Bridging the Gap Between Intensional and Extensional Query Evaluation in Probabilistic Databases

Abhay Jha
abhaykj@cs.washington.edu

Dan Olteanu
dan.olteanu@comlab.ox.ac.uk

Dan Suciu
suciu@cs.washington.edu

March 12, 2010

Contents

Abstract

There are two broad approaches to query evaluation over probabilistic databases: (1) Intensional Methods proceed by manipulating expressions over symbolic events associated with uncertain tuples. This approach is very general and can be applied to any query, but requires an expensive postprocessing phase, which involves some general-purpose probabilistic inference. (2) Extensional Methods, on the other hand, evaluate the query by translating operations over symbolic events to a query plan; extensional methods scale well, but they are restricted to *safe* queries.

In this paper, we bridge this gap by proposing an approach that can translate the evaluation of any query into extensional operators, followed by some post-processing that requires probabilistic inference. Our approach uses characteristics of the data to adapt smoothly between the two evaluation strategies. If the query is safe or becomes safe because of the data instance, then the evaluation is completely extensional and inside the database. If the query/data combination departs from the ideal setting of a safe query, then some intensional processing is performed, whose complexity depends only on the distance from the ideal setting.

1 Introduction

The problem of query evaluation on probabilistic databases spans an increasing amount of interest in the database research community due to its applications to scientific data management, data cleaning, data integration, and sensor networks. This problem is known to be notoriously hard. A fundamental result concerning its complexity is the dichotomy for conjunctive queries without self-joins on so-called tuple-independent probabilistic databases: A query is either tractable, that is, it can be evaluated in polynomial time data complexity, or it is #P-hard [8]. The former queries are called *safe* (or hierarchical), whereas the latter are *unsafe*.

A safe query can be evaluated by using an *extensional* query plan. This is a regular relational plan, where every operator performs some simple manipulation of the probabilities, e.g. using multiplication or aggregation [8, 18]. However, such plans can only exist for safe queries: every unsafe query can be shown to be #P hard, and therefore is unlikely to admit any efficient evaluation algorithm. The common practice to evaluating unsafe queries is to use *intensional* probabilistic inference techniques. In this approach one first computes the *lineage* for each tuple in the query's output: this is a symbolic expression, usually a DNF formula, over symbolic events associated with the tuples in the input database. Second, a general purpose probabilistic inference algorithm is run on every lineage expression to compute that tuple's probability, e.g. sampling [21], or inference on graphical models [25], or variable elimination (DPLL) [16].

Thus, there is a significant efficiency gap between safe queries and unsafe queries. The query processor has to make a decision whether the query is safe, and then it can be evaluated using an extensional plan, or unsafe, in which case a much slower intensional method is used.

In this paper we propose a new method that fills in the gap between safe and unsafe queries. Our starting observation is that, even an unsafe query becomes safe if certain functional dependencies hold, or some of the tuples are actually deterministic: we say that the query is safe on that particular database instance. In general, we give a new *safety* criterion for each relational operator based on its input relation instances, that guarantees its output is an independent relation. For a query plan to be safe, every operator must be safe. When this holds, then our method simply evaluates the query plan extensionally as in [8]. If these properties fail, then we identify a set of *offending* tuples, to which we add a *lineage* variable. Our approach is to evaluate the query plan using a mixed extensional/intensional semantics. We process the query plan using extensional operators that use both lineage and probability, unlike in [8] where they only work with probabilities. The output of query plan is a lineage expression of new kind, in that it mixes symbols (for the offending tuples) with probabilities (computed from the non-offending tuples), and we call it a *partial lineage*. In a final processing step, we compute the probabilities of the output tuples by doing probabilistic inference on the partial lineage expressions.

Thus, our approach combines the best aspects of both extensional and intensional evaluation, and bridges the gap between the two. In the worst case, when all input tuples to an operator are "offending", our approach falls back to the intensional approach: compute the full lineage expression. In the best case, when there are no offending tuples, the output of the query plan consists directly of probabilities, like in a safe plan. The common case is somewhere in between, where some processing is done extensionally, while some intensional evaluation needs to be performed on the offending tuples. We show both analytically and experimentally that our approach combines indeed the best of the two worlds, and obtains orders of magnitude performance improvements when compared with generic inference techniques.

The main contributions of this paper are as follows:

- We define a *data safe* query plan, which is a query plan P that can be evaluated using extensional semantics on a database instance D; in contrast, the standard definition of a safe plan is one where the extensional semantics is correct on any instance D. When a query plan P is not data safe for an instance D, then we define a set of "offending tuples", that describes the distance of this instance from the ideal setting, where the plan would have been *data safe* (Sec. 3).

- We propose a new method for evaluating queries over probabilistic databases that consists of evaluating safe plans *conditioned* on the presence/absence of offending tuples (Introduced in Sec. 4.1, and detailed in Sec. 5).

- We introduce the notion of *partial lineage*. This is a mixed symbolic/numeric expression that allows us to combine extensional query evaluation with intensional symbolic processing. The partial lineage has symbols only for the offending tuples in the data: all the other tuples are treated extensionally (Introduced in Sec. 4.2, and detailed in Sec. 5).

- We propose a new representation of lineage expression called AND-OR networks that generalize the factor graphs in [25] and are a special case of Bayesian Networks, and we provide some theoretical results justifying their effectiveness on partial lineage (Introduced in Sec. 4.3 and detailed in Sec. 5).

- We evaluate empirically our approach. We have implemented the query evaluation method as a Java frontend on top of SQL Server 2005. Our experiments show that when the data is nearly safe, our approach can vastly outperform existing approaches, while at the same time, when the data is unsafe, it transitions smoothly and still compares just as well. Our approach performs thus just as well with the best known approaches in the three corner cases: completely deterministic data, safe query, and completely symbolic evaluation, while at the same time transitions smoothly in a mixed setting (Sec. 6).

This is the full version of a paper that appeared in EDBT'10 [14].

2 Background

A *probabilistic relation* $\mathcal{R} = (R, \rho)$ represents a probability distribution over all subsets of R, also called instances, given by $\rho : 2^R \rightarrow [0,1]$ s.t. $\sum_{\omega \subseteq R} \rho(\omega) = 1$. Given k probabilistic relations $(R_1, \rho_1), \ldots, (R_k, \rho_k)$, a *probabilistic database* is the product space $D = (\mathbf{R}, \rho)$, where $\mathbf{R} = (R_1, \ldots, R_k)$ and $\forall 1 \leq i \leq k, \omega_i \subseteq R_i : \rho(\omega_1, \ldots, \omega_k) = \rho_1(\omega_1) \cdot \ldots \cdot \rho_k(\omega_k)$.

We say that a probabilistic relation is *independent* if its tuples are pairwise independent, that is, the presence of a tuple does not constrain the presence or absence of any other tuple in an instance of that probabilistic relation. In that case, the relation can be given by a pair (R, p), where $p : R \rightarrow [0,1]$, and represents the probabilistic relation $\mathcal{R} = (R, \rho)$, where for any $\omega \subseteq R$:

$$\rho(\omega) = \prod_{t \in \omega} p(t) \prod_{t \in (R - \omega)} (1 - p(t)) \tag{1}$$

In this paper, we assume that the input database consists of k independent relations. The application of relational operators can lead, however, to intermediate relations that are no longer independent.

Consider a relational operator $\wp$. When the operator is applied to a probabilistic database, it returns a probabilistic relation, defined formally as follows:

Definition 2.1 (Possible Worlds Semantics) *Let $\wp$ be a relational operator and $D = (\mathbf{R}, \rho)$ a probabilistic database. Then $\wp(D) = (\wp\mathbf{R}, \rho')$ where for any $\omega \subseteq \wp\mathbf{R}$:*

$$\rho'(\omega) = \sum_{\Omega \subseteq \mathbf{R}, \wp\Omega = \omega} \rho(\Omega)$$

The evaluation of a Boolean query q on a probabilistic database $\mathbf{D}$ is defined by $Pr(q)$, which is the sum of probabilities of those instances of $\mathbf{D}$ that satisfy q. In this paper we study efficient techniques for evaluating $Pr(q)$. We consider Boolean conjunctive queries made up using three relational operators: selections σ, projections π, and eq-joins $\bowtie$. We leave out self-joins since queries with self-joins are known to be very difficult [9] and the existing algorithm cannot be translated into a database plan. Furthermore, we consider connected queries; Otherwise, a query q can be expressed as the relational product $q = q_1 q_2$ of two unconnected queries q_1 and q_2, in which case $Pr(q) = Pr(q_1)Pr(q_2)$.

3 Data Safety

Suppose the input relations to an operator are independent. Then, we define the *extensional semantics* of the relational operators as follows:

$$\sigma^e(R, p) = (\sigma R, p) \quad (2)$$

$$\pi^e(R, p) = (\pi R, p_\pi) \quad (3)$$

$$(R_1, p_1) \bowtie^e (R_2, p_2) = (R_1 \bowtie R_2, p_\bowtie) \quad (4)$$

where $p_\pi(t) = 1 - \prod_{\pi t' = t}(1 - p(t'))$ for $t \in \pi R$; $p_\bowtie(t_1 \bowtie t_2) = p_1(t_1)p_2(t_2)$ for $t_1 \in R_1$, $t_2 \in R_2$. We drop the superscript e when it is clear from the context. Extensional operators can be computed efficiently, and they return what looks like a representation of an independent relation.

Suppose we take the output of an extensional operator, and interpret it as an independent probabilistic relation, using Eq. (1). If this probabilistic relation is the same as the possible worlds semantics (Definition 2.1), then we say that the operator is data-safe:

Definition 3.1 *Let P be a query plan and D a probabilistic database instance. A relational operator in P is called* data safe *if its extensional semantics is equal to the possible worlds semantics. The plan P is called* data safe *if each of its operators is data safe.*

Note that the definition of data-safety depends both on the relational plan P and on the instance D. We give below necessary and sufficient conditions for data-safety.

Proposition 3.2 *The selection and projection operators are always data-safe. A join operator $(R, p) \bowtie^e_{A=B} (S, q)$ is data-safe iff for any tuple $t \in R$, if $p(t) < 1$ then $|\sigma_{B=t.A}(S)| \leq 1$, and for any tuple $t \in S$, if $q(t) < 1$ then $|\sigma_{A=t.B}(R)| \leq 1$.*

Proof: Selection and projection can be easily shown to be always data-safe. We only consider the case of joins here.
For the *if* direction, given any world $\omega \subseteq R \bowtie S$: by extensional semantics

$$Pr^e(\omega) = \prod_{t_1 \bowtie t_2 \in \omega} p(t_1)q(t_2) \prod_{t_1 \bowtie t_2 \notin \omega} (1 - p(t_1)q(t_2))$$

Let $\omega_R = \{t \in R \mid t \bowtie t' \in \omega, p(t)\}$ and similarly define ω_S. Then we can rewrite

$$Pr^e(\omega) = \prod_{t \in \omega_R} p(t) \prod_{t \in \omega_S} q(t) \prod_{t_1 \bowtie t_2 \notin \omega} (1 - p(t_1)q(t_2))$$

This is because if any $t \in R, S$ joins with more than one tuple, then it must be deterministic. Now consider all $\omega_1 \subseteq R$ and $\omega_2 \subseteq S$ s.t. $\omega_1 \bowtie \omega_2 = \omega$. Then clearly $\omega_1 \supseteq \omega_R$ and $\omega_2 \supseteq \omega_S$. Also for any tuple $(t_1, t_2) \notin \omega$, we require that both t_1 and t_2 cannot be present in ω_1, ω_2. This gives

$$\begin{aligned} Pr(\omega) &= \prod_{t \in \omega_R} p(t) \prod_{t \in \omega_S} q(t) \prod_{t_1 \bowtie t_2 \notin \omega} (1 - p(t_1)q(t_2)) \\ &= Pr^e(\omega) \end{aligned}$$

We now prove the *only if* part. Assume w.l.o.g. that $R(x, y)$ and $S(y, z)$ are binary relations, and the join is a natural join. Consider a tuple $(a, b) \in R$ s.t. $p(a, b) < 1$ and $(a, c_1), (a, c_2) \in S$. Let $(U, r) = (R, p) \bowtie (S, q)$, then

$$Pr(U(a, b, c_1) \wedge U(a, b, c_2)) = p(a, b)q(b, c_1)q(b, c_2)$$

while

$$Pr(U(a, b, c_1))Pr(U(a, b, c_2)) = p^2(a, b)q(b, c_1)q(b, c_2)$$

But $p(a, b)q(b, c_1)q(b, c_2) \neq p^2(a, b)q(b, c_1)q(b, c_2)$ unless $p(a, b)$ is 1 or 0, a contradiction. Therefore, the tuples $U(a, b, c_1)$ and $U(a, b, c_2)$ are not independent. Hence proved. □

Call a join operation $R(\mathbf{x}) \bowtie S(\mathbf{y})$ 1-1 if $\mathbf{x} = \mathbf{y}$. Note that 1-1 join are always data-safe irrespective of the data instance. Previous work that has mostly focused on the notion of *safety* [8], use only 1-1 joins.

Definition 3.3 *A query plan P is called a* safe plan *if it is data safe for any input database D. A query that has a safe plan is called a* safe query.

It follows from our discussion that a query plan is safe iff all its joins are 1-1. For example, consider the Boolean query $q = R(x, y), S(x, z)$. The query plan $\pi_\emptyset(R \bowtie S)$ is not safe, because the join is not 1-1. Suppose, however, that the two independent probabilistic relations $R(x, y)$ and $S(x, z)$ are such that x is a key, both in R and in S. Then the plan above is data-safe, because the join happens to be 1-1. On the other hand, the plan $\pi_\emptyset(\pi_x(R) \bowtie \pi_x(S))$ is safe on any input database, because its join is always 1-1.

Finally, we define the set of offending tuples:

Definition 3.4 *Consider a probabilistic database instance D and a query plan P. Let $\wp$ be an operator in P. A set T of input tuples to $\wp$ is called* set of offending tuples *if $\wp$ becomes data-safe after removing the tuples in T.*

Given a fixed query plan P, operators $\wp$ and data instance D, a set of offending tuples can be computed efficiently, using standard SQL. For example, consider the plan $P = \pi_{\emptyset}(R(x,y) \bowtie S(x,z))$ for the query mentioned above. The inputs to the join operator are the two independent probabilistic relations (R,p) and (S,q). The set of offending tuples T for this operator can be computed as:

$$\begin{aligned} T &= \{(a,b) \in R \mid \text{s.t. } |\{(a,c) \in S \mid q(a,c) < 1.0\}| \geq 2\} \\ &\cup \{(a,c) \in S \mid \text{s.t. } |\{(a,b) \in T \mid p(a,b) < 1.0\}| \geq 2\} \end{aligned}$$

Note that the set of offending tuples do not necessarily come from the database instance; they could also be intermediate tuples generated during the query plan. So many tuples in the database instance, that make the query unsafe, may actually correspond to just one offending tuple for the query plan. Note that a safe plan has no offending tuples and as we will see later, the output of any plan is an expression with symbols from only the offending tuples. Hence the number of offending tuples is a measure of how safe/unsafe a plan really is for a given database instance.
In this paper our focus is on how to use a plan P and set of offending tuples for all its operators in order to evaluate the query efficiently. In probabilistic databases, query evaluation is significantly more expensive than query optimization, and this is our main focus here. We do not address the problem of finding the plan P and computing the offending tuples T. As suggested here, T can be computed using standard SQL, while P can be obtained using standard optimization algorithm, by searching a space of query plans: both problems are outside the scope of this paper.

A safe query can be evaluated efficiently, by using the extensional versions of the relational operators. By contrast, as shown in [8], every unsafe query is #P-hard, which means that no algorithm exists that can evaluate it efficiently on *every* database. The state of the art for evaluating unsafe queries is by computing their lineage.

Definition 3.5 *Let q be a Boolean conjunctive query q and $D = (\mathbf{R}, \rho)$ a probabilistic database. The* lineage *of q is a DNF formula $F(q,D)$ obtained by grounding q with tuples from D.*

Example 3.6 Let $q = R(x,y), S(y,z)$ and $R = S = \{(1,1),(1,2),(2,1),(2,2)\}$. Then, $F(q,D) = r_{11}s_{11} \vee r_{11}s_{12} \vee r_{12}s_{21} \vee r_{12}s_{22} \vee r_{21}s_{11} \vee r_{21}s_{12} \vee r_{22}s_{21} \vee r_{22}s_{22}$, where r_{ij} and s_{ij} are random variables corresponding to tuples (i,j) in R and S respectively.

In probabilistic database systems [2, 1, 4], if the query is unsafe then the system first computes its lineage, then uses some general-purpose probabilistic inference method for computing the probability of the lineage; it is known that the probability of a query, $Pr(q)$ is the same as the probability of its lineage expression.

4 Overview of Our Approach

We develop three techniques that exploit the data in order to compute unsafe queries more efficiently. We give here an overview of the techniques and introduce them formally in Sec. 5.3.

4.1 Combining Safe Plans with Conditioning

The first technique that we introduce consists of combining safe plans with conditioning. We illustrate this technique on the query $q_u :- R(x), S(x,y), T(y)$, which is known to be an *unsafe* query, because none of the joins $R \bowtie S$ or $S \bowtie T$ is 1-1. The complexity of this query is #P-hard.

Consider a probabilistic database instance where S satisfies the functional dependency $x \rightarrow y$. In this case the plan $\pi_y(R \bowtie S) \bowtie T$ is data-safe, because on this particular instance both joins are 1-1. Thus, if the functional dependency holds in S, then we can compute the query very efficiently using only extensional operators.

Consider now the case when most, but not all tuples in $S(x,y)$ satisfy the functional dependency $x \rightarrow y$. In this case the plan above is no longer data-safe, and we cannot use it to compute the query. We use a new approach instead. To describe it, suppose first that a single value $a \in \pi_x(S)$ violates the functional dependency: that is, we assume that the functional dependency $x \rightarrow y$ holds on the relation $\sigma_{x \neq a}(S)$. Then:

$$\begin{aligned} Pr(q_u) &= Pr(q_u|\neg R(a))(1 - Pr(R(a))) \\ &+ Pr(q_u|R(a)) \cdot Pr(R(a)) \end{aligned}$$

To compute the conditional probability $Pr(q_u|\neg R(a))$, we modify the probabilistic relation R by removing $R(a)$, then evaluate q_u: by Proposition 3.2 the plan $\pi_y(R \bowtie S) \bowtie T$ is still data-safe on this database, since now the offending tuples in S do not join with anything in R. To compute $Pr(q_u|R(a))$, modify R by setting $Pr(R(a)) = 1$. By Proposition 3.2 the join operator $R \bowtie S$ is still data-safe, since no condition needs to be checked for tuples whose probability is 1.

In general, call a set of tuples $a_1, \ldots, a_k$ in R *offending tuples* if these are all the tuples that lead to violations of the functional dependency $x \rightarrow y$ in S; in other words, $\sigma_{x \notin \{a_1,\ldots,a_k\}}(S)$ satisfies the functional dependency. Then, we compute $Pr(q)$ by evaluating the safe plan $\pi_y(R \bowtie S) \bowtie T$ on 2^k probabilistic relations. Thus, we have shown how, by using conditionals on a set of k values in R, we can reduce the evaluation problem of q_u to computing 2^k safe plans.

However, evaluating 2^k safe plans is not practical, except for very small values of k. While, in general, no algorithm can escape the fact that q_u is #P-hard, in practice one can compute the lineage of q_u first, then apply some optimized probabilistic inference method on the resulting DNF formula and obtain a much better performance than evaluating 2^k safe plans. We describe next the second step of our approach, which ensures that our algorithm falls back on a lineage-based inference when the data is unfavorable for safe plans.

4.2 Partial Lineage

The key observation is that in all 2^k probabilistic instances we need to evaluate *the same* safe plan. We exploit this by introducing a *partial lineage*, which, in essence, keeps track of the 2^k distinct probabilistic databases. Informally, a partial lineage is a lineage expression over k propositional variables *and* probabilities (numbers). Each number stands for an independent, and anonymous propositional variable. We illustrate this on our running example. Assume that two values a_1, a_2 in R violate the functional dependency in S. Then, when computing the join $R \bowtie S$ we compute the lineage by using only the variables r_1, r_2, corresponding to the tuples a_1 and a_2 in R: for all others, we compute their probabilities using the equations for the extensional operators. For example, assume the following probabilistic instance:

$$
\begin{aligned}
R &= \{(a_1, 0.1), (a_2, 0.2), (a_3, 0.3), (a_4, 0.4)\} \\
S &= \{(a_1, b_1, 0.11), (a_1, b_2, 0.12), (a_2, b_1, 0.13), \\
&\quad (a_2, b_2, 0.14), (a_3, b_1, 0.15), (a_4, b_1, 0.16)\} \\
T &= \{(b_1, 0.21), (b_2, 0.22)\}
\end{aligned}
$$

The numbers indicate the tuple probabilities. Note that here both a_1 and a_2 violate the functional dependency $x \rightarrow y$ in S; in contrast, a_3 and a_4 satisfy the dependency. Then, the following illustrates the first two steps for computing the plan $\pi_y(R \bowtie S) \bowtie T$:

$$
\begin{aligned}
R \bowtie S &= \{(a_1, b_1, 0.11r_1), (a_1, b_2, 0.12r_1), \\
&\quad (a_2, b_1, 0.13r_2), (a_2, b_2, 0.14r_2), \\
&\quad (a_3, b_1, 0.045), (a_4, b_1, 0.064)\} \\
\pi_y(R \bowtie S) &= \{(b_1, 0.11r_1 \vee 0.13r_2 \vee 0.10612), \\
&\quad (b_2, 0.12r_1 \vee 0.14r_2)\}
\end{aligned}
$$

The *partial lineage* expressions above combine Boolean variables r_1, r_2 with numbers. The interpretation is that every number stands for a separate Boolean variable, which is independent from everything else: the number gives the probability of this anonymous Boolean variable. For example, the first lineage expression $0.11r_1$ stands for $s_1 r_1$ where s_1 is a propositional variable with probability 0.11. The key idea in the partial lineage is that we do not need the symbol s_1 any further, and instead replace it with its probability 0.11. The safe plan manipulates probability numbers directly. For example, when joining a_3 and (a_3, b_1), their probabilities are multiplied $0.3 \cdot 0.15 = 0.045$. Similarly, after we perform the projection/duplicate-elimination operation π_y, the partial lineage $0.11r_1 \vee 0.13r_2 \vee 0.10612$ has an intensional component $0.11r_1 \vee 0.13r_2$, and an extensional component $0.10612 = 1 - (1 - 0.045) * (1 - 0.064)$.

Continuing the evaluation of the query plan in this way we arrive at a partial lineage expression for the entire query; on this we run any general purpose probabilistic inference algorithm.

It is interesting to compare this approach with the standard approach of computing the full lineage first, then running an inference algorithm. The partial lineage is always a strict subset of the full lineage, because it only uses a subset of Boolean variables used by the full lineage, namely the offending tuples. Hence, the final step, performing a probabilistic inference on the final lineage expression, is no harder for partial lineage than for full lineage. On the other hand, when the set of offending tuples is small, then the safe plan ends up manipulating mostly numbers (i.e. probabilities). In the best case, there are no offending tuples at all, and in that case our approach consists entirely of a safe plan.

4.3 Complexity of Query Evaluation

In this section we give some theoretical results comparing the complexity of different approaches to query evaluation proposed so far, and then explain how our algorithm compares with them. We are only interested in data complexity here and hence assume query size to be bounded.

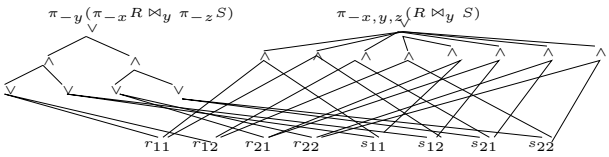

Figure 1: AND/OR-factor graphs for $q = R(x, y), S(y, z)$ in Example 3.6.

4.3.1 Lineage

The intensional approaches to query evaluation compute the probability of the query lineage represented by DNF formulas. The techniques used here are similar to those for SAT and #SAT problems and generally involve using some heuristic to eliminate the variables and simplify the formulae. There are some classes of formulae such as symmetric and read-once functions [24] that can be solved efficiently. The most effective methods rely on finding a good variable order; however, finding the best order is itself an intractable problem, and there are no guarantees that one can find such a good order based on the data.

The approaches that provide some theoretical guarantees on the performance [12, 10, 17] have their running time exponential in the *treewidth* of the lineage. We next define treewidth.

A hypergraph is a pair $H = (V, E)$, where V is the set of vertices and $E \subseteq 2^V$ is a set of subsets of V. With each dnf-formulae $\mathcal{F} = \bigvee_{i=1}^{n} \bigwedge_{j=1}^{k} a_{ij}$, we associate the hypergraph $H(\mathcal{F}) = (V, E)$, where V is the set of vars in $\mathcal{F}$ and

$$E = \{\{a_{i1}, \ldots, a_{ik}\} | 1 \leq i \leq n\}$$

A tree decomposition for a hypergraph $H = (V, E)$ is a pair (X, T), where $X = X_1, ..., X_n$ is a family of subsets of V, and T is a tree whose nodes are the subsets X_i, satisfying the following properties

1. $\bigcup X_i = V$
2. $\forall e \in E,\ \exists X_i$ s.t. $e \subseteq X_i$
3. For each $v \in V$, the set $\{X_i | v \in X_i\}$ forms a connected component in T.

The *width* of tree decomposition is $\max\{|X_i| - 1 | X_i \in X\}$. The treewidth $tw(H)$ of a hypergraph H is the minimum width among all possible tree decompositions of H. The treewidth [1] $tw(\mathcal{F})$ of a DNF formula $\mathcal{F}$ is the treewidth of the associated hypergraph $H(\mathcal{F})$. The complexity of lineage-based approaches is exponential in the treewidth of the lineage. We first prove that queries with lineage of bounded treewidth correspond precisely to a strict subset of safe queries.

Definition 4.1 *A query q is called* strictly hierarchical *if q can be written as $R_1(\bar{x}_1), \ldots, R_m(\bar{x}_m)$ s.t. $\bar{x}_1 \subseteq \ldots \subseteq \bar{x}_m$.*

Theorem 4.2 *The following are equivalent: (1) q is a strictly hierarchical query and (2) there exists a constant c such that for any database instance D, $tw(F(q, D)) < c$.*

[1] The treewidth notion here is the treewidth of the primal graph. Treewidth can alternatively be defined on the *incidence* graph, but assuming bounded query size, the two widths are related by a constant factor.

Thus, only a subset of the safe or hierarchical queries have lineage of bounded treewidth. For example, the query $R(x, y), S(x, z)$ is safe, but it is not strictly hierarchical, hence its lineage does not have a bounded treewidth. Moreover, it is important to observe how the treewidth grows. A cartesian product occurring in a subquery makes the treewidth of the query at least the size of one of the two sets. So any query with many-many join will have high treewidth lineage.

Hence the class of intensional methods based on lineage treewidth cannot even compute safe queries in PTIME. One can however use the hierarchical structure of safe queries to efficiently factorize the lineage into a tractable form, called one-occurrence form [17], for which probability computation can be performed in linear time. But this approach does not generalize to unsafe queries and this motivates the need for a general model that would be in PTIME for safe queries.

4.3.2 Factor Graphs

[25] propose modeling queries as *AND/OR- factor graphs*, where there are two kinds of factors: AND and OR. An AND factor is true iff its inputs are both true, and an OR factor is false iff both its inputs are false, just like conventional gates. Their model takes as input not a query, but a query plan. So the same query may be expressed as two graphs. Figure 1 shows how this approach would model the query in Example 3.6 for two different plans. AND/OR-factor graphs are a special case of Bayesian Networks, and the inference in any Bayesian Network G is carried out by moralizing the graph(connect all the parents of every node)G to convert it into a markov network $M(G)$. This would lead to networks of very high treewidth, but [25] exploit *decomposability* [22] to reduce the factors in G to size 3 factors: call this graph $D(G)$. Figure 2 illustrates the construction of $M(G)$ and $D(G)$. Hence the complexity of their approach depends on the treewidth of the graph $M(D(G))$ obtained by moralizing the graph with *decomposed* factors. This model has the advantage that for safe plans $tw(M(D(G))) = 2$, and hence this approach is superior to lineage-based approaches in that respect. It is known that in general $tw(G) \leq tw(M(D(G))) \leq tw(M(G))$ [22].

4.3.3 Our Approach: Partial Lineage

To allow the partial lineage approach to take full advantage of these techniques, we use AND-OR networks (Sec. 5.1), which are a generalization of the AND-OR factor graphs, to represent partial lineage. Given a query plan P, let G_f be the AND-OR factor graph representing it and G_n be the AND-OR network representing the partial lineage. Then our approach is exponential in $tw(G_n)$. This is analogous to the factor graphs approach whose complexity depends on $tw(M(D(G_f)))$. We can show that

Proposition 4.3 *G_n is a minor of G_f.*

Using the fact that treewidth of a minor is at most the treewidth of the graph, we get that the treewidth of the AND-OR Network modeling the partial lineage is less than that of the AND-OR factor graph i.e. $tw(G_n) \leq tw(G_f)$. Since $tw(G_f) \leq tw(M(D(G_f)))$,

Corollary 4.4 $tw(G_n) \leq tw(M(D(G_f)))$.

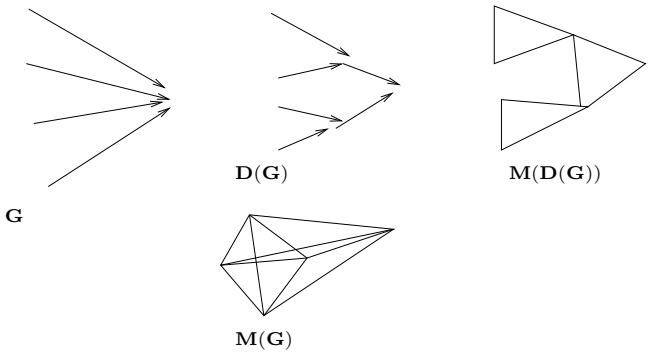

Figure 2: Decomposing a factor G into small factors $D(G)$ and moralizing these graphs into $M(G)$ and $M(D(G))$.

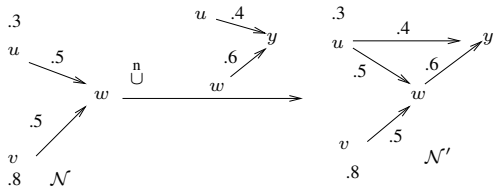

Figure 3: And-Or networks $\mathcal{N}$ and $\mathcal{N}'$; $\mathcal{N}'$ is obtained by *augmenting* $\mathcal{N}$.

As we discussed, the inference algorithm in [25] is exponential in $tw(M(D(G_f)))$. Hence we see that our approach is at least as good as in complexity as the other approaches, if not better.
To summarize, representing lineage as a DNF formula is a poor choice since we lose information about its structure and many inference algorithms on the lineage won't even evaluate the safe queries in PTIME. Representing them as factor graphs is better in that respect, and hence we employ a similar approach in representing partial lineage and indeed our approach has just as good worst-case guarantee as these approaches.

5 Query Evaluation with Partial Lineage

We describe here our approach to query evaluation using partial lineage. First, we introduce And-Or networks, which are our optimized representation of lineage expressions. They are a special case of Bayesian networks, which represent exactly the kind of dependencies introduced by relational operators. We then define *pL-relations*, which extend the independent relational model using And-Or networks, so that they can model the intermediate relations generated during query evaluation on independent relations. Finally, we show how to execute relational operators over pL-relations and conclude the section by performing the theoretical analysis outlined in Sec. 4.3.

5.1 And-Or Networks

An And-Or network $\mathcal{N} = (V, E, P, Lab)$ can be represented as a directed acyclic graph $G = (V, E)$, where $Lab : V \rightarrow \{And, Or, Leaf\}$, and $P : E \cup V_L \rightarrow [0, 1]$ with $V_L = \{v \mid v \in V, \nexists w \in V \; s.t. \; (w, v) \in E\}$. We call the nodes in V_L as the leaves of G. $Lab(V_i) = Leaf$ iff it is a leaf in G. Every node in V is treated as a random Boolean variable. Let x be a Boolean assignment over V. We use x_v to denote $x(v)$,$v \in V$ and x_W to denote $x|_W$, $W \subseteq V$. For

every node $v \in V$, we define a conditional probability distribution, conditioned on its parents $par(v) = \{w \mid (w, v) \in E\}$ as

$$\phi(x_v = 1|x_{par(v)}) = \begin{cases} 1 - \prod_{w \in par(v)} (1 - x_w P(w, v)), & v \text{ is Or} \\ \prod_{w \in par(v)} x_w P(w, v), & v \text{ is And} \\ P(v), & v \text{ is leaf} \end{cases}$$

The And-Or Network $\mathcal{N}$ represents a joint probability distribution over V, where

$$\mathcal{N}(x) = \prod_{v \in V} \phi(x_v|x_{par(v)})$$

For the sake of simplicity we use $\mathcal{N}$ to denote the distribution. Note that And-Or Networkds are a special case of *Bayesian Networks* and hence represents a valid probability distribution. As we will see, Or nodes represent the dependency introduced by projection, while And nodes represent join.

The marginal probability $\mathcal{N}^0(y)$ where y is a Boolean assignment to $W \subseteq V$, is given by

$$\mathcal{N}^0(y) = \sum_{x, x_W = y} \mathcal{N}(x)$$

Given a network $\mathcal{N}$, we denote V, E by $V(\mathcal{N}), E(\mathcal{N})$ respectively.

Example 5.1 Figure 3 shows an And-Or Network $\mathcal{N}$ on vertices u, v, w. Note that u, v are the leaves, and assume w to be an Or node. We did not denote nodes as And/Or on the graph. The probability values P for leaves and edges are written beside them. Consider any assignment x to the nodes of $\mathcal{N}$, then

$$\mathcal{N}(x) = \phi(x_w|x_u, x_v)\phi(x_u)\phi(x_v)$$

Let $x = \{0, 1, 0\}$ on $\{u, v, w\}$, then $\mathcal{N}(x) = (1 - 0 \cdot 0.5)(1 - 1 \cdot 0.5) \cdot (1 - .3) \cdot .8 = .28$.

Augmenting an And-Or Network: Given $\mathcal{N} = (V, E, P, Lab)$ and a new node $w \notin V$, we grow the And-Or Network by connecting w to few or none of the nodes in V as its parents. We represent this operation by $\mathcal{N}' = \mathcal{N} \overset{\text{n}}{\cup} (w, E', P', lab) = (V \cup \{w\}, E \cup E', P \cup P', Lab \cup \{(w, lab)\})$, where $E' \subseteq V \times \{w\}$. If $E' = \emptyset$, then lab has to be leaf and $P' : \{w\} \to [0, 1]$; else lab is And/Or and $P' : E' \to [0, 1]$. It should be easy to see that $\mathcal{N}'$ is also a valid And-Or Network. We can similarly extend it to add not just one vertex w, but a set of vertices W all of the same label lab as $\mathcal{N} \overset{\text{n}}{\cup} (W, E', P', lab)$.
Figure 3 shows the network $\mathcal{N}'$ obtained by augmenting $\mathcal{N}$ with a node y with u, w as its parents.

5.2 Relations with Partial Lineage

We define some shorthand notation before this section. We use $z \in 2^S$ to denote that $z : S \to \{0, 1\}$. $P_I(\omega, p(t))$ stands for the independent probability distribution $\prod_{t \in \omega} p(t) \prod_{t \notin \omega} (1 - p(t))$, assuming R is clear from the context; otherwise we will write it as $P_I^R(\omega, p(t))$. As we have already learnt from proposition 3.2, the independent relation model is not enough to represent the

intermediate distributions resulting from relational operators. We define pL-relation because as we will see, this can *succinctly* represent the intermediate distributions and is still closed with respect to relational operators. In particular, independent relations can be modeled with just one node And-Or Network.

Definition 5.2 *A pL-relation $\mathcal{R} = (R, p, l, \mathcal{N})$, where $p : R \to [0, 1]$, $l : R \to V(\mathcal{N})$, and $\mathcal{N}$ is an And-Or network ; represents a probability distribution ρ over all subsets ω of R as*

$$\rho(\omega) = \sum_{z \in 2^{V(\mathcal{N})}} \mathcal{N}(z) P_I(\omega, z_{l(t)} p(t)) \tag{5}$$

We often use $t \in \mathcal{R}$ to mean $t \in R$. Given $\mathcal{N}$, $\mathcal{R}$ can be represented tabularly by appending columns l, p to R. Its easy to show that ρ is a probability distribution, as will be clear from the following discussion.

Example 5.3 Consider the case where $\mathcal{N}$ has just one node ε with $P(\varepsilon) = 1$. So $\mathcal{N}(z) = 1$ if $z_\varepsilon = 1$ else 0. Now consider the pL-relation $\mathcal{R}$ given by

$$\begin{pmatrix} \text{A} & \text{l} & \text{p} \\ 1 & \varepsilon & .6 \\ 2 & \varepsilon & .3 \\ 3 & \varepsilon & .5 \end{pmatrix}$$

Then $\rho(\omega) = P_I(\omega, p(t))$, the probability distribution of the independent relation (R, p). To represent any independent relation as a pL-relation, we just need to set $l(t) = \varepsilon$ for all tuples t.

Example 5.4 Let $\mathcal{N}$ be the one given by figure 3. Now consider $\mathcal{R}$ as given by

$$\begin{pmatrix} \text{A} & \text{l} & \text{p} \\ 1 & \text{u} & 1 \\ 2 & \text{v} & 1 \\ 3 & \text{w} & 1 \end{pmatrix}$$

Note that $\forall t \; p(t) = 1$ and hence $z_{l(t)} p(t) = z_{l(t)}$. $P_I(\omega, z_{l(t)}) = 0$ if $\exists t \in \omega \; s.t. \; z_{l(t)} = 0$ and vice-versa if $t \notin \omega$ and $z_{l(t)} = 0$. Hence the summation in Eq. (5) has only one non-zero term, corresponding to the assignment z^0 s.t. $z^0_{l(t)} = \mathbf{1}_{t \in \omega}$. Therefore $\rho(\omega) = \mathcal{N}(z^0)$. Hence in this case the relation just represents the And-Or Network $\mathcal{N}$.

Between these two extremes, that is, And-Or networks and independent relations, the pL-relations represent a combination of many independent relations each weighted according to the probability distribution of an And-Or network.

Mixture of Independent Relations A mixture of independent relations is a convex combination of many independent relations. A mixture

$$\{(w_i, (R, p_i)) \mid w_i \in \mathbb{R}, \sum_i w_i = 1, p_i : R \to [0,1]\}$$

defines a probability distribution ρ, where

$$\rho(\omega) \quad = \quad \sum_i w_i P_I(\omega, p_i(t)) \tag{6}$$

From Eq. (5), one can see that the pL-relation $\mathcal{R}$ can be expressed as the mixture $\{(\mathcal{N}(z), (R, p_z)) \mid z \in 2^{V(\mathcal{N})}, p_z(t) = z_{l(t)}p(t)\}$. Given any pL-relation, we call this the *standard mixture* as it follows from the definition naturally.

Example 5.5 Let $\mathcal{N}$ be the network in Fig. 3, and consider the pL-relation $\mathcal{R}$ given by

$$\begin{pmatrix} \text{A} & l & p \\ 1 & w & 1 \\ 2 & \varepsilon & .3 \\ 3 & \varepsilon & 0.6 \end{pmatrix}$$

The standard mixture is

$$\{(\mathcal{N}(z), (R, p_z)) \mid z \in 2^{\{u,v,w\}}, p_z(t) = z_{l(t)}p(t)\}.$$

$\mathcal{R}$ can also be expressed as another mixture

$$\left\{(\phi(z_u)\phi(z_v), (R, p_z)) | z \in 2^{\{u,v\}}\right\}, \text{ where}$$
$$p_z(t) = \begin{cases} z_{l(t)}p(t), & t \neq 1 \\ \phi(z_w = 1 | z_u, z_v), & t = 1 \end{cases}$$

Since $p(1)$ is 1, the lineage variable w and the tuple 1 represent the same event. So instead of summing over $w = 0, 1$ in the probability expression for $\mathcal{N}$, we move the factor $\phi(z_w = 1|z_u, z_v)$ to the probability value of tuple 1, and get rid of w from $\mathcal{N}$. The resulting distribution is still the same.

Our earlier example demonstrates another way of representing pL-relations as mixtures.

Proposition 5.6 *Given a pL-relation $\mathcal{R} = (R, p, l, \mathcal{N})$, let $S \subseteq R$ be a set of tuples s.t. $\forall t \in S,\ p(t) = 1$. Define $V_S = \{l(t) \mid t \in S\}$, and let $\mathcal{N}'$ be the And-Or Network obtained by removing the nodes V_S from $\mathcal{N}$. Then*

$$\left\{(\mathcal{N}'(z), (R, p_z)) \mid z \in 2^{V(\mathcal{N}) \setminus V_S}\right\}, \textit{ where}$$
$$p_z(t) = \begin{cases} z_{l(t)}p(t), & t \notin S \\ \phi(z_{l(t)} = 1 | z_{par(l(t))}), & t \in S \end{cases}$$

is a valid mixture representing $\mathcal{R}$. We denote it by mixture$(\mathcal{R}, S)$.

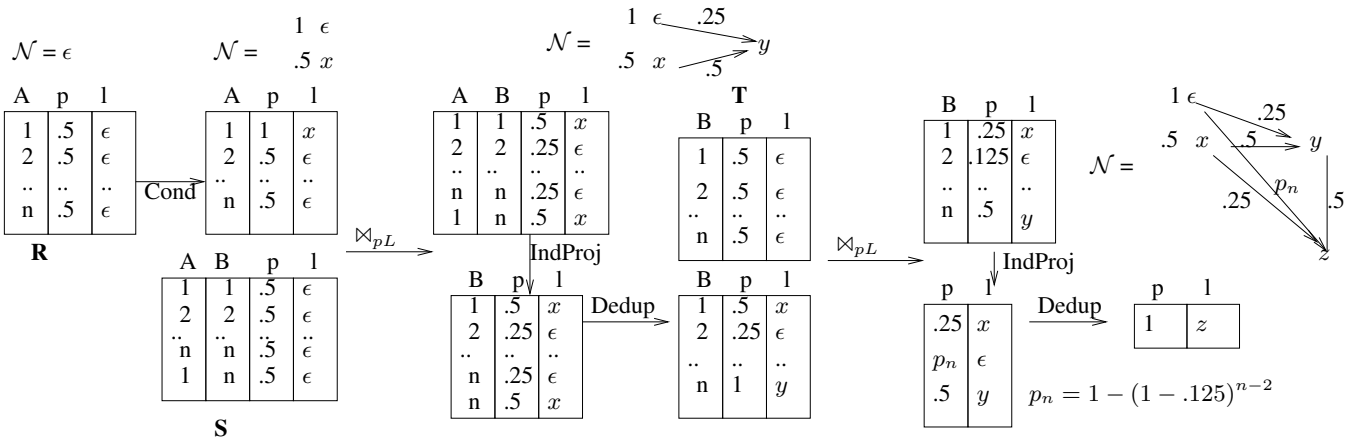

Figure 4: Evaluating $q : -R(x)S(x,y)T(y)$. $Pr(q)$ is given by $\mathcal{N}^0(z=1)$. P values for leaves and edges in $\mathcal{N}$ are written besides them.

5.3 Relational Operators over pL-relations

In this section, we will describe the select-project-join operators over pL-relations. There can be more than one way of doing this; we have defined them in a way so that i)for independent relations, they are the same as extensional operators and ii)in general exploit extensional operators as far as possible. We will also show that each operator obeys the possible world semantics. For this, we will use the existing query evaluation results for independent relations. Since a pL-relation is a mixture of independent relations, the result of any operator can be shown to be the same mixture where the operator has been applied to each independent relation in the mixture individually. Formally

Proposition 5.7 *Consider any unary relational operator $\wp$ and $\mathcal{R} = (R, p, l, \mathcal{N})$ a pL-relation. Let $\{(\mathcal{N}(z), (R, p_z)) \mid z \in 2^{V(\mathcal{N})}\}$ be the standard mixture. Assuming $\wp(R, p_z)$ is also an independent relation $\forall z \in 2^{V(\mathcal{N})}$, then*

$$\{(\mathcal{N}(z), \wp(R, p_z)) \mid z \in 2^{V(\mathcal{N})}\} = \wp\mathcal{R}.$$

This can be extended similarly for binary operators too.
Hence one way of showing that $\mathcal{R}' = \wp\mathcal{R}$ is to first show that $\wp$ is *safe* for all (R, p_z), and then show that the mixture $\{(N(z), \wp(R, p_z)) \,|z \in 2^{V(\mathcal{N})}\}$ is equal to $\mathcal{R}'$. We discuss below the case of selection, projection, and join.

5.3.1 Selection

Given a pL-relation $\mathcal{R} = (R, p, l, \mathcal{N})$, and $A \subseteq attribs(R)$, let $\mathcal{R}' = (\sigma_{A=a}R, p, l, \mathcal{N})$.
First of all note that selection on independent relations is always safe; $\sigma(R, p) = (\sigma R, p)$ and then $\{(N(z), (\sigma_{A=a}R, p_z))|z \in 2^{V(\mathcal{N})}\}$ is actually the standard mixture of $\mathcal{R}'$. Hence by proposition 5.7 $\mathcal{R}' = \sigma_{A=a}\mathcal{R}$. Therefore selection is performed, just like in independent relations, by simple relational selection operator.

5.3.2 Projection

Given a pL-relation $\mathcal{R} = (R, p, l, \mathcal{N})$, and $A \subset attribs(R)$, we want to compute $\pi_A\mathcal{R}$. For the sake of simplicity, we describe this operation in two stages.

Independent Project : This is just like the independent project operation for independent relations, except that here we project only on tuples with same lineage l. Let $A' = A \cup \{l\}$. We compute $\mathcal{R}^i = (\pi_{A'}R, p^i, l^i, \mathcal{N})$; where for any $a = \pi_{A'}t$, $l^i(a) = l(t)$, $p^i(a) = 1 - \prod_{\pi_{A'}t=a}(1 - p(t))$

Example 5.8 Figure 4 has two Independent Projection operations, listed as IndProj. The first one is inconsequential, as there are no candidate tuples for independent project. But consider the second IndProj operator that projects the tuples $(i, .125, \epsilon), 2 \leq i \leq n-1$ on l to (p_n, ϵ). Note that independent project does not change $\mathcal{N}$.

Deduplication : Now we describe how to calculate $\pi_A\mathcal{R}^i$, by removing duplicate tuples. Define

$$\begin{aligned} Pj &= \{(a, a') \mid \pi_A a' = a, a' \in \mathcal{R}'\} \\ S &= \{a \mid \#\{a' | Pj(a, a')\} > 1\} \end{aligned}$$

Let $l^d(a) = l^i(a)$ if $a \notin S$ else $h(\{(l^i(a'), p^i(a')) | Pj(a, a')\})$, where h is a hash function. $E = \{(l^d(a), l^i(a')) \mid Pj(a, a'), a \in S\}$ and $Q(l^d(a), l^i(a')) = p^i(a')$. Let $V = \{l^d(a) \mid a \in S\}$. Define an And-Or network $\mathcal{M} = \mathcal{N} \overset{n}{\cup} (V, E, Q, Or)$.
Then $\pi_A\mathcal{R} = \mathcal{R}^d = (\pi_A R, p^d, l^d, \mathcal{M})$, where

$$p^d(a) = \begin{cases} p^i(a) & \text{if} a \notin S \\ 1 & \text{otherwise.} \end{cases}$$

Deduplication replaces duplicate tuples with a unique tuple. The probability of the new tuples is 1, while the new lineage is just computed by hashing the lineage,probability of duplicate tuples. The existing And-Or Network is then augmented with this new lineage by connecting it to the lineages of the duplicate tuples. In case of no duplicate tuples, nothing is done.

Example 5.9 Figure 4 illustrates Deduplication, written as Dedup, on two relations. In the first case for example, the duplicate tuples with $B = n$ are reduced to a new tuples with a new lineage y produced by hashing. $\mathcal{N}$ is then *augmented* with y as an Or node, where x, ε, the lineage of duplicate tuples, are set as parents of y.

Theorem 5.10 $\mathcal{R}^d = \pi_A\mathcal{R}$.

Proof: Projection is also always safe, hence it suffices to show that $\mathcal{V} = (\mathcal{N}(z), (\pi_A(R, p_z))) = (\mathcal{N}(z), (\pi_A R, p'_z))$ is a mixture for $\mathcal{R}^d$, where

$$\begin{aligned} p'_z(a) &= 1 - \prod_{\pi_A t=a} (1 - p_z(t)) && (7) \\ &= 1 - \prod_{\pi_A t=a} (1 - z_{l(t)} p(t)) && (8) \end{aligned}$$

Let $S^c = \pi_A R \setminus S$. Consider $\mathcal{V}1 = mixture(\mathcal{R}^d, S)$, we will show that $\mathcal{V}1$ is in fact $\mathcal{V}$. Note that all the new lineage variables are added to tuples in S and their probabilities are 1; hence $\mathcal{V}1 = \{(\mathcal{N}(z), (\pi_A R, p''))\}$, where

$$p''(a) = \begin{cases} p^i(a) & a \in S^c \\ \phi(z_{l^d(a)} = 1 | z_{par(l^d(a))}) & a \in S \end{cases} \quad (9)$$

But we have

$$\phi(z_{l^d(a)} = 1 | z_{par(l^d(a))}) = \prod_{\pi_A t = a} (1 - z_{l^i(a)} p_i(t)) \quad (10)$$

$$z_k p_i(t) = 1 - \prod_{\pi_A t = a, l^i(a) = k} (1 - z_k p(t)) \quad (11)$$

$$\text{where } k = l^i(a) \quad (12)$$

Eq. (11) can be verified by setting $z(k) = 0, 1$ on both sides. Eq. (8), (9), (10), (11) show that $p'_z = p''$, and hence proved. □

5.3.3 Join

Before we join two pL-relations, we need to go through an operation *conditioning*, which as the name suggests conditions on a tuple in the relation.

Conditioning This operation takes a pL-relation $\mathcal{R}$ and a tuple $tu \in R$ and returns another pL-relation

$$\text{Cond}(\mathcal{R}, tu) = (R, p', l', \mathcal{N} \overset{\text{n}}{\cup} (w, \emptyset, p(tu), leaf), \text{ where}$$

$$p'(t) = \begin{cases} p(t) & t \neq tu \\ 1 & t = tu \end{cases}, \qquad l'(t) = \begin{cases} l(t) & t \neq tu \\ w & t = tu \end{cases}$$

Example 5.11 Conditioning, written as Cond, is the first operation done in figure 4. R is conditioned on the tuple $(1, 0.5, \epsilon)$. All conditioning does is change the probability of the tuple to 1, assign to it a new lineage x, and then add x as a leaf in the And-Or network N with $P(x) = 0.5$.

Lemma 5.12 *Cond$(\mathcal{R}, tu)$ and $\mathcal{R}$ represent the same distribution.*

Proof: Let ρ_c, ρ denote the probability distributions corresponding to Cond$(\mathcal{R}, tu)$ and $\mathcal{R}$. Then for any world $\omega \subseteq R$

$$\begin{aligned} \rho_c(\omega) &= \sum_{z \in 2^{V(\mathcal{N})}} \mathcal{N}(z) I_{tu \in \omega} p(tu) P_I^{R \setminus \{tu\}}(\omega, z_{l(t)} p(t)) \\ &+ \mathcal{N}(z) I_{tu \notin \omega} (1 - p(tu)) P_I^{R \setminus \{tu\}}(\omega, z_{l(t)} p(t)) \\ &= \sum_{z \in 2^{V(\mathcal{N})}} \mathcal{N}(z) P_I^R(\omega, z_{l(t)} p(t)) \\ &= \rho(\omega) \end{aligned}$$

□

So, in short, the usefulness of conditioning lies in the fact that it makes the tuple deterministic in the table i.e. sets $p'(tu) = 1$. As we know, deterministic tuples never count as offending tuples; hence one way of making offending tuples non-offending is to just *condition* on them. We'll now define join for pL-relations, using operators similar to the ones for join of independent relations. We will again face similar problems as for independent relations i.e the resulting relation may not be *safe* i.e. obey possible worlds semantics. But then we will show that after *conditioning* on the right set of tuples, the join is safe.

Definition 5.13 *Define*

$$\mathcal{R}_1 \bowtie_{pL} \mathcal{R}_2 = (R_1 \bowtie R_2, p^{12}, l^{12}, \mathcal{N}^{12})$$

where the new variables are defined below. Let

$$S = \{(t_1, t_2) \mid l_1(t_1) \neq \varepsilon \wedge l_2(t_2) \neq \varepsilon \wedge t_1 \in \mathcal{R}_1 \wedge t_2 \in \mathcal{R}_2\}$$

and g be a hash function, then

$$l^{12}(t_1 \bowtie t_2) = \begin{cases} g(\{l_i(t_i), p_i(t_i), i = 1, 2\}) & (t_1, t_2) \in S \\ l_1(t_1) & l_2(t_2) = \varepsilon \\ l_2(t_2) & l_1(t_1) = \varepsilon \end{cases}$$

$$p^{12}(t_1 \bowtie t_2) = \begin{cases} p_1(t_1)p_2(t_2) & (t_1, t_2) \notin S \\ 1 & (t_1, t_2) \in S \end{cases}$$

Let

$$V = \{l^{12}(t_1 \bowtie t_2) \mid (t_1, t_2) \in S\}$$
$$E = \{(l^{12}(t_1 \bowtie t_2), l_1(t_1)), (l^{12}(t_1 \bowtie t_2), l_2(t_2)) \mid (t_1, t_2) \in S\}$$
$$Q(l^{12}(t_1 \bowtie t_2), l_i(t_i)) = p_i(t_i)\ i = 1, 2$$

$\mathcal{N}^{12} = (\mathcal{N}_1 \uplus \mathcal{N}_2) \overset{n}{\cup} (V, E, Q, And)$.

$\bowtie_{pL}$ is very much like the independent join operation unless both of the joining tuples have non-trivial lineage i.e not ϵ; in which case we assign a new lineage using hashing and the new probability is 1. Fig 4 has two examples of $\bowtie_{pL}$ operations; none of the joins introduce any new lineage variables.

Now we define $cSet$, which essentially are the set of offending tuples present in each relation.

Definition 5.14 *Define*

$$cSet(\mathcal{R}_1, \mathcal{R}_2) = \{t \in R_1 \mid p_1(t) < 1 \wedge \#\{\{t\} \bowtie R_2\} > 1\}$$

As we would expect, conditioning on this set makes the join *safe*.

Proposition 5.15 $\mathcal{R}_1 \bowtie_{pL} \mathcal{R}_2 = \mathcal{R}_1 \bowtie \mathcal{R}_2$ *if* $cSet(\mathcal{R}_1, \mathcal{R}_2)$ = $cSet(\mathcal{R}_2, \mathcal{R}_1) = \emptyset$.

Proof: (Sketch) From Proposition 3.2, it is easy to see that $c - Set = \emptyset$ implies the join is safe, and hence we again only need to show that $\mathcal{V} = (\mathcal{N}(z), (R_{1z} \bowtie R_{2z}, p'_z))$, where $p'_z(t_1 \bowtie t_2) = p_{1z}(t_1)p_{2z}(t_2)$, is a mixture for $\mathcal{R}_1 \bowtie_{pL} \mathcal{R}_2$. Let

$$S' = \{t_1 \bowtie t_2 \mid (t_1, t_2) \in S\}$$

We can show that $\mathcal{V}$ is the same as $mixture(\mathcal{R}_1 \bowtie_{pL} \mathcal{R}_2, S')$. The proof is very similar to that of theorem 5.10 ; S' captures exactly the set of tuples with new lineage and they all have probability 1. Afterwards its just a routine verification. □

Now observe that $cSet(Cond(\mathcal{R}_1, cSet(\mathcal{R}_1, \mathcal{R}_2)), \mathcal{R}_2) = \emptyset$. This implies that

Theorem 5.16 *Let $\mathcal{R}'_1 = Cond(\mathcal{R}_1, cSet(\mathcal{R}_1, \mathcal{R}_2))$ and $\mathcal{R}'_2 = Cond(\mathcal{R}_2, cSet(\mathcal{R}_2, \mathcal{R}_1))$; then $\mathcal{R}'_1 \bowtie_{pL} \mathcal{R}'_2 = \mathcal{R}'_1 \bowtie \mathcal{R}'_2$*

This tells us that to compute the join of any two pL-relations, it suffices to first condition them on their $cSets$ and then do the join as stated in definition 5.13 Now that we know all operations, Figure 4 walks through our approach on the motivating example in Sec. 4.1, where only one tuple in S violates the FD. To compute $R \bowtie S$, we first condition on the tuple $R(1)$ since there are two tuples in S with $A = 1$. Then we perform $\bowtie_{pL}$. Projection is carried out by first *Independent Project*(IndProj) and then *Deduplication*(Dedup). For the next join, no conditioning is needed as its a 1-1 join. The final probability is given by $\mathcal{N}^0(z = 1)$.

Inference over an And-Or Network

Theorem 5.17 *Let $\mathcal{N}$ be an And-Or network and $G = (V(G), E(G))$ be the directed graph representation of it. Let $\overline{G}$ be the undirected graph obtained by ignoring the direction of edges in G. Then given a tree decomposition of G and any $W \subseteq V(G)$, $x : W \to \{0, 1\}$, the marginal probability $\mathcal{N}^0(x)$ can be computed in time $O(|G|16^{tw(\overline{G})})$.*

Note that [3] has already shown that tree-decomposition for graphs of bounded treewidth can be found in linear time.

Proof: First we state a known fact about treewidth that will enable us to present the algorithm in a simpler way.

Fact 5.18 *Let $tw(G) = k$; then G can be expressed as $G = G_1 \cup G_2$ where G_1, G_2 are two subgraphs of G s.t. $|V(G_1) \cap V(G_2)| <= k$ and $\forall v_1 \in V(G_1) \setminus V(G_2), v_2 \in V(G_2) \setminus V(G_1)$: $(v_1, v_2) \notin E(G)$.*

Let $G = G_1 \cup G_2$ where G_1, G_2 are as in Fact 5.18. Let $G_{12} = G_1 \cap G_2$. For the sake of simplicity we'll assume no leaf nodes by connecting every leaf node to itself ; this way its the parent of itself. It can be treated as And/Or ; does not make a difference. This way P is defined only for edges. Also given $S, z : S \to \{0, 1\}$, we use S_z to denote $\{s \mid z(s) = 1\}$.
Given two graphs H_1, H_2, $H_2 \subseteq H_1$, we define $H_1 \ominus H_2 = (V(H_1), E(H_1) \setminus E(H_2))$. Also let $par_H(v) = \{w \mid (w, v) \in E(H)\}$, for any graph H and $v \in V(H)$. Given any subgraph H of G and $W \subseteq V(H)$; $x, y, z : W \to \{0, 1\}$, define

$$S^{x,y,z}(H, W) = \sum_{\substack{x' \in 2^{V_0(H)} \\ x'_V = x_W}} \Big(\prod_{v \notin W} \phi(x'_v | x'_{par_H(v)}) \tag{13}$$

$$\times \prod_{\substack{v \in W_z \\ par_H(v) \neq \emptyset}} \phi(y_v | x'_{par_H(v)}) \Big) \tag{14}$$

where $V_0(H) = W \cup$

$$\{v \mid v \in V(H), \exists w \ (v, w) \in E(H) \vee (w, v) \in E(H)\}$$

Clearly

$$\mathcal{N}^0(\varphi) = S^{\varphi,\varphi,\mathbf{1}}(G, W)$$

Note that for G, $V_0(G)$ can be assumed to be the same as $V(G)$ since removing isolated vertices does not change the sum ; and after connecting leaves to themselves $par_G(v) \neq \emptyset$ for any v. We will now address the problem of evaluating $S^{x,y,z}(G, W)$ in general for any G, W.
Let $GU = G_{12} \cup W$. We decompose the above sum as :

$$S^{x,y,z}(G, W) = \sum_{a \in 2^{V(G_{12}) \setminus V(W)}} S^{x \uplus a, y \uplus a, z \uplus \mathbf{1}}(G, GU)$$

Note that the conditional probability functions ϕ of And-Or Networks have the property that $\phi(x_v|y_V) = \phi(x_v|y_{V'})\phi(x|y_{V''})$or $1 - \phi(x_v|y_{V'})\phi(x|y_{V''})$ for any two disjoint subsets V', V'' of V. This is the crucial property that we will use in the proof of this theorem and the following lemma.

Lemma 5.19 $S^{x,y,z}(G, GU) =$

$$\sum_{z' \in 2^{V_{0z}(G_{12})}} \Bigg(\prod_{v \in V_{0z}(G_{12})} C(v, y, z') S^{x,y',z''}(G_{12}, G_{12}) \\ \times S^{x,y',z''}(G_1 \ominus G_{12}, GU \cap G_1) S^{x,y',z''}(G_2 \ominus G_{12}, GU \cap G_2) \Bigg)$$

where $V_{0z} = \{v \mid v \in V_z(G_{12}), par_{G \ominus G_{12}}(v) \neq \emptyset\}$*;* $y'_v = 0$ *if* v *is an Or gate else 1 ;* $z''_v = 0$ *if* $z_v = 0$ *else* z'_v*.*

$$C(v, y, z') = \begin{cases} 0 & \text{if } z'_v = 0 \wedge y_v = y_v^{(1)} \\ -1 & \text{if } z'_v = 1 \wedge y_v \neq y_v^{(1)} \\ 1 & \text{otherwise} \end{cases}$$

Proof: For $H1, H2$ subgraphs of G s.t. $V(H2) \subseteq V(H1)$,$x' \in 2^{V(H1)}$,$y, z \in 2^{V(H2)}$, define $T_{H1}(H2, x, y, z) =$

$$\prod_{v \notin H2} \phi(x'_v | x'_{par_{H1}(v)}) \prod_{\substack{v \in H2_z \\ par_{H1}(v) \neq \emptyset}} \phi(y_v | x'_{par_{H1}(v)})$$

We will show that $T_G(GU, x', y, z) =$

$$\sum_{z' \in 2^{V_{0z}(G_{12})}} \Bigg(\prod_{v \in V_{0z}(G_{12})} C(v, y, z') T_{G_{12}}(G_{12}, x', y, z) \tag{15}$$

$$\times T_{G_1 \ominus G_{12}}(GU \cap G_1, x', y, z) T_{G_2 \ominus G_{12}}(GU \cap G_2, x', y, z) \Bigg) \tag{16}$$

The lemma then follows if one sums over both sides such that $x'_{GU} = x$. Note that given x, the sums S over $G_1 \ominus G_{12}$ and $G_2 \ominus G_{12}$ are independent while that over G_{12} is not really a sum, but fixed by the assignments x, y, z. The above equation essentially shows that each term can be broken down into these independent components which when summed over prove the lemma.
We prove equation (15) by induction over G. The base graph has all the nodes, but no edges. This

case is trivial since both sides are 1. Now we add edges by adding every vertex to all of its parents. Lets suppose w was the last vertex to be added to form G and G' be the previous graph in which w was not connected to its parents. We consider two cases :
$\mathbf{w \in V(G_1) \setminus V(G_{12})}$: assume $w \notin GU$. Then $T_G(GU, x'y, z) = \phi(x'_w|x'_{par_G(w)})T_{G'}(GU, x', y, z)$ and $T_{G_1 \ominus G_{12}}(GU \cap G_1, x', y, z) = \phi(x'_w|x'_{par_G(w)})T_{G'_1 \ominus G_{12}}(GU \cap G_1, x', y, z)$ as well. Hence we have both L.H.S and R.H.S are multiplied by the same factor. $w \in GU$ can be similarly proved and the case when $w \in V(G_2) \setminus V(G_{12})$ is also similar.
$\mathbf{w \in G_{12}}$: W.l.o.g assume w is an And node. We will only show the case when $y_w = 0$; the other case is easy to see after this one. If $z_w = 0$, then there is no difference in the sum, so assume $z_w = 1$. Also if $par_{G \ominus G_{12}}(w) = \emptyset$, then this is just like the previous case where only term $T_{G_{12}}$ is affected.
Let $par_G(w) = W^p = W_1^p \uplus W_2^p \uplus W_{12}^p$, where $W_i^p = par_{G_i}w$. Then $\phi(y_w|x'_{W^p}) = 1 - \prod_{j=1,2,12} \phi(y_w|x'_{W_j^p})$.

$$\begin{aligned} \therefore T_G(GU, x', y, z) &= T_{G'}(GU, x', y, z) \\ &- \prod_{j=1,2,12} \phi(y_w|x'_{W_j^p})T_{G'}(GU, x', y, z) \end{aligned}$$

Also note that
$T_{G_{12}}(G_{12}, x', y, z') = T_{G'_{12}}(G'_{12}, x', y, z)\phi(y_w|x'_{W_{12}^p})$ if $z'_w = 1$ and $T_{G'_{12}}(G'_{12}, x', y, z)$ otherwise ; similar equality holds for for $T_{G_2 \ominus G_{12}}$ and $T_{G_1 \ominus G_{12}}$. This case also follows by plugging the values in equation (15), once we observe that $C(v, y, z') = C(v, y, z)$, $z'_w = 0$ and $-C(v, y, z')$ otherwise. □

The above lemma provides an inductive way of calculating S^G using S^{G_1}, S^{G_2}. So one can proceed by dynamic programming and calculate S^G by repeatedly applying fact 5.18. Bodlaender[3] showed that a tree-decomposition can be calculated in linear-time for a graph of constant treewidth; and G_1, G_2 in fact 5.18 can be recursively computed from the tree-decomposition. This proves the theorem. □

5.4 Analysis and Comparisons

In this section we complete the theoretical analysis, that we started in Sec. 4.3. First we prove that only strictly hierarchical queries have lineage of bounded treewidth.

Proof: [of theorem 4.2] We state the following well-known facts without proof.

Fact 5.20 $tw(K_{m \times n}) = min(m, n)$

Fact 5.21 *The treewidth of a graph is at least as big as the treewidth of any of its subgraph.*

Let $Sg(x)$ for any variable x of q be the set of subgoals of q which contain the variable x. We first prove that $tw(F(q, D)) < k$ for any strictly hierarchical query q with k subgoals by induction on k.

$\underline{k = 1:}$ In this case $H(F(q, D))$ has no edges, and hence has treewidth 0.
$\underline{k > 1:}$ By definition 4.1, one can order the variables of q so that $x \prec y$ if $Sg(x) \subset Sg(y)$. Let

$\bar{x}$ be the variables at the top of hierarchy i.e. $\forall y \in Vars(q), x \in \bar{x}\ Sg(y) \subset Sg(x)$ if $y \notin \bar{x}$ else $Sg(x) = Sg(y)$. Then consider a subgoal $R(\bar{x})$ containing only $\bar{x}$ as variables. There has to be a subgoal like this by definition of $\bar{x}$. Let $(X_{\bar{a}}, T_{\bar{a}})$ be the tree-decomposition of $F(q'[\bar{a}/\bar{x}], D),\ \bar{a} \in dom(\bar{x})$ where q' is q with subgoal $R(\bar{x})$ removed. Note that for $\bar{a}, \bar{b} \in dom(\bar{x}), \bar{a} \neq \bar{b}, X_{\bar{a}} \cap X_{\bar{b}} = \emptyset$. Consider $\bigcup(X'_{\bar{a}}, T_{\bar{a}})$, where $X'_{\bar{a}} = \{x \cup \{R(\bar{a})\} | x \in X_{\bar{a}}\}$. Its easy to see that it represents a tree-decomposition for $F(q, D)$, and its width is at most $k-1$ by IH.

Now consider q which is not a strictly hierarchical query. Then let $R(x, \bar{z}, \bar{z}_1)$ and $S(y, \bar{z}, \bar{z}_2)$ be two subgoals of q s.t. $x \notin \bar{z} \cup \bar{z}_2$ and $y \notin \bar{z} \cup \bar{z}_1$ and $\bar{z}_1 \cap \bar{z}_2 = \emptyset$. If $\bar{z} = \emptyset$, then the hypergraph for $F(R(x, \bar{z}_1)S(y, \bar{z}_2), D)$ is just $K_{m \times n}$, where $m = |R(x, \bar{z}_1)|$ and $n = |S(y, \bar{z}_2)|$. The proof follows from facts 5.20. If $z \neq \emptyset$, then consider $F(R(x, \bar{a}, \bar{z}_1)S(y, \bar{a}, \bar{z}_2), D)$ and use 5.21 and the argument above again. □

Now we show that our And-Or Network is actually a minor of the AND-OR factor graph constructed in [25].

Proof: [of Proposition 4.3] (Sketch) Recall that H is a minor of G if a graph isomorphic to H can be obtained from G by contracting some edges, deleting some edges, and deleting some isolated vertices. Lets consider the the three operations one by one. For selection, we only keep the nodes selected, and get rid of the unselected nodes from the graph. A factor graph would make a unary factor out of every node and give non-zero weight to only the selected ones; hence our graph is a minor of the factor graph. For projection, a factor graph would connect the new node(output) representing projected tuple to all the tuples(input nodes) that project to that node. Our network is distinct in two ways (i) we only retain a subset of the input nodes; those that result from the independent project. So we get rid of some input nodes (ii) the output node may be same as an existing node in the network, because of hashing. Again this operation leads to a minor of the factor graph. The join operation follows similarly as well. □

Actually, the treewidth of the And-Or Network would be exactly the same as treewidth of the AND/OR-factor graph(i.e. original directed G and not $M(D(G))$; the complexity of factor graph inference depends on $tw(M(D(G)))$)if it were not for hashing. Note that whenever we create new And/Or nodes, we do so by hashing its parents. This may reduce the treewidth of the graph. For example take the query $q : -R(x)S(x, y)T(y)$, where $R = T = \{a_i, p_i \mid 1 \leq i \leq n\}, S = \{(a_i, a_j, 1) \mid 1 \leq i, j \leq n\}$, and S is deterministic. This is actually in PTIME. But the treewidth of the factor graph is n. This is because the graph can't capture the fact that S is deterministic. Now consider what happens when we project $R \bowtie_x S$ on y. $R \bowtie S = \{(a_i, a_j, x_i, p_i) \mid 1 \leq i, j \leq n\}$, where we have added the lineage x_i to tuples $(a_i, \_)$. Independent Project does nothing ; but after deduplication the result is $\{(a_j, y, 1) \mid 1 \leq j \leq n\}$, where $y = h(\{(x_i, p_i) \mid 1 \leq i \leq n\})$, h is some hash function. Note that all the tuples have the same lineage. After the next join with T, the result is $\{(a_j, y, p_i) \mid 1 \leq j \leq n\}$.All these tuples having the same lineage will be useful as independent project would aggregate them all to $(z, 1)$. At last the And-Or network left would be a tree connecting z to y and y to all $x_i, 1 \leq i \leq n$. This shows how hashing can actually make intractable problems tractable at times. Note that in the earlier example S did not have to be deterministic; even if all its tuples had the same probability, the result would still hold.

Name	Query	Join Order (left-deep plans)
P1/S1	$q(h) : -R_1(h,x), S_1(h,x,y), R_2(h,y)$	R_1, S_1, R_2
P2	$q(h) : -R_1(h,x), S_1(h,x,y), S_2(h,y,z), R_2(h,z)$	R_1, S_1, S_2, R_2
P3	$q(h) : -R_1(h,x), S_1(h,x,y), S_2(h,y,z), S_3(h,z,u), R_2(h,u)$	R, S_1, S_2, S_3, T
S2	$q(h) : -R_1(h,x), T_1(h,x,y,z), R_2(h,y), R_3(h,z)$	R_1, T_1, R_2, R_3
S3	$q(h) : -R_1(h,x), T_2(h,x,y,z,u), R_2(h,y), R_3(h,z), R_4(h,u)$	R_1, T_2, R_2, R_3, R_4

Table 1: Queries and query plans used in experiments.

6 Experiments

Our experiments show that when the query is almost safe, that is there are only a few offending tuples in the dataset, our system has two orders of magnitude time improvements over MayBMS, a state-of-the-art DBMS for probabilistic data. We also verified that as the query plans move from data safety towards increasing data unsafety, our system transits smoothly with a small slope, while it remains at all times competitive with MayBMS.

6.1 Dataset and Queries

Following Proposition 3.2, two parameters are essential to determining the data safety condition: the fraction of offending tuples, i.e., of those tuples violating functional dependencies (FFD), and the fraction of non-deterministic tuples (FDT). Both parameters control how *safe* the given data instance is: this is because offending tuples are always non-deterministic, by making tuples deterministic we remove them from the set of offending tuples.

To generate the data, we consider the parameters N, m, $fanout$, and $r_f, r_d \in [0,1]$. The parameter N gives the size of the final query answer; this is equivalent to executing a boolean query N times on different data instances; m controls the size of *complete* (not partial) lineage. The parameter $fanout$ determines the density of the data, where higher density means higher treewidth.

The tables used in the experiments are as follows:

- Four tables $R_1, \ldots, R_4$ with the same attributes H and A. We set $dom(H) = [N], dom(A) = [m]$ and generate $R_i = \{(h,a) \mid h \in dom(H), a \in dom(A)\}$, $1 \le i \le 4$. The probability of each tuple is set to 1 with probability $1 - r_d$, otherwise a random number in $(0,1)$.

- Three tables $S_i(H,A,B)$ $(1 \le i \le 3)$. The tuples are created as follows: for each $h \in [N], a \in [m]$ with probability $1 - r_f$, we choose at random $b \in [m]$ and add (h,a,b) to S_i; otherwise we choose at random a number f from 2 to $fanout$ and then randomly select $b_1, \ldots, b_f \in [m]$ and add $(h,a,b_j), j \le f$ to S_i. For the sake of uniformity we want $|\sigma_{H=h}S_i|$ to be m and hence the above process is stopped as soon as for any h, we have added m tuples with $H = h$ and we start with $h+1$. Every tuple is non-deterministic here.

- Two tables $T_i(H,A,B,C)$ $(1 \le i \le 2)$. We use the previous construction of S_i to generate $T'_i(H,B,C)$, then generate $T_i(H,A,B,C)$, where for each $h \in [N], a \in [m]$, we choose b, c from $\pi_{B,C}\sigma_{H=h}T'_i$ just as we chose b from $[m]$ in the case of S_i. This process controls the FD violations of $B \to C$ and $A \to B, C$. Here again the tuples are all non-deterministic.

The rationale for this data generation is that it ensures that at most r_f fraction of tuples are offending, i.e., violate the functional dependencies. The tables R_i have r_d fraction of tuples deterministic. If $r_f = 0$ or $r_d = 0$, then the query is safe. The construction above ensures that the size of each relation is exactly $N*m$.

We experiment with Path queries and Star queries. They are listed in Table 1, along with the join-ordering used for their plan.

6.2 Setup

The experiments were run on a Windows Server 2003 machine with Quad Core 2.0GHz and 8GB of RAM. Our implementation was done in Java, wherein the program sends a batch of SQL statements to an SQL Server 2005 database, which in turn writes the resulting output AND-OR network relationally to a temporary table to be read by the same program. A temporary table L contains the description of the network. Each node v is represented by the set of tuples (v, w, p), where w iterates over the parents of v and $p = P((w, v))$. We use ε at place of missing parents. After the execution of SQL commands, the program has an AND-OR network on which it does exact inference.

The competitor, MayBMS, is implemented in C as an extension of the PostgreSQL 8.3 backend. We only considered the MayBMS exact evaluation technique for arbitrary conjunctive queries on c-table-like probabilistic data models, which is detailed in [16]. More specific MayBMS evaluation techniques for safe queries were not considered, for they cannot be employed for our unsafe queries.

In the following, we report wall-clock execution times of queries run from a JDBC link with a warm cache obtained by running a query once and then reporting the second running time.

6.3 Scalability

This experiment demonstrates that when the query is almost safe, our approach scales very well when compared to MayBMS.

We generated the tables with parameters $N = 100$, $m = 10000$, $r_f = 0.01$, $r_d = 1$, $fanout = 4$. This means that we are evaluating the queries on 100 different instances, each with 10,000 conjuncts over tables of size 1M and we report the average execution time per query. There is only 1% of offending tuples. Every tuple is uncertain (has probability less than 1). Figure 5 shows the execution time of our system(Partial Lineage) versus MayBMS. As the lineage becomes more complex, the difference in running time increases. Our execution times are better by an order of magnitude in this setting, which is to be expected as the query plan is almost data safe in this case; MayBMS though cannot recognize and take advantage of this.

6.4 Varying the number of offending tuples

We now measure the influence of the number of offending tuples on query execution time. Here the parameters chosen are $N = 10$, $m = 1000$, $r_d = 1$, $fanout = 3$. The parameter r_f varies from 0 to 1 and r_d is 1 to make sure that the FDT factor does not influence the execution time. We have chosen a smaller scale for this problem because as r_f gets bigger, the problem gets increasingly

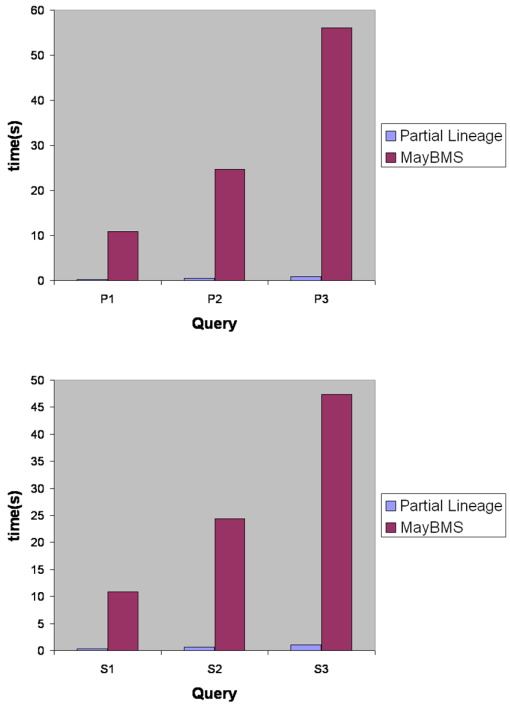

Figure 5: Behaviour for 1% offending tuples.

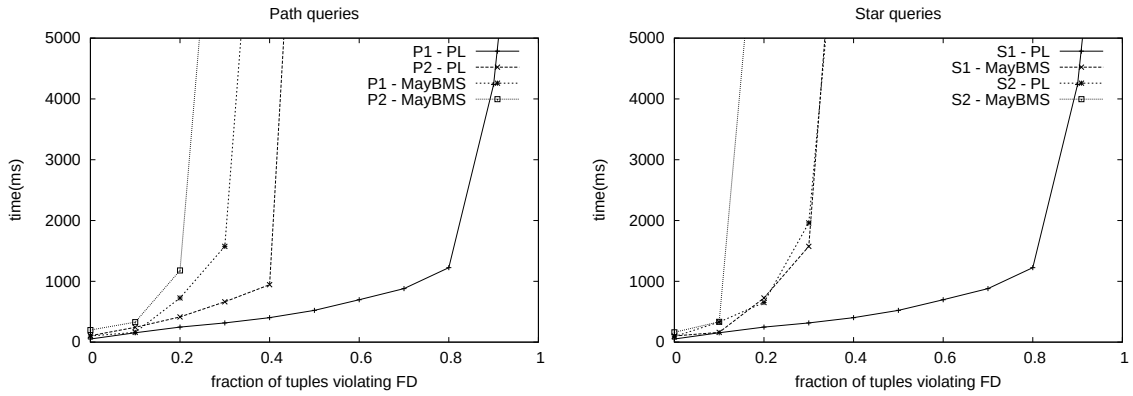

Figure 6: Varying the number of offending tuples.

intractable and execution times shoot up considerably. Here again, we run the query on 10 different instances and report the average running time.

Figure 6 plots the execution time when varying r_f. As we increase r_f, the data becomes denser and the treewidth increases. As shown in the figure, the execution time shoots up considerably at one point; this is the moment when the treewidth has increased to the the point where exact computation is no more feasible or a phase transition has occurred. The realm of exact computation with our method lies only up to this point, and beyond this one must resort to approximate computations. It can be observed, though, that in the tractable region the slope is not very high. This shows that if it were not for the treewidth, our method would have gracefully scaled as the query plans move from data safety to increasing unsafety. The line representing MayBMS follows that of our method, though an additional clear overhead is visible: This may be due to the considered heuristic for variable elimination [16], or additional implementation overhead. MayBMS shoots

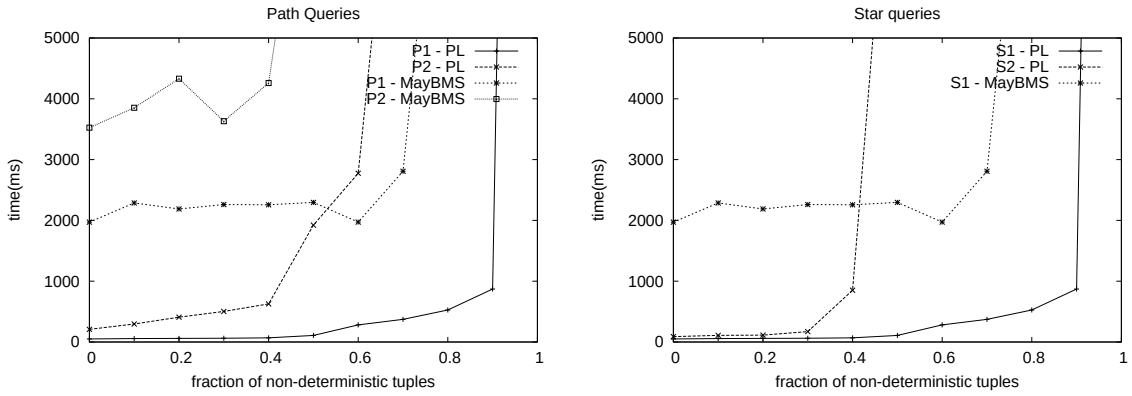

Figure 7: Varying the number of deterministic tuples.

up earlier and the slope increases faster too.

6.5 Varying the number of deterministic tuples

We use the same settings as for the previous experiment, where in addition $r_f = 1$ and r_d varies from 0 to 1. Every tuple is thus offending. For $r_d = 1$, the queries become very hard to compute and both systems fail. We focus instead on the cases where r_d is small, making the problem tractable. We observe that our system performs very well for low values of r_d, as illustrated in Figure 7. For the query S2, MayBMS could not execute any instance in the chosen y-range, hence it does not appear in the plot.

7 Related Work

The problem of query evaluation on probabilistic databases has been well studied in recent years and many different approaches have been proposed. For exact extensional evaluation, [8] showed that *safe* queries can be evaluated completely extensionally by reducing inference to safe plans using standard database operators. [18] later showed that *safe* queries can be evaluated using relational plans with any join ordering, and not only those orderings required by safe plans. Intensional approaches can evaluate any conjunctive query by using general-purpose inference techniques on query lineage [2], or specialized compilation techniques [16, 17]. [25] construct a Bayesian Network, instead of lineage, and evaluate the answer probability of tuples by doing inference over these networks.

Exact evaluation is not always feasible for unsafe queries, and hence many approximation strategies have also been proposed based on sampling [21, 13] or compilation [19]. There are other approximate inference approaches in graphical models[15, 6, 26] that can also be leveraged. Note that these approximation strategies can be used on the And-Or Networks as well. Our method basically reduces the original problem into an inference problem of smaller scale. This means it takes less time to sample the data and more samples mean better approximation.

8 Conclusion and Future Work

This paper proposes a novel approach to query evaluation on probabilistic databases that combines intensional and extensional approaches: It can perform extensional evaluation inside the database even for #P-hard unsafe queries and, if necessary, completes it with intensional computation outside the database. In contrast, existing methods either evaluate extensionally *safe* queries only, or perform the entire probability computation intensionally. We also study the parametrized complexity of our algorithm and give the best theoretical guarantees proposed so far for this problem.

We leave many questions unanswered. It is open how to choose a query plan, that minimizes (i) the size or (ii) the treewidth of output network of the output network. Answering (ii) is crucial, since our algorithm being exponential in treewidth is very sensitive to it. We do not know the query complexity of finding the optimal query plan, i.e., one that has minimum treewidth. For the queries considered in this paper, the plans are the same as the optimal query plans in the traditional sense, but it is not clear whether this is true in general. We also need to extend the approach to self-joins and evaluate queries over more complicated models. In the latter case, we may not be able to do many computations extensionally, and hence this raises the question whether the second stage symbolic evaluation that we currently do outside the database can be converted to database operators. This is particularly advantageous when the scale of the data is huge and treewidth is very small.

Acknowledgements. Abhay Jha and Dan Suciu were partially funded by NSF IIS-0513877. Dan Olteanu acknowledges the financial support of the FP7 European Research Council grant agreement FOX number FP7-ICT-233599.

References

[1] L. Antova, T. Jansen, C. Koch, and D. Olteanu. Fast and simple relational processing of uncertain data. In *ICDE*, pages 983–992, 2008.

[2] O. Benjelloun, A. D. Sarma, A. Y. Halevy, and J. Widom. Uldbs: Databases with uncertainty and lineage. In *VLDB*, pages 953–964, 2006.

[3] H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 226–234, New York, NY, USA, 1993. ACM.

[4] J. Boulos, N. N. Dalvi, B. Mandhani, S. Mathur, C. Ré, and D. Suciu. Mystiq: a system for finding more answers by using probabilities. In *SIGMOD Conference*, pages 891–893, 2005.

[5] R. G. Cowell, S. L. Lauritzen, A. P. David, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[6] P. Dagum and M. Luby. An optimal approximation algorithm for bayesian inference. *Artif. Intell.*, 93(1-2):1–27, 1997.

[7] N. Dalvi and D. Suciu. Management of probabilistic data: foundations and challenges. In *PODS*, pages 1–12, New York, NY, USA, 2007. ACM Press.

[8] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, pages 864–875, 2004.

[9] N. N. Dalvi and D. Suciu. The dichotomy of conjunctive queries on probabilistic structures. In *PODS*, pages 293–302, 2007.

[10] E. Fischer, J. A. Makowsky, and E. V. Ravve. Counting truth assignments of formulas of bounded tree-width or clique-width. *Discrete Applied Mathematics*, 156(4):511–529, 2008.

[11] J. Huang, L. Antova, C. Koch, and D. Olteanu. "MayBMS: A Probabilistic Database Management System". In *Proc. SIGMOD*, 2009.

[12] J. Huang and A. Darwiche. Using dpll for efficient obdd construction. In *SAT*, 2004.

[13] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. M. Jermaine, and P. J. Haas. Mcdb: a monte carlo approach to managing uncertain data. In *SIGMOD Conference*, pages 687–700, 2008.

[14] A. Jha, D. Olteanu, and D. Suciu. Bridging the gap between intensional and extensional query evaluation in probabilistic databases. *EDBT*, 2010.

[15] M. Jordan, Z. Ghahramani, T. Jaakkola, and L. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233, 1999.

[16] C. Koch and D. Olteanu. Conditioning probabilistic databases. *PVLDB*, 1(1):313–325, 2008.

[17] D. Olteanu and J. Huang. Using obdds for efficient query evaluation on probabilistic databases. In *SUM*, pages 326–340, 2008.

[18] D. Olteanu, J. Huang, and C. Koch. Sprout: Lazy vs. eager query plans for tuple-independent probabilistic databases. In *ICDE*, pages 640–651, 2009.

[19] D. Olteanu, J. Huang, and C. Koch. Approximate confidence computation on probabilistic databases. In *ICDE*, 2010.

[20] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[21] C. Re, N. N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, pages 886–895, 2007.

[22] I. Rish. *Efficient reasoning in graphical models*. PhD thesis, UCI, 1999. Chair-Dechter, Rina.

[23] A. D. Sarma, M. Theobald, and J. Widom. Exploiting lineage for confidence computation in uncertain and probabilistic databases. In *ICDE*, pages 1023–1032, 2008.

[24] M. Sauerhoff, I. Wegener, and R. Werchner. Optimal ordered binary decision diagrams for read-once formulas. *Discrete Applied Mathematics*, 103(1-3):237–258, 2000.

[25] P. Sen and A. Deshpande. Representing and querying correlated tuples in probabilistic databases. In *In ICDE*, 2007.

[26] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Generalized belief propagation. In *NIPS*, pages 689–695, 2000.