluating $P(q)$. 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 3.1** 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:

$$q = R(x), R'(x), S(x, y, y), T(y, z, b)$$

$$\Rightarrow R(x), S(x, y, y), T(y, z, b)$$

$$\Rightarrow^* R(x), S(x, y), T(y) = h_1$$

$$q' = R(x, y), S(y, z), T(z, x), U(y, x)$$

$$\Rightarrow R(x, y), S(y, x), T(x, x), U(y, x)$$

$$\Rightarrow^* R(x, y), S(y, x), U(y, x) = h_3^+$$

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 3.2** $h_1, h_2^+, h_3^+$ are the only final queries.

This implies immediately the dichotomy property:

**Theorem 3.3** 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.

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 $q = R(x,y), R(y,z)$ 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 $q[a/y] = R(x,a), R(a,z)$ and $q[b/y] = R(x,b), R(b,z)$ 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, see. 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 $R$ and conjunctive query $q$ without self-joins over $R$, decide whether $q$ is safe. 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, $\text{Key}(R) = \text{Attr}(R)$, hence $sg(x)$ becomes: $sg(x) = \{ y \mid x \in Vars(g) \}$.

**Definition 3.1** 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 $q = R(x), S(x,y)$ 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 3.2** Let $SG$ be a binary relation name. We represent a pair $R, q$, where $R$ is a relational schema for an independent database and $q$ a conjunctive query without self-joins, as an instance over $SG$, as follows. The constants are $R \cup Vars(q)$, 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 $R, q$, $q$ is unsafe” can be expressed in FO over the vocabulary $SG$.

---

4 For a fixed $R$ there are only finitely many queries without self-joins: this is the reason why $R$ 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 $q$. 

11
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 in the Appendix:

**Proposition 3.3** AGAP is reducible in LOGSPACE to the following problem: given a schema $R$ and a query $q$ without self-joins, check if $q$ is safe. In particular, the latter is PTIME-hard.

## 4 Materialized Views on Probabilistic Databases

The main results in this section are from [25], but the presentation is quite different, and we have added several new results to shed more light on materialized views. In this section we include most of the proofs.

Materialized views are a widely used today to speedup query evaluation. Early query optimizers used materialized views that were restricted to indexes (which are simple projections on the attributes being indexed) and join indexes [26]; modern query optimizers can use arbitrary materialized views [2].

Materialized views on probabilistic databases can make dramatic impact. Suppose we need to evaluate a Boolean query $q$ on a BID probabilistic database, and assume $q$ is unsafe. Normally, the only available technique is Luby and Karp’s FPTRAS, and its performance is two orders of magnitudes or more worse than a safe plan. 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.

The major challenge is how to represent 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 [7]), 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 need to store their lineage.

- We compute a partial representation for the view. This is a schema-level (i.e. data independent) information about the independence/disjointness/correlations of the tuples in the view, obtained from static analysis on the view definition.
• To evaluate a query \( q \) we first check if the query can be rewritten in terms of the view (using standard techniques [18]), then we check if the rewritten query is well defined based on the partial representation. If both are true, then we evaluate \( q \) on using the view as any base tables, with marginal tuple probabilities.

Let \( e_1, \ldots, e_n \) be \( n \) events over a probabilistic space. They are called independent if \( \mathbf{P}(e_1 \land \cdots \land e_n) = \mathbf{P}(e_1) \cdots \mathbf{P}(e_n) \); they are called 2-way independent, or 2-independent, if for all \( i, j \), \( \mathbf{P}(e_i \land e_j) = \mathbf{P}(e_i)\mathbf{P}(e_j) \).

**Definition 4.1** Let \( V \) be a probabilistic database consisting of a single table.

• Let \( L \subseteq \text{Attr}(V) \). We say that \( V \) is \( L \)-block independent if for all \( n > 1 \) and any tuples \( t_1, \ldots, t_n \in V \) with distinct values for the attributes \( L \) (i.e. \( t_i.\bar{L} \neq t_j.\bar{L} \), for all \( i \neq j \)) are independent.

• Let \( K \subseteq \text{Attr}(V) \). We say that \( V \) is \( K \)-block disjoint if for any two tuples \( t, t' \), if \( t.\bar{K} = t'.\bar{K} \) then \( t, t' \) are disjoint (i.e. \( \mathbf{P}(t \land t') = 0 \)). Equivalently, \( K \) is a key in each possible world of \( V \).

Obviously, if \( V \) is \( L \)-block independent and \( L \supseteq L' \) then \( V \) is also \( L' \)-block independent, and any \( V \) is \( \emptyset \)-block independent. Our objective is to find a large set \( L \) s.t. \( V \) is \( L \)-block independent. Similarly, if \( V \) is \( K \)-block disjoint and \( K \subseteq K' \) then \( V \) is also \( K' \)-block independent (a superset of a key is also a key). In particular, by increasing \(^6\) \( K \) we can always ensure that \( L \subseteq K \).

**Definition 4.2** Let \( V \) a probabilistic database consisting of a single table. A partial representation for \( V \) consists of a pair \((L, K)\) s.t. \( L \subseteq K \subseteq \text{Attr}(V) \) and \( V \) is \( L \)-block independent and \( K \)-block disjoint. If \( L = K \) then we call it a total representation, or a representation for short.

Thus, our goal is to find a “good” partial representation for \( V \), i.e. with a large \( L \) and a small \( K \). As we shall see, it is always possible to find a largest \( L \), but not always possible to find a smallest \( K \), since in general there may be several minimal keys. In that case we will keep several partial representations, \((L, K_1), (L, K_2), \ldots \). One should think of partial representations as being similar to functional dependencies: we collect all we can, and use them to statically analyze a query that refers to \( V \).

\(^6\)By increasing \( K \) we do change the partial representation, while if we decreased \( L \) then we would make it a strictly weaker representation. To see the former, note that whenever \( V \) is \( L \)-block independent and \( K \)-block disjoint, then the functional dependency \( K \rightarrow L \) holds on the set of possible tuples in \( T \); because if \( t, t' \in T \) are such that \( t.\bar{K} = t'.\bar{K} \) then they are disjoint, hence cannot also be independent (assuming the tuples in \( T \) have non-zero probability), hence \( t.\bar{L} = t'.\bar{L} \). It follows that the set of pairs of tuples \( t, t' \) for which \( t.\bar{K} = t'.\bar{K} \) is the same as the set of pairs of tuples for which \( t.(\bar{L} \cup K) = t'.(\bar{L} \cup K) \). Thus, the partial representation \((L, K)\) makes exactly the same statements on the possible tuples \( T \) as the partial representation \((L, L \cup K)\). In contrast, if we decrease \( L \) to \( L \cap K \), then we lose information about the independence of certain tuples.
Note that \( V \) is a \( BID \) table iff it has a total representation: in this case \( \text{Key}(V) = L = K \).

We start with a negative result: with our current definition, there is no largest \( L \). This can be shown from the following example.

**Example 4.1** Consider a probabilistic table \( V(A, B, C) \) with four possible tuples:

\[
\begin{array}{ccc|c}
A & B & C & t_1 \\
\hline
a & b & c & t_1 \\
a & b' & c & t_2 \\
a & b' & c' & t_3 \\
\end{array}
\]

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 \( \mathbb{P}(t_1) = \mathbb{P}(t_3) = 1/2 \) and \( \mathbb{P}(t_1,t_2) = \mathbb{P}(t_1,t_3) = \mathbb{P}(t_2,t_3) = 1/4 \). \( V \) 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: \( \mathbb{P}(t_1,t_2,t_3) = 0 \). This shows that there is no largest set \( L \): both \( AB \) and \( AC \) are maximal.

Thus, for a general probabilistic databases we cannot hope to have a best partial representation. However, we can prove the following weaker result, which we will use later:

**Lemma 4.1** For a set \( L \subseteq \text{Attr}(V) \) 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. Then, any probabilistic table \( V \) has a largest set \( L \) s.t. \( V \) is \( L \)-block 2-independent.

**Proof:** It suffices to prove the following. Let \( L_1, L_2 \subseteq \text{Attr}(V) \) be s.t. \( V \) is \( L_i \)-block 2-independent for each \( i = 1,2 \); then, denoting \( L = L_1 \cup L_2 \), \( V \) is \( L \)-block 2-independent. 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.

Continuing Example 4.1 we note that \( V \) is \( ABC \)-block 2-independent, since any two of the tuples \( t_1,t_2,t_3 \) are independent.

### 4.1 Materialized Views Expressed by \( c \)-Tables

Despite Example 4.1, it turns out that \( c \)-tables do admit a largest set \( L \), for an appropriate definition of \( L \)-block independence. Recall that a \( pc \)-table \( V \) consists of two parts: \( V = (CV,P) \), where \( CV \) is a \( c \)-table and \( P \) a product probability space on the set of variables \( \bar{X} \).

**Definition 4.3** Let \( CV \) be a \( c \)-table of arity \( k \).

- Let \( L \subseteq \text{Attr}(CV) \). We say that \( CV \) is \( L \)-block independent if for any \( pc \)-table \( V = (CV,P) \), \( V \) is \( L \)-block independent.
Let $K \subseteq \text{Attr}(CV)$. We say that $CV$ is $K$ block disjoint if for any pc-table $V = (CV, P)$, $V$ is $K$-block disjoint.

In general there is no smallest set $K$ s.t. $V$ is $K$-block disjoint: see Example 4.4 below. On the other hand, we prove the following in this section:

**Theorem 4.1** For any c-table $CV$ there exists a largest set of attributes $L \subseteq \text{Attr}(CV)$ s.t. $CV$ is $L$-block independent.

It is easy to see that every c-table $CV$ has a largest set of attributes $L$ s.t. $CV$ is $L$-block 2-independent: the proof is similar to that of Lemma 4.1. We need to prove, however, that $CV$ is also $L$-block independent.

To prove this, we establish a few simple results. First, consider a Boolean formula $\varphi$ over variables $\bar{X} = \{X_1, \ldots, X_m\}$: the atomic formulas in $\varphi$ are expressions $X_i = v$, where $v \in \text{Dom}(X_i)$. A valuation is a function $\theta : \bar{X} \rightarrow \prod \text{Dom}(X_i)$, and we denote with $\varphi[\theta]$ the truth value of the formula $\varphi$ under the valuation $\theta$.

Let $\varphi_1, \ldots, \varphi_n$ be $n$ formulas over the same sets of variables $\bar{X}$. We say that they are independent if for any product probability space $P$ the events $\varphi_1, \ldots, \varphi_n$ are independent. Similarly, we say that they are 2-way independent if for any probability spaces for its variables, they are 2-way independent events. Theorem 4.1 obviously follows from the following:

**Proposition 4.1** A set of formulas $\varphi_1, \ldots, \varphi_n$ is independent iff it is 2-way independent.

Let’s see first how this implies Theorem 4.1: if $L$ is the largest set of attributes s.t. $V$ is $L$-block 2-independent: then the proposition implies that $V$ is also $L$-block independent, and $L$ is obviously the largest such set.

To prove the proposition we give an alternative characterization of independence in terms of critical variables.

**Definition 4.4** A variable $X_j$ is called a critical variable for $\varphi$ if there exists a valuation $\theta$ for the variables $\bar{X} - \{X_j\}$ and two values $v', v'' \in \text{Dom}(X_j)$ s.t. $\varphi[\theta \cup \{(X_j, v')\}] \neq \varphi[\theta \cup \{(X_j, v'')\}]$.

In other words $X_j$ is a critical variable if there is a choice of values for the other variables for which $X_j$ makes a difference: when $X_j$ changes $v'$ to $v''$, then $\varphi$ changes from false to true. If the expression $\varphi$ does not mention the variable $X_j$ at all, then it is obviously not a critical variable. Conversely, if $X_j$ is not a critical variable for $\varphi$ then one can rewrite $\varphi$ as an expression that does not mention $X_j$. For example, consider $\varphi = X_1 \lor (X_1 \land X_2)$ (where the variables are assumed to be Boolean). Here $X_1$ is a critical variable, but $X_2$ is not: in fact $\varphi$ can be rewritten as $\varphi = X_1$. In general:

**Proposition 4.2** Deciding whether $X_j$ is a critical variable for $\varphi$ is NP-complete.
Membership in NP follows by definition, while hardness follows from the fact that if $\psi$ is a Boolean expression that does not mention $X_j$, then $X_j$ is critical for $\psi \land X_j$ iff $\psi$ is satisfiable.

The connection between independence and critical variables is the following:

**Theorem 4.2** Let $\varphi, \psi$ be two Boolean formulas over the same set of variables $\bar{X}$. Then $\varphi, \psi$ are independent iff they have no common critical variables.

The “if” direction is trivial: if $\varphi, \psi$ use disjoint sets of variables, then they are clearly independent. Before we prove the “only if” direction, we note that the theorem immediately implies Proposition 4.1: if the formulas are 2-way independent, then each $\varphi_i$ uses a set of variables that is disjoint for the variables used by any other $\varphi_j$, hence they are independent.

**Proof:** ("Only if" of Theorem 4.2). The “only if” direction was shown in [22] for the case when all variables $X_j$ are Boolean, i.e. $|\text{Dom}(X_j)| = 2$. We briefly review the proof here. Given a probability spaces $(\text{Dom}(X_j), P_j)$, denote $x_j = P_j(X_j = 1)$, hence $P(X_j = 0) = 1 - x_j$. Then $P(\varphi)$ is a polynomial in the variables $x_1, \ldots, x_m$ where each variable has degree $\leq 1$. (For example, if $\varphi = \neg(X_1 \otimes X_2 \otimes X_3)$ (exclusive or) then $P(\varphi) = x_1x_2(1 - x_3) + x_1(1 - x_2)x_3 + (1 - x_1)x_2x_3 + (1 - x_1)(1 - x_2)(1 - x_3)$, which is a polynomial of degree 1 in $x_1, x_2, x_3$.) The identity $P(\varphi)P(\psi) = P(\varphi \land \psi)$ must hold for any values of $x_1, \ldots, x_m$. If $X_j$ is a common critical variable for $\varphi$ and $\psi$ then the left hand side is a polynomial of degree 2 in $x_j$, while the right hand side has degree 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 $x_{ij} = P(X_j = i)$, for $i = 1, \ldots, d_j$, thus $P(X_j = 0) = 1 - x_{1j} - x_{2j} - \cdots - x_{d_jj}$. As before $P(\varphi)$ is a polynomial of degree 1 in the variables $x_{ij}$ with the additional property that if $i_1 \neq i_2$ then $x_{i_1j}$ and $x_{i_2j}$ cannot appear in the same monomial. We still have the identity $P(\varphi)P(\psi) = P(\varphi \land \psi)$, for all values of the variables $x_{ij}$ (since the identity holds on the open set $x_{ij} \geq 0$ for all $i, j$, and $\sum x_{ij} \leq 1$, for all $j$). If $X_j$ is a critical variable for $\varphi$ then $P(\varphi)$ must have a monomial containing some $x_{ij}$; if it is also critical for $\psi$, then $P(\psi)$ has a monomial containing $x_{ij}$. Hence their product contains $x_{i_1j} \cdot x_{i_2j}$, contradiction. □

### 4.2 Materialized Views Expressed by Conjunctive Queries

The conjunctive queries that we consider in this section may have self-joins, unless we explicitly say that they don’t have self-joins. We fix the schema $\mathcal{R}$ of a BID database, and consider the case when $V$ is defined by a conjunctive query over $\mathcal{R}$. That is, $V : -v$, where $V$ is the name of the view, and $v$ is the conjunctive query defining it.

**Definition 4.5** Let $L \subseteq \text{Attr}(V)$: $V$ is $L$-block independent if for any input BID database $PDB$, the probabilistic table $v(PDB)$ is $L$-block independent.
Let $K \subseteq \text{Attr}(V)$: $V$ is $K$-block independent if for any input BID database PDB, the the probabilistic table $v(PDB)$ is $K$-block disjoint.

We illustrate with three examples.

**Example 4.2** Consider the relational schema $R(C, A)$, $S(C, A, B)$, $T(C, B)$, and the following view:

$$v(z) : = R(z, x), S(z, x, y), T(z, y)$$

Denote $V(Z)$ the schema of the materialized view. Then $V$ is $Z$-block independent: in other words all tuples in $V$ are independent. Note that this query is a hard query, since it reduces to $h_1$ from Theorem 3.2, but it is fully representable as a table where all tuples are independent. That means that we need to pay a high price to materialize the view $V$: the result is the set of all possible tuples in $V$, together with their marginal probabilities. But later we can use $V$ freely in queries, and we know that all tuples are independent. Importantly, we can use $V$ in queries that have RST as a subquery. For example consider the Boolean query $q : = -R(z, x), S(z, x, y), T(z, y), U(z, v)$, where $U(C, D)$ is another relation. Then $q$ is $\#P$-hard, but after rewriting it as $q : = -V(z), U(z, v)$ it becomes a safe query, i.e. it is in PTIME. Thus, by using $V$ to evaluate $q$ we obtain a dramatic reduction in complexity.

**Example 4.3** For a second example, consider the schema $R(A)$, $S(A, B, C)$, and the query:

$$v(x, y, z) : = R(x), S(x, y, z)$$

This query is safe (there are no projections), but is not fully representable. Denoting $V(X, Y, Z)$ the output schema, the best we can say is that $V$ is $X$-block independent, and $XY$-block disjoint. Thus, we know that the tuples $V(a, b, c)$ and $V(a', b, c)$ are independent, and the tuples $V(a, b, c)$, $V(a, b, c')$ are disjoint. But we do not know the correlation between the tuples $V(a, b, c)$ and $V(a, b', c)$. In fact, we cannot answer the Boolean query $q = V(a, b, c), V(a, b', c)$ by examining only the view $V$: to answer $q$ we need to expand it and answer it from the base relations.

**Example 4.4** Finally, consider the schema $R(A, B, C)$, $S(A, C, B)$ and the view:

$$v(x, y, z) : = R(x, y, z), S(x, z, y)$$

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. In practice, in this case we will keep both partial representations $(X, XY)$ and $(X, XZ)$. 

17
Recall that a BID instance $PDB$ is a special case of a (set of) pc-tables. Thus, we can express it as $PDB = (CDB, P)$ separating the c-tables from the probability $P$. This implies that the view $V = v(PDB)$ can be expressed as follows: it has a c-table $v(CDB)$, which can be computed by computing separately the lineage of each output tuple, and it has a probability space $P$ on the variables $\bar{X}$. Both the variables $\bar{X}$ and the product probability $P$ are the same as for $PDB$. From this it follows that $V$ is $L$-block independent if for any input c-tables $CDB$, the c-table $v(CDB)$ is $L$-block independent, and is $K$-block disjoint if for any input c-tables $CDB$, $v(CDB)$ is $K$-block disjoint. The following are immediate:

**Proposition 4.3** (a) $V$ is $L$-block 2-independent iff it is $L$-block independent. 
(b) There exists a largest set $L$ s.t. $V$ is $L$-block independent.

**Proof:** (a) follows immediately from the corresponding fact for c-tables. (b) for each c-table $CDB$, there exists a largest set $L_{CDB}$ s.t. $v(CDB)$ is $L_{CDB}$-block independent. Then the largest set $L$ s.t. $V$ is $L$-block independent is $\bigcap_{CDB} L_{CDB}$ (the intersection ranges over infinitely many CDBs).

Thus, there exists a “best” (largest) choice for the attributes $L$. On the other hand, we know that there is no “best” (smallest) choice for the $K$ attributes, as shown in Example 4.4.

The proof above gives no clue how to actually search for the largest set $L$. To find $L$ we can iterate over all subsets $L \subseteq Attr(V)$, but we still need a criteria to check if for a given set $L$, the view $V$ is $L$-block independent. For that we use the notion of critical tuples, introduced in [22].

Given a Boolean query $q$, a critical tuple is a ground tuple $t$ for one of the relations $R_i$ occurring in $q$ s.t. there exists a (conventional) database instance $I$ s.t. $q(I) \neq q(I \cup \{t\})$. For a simple illustration, consider the Boolean query $q : \neg R(x, x), S(a, x, y)$, 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: $q : \neg R(x, y, z, u), R(x, x, y, y)$. 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. In general:

**Theorem 4.3** [22] The problem: given $q, t$, check whether $t$ is a critical tuple for $q$, is $\Sigma_2^p$-complete.

The problem: given two Boolean queries $q, q'$, check whether they have no common critical tuples is $\Pi_2^p$ complete.

For the first statement, membership in $\Sigma_2^p$ follows from the observation that the size of $I$ can be bounded by the number of variables occurring in $q$ plus the
constants occurring in \( q \) and \( t \). The second statement follows immediately from
the first (hardness follows by taking \( q' = t \)).

There is a strong connection between critical variables of a Boolean formula
and critical tuples of a Boolean query. Let \( CDB \) be a c-table database, and let
\( \varphi \) be the lineage formula for \( q(CDB) \). Then if \( X_j \) is a critical variable for \( \varphi \),
then at least one of the tuples in \( CDB \) that is annotated with an expression
\( X_j = v \) is a critical tuple for \( q \). Conversely, if \( t \) is a critical tuple for \( q \) then one
can find a c-table database \( CDB \) containing \( t \) s.t. the variable \( X_j \) annotating \( t \)
is a critical variable for the lineage formula for \( q(CDB) \).

This implies:

**Proposition 4.4** Let \( q, q' \) be two Boolean queries over a common BID schema
\( \mathcal{R} \). Then the following are equivalent:

- For any input probabilistic database \( q \) and \( q' \) are independent.
- \( q \) and \( q' \) do not have any common critical tuples.

**Corollary 4.4** Checking whether \( q, q' \) are independent for all BID databases is
\( \Pi_2^p \)-complete.

We use this to derive a necessary and sufficient condition for \( V \) to be \( L \)-
block 2-independent, which, as we have shown, implies that \( V \) is also \( L \)-block
independent. Recall that \( k \) is the arity of \( V \) (\( k = |\text{Attr}(V)| \)).

**Proposition 4.5** Let \( V \) be a view defined by a conjunctive query over a BID
schema. For any \( L \subseteq \text{Attr}(V) \), the following two conditions are equivalent:

- \( V \) is \( L \)-block independent.
- \( V \) is \( L \)-block 2-independent.

- For any two ground tuples \( t, t' \in D^k \) s.t. \( t.L \neq t'.L \), the two Boolean
queries \( v(t) \) and \( v(t') \) have no common critical tuples. Here \( v(t) \) denotes
the Boolean query obtained by substituting the head variables in \( v \) with the
tuple \( t \), and similarly \( v(t') \).

Thus, an upper bound on the complexity is \( \Pi_2^p \). It turns out that the problem
is also hard for this class, as shown in the full version of [25].

**Theorem 4.5** Checking whether \( V \) is \( L \)-block independent is \( \Pi_2^p \)-complete.

We can now better understand the set of attributes \( L \) for which \( V \) is \( L \)-block
independent. Any such set consists only of all attributes \( A \) s.t. \( V \) is \( A \)-block
independent; and the maximal set \( L \) is precisely the set of all such attributes
\( A \). Moreover, given an attribute \( A \), checking whether \( V \) is \( A \)-block independent
is precisely the safety test for an independent project on \( A \). We have shown in
Sec. 3 that in the case of queries without self joins, this happens iff \( A \) appears
in a key position in each subgoal of \( v \). On the other hand, if \( v \) is allowed to have self-joins, then checking if a safe project on \( A \) is possible is \( \Pi_2^P \)-complete.

Finally, we briefly comment on how to compute the set \( K \) that describes disjoint tuples. This is simply a key \( K \subseteq Attr(V) \), in the conventional sense, where the relations mentioned in the query \( v \) have explicit keys. There are standard procedures for computing the set of keys in a conjunctive query.

### 4.3 Querying Partially Represented Views

We now turn to an interesting question: given a partial representation \((L, K)\) for a materialized view \( V \), and a query \( q \) that uses the view, check whether \( q \) can be answered from \( V \). Notice that this is orthogonal 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. Here we assume that the rewriting has already been done, thus \( q \) already mentions the view(s). The problem is whether \( q \) is well-defined: we saw in Example 4.4 a case when the query is not well-defined, because the partial representation \((L, K)\) and the marginal tuple probabilities do not uniquely define a probabilistic database for the view.

In this section we restrict the query \( q \) to be over a single view \( V \), and mention no other relations. Thus, \( q \) can perform selections and self-joins over \( V \) only. Our discussion of well-definedness extends immediately to the case when \( q \) is written over multiple views, each with its own partial representation, and even over input BID tables (which for this purpose are views with a complete representation), provided that all views and base tables are independent.

**Definition 4.6** Let \( PV \) be a probabilistic relation of schema \( V \). We write \( PV \models (L, K) \) if \( PV \) is \( L \)-block independent and \( K \)-block disjoint.

**Definition 4.7** Let \( q \) be a Boolean query over the single relation name \( V \). We say that \( q \) is well-defined given the partial representation \((L, K)\), if for all \( PV, PV' \) s.t. \( PV \models (L, K) \), \( PV' \models (L, K) \), and \( \forall t \in P(t) = P'(t) \), \( P(q) = P'(q) \).

Thus, \( q \) is well defined if \( P(q) \) depends only on the marginal tuple probabilities \( P(t) \) (which we know), and not on the entire distribution (which we don’t know). We will give now a necessary and sufficient condition for \( q \) to be well defined. For that we first need some background on numerical functions.

#### 4.3.1 Numerical Functions and Differentials

Recall that we have a fixed domain \( D \) and denote \( Tup \) the set of tuples (over a given relational schema \( R \)) that can be constructed with constants from \( D \). Let \( Inst = P(Tup) \) be the set of instances. A **numerical function** is a function of the form \( f : Inst \rightarrow R \), where \( R \) is the set of reals. A Boolean query is a particular numerical function: \( q(I) = 0 \) if the query is false at \( I \), and \( q(I) = 1 \) when the query is true.
Definition 4.8 Let \( s \in \text{Tup} \) be a ground tuple. The differential of \( f \) w.r.t. \( s \) is:

\[
\Delta_s f(I) = f(I) - f(I - \{s\})
\]

Iterating this definition gives us the differential \( \Delta_S \) for a set of tuples \( S \): for \( s \notin S \) define \( \Delta_{\{s\} \cup S} f = \Delta_s (\Delta_S f) \). Note that the differential of a monotone Boolean query is also a Boolean query, but it is not necessarily monotone; furthermore, the differential of a non-monotone query may take the value \(-1\).

Definition 4.9 A set of tuples \( C \) is critical for \( f \) if \( \exists I \) s.t. \( \Delta_C f(I) \neq 0 \).

This generalizes the previous definition of a critical tuple. Indeed, a tuple \( t \) is critical for a query \( q \) iff the set \( \{t\} \) is critical for the numerical function associated to \( q \).

The following identities can easily be derived:

\[
\begin{align*}
  f(I) &= f(I - \{s\}) + \Delta_s f(I) \\
  f(I) &= f(I - \{s_1, s_2\}) + \Delta_{s_1} f(I - \{s_2\}) + \Delta_{s_2} f(I - \{s_1\}) + \Delta_{s_1, s_2} f(I)
\end{align*}
\]

In order to generalize the latter formula, we introduce another definition:

Definition 4.10 Let \( T \subseteq \text{Tup} \) be a set of tuples. The restriction of \( f \) to \( T \) is:

\[
f_T(I) = f(I \cap T)
\]

In particular \( f = f_{\text{Tup}} \), where \( \text{Tup} \) is the set of all tuples over the domain. If \( q \) is a Boolean query then \( q^T \) is also a Boolean query; if moreover \( q \) is monotone, then \( q^T \) is also monotone.

Proposition 4.6 For any set of tuples \( T \):

\[
f = \sum_{S \subseteq T} \Delta_S f_{\text{Tup} - (T - S)}
\]

In particular, by taking \( T = \text{Tup} \) we obtain \( f = \sum_{S \subseteq \text{Tup}} \Delta_S f^S \).

For any set of tuples \( S \):

\[
\Delta_S f = \sum_{T \subseteq S} (-1)^{|T|} f_{\text{Tup} - T}
\]

The proofs are immediate: the first equation generalizes the two identities above, while the second equation generalizes identities like:

\[
\begin{align*}
  \Delta_s f(I) &= f(I) - f(I - \{s\}) \\
  \Delta_{s_1, s_2} f(I) &= f(I) - f(I - \{s_1\}) - f(I - \{s_2\}) + f(I - \{s_1, s_2\})
\end{align*}
\]

Finally, we state the following (which is easy to check):

Proposition 4.7 (1) If \( C \) is critical for \( \Delta_S f \), then \( C \) is critical for \( f \). (2) If \( C \) is critical for \( f^T \), then \( C \) is critical for \( f \).
4.3.2 The Well-definedness Condition

We call two tuples \( t, t' \) *intertwined* if \( t.L = t'.L \) and \( t.K \neq t'.K \). If two tuples are not intertwined, then they are either independent (when \( t.L \neq t'.L \)), or disjoint (when \( t.K = t'.K \)). Thus, intertwined tuples are correlated in ways that we cannot determine.

**Theorem 4.6** Let \( q \) be a monotone Boolean query over \( V \). Then \( q \) is well defined iff for any two intertwined tuples \( t, t' \) the set \( \{t, t'\} \) is not critical for \( q \) (in other words, \( \Delta_{t,t'} q = 0 \)).

Before proving the theorem we illustrate with an example:

**Example 4.5** Let \( V(A, B, C) \) have the following partial representation: \( L = A \), \( K = AB \). Consider the following queries:

\[
q_1 : \quad V(a, -, -) \\
q_2 : \quad V(-, b, -) \\
q_3 : \quad V(-, -, c)
\]

Of the three, \( q_2 \) is the only query that is well-defined. We first explain intuitively why \( q_2 \) is well-defined. Its value depends only on the tuples of the form \( (a_i, b_j, c) \): these can be partitioned by \( a_i \) into independent sets, while the tuples in each set are disjoint. In other words, \( q_2 \) depends only on a subset of tuples that form a BID table. One can also see that \( q_2 \) has no two intertwined, critical tuples: if \( t_1, t_2 \) is a critical set then each of \( t_1, t_2 \) must be critical, hence they must be of the form \( (a_i, b_j, c) \): but then they are not intertwined (they are either independent \( (t_1.A \neq t_2.A) \) or disjoint \( (t_1.AB = t_2.AB) \), but \( t_1.C \neq t_2.C \)).

In contrast, neither \( q_1 \) nor \( q_2 \) are well-defined. To see this, consider a view \( V \) with two tuples: \( t_1 = (a, b_1, c) \) and \( t_2 = (a, b_2, c) \); these tuples are intertwined, i.e. the correlation of \( t_1 \) and \( t_2 \) is unknown. Further, \( P'[q_1] = P'[q_3] = P[t_1 \lor t_2] \) and so neither \( q_1 \) nor \( q_3 \) is well-defined. They also form a set of critical tuples: denoting \( I = \{t_1, t_2\}, q_1(I) = q_1(I - \{t_1\}) = q_1(I - \{t_2\}) = 1, q_3(I - \{t_1, t_2\}) = 0, hence \( \Delta_{t_1, t_2} q_1(I) = -1 \); similarly for \( q_3 \).

We now prove the theorem.

**Proof:** We start with the “only if” direction. Let \( t, t' \) be a critical set of two intertwined tuples. By definition there exists an instance \( I \) s.t. \( q(I) - q(I - \{t\}) - q(I - \{t'\}) + q(I - \{t, t'\}) \neq 0 \). Since \( q \) is monotone we have \( q(I) = 1, q(I - \{t, t'\}) = 0 \), and either \( q(I - \{t\}) = q(I - \{t'\}) = 0 \) or \( q(I - \{t\}) = q(I - \{t'\}) = 1 \). Without loss, we assume that \( q(I - \{t\}) = q(I - \{t'\}) = 0 \). Then we define two probabilistic databases \( PV = (W, P) \) and \( PV' = (W, P') \) as follows. Each has four possible worlds: \( I, I - \{t\}, I - \{t'\}, I - \{t, t'\} \). In \( PV \) these worlds are assigned probability \( P = (0.5, 0, 0, 0.5) \), respectively; here, \( t_1 \) and \( t_2 \) are positively correlated. In \( PV' \), all worlds are assigned probability 0.25.
i.e. tuple independence. Observe that in both cases, the marginal probability of any tuple is the same, \( P(t) = P(t') = 0.5 \) and all other tuples probability 1, then \( P[q] = 0.5 \) while \( P'[q] = 0.25 \), so the value of \( q \) is not well-defined.

Next we prove the “if” part. 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 \( I \). We need to make this intuition formal, and we start with a lemma. We say that a set of tuples \( T \) is non-intertwined, or NIT, if \( \forall t, t' \in T, t \) and \( t' \) are not intertwined.

**Lemma 4.2** Let \( q \) be a monotone, Boolean query without critical pairs of intertwined tuples, and let \( T \) be a NIT set of tuples. Then the Boolean query \( q^T \) is well defined.

**Proof:** A minterm for \( q^T \) is a minimal instance \( J \) s.t. \( q^T(J) \) is true (that is if \( J' \subset J \) and \( q^T(J') \) is true then \( J = J' \)). 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 the set \( M \) of all its minterms: \( q^T(J) = \bigvee_{J \in M} (J \subseteq I) \). Denoting \( r^J \) the boolean query \( r^J(I) = (J \subseteq I) \), we apply the inclusion-exclusion formula to derive \( P(q^T) = P(\bigvee_{J \in M} r^J) = \sum_{N \subseteq M, N \neq \emptyset} (-1)^{|N|} P(r^N) \). Finally, we observe that for each \( N \subseteq M \), the expression \( P(r^N) \) 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 \( P(r^J) = P(t_1 t_2 \cdots) \). If the set \( J \) contains two disjoint tuples \( (t_i, K = t_j, K) \) then \( P(t_1 t_2 \cdots) = 0 \). Otherwise, it contains only independent tuples \( (t_1, L \neq t_j, L) \), hence \( P(t_1 t_2 \cdots) = P(t_1) P(t_2) \cdots \). In either cases it is well-defined and, hence, so is \( P(q^T) \). \( \Box \)

Thus, we know that, for every NIT \( T \), \( q^T \) is well defined: we need to prove that \( q \) is well defined, and for that we use the expansions above. Let \( PV \) be a probabilistic database s.t. \( PV \models (L, K) \), and let \( Tup \) be the set of possible tuples in \( PV \). Then:

\[
q = \sum_{T \subseteq Tup} \Delta_T q^T = \sum_{T \text{ is NIT}} \Delta_T q^T = \sum_{T \text{ is NIT}} \sum_{S \subseteq T} (-1)^{|S|} q^{T-S}
\]

Here we used the fact that if \( T \) contains any two intertwined tuples, then \( \Delta_T q^T = 0 \): indeed, suppose \( t, t' \in T \) are two intertwined tuples, and denote \( S = T - \{t, t'\} \). Then there exists \( I \) s.t. \( \Delta_T q^T(I) = \Delta_{t,t'} \Delta_S q^T(I) \neq \emptyset \), hence
Next we apply the expectation on both sides, and use the linearity of expectation plus \( P(q) = E[q] \):

\[
P(q) = E[q] = \sum_{T \text{ is NIT}} \sum_{S \subseteq T} (-1)^{|S|} E[q^{T-S}]
\]

\[
= \sum_{T \text{ is NIT}} \sum_{S \subseteq T} (-1)^{|S|} P(q^{T-S})
\]

Finally, we use the lemma to argue that each expression \( P(q^{T-S}) \) is well defined.

Finally, we mention without proof the complexity (the proof can be found in the full version of [25]):

**Theorem 4.7** Checking whether \( q \) is well defined w.r.t. \((L, K)\) is \( \Pi_2^p \)-complete.

As a final comment, we remark that once we have determined that a query \( q \) is well defined for a given partial representation \((L, K)\), of a view \( V \) it is easy to evaluate it, using e.g. the techniques from Sec. 3: for that it suffices to pretend that \( V \) is a BID, either with \( Key(V) = L \) or with \( Key(V) = K \): both assumptions will lead to the same value for \( q \) because \( q \) is well-defined.

## 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 \( \text{PTIME} \). 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 \( \text{PTIME} \), and have described static analysis techniques to derive a partial representation for the view, and to further use it in query evaluation.

**References**


