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

September 16, 2009

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 involve 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, called a safe 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

Query Evaluation for Probabilistic Databases even over independent relations is known to be a #P-hard problem [4]. There is a dichotomy for conjunctive queries without self-joins which says every query is either in PTIME(*safe*) or #P-hard(*unsafe*). There is an efficient way to evaluate the safe queries using just database operators[5, 11]. Unsafe queries, on the other hand, require the probabilistic inference approaches like sampling[13], graphical models [17], DPLL [9] etc that are expensive and don't scale well.

There has been no similar work to translate the evaluation of unsafe queries to database operators. This is because its a #P-complete problem and encompasses problems like probabilistic inference, #SAT etc, for which symbolic approaches and more complicated algorithms are required. Deploying them inside a database may not lead to much benefit. In this paper, we propose an approach that tries to bridge this gap by combining both approaches. In the first stage the evaluation is translated into database operators, but the output is not a value, but a structure that is evaluated symbolically in the second stage. If the query is safe or becomes safe because of the data instance, no work is done in the second stage. In general, the initial problem is reduced to an inference problem that may be much smaller than the original scale of data or just as bad depending on the data. Hence this is the first approach which, to the best of our knowledge, makes a smooth transition from safe queries to unsafe queries. It compares well with safe plans on safe queries and is just as fast as other approaches when the query gets unsafe as well.

We accomplish this by first translating the notion of *safe* from just query to query and data i.e. any query can be safe depending on the data. We'll see that two features of the data: i)deterministic tuples and ii) functional dependencies, determine this safety criterion. To solve the unsafe cases then, just like the intesional approaches, we propose a new model that can capture the correlations that are introduced during query

evaluation. We then extend the existing relational operators over independent relations to work over these models, so that the safe part of the data(deterministic tuples and ones satisfying FDs) is evaluated extensionally and the rest is pushed into the symbolic evaluation to be done after query evaluation. Unsafe queries are #P-hard in general but they are known to be in PTIME when you assume a parameter of the query/data, the *treewidth* to be bounded. We study the parametrized complexity of our approach and propose a parameter that is better than the ones proposed earlier in the literature.
Our approach performs just as well with the best known approaches in the three corner cases : i) completely deterministic data ii) safe query iii)completely symbolic evaluation ; and at the same time transitions smoothly in general in a mixed setting.

2 Background

We first review some basics of query evaluation semantics in probabilistic databases below :
A Probabilistic Relation $\mathcal{R} = (R, \rho)$ represents a probability distribution over all the subsets of R according to $\rho : 2^R \rightarrow [0, 1]$ s.t. $\sum_{\omega \subseteq R} \rho(\omega) = 1$. Assuming ρ is clear from the context, we use $Pr(\phi)$ to denote the marginal probability of a boolean formulae ϕ over tuples of R i.e.$Pr(\phi) = \sum_{\omega \models \phi} \rho(\omega)$.
To evaluate a boolean query q means computing $Pr(q)$. In this paper, we are interested only in conjunctive queries q without self-joins. We further assume that $q \neq q_1 q_2,\ Vars(q_1) \cap Vars(q_2) = \emptyset$, since otherwise q_1,q_2 are independent components and $Pr(q) = Pr(q_1)Pr(q_2)$. We will only be studying the data complexity and hence the size of query is assumed to be bounded throughout the paper.

2.1 Relational Operators on Probabilistic Databases

Definition 2.1 *For any relational operator $\wp$, the semantics given over probabilistic relations are : $\wp(\mathbf{R}, \rho) = (\wp\mathbf{R}, \rho')$ where for any $\omega \subseteq \wp\mathbf{R}$*

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

Independent Relations A relation where all tuples are assumed to be mutually independent, is called an *Independent Relation*. Formally, its a pair (R, p), where $p : R \rightarrow (0, 1]$ and represents the probabilistic relation $\mathcal{R} = (R, \rho)$, where for any $\omega \subseteq R$:

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

Consider the following syntax for projection,join operators on independent relations.

$$\pi(R, p) = (\pi R, p_\pi) \tag{2}$$

$$(R_1, p_1) \bowtie (R_2, p_2) = (R_1 \bowtie R_2, p_\bowtie) \tag{3}$$

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 $t_1 \in R_1$, $t_2 \in R_2$. The great thing about the above operators is that they can be easily rewritten into database operators ; hence query evaluation is very fast and scalable.

Proposition 2.2 *$(R(x, y), p) \bowtie_x (S(x, z), q)$ is an independent relation iff $\forall(a, b) \in R$ if $p(a, b) < 1$ then$|\{c \mid (a, c) \in S\}| \leq 1$. We call $\bowtie$ a 1-1 join if the aforementioned condition is true.*

Proof: The *if* part is easy and follows from the work on independent relations[5]. To prove the other direction assume $p(a, b) < 1$ 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 $p(a, b) = 1$ or 0, a contradiction. $\therefore$ the tuples $U(a, b, c_1)$ and $U(a, b, c_2)$ are not independent. Hence proved. □

[5] proved that equations (2) and (3) hold for a subset of conjunctive queries called *safe* queries. Safe queries, hence have a plan(safe plan) where all joins are 1-1 irrespective of data instance or probability values. Its easy to show that this entails that every join is of the form $R(x) \bowtie_x S(x)$. We can in light of proposition 2.2 extend the defintion of safe plans,queries so that its no longer agnostic of data.

Definition 2.3 *A plan where all joins are 1-1 is called a safe plan. A query that has a safe plan is called a safe query.*

Note that depending on the data, a query may be in PTIME, yet not be safe. For example $R(x)S(x, y)T(y)$ is in PTIME if S is deterministic and the gaifman graph representing S is complete-bipartite. But still there is no plan to evaluate this query with the join and project operators defined in equations (2),(3). So definition 2.3 captures only the queries that can be evaluated by a database plan with the independent join and project operators. [11] have a similar result where they rewrite queries using FDs and queries that can be rewritten into a safe query admit a safe plan. Proposition 2.2 though offers an if and only if criterion.

To solve unsafe queries though, one has to be able to evaluate joins which are not *1-1* and for that we need models more complicated than independent relations as demonstrated by Proposition 2.2. Infact the problem is #P-complete for conjunctive queries ; hence subsequently other approaches that deal with a more general set of models have been proposed. Most of these approaches have been derived from the literature of *probabilistic inference*[12, 2] which has proposed many models for representing correlations/dependencies along with the algorithms to infer probability of any query. We talk about these approaches below :

2.2 Intensional Evaluation

In this paper we frequently use terms like intensional and symbolic evaluation to mean the same thing : inference techniques that proceed by some sort of symbolic manipulation i.e. where the resulting probability can't be just expressed as a function of the

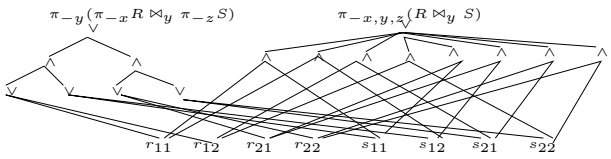


Figure 1: Factor graphs for $q = R(x, y)S(y, z)$ in example 2.5

input probabilities of the tuples as in the former case of safe queries. We classify the existing approaches by the complexity of their model into two categories :

1. *queries as dnf-formula:*

   Definition 2.4 Lineage:*Given a boolean conjunctive query q over database D, we denote by $F(q, D)$, the dnf formulae one would get by grounding q with tuples from D. Its common in literature to use the term lineage for $F(q, D)$.*

   Example 2.5 Let $q = R(x, y)S(y, z)$ and $R = \{(1, 1), (1, 2), (2, 1), (2, 2)\}, S = \{(1, 1), (1, 2), (2, 1), (2, 2)\}$; then $F(q, D) \equiv r_{11}s_{11} \vee r_{11}s_{12} \vee r_{12}s_{21} \vee r_{12}s_{22} \vee r_{21}s_{11} \vee r_{21}s_{12} \vee r_{22}s_{21} \vee r_{22}s_{22}$.

   These approaches[16, 9] have the same complexity of evaluating a boolean query as counting the number of assignments of a boolean formulae expressed as $F(q, D)$. The way in which they evaluate it or come to this formulae may be different though. For example one may choose to use bayesian networks or #SAT techniques, but the underlying complexity would be determined by the structure of the dnf-formula.

2. *factoring lineage using query plans:* Consider example 2.5 ; $F(q, D)$ may be rewritten as $((r_{11} \vee r_{21}) \wedge (s_{11} \vee s_{12})) \vee ((r_{12} \vee r_{22}) \wedge (s_{21} \vee s_{22}))$. Its easy to see that this representation is more efficient than the dnf representation. And in fact one gets this representation if q is thought of as $R'(y)S'(y)$ where $R'(y) = \pi_y R(x, y)$ and $S'(y) = \pi_y S(x, y)$. [17] exploit this by modeling queries as *factor graphs*, where there are two kinds of factors : 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 2.5 for two different plans.

The exact algorithms vary from sampling to message passing,etc., but these algorithms are very general and could be used for both models. So we don't delve into the algorithms, rather into the model that is fed into these algorithms. As we will show in the next section, the second model can theoretically outperform first model by any arbritary margin.

3 Motivation

In this section, we outline two weaknesses in the existing approaches that we wish to address. The first one expresses the need for a system with both extensional and intensional techniques and second one shows how different ways to model query in section 2.2 differ.

3.1 Unsafe Query via Safe Plans

Let $q_u : -R(x)S(x,y)T(y)$. Its known to be an *unsafe* query.

Now let $R = \{a_i | 1 \leq i \leq n\}$,$S = \{(a_i, b_i) | 1 \leq i \leq n\}$,$T = \{b_j | 1 \leq j \leq n\}$. By Proposition 2.2 q_u can be evaluated completely extensionally by a safe plan. So the existing systems could decide to evaluate q_u by a safe plan in this case. Now add the tuple (a_1, b_n) to S. The query is still *almost* safe ; so does it make sense to use an intensional approach. In fact we can show that

Proposition 3.1

$$Pr(q_u) = p(a_1)P_1 + (1 - p(a_1))P_2$$

where P_1,P_2 *are* $Pr(q_u)$ *evaluated over the databases with* $p(a_1) = 1$ *and* $R(a_1)$ *missing respectively.*

Proof:

$$\begin{aligned} Pr(q_u) &= Pr(q_u | R(a_1))p(a_1) + Pr(q_u | \neg R(a_1))(1 - p(a_1)) \\ &= p(a_1)P_1 + (1 - p(a_1))P_2 \end{aligned}$$

□

By Proposition 2.2, we have that both P_1 and P_2 can be evaluated efficiently by safe plans. When n is very large, there can be an order of magnitude difference between evaluating q_u above intensionally and evaluating it by decomposing into two safe plans as shown.

3.2 Complexity of Counting Conjunctive queries

The complexity of probabilistic inference algorithms viz. variable elimination, junction-tree that have been employed by the other intensional approaches depend intimately on a parameter of the input structure called treewidth. These algorithms have running time exponential in treewidth and hence they are in PTIME iff treewidth is bounded.

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 $tw(H)$ is the minimum width among all possible tree-decompositions of H.
The treewidth of a dnf-formulae $\mathcal{F}$ $tw(\mathcal{F})$ is the treewidth of the associated hypergraph $H(\mathcal{F})$. We state the following well-known facts without proof.

Fact 3.2 $tw(K_{m\times n}) = min(m, n)$

Fact 3.3 *The treewidth of a hypergraph is at least as big as the treewidth of any of its subgraph.*

Now we are ready to characterise the treewidth of lineage $F(q, D)$ of a conjunctive query q.

Definition 3.4 *A CQ q is called* strictly hierarchical *if q can be written as* $R_1(\bar{x}_1) \ldots R_m(\bar{x}_m)$ *s.t.* $\bar{x}_1 \subseteq \ldots \bar{x}_m$.

Theorem 3.5 *If q is a strictly hierarchical query then* $\forall D\ tw(F(q, D)) < c$ *for some constant c. Otherwise* $\forall c\ \exists D$ *s.t.* $tw(F(q, D)) > c$.

Proof: 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 3.4, 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. 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 $\exists x, y \in Vars(q)$ s.t. $Sg(x) \cap Sg(y) = \emptyset$. Let $R(x, \bar{z}, \bar{z}_1)$ and $S(y, \bar{z}, \bar{z}_2)$ be two subgoals of q s.t. $x \notin z \cup z_2$ and $y \notin z \cup z_1$ and $z_1 \cap z_2 = \emptyset$. 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 3.2 and 3.3. If $z \neq \emptyset$, then consider $F(R(x, \bar{a}, \bar{z}_1)S(y, \bar{a}, \bar{z}_2), D)$ and use the argument above again. □

So this shows that only a subset of the safe or hierarchical queries have lineage of bounded treewidth. But what is more important to observe is 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.
Now the first class of intensional methods as mentioned work with this model. Hence this class of intensional methods can't even compute safe queries in PTIME. Of course one can use safe plans or other heuristics to simplify this lineage; [10] for example observe that safe plans produce lineage that are of a particular form called 1OF. But these don't generalize to unsafe queries and this motivates the need for a general model that would be in PTIME for safe queries.
Now lets consider the second model of factor graphs. The factor graph produced by any safe plan is a tree. The inference algorithm used by [17] though first transforms the factor graph into some other graph by decomposing each factor into factors of size at most 3. Then the graph is moralized by connecting the parents, and the complexity of inference depends on the treewidth of this graph. For safe queries, this graph has treewidth at most two. But in general, we don't know if the treewidth of this graph can be bounded as a function of the treewidth of original factor graph and in fact different transformations of the same graph may lead to different treewidth graphs [15]. We settle this issue by proving that a boolean conjunctive query can be evaluated in PTIME if there is a query plan whose corresponding factor graph has bounded treewidth. Our aim now is to propose an approach that combines both goals listed in this section i.e. pushes as much evaluation as possible into the database engine like in safe plans, but it should be in PTIME if the treewidth of the factor graph is bounded.

4 Our Approach

And-Or Network An And-Or network $\mathcal{N} = (V, E, P, Lab)$ can be represented as a directed acylic graph(DAG) $G = (V, E)$, where $Lab : V \rightarrow \{And, Or, Leaf\}$; $P : E \cup V_L \rightarrow [0, 1]$, where $V_L = \{v \mid v \in V, \nexists w \in V \; s.t. \; (w, v) \in E\}$. We call these nodes in V_L as the leaves of G. $Lab(V_i) = Leaf$ iff its 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

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

The And-Or Network $\mathcal{N}$, then 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 $\mathcal{N}$ is a special case of *Bayesian Networks* and hence represents a valid probability distribution. 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.

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 $P' \in [0, 1]$; 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.

pL-Relation : Relations with Partial Lineage 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

$$\rho(\omega) = \sum_{z : V(\mathcal{N}) \to \{0,1\}} \mathcal{N}(z) \prod_{t \in \omega} z(l(t))p(t) \prod_{t \notin \omega} (1 - z(l(t))p(t)) \tag{4}$$

We often use $t \in \mathcal{R}$ to mean $t \in R$. Its easy to show that ρ is a probability distribution, as will be clear from the following discussion.

To understand this definition better, lets 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. The lineage for every tuple is ε. Hence equation (4) can be seen to be the same as equation (1), the probability distribution of the independent relation (R, p). $\mathcal{R}$ therefore is an independent relation in this case.

On the other hand, assume every tuple has a different lineage l and $\forall t\ p(t) = 1$; then $\rho(\omega) = \mathcal{N}(z)$, where $z(l(t)) = \mathbf{1}_{t \in \omega}$. Hence in this case the relation just represents an And-Or Network $\mathcal{N}$. Between these two extremes(an And-Or network,independent relation) as we discuss below, the pL-relation represents a combination of many independent relations each weighted according to the probability distribution of an And-Or network.

For each z, $\rho(\omega)$ is $\mathcal{N}(z)$ times the probability of the world ω in an independent relation $\mathcal{R}_z = (R, p_z)$, where $p_z(t) = z(l(t))p(t)$. Let ρ_z be the distribution function for $\mathcal{R}_z$, then

$$\rho(\omega) = \sum_{z : V(\mathcal{N}) \to \{0,1\}} \mathcal{N}(z)\rho_z(\omega)$$

where $\sum_{z : V(\mathcal{N}) \to \{0,1\}} \mathcal{N}(z) = 1$. Hence $\sum_\omega \rho(\omega) = 1$.

So a pL-relation compactly represents a convex combination of many independent relations $\mathcal{R}_z$. Such distributions are often called *mixture models* or *hierarchical models* in

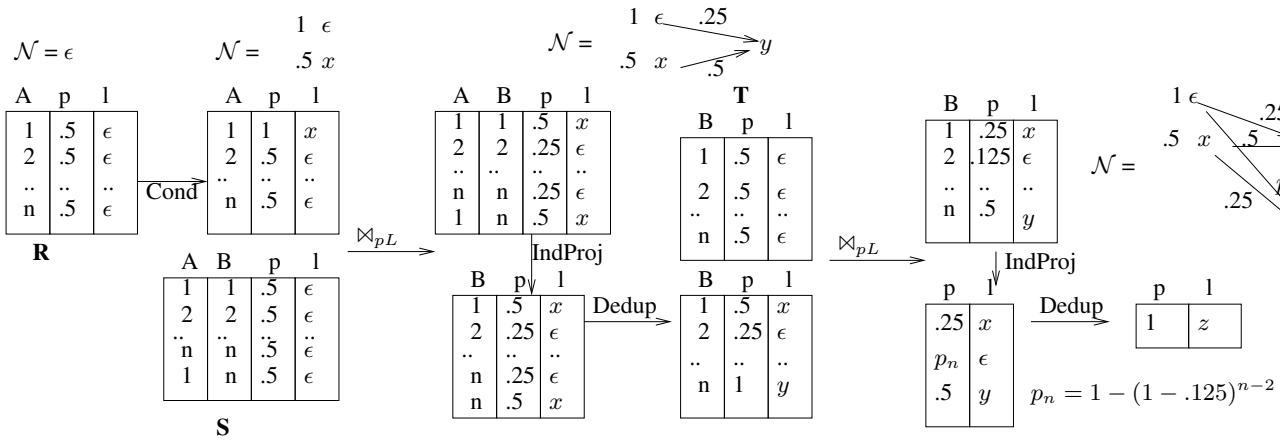


Figure 2: Evaluating $q : -R(x)S(x,y)T(y)$. $Pr(q)$ is given by $\mathcal{N}^0(z=1)$. P values for leaves and edges in $\mathcal{N}$ are written besides them.

AI and statistics. The idea is to first pick one distribution(R_z) with some probability($\mathcal{N}(z)$) and then choose a world(ω) from the picked distribution(according to ρ_z). As we already learnt from proposition 2.2, the independent relation model is not enough to represent the intermediate distributions resulting from relational operators. We defined pL-relation because as we will see, this can compactly represent the intermediate distributios and is still closed with respect to relational operators.

4.1 Relational Operators over pL-relations

4.1.1 Selection

Given a pL-relation $\mathcal{R} = (R, p, l, \mathcal{N})$, and $A \subseteq attribs(R)$, we will show that $\sigma_{A=a}\mathcal{R} = \mathcal{R}' = (\sigma_{A=a}R, p, l, \mathcal{N})$. Let ρ and ρ^s denote the probability distributions of $\mathcal{R}$ and $\mathcal{R}'$ respectively. Then

$$\begin{aligned}\sum_{\omega,\sigma\omega=\omega'} \rho(\omega) &= \sum_{z:V(\mathcal{N})\to\{0,1\}} \mathcal{N}(z) \sum_{\omega,\sigma\omega=\omega'} \rho_z(\omega) \\ &= \sum_{z:V(\mathcal{N})\to\{0,1\}} \mathcal{N}(z)\rho_z^s(\omega') \\ &= \rho^s(\omega')\end{aligned}$$

The equality between first and second equations follows from the semantics of selection over independent relations. Hence $\mathcal{R}' = \mathcal{R}$ by definition 2.1.

4.1.2 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.

1. **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. The result in not a valid pL-relation since we could have duplicate tuples, but we still abuse the notation to represent this intermediate distribution by a pL-relation by appending the lineage as a column L inside the relation. Let $A' = A \cup \{l\}$. We compute $\mathcal{R}' = (\pi_{A'} R, p', l', \mathcal{N})$; where $l'(a) = a.L$, $p'(a) = 1 - \prod_{\pi_{A'} t = a}(1 - p(t))$.
   Figure 2 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 this operation doesn't change $\mathcal{N}$.

2. **Deduplication** Now we describe how to calculate $\pi_A \mathcal{R}'$, by removing duplicate tuples. Define $Pj = \{(a, a') \mid \pi_A a' = a, a' \in \mathcal{R}'\}$. Let $S = \{a \mid \#\{a' | Pj(a, a')\} > 1\}$. Define $l''(a) = l'(a)$ if $a \notin S$ else $h(\{(l'(a'), p'(a')) | Pj(a, a')\})$, where h is some hash function ; $E = \{(l''(a), l'(a')) \mid Pj(a, a'), a \in S\}$ and $Q(l''(a), l'(a')) = p'(a')$. Let $V = \{l''(a) \mid a \in S\}$. Define an And-Or network $\mathcal{M} = \mathcal{N} \overset{\cdot}{\cup} (V, E, Q, Or)$.
   Formally $\mathcal{R}'' = (\pi_A R, p'', l'', \mathcal{M})$, where

$$p''(a) = \begin{array}{l} p'(a) \text{ if } a \in S \\ 1 \text{ otherwise} \end{array}$$

Theorem 4.1 $\mathcal{R}'' = \pi_A \mathcal{R}$

Proof: Let ρ and ρ'' denote the probability distributions of $\mathcal{R}$ and $\mathcal{R}''$ respectively. Then

$$\sum_{\omega, \pi\omega = \omega'} \rho(\omega) = \sum_{z: V(\mathcal{N}) \to \{0,1\}} \mathcal{N}(z) \sum_{\omega, \pi\omega = \omega'} \rho_z(\omega) \tag{5}$$

$$= \sum_{z: V(\mathcal{N}) \to \{0,1\}} \mathcal{N}(z) \prod_{t \in \omega'} p^p(t) \prod_{t \notin \omega'} (1 - p^p(t)) \tag{6}$$

where $p^p(t) = 1 - \prod_{\pi t' = t}(1 - z(l(t'))p(t'))$. This follows from the projection semantics over independent relations. Note that if $t \notin S$, then $\exists k s.t. \forall t' \pi t' = t \; z(t') = k$. $\therefore p^p(t) = 1 - \prod_{\pi t' = t}(1 - z(k)p(t')) = z(k)\,(1 - \prod_{\pi t' = t}(1 - p(t')))$. This can be

easily verified by setting $z(k)$ to 0/1 and comparing both sides. Now

$$\rho''(\omega') = \sum_{z'':V(\mathcal{M})\to\{0,1\}} \mathcal{M}(z'') \prod_{t\in\omega'} z''(t)p''(t) \prod_{t\notin\omega'} (1-z''(t)p''(t)) \quad (7)$$

$$= \sum_{z:V(\mathcal{N})\to\{0,1\}} \mathcal{N}(z) \prod_{t\in\omega'} p'''(t) \prod_{t\notin\omega'} (1-p'''(t)), \text{ where} \quad (8)$$

$$p'''(t) = \begin{cases} z(l''(t'))\,(1-\prod_{\pi t'=t}(1-p(t'))) & t\notin S \\ \phi(z_{l(t)}=1|z'''_{par(l(t))}) & t\in S \end{cases} \text{ and} \quad (9)$$

$$\phi(z_{l(t)}=1|z'''_{par(l(t))}) = 1-\prod_{\pi t'=t}(1-z(l(t'))p(t')) \quad (10)$$

$$= p^p(t) \quad (11)$$

Equation (8) follows from (7) since for $t\in S$, $p''(t)=1$, hence $z(l(t))$ can be assumed to be $\omega|_t$ for the new lineage variables which are all from S. From equations (6),(8) the proof follows. □

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

$$p'(t) = \begin{cases} p(t)\ t\neq tu \\ 1\ t=tu \end{cases}$$

$$l'(t) = \begin{cases} l(t)\ t\neq tu \\ w\ t=tu \end{cases}$$

Conditioning, written as Cond, is the first operation done in figure 2. 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 4.2 *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 $\omega \subseteq R$

$$\begin{aligned}
\rho_c(\omega) &= \sum_{z:V\to\{0,1\}} \mathcal{N}(z) I_{tu\in\omega} p(tu) \prod_{t\in\omega, t\neq tu} p(t) z(l(t)) \prod_{t\notin\omega} (1-p(t)) \\
&+ \mathcal{N}(z) I_{tu\notin\omega} (1-p(tu)) \prod_{t\in\omega} p(t) \prod_{t\notin\omega, t\neq tu} (1-p(t)) \\
&= \sum_{z:V\to\{0,1\}} \mathcal{N}(z) \prod_{t\in\omega} p(t) z(l(t)) \prod_{t\notin\omega} (1-p(t)) \\
&= \rho(\omega)
\end{aligned}$$

□

Definition 4.3 *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*

$$\begin{aligned}
l^{12}(t_1 \bowtie t_2) &= \begin{cases} g\left(l_1(t_1), l_2(t_2), p_1(t_1), p_2(t_2)\right) & (t_1,t_2) \in S \\ l_1(t_1) & l_2(t_2) = \varepsilon \\ l_2(t_2) & l_1(t_1) = \varepsilon \end{cases} \\
p^{12}(t_1 \bowtie t_2) &= \begin{cases} p_1(t_1)p_2(t_2) & (t_1,t_2) \notin S \\ 1 & (t_1,t_2) \in S \end{cases}
\end{aligned}$$

Let $V = \{l^{12}(t_1 \bowtie t_2) \mid (t_1, t_2) \in S\}$ *;*
$E = \{(l^{12}(t_1 \bowtie t_2), l_1(t_1)), (l^{12}(t_1 \bowtie t_2), l_2(t_2)) \mid (t_1, t_2) \in S\}$ *;*
$\forall e = (l^{12}(t_1 \bowtie t_2), l_1(t_1)) \in E$ *define* $Q(e) = p_1(t_1)$ *and vice-versa.* $\mathcal{N}^{12} = (\mathcal{N}_1 \uplus \mathcal{N}_2) \overset{n}{\cup} (V, E, Q, \bar{A}nd)$.

Definition 4.4 *Define* $c - Set(\mathcal{R}_1, \mathcal{R}_2) = \{t \in R_1 \mid p_1(t) < 1 \wedge \#\{\{t\} \bowtie R_2\} > 1\}$

Proposition 4.5 $\mathcal{R}_1 \bowtie_{pL} \mathcal{R}_2 = \mathcal{R}_1 \bowtie \mathcal{R}_2$ *if* $c{-}Set(\mathcal{R}_1, \mathcal{R}_2) = c{-}Set(\mathcal{R}_2, \mathcal{R}_1) = \emptyset$.

Proof: Let ρ^1, ρ^2, ρ denote the probability distributions of $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_1 \bowtie_{pL} \mathcal{R}_2$. Let $\mathcal{N} = \mathcal{N}_1 \uplus \mathcal{N}_2$, $V = V(\mathcal{N})$. Then

$$\begin{aligned}
\sum_{\omega_1 \bowtie \omega_2 = \omega} \rho^1(\omega_1)\rho^2(\omega_2) &= \sum_{z:V\to\{0,1\}} \mathcal{N}(z) \sum_{\omega_1\bowtie\omega_2=\omega} \rho^1_z(\omega_1)\rho^2_z(\omega_2) \qquad (12) \\
&= \sum_{z:V\to\{0,1\}} \mathcal{N}(z) \prod_{t\in\omega} p^j(t) \prod_{t\notin\omega} (1-p^j(t)) \text{ where} \qquad (13) \\
p^j(t) &= z(l_1(t_1))p_1(t_1)z(l_2(t_2))p_2(t_2) \qquad (14)
\end{aligned}$$

We can assume that $z(\varepsilon) = 1$, since otherwise the probability is 0($P(\varepsilon) = 1$). So if $(t_1, t_2) \notin S$, then its easy to see that $p^j(t) = z(l^{12}(t))p^{12}(t)$(just enumerate the cases

when either or both of lineage is ε). Now lets analyze ρ.

$$\rho(\omega) = \sum_{z:V(\mathcal{N}^{12})\to\{0,1\}} \mathcal{N}^{12}(z) \prod_{t\in\omega} z(l^{12}(t))p^{12}(t) \prod_{t\notin\omega}(1 - z(l^{12}(t))p^{12}(t)) \quad (15)$$

$$= \sum_{z:V(\mathcal{N})\to\{0,1\}} \mathcal{N}(z) \prod_{t\in\omega} p^z(t) \prod_{t\notin\omega}(1 - p^z(t)) \text{ where} \quad (16)$$

$$p^z(t) = \begin{cases} z(l^{12}(t))p^{12}(t) & t = t_1 \bowtie t_2, (t_1,t_2) \notin S \\ \phi(z_{l(t)} = 1 | z_{par(l^{12}(t))}) & \text{otherwise} \end{cases} \quad and \quad (17)$$

$$\phi(z_{l(t)} = 1 | z_{par(l^{12}(t))}) = z(l_1(t_1))p_1(t_1)z(l_2(t_2))p_2(t_2) \quad (18)$$

$$(19)$$

The proof then follows from equations (13) and (16). □

This tells us that to compute the join of any two pL-relations, it suffices to first condition them and do the join as stated in definition 4.3.
Note that the And-Or network remains a DAG after each of the above operations. Figure 2 illustrates our approach on the motivating example in section 3.1. 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 $\mathcal{N}^0(z = 1)$.

4.2 Inference over an And-Or Network

Theorem 4.6 *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 any $W \subseteq V(G)$ and $\varphi : W \to \{0,1\}$, the marginal probability $\mathcal{N}^0(\varphi)$ can be computed in time $O(V(G))$ if $tw(\overline{G}) < k$ for some constant k.*

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

Fact 4.7 *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 4.7. 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 ; doesn't 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

$$S^{x,y,z}(H, W) = \sum_{\substack{x':V_0(H)\to\{0,1\} \\ x'_V = x_W}} \prod_{v\notin W} \phi(x'_v | x'_{par_H(v)}) \prod_{\substack{v\in W_z \\ par_H(v)\neq\emptyset}} \phi(y_v | x'_{par_H(v)}) \quad (20)$$

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 doesn't 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 :

$$S^{x,y,z}(G,W) = \sum_{a:V(G_{12})\setminus V(W)\to\{0,1\}} S^{x\uplus a, y\uplus a, z\uplus \mathbf{1}}(G, GU) \tag{21}$$

$$\tag{22}$$

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 4.8

$$S^{x,y,z}(G,GU) = \sum_{z':V_{0z}(G_{12})\to\{0,1\}} \Big(\Big(\prod_{v\in V_{0z}(G_{12})} C(v,y,z') \Big) S^{x,y',z''}(G_{12}, GU \cap G_{12}) \tag{23}$$

$$\times\ S^{x,y',z''}(G_1 \ominus G_{12}, GU \cap G_1) S^{x,y',z''}(G_2 \ominus G_{12}, GU \cap G_2) \Big) \tag{24}$$

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. $V(H2) \subseteq V(H1)$,$x' : V(H1) \to \{0,1\}$,$y, z : V(H2) \to \{0,1\}$, 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

$$T_G(GU, x', y, z) = \sum_{z':V_{0z}(G_{12})\to\{0,1\}} \prod_{v\in V_{0z}(G_{12})} C(v,y,z') T_{G_{12}}(G_{12}, x', y, z) \tag{25}$$

$$\times\ T_{G_1\ominus G_{12}}(GU \cap G_1, x', y, z) T_{G_2\ominus G_{12}}(GU \cap G_2, x', y, z) \tag{26}$$

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 (25) 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 wasn't connected to its parents. We consider two cases

1. $w \in V(G_1) \backslash V(G_{12})$: 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.

2. $w \in G_{12}$: 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 $\phi(y_w | x'_{W^p}) = 1 - \prod_{j=1,2,12} \phi(y_w | x'_{W_j^p})$. $\therefore T_G(GU, x', y, z) = T_{G'}(GU, x', y, z) - T_{G'}(GU, x', y, z) \prod_{j=1,2,12} \phi(y_w | x'_{W_j^p})$.
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 (25), 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 4.7. □

4.3 Analysis and Comparisons

In section 3.2, we claimed that a query can be evaluated in PTIME if there is plan for which the factor graph has bounded treewidth. This actually follows from theorem 4.6, since the factor graphs are a special case of And-Or networks. But we want to analyze the complexity of our approach for which we need to find the treewidth of the And-Or network generated by our algorithm. We will show that it is at most the treewidth of the factor graph. It could be lot lesser in size as demonstrated in figure 2 ; and also lesser in treewidth because of hashing, as we will show later on.

Proposition 4.9 *Given a query plan, let G be the And-Or Network output by our algorithm and F be the factor graph; then $tw(G) \leq tw(F)$.*

Proof: Consider any operator(selection/projection/join). Its easy to observe from the algorithm that the graph constructed by our approach is actually a subgraph of the factor graph. From Fact 3.3, the result follows. □

Actually, the treewidth of the two approaches would be exactly the same if it weren't 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 as we briefly mentioned in section 2.1. 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 all these tuples having the same lineage will be useful in independent project. At last the And-Or network left would be a tree connecting 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.

5 Experiments

With Proposition 2.2 and Definition 2.3, we shifted the definition of *safety* from merely query to both query/data. The main contribution of this paper is to be able to evaluate efficiently not only the safe queries but also those which are *almost safe* i.e. when there are very few tuples which violate the 1-1 join condition expressed in Proposition 2.2. We define the following two criterion to quantify how *safe* the query/data is :

1. fraction of tuples violating FD(FFD) : Note that $R(x,z) \bowtie_x S(x,y)$ is a 1-1join if $x \to y, x \to z$ hold on S, R respectively ; $R(x) \bowtie_x S(x,y)$ is 1-1join if $x \to y$ held on S. Given a query plan, we find out the fraction of tuples that violate this condition. For e.g. $|\{a \mid a \in R \wedge \exists b_1, b_2\ b_1 \neq b_2 \wedge (a,b_1), (a,b_2) \in S\}|/|R|$ in case of $\pi_{-y}(T(y) \bowtie_y \pi_{-x}(R(x) \bowtie_x S(x,y)))$. Note that we count the tuples in R and not in S; so $|\{y \mid (a,y) \in S\}|$ could be 2 or 20, but the fraction of tuples violating FD would be same. When this number is 0, query plan is safe; as more and more tuples violate this condition, we expect the computation to get more Intensional.

2. fraction of non-deterministic tuples(FDT) : $R(x) \bowtie_x S(x,y)$ is infact also a 1-1join if R is deterministic. Given a query plan, we can then find the set of tuples that we expect to be deterministic for the plan to be safe. This metric measures the fraction of those tuples that are not deterministic. When this ratio is 0, the plan is again safe and as it appraoches 1, the evaluation gets more intensional.

We will measure the performance of our algorithm vs another system MayBMS[8] as we vary these two metrics.

DataSet/Queries We will experiment with Path queries and Star queries. They are listed in table 1, along with the join-ordering used for their plan. To generate the data,

Name	Query	Join Order
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: Query/Plans used for experiments

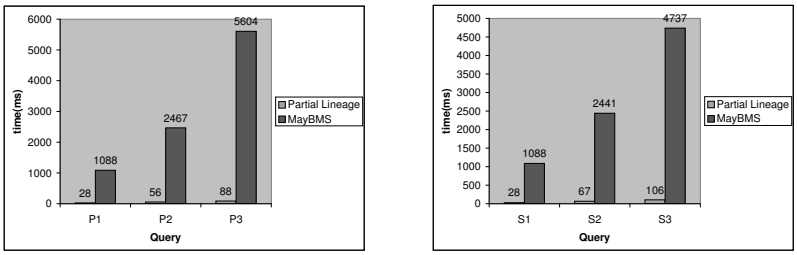
we take the parameters $r_f, r_d \in [0,1]$,fanout,N,m and generate the tables as follows :
$\mathbf{R_i}$: All tables $R_i, i \leq 4$ have the same schema, say $R_i(H, A)$. Then we set $dom(H) = [N], dom(A) = [m]$ and generate R_i accordingly. The probability of each tuple is set to 1 with probability $1 - r_d$, otherwise anything in $(0, 1)$.
$\mathbf{S_i}$: For any $S_i(H, A, B)$, we add tuples iteratively 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 radom a number f from 2 to fanout and then radomly select $b_1, \ldots, b_f \in [m]$ and add $(h, a, b_j), j \leq f$ 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.
$\mathbf{T_i}$: We illustrate for $T_1(H, A, B, C)$ and T_2 follows similarly. Use the previous construction of S_i to generate $T_1'(H, B, C)$. And then generate $T_1(H, A, B, C)$, where for each $h \in [N], a \in [m]$, we choose b, c from $\pi_{B,C}\sigma_{H=h}T_1'$ just as we chose b from $[m]$ in the case of S_i. This process controls the FD violations of $B \rightarrow C$ and $A \rightarrow B, C$. Here again the tuples are all non-deterministic.
The rationale of generating the data as discussed above is that it ensures that at most r_f fraction of tuples violate the functional dependencies that make the query safe; and all the base tables R_j have r_d fraction of tuples deterministic. If $r_f = 0$ or $r_d = 0$, then the query is safe. N controls the number of boolean queries executed while m controls the size of each query. The construction above ensures that the size of each relation is actually exactly $N*m$. fanout is a parameter that determines how dense the data is; the extensional part isn't affected much by this choice, but dense data means higher treewidth and so to keep problem tractable, we want small fanout. On the other hand we don't want the data to be too sparse, so that each query is essentially just many small queries and hence m loses its meaning. So fanout is chosen experimentally for the right balance.

Setup The experiments were run on a Windows Server 2003 machine with Quad Core 2.0GHz, 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 temp table read by the same program. So after the execution of sql commands, the program has an And-Or Network on which it does exact inference using its tree-decomposition. The program to do inference isn't very optimized, since optimizing this part isn't the focus of our paper.

5.1 Scalability

In the first experiment we wanted to demonstrate that when the query is almost safe, then our approach will scale while other existing approaches can't. So we generated the tables with $N = 100, m = 10000, r_f = 0.01, r_d = 1, fanout = 4$. This means that we are evaluating 100 boolean queries, each with 10,000 clauses over tables of size 1M. There is only 1% of tuples that violate FDs though. We are not using determinism at all, and every tuple is non-deterministic. Figure 5.1 shows the execution time of our system(Partial Lineage) vs MayBMS. Note that as the queries get more complex, the difference in running time gets even bigger. Our execution times are better by an order of magnitude in this condition, which is to be expected as the query is almost safe in this case; MayBMS though cannot recognize and take advantage of this.

Figure 3: Execution time with 1% FD violations

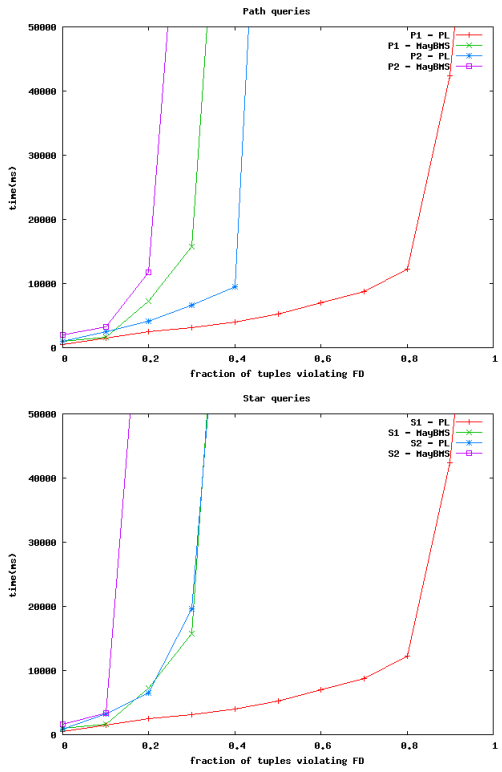


5.2 Effect of FFD

We now want to measure the influence of FFD on execution time. Here the parameters chosen were $N = 10, m = 1000, r_d = 1, fanout = 3$. r_f was varied from 0 to 1 and r_d is 1 to make sure that we don't have any influence of FDT factor. We have chosen a smaller scale for this problem because as r_f gets bigger, query gets increasingly intractable and execution times shoot up considerably. Figure 4 plot the execution time vs r_f. Note that as you increase r_f, the data gets denser and the treewidth also increases. So there is another factor at work here as well. As one can observe from the plots, at one point the execution time shoots up considerably, which is the moment when treewidth has increased to the point where exact computation is no more feasible. The realm of exact computation lies only uptil this point, and after this one must resort to approximate computations. The focus of this paper is only on exact query evaluation, so we are not concerned with the intractable region of high treewidth. Observe though that in the tractable region the slope is not very high, which is an encouraging sign. This shows that if it weren't for treewidth, our method gracefully scales as the query/data change from safe to increasingly unsafe. The line representing MayBMS isn't very interesting to look at, since it seems because of some poor choice of heuristics their estimate of treewidth isn't very good; and hence their running time shoots up much

before it really should. The lesson to take away from these graphs is that our system transitions smoothly as data gets unsafe until the point the problem gets intractable.

Figure 4: Effect of FFD



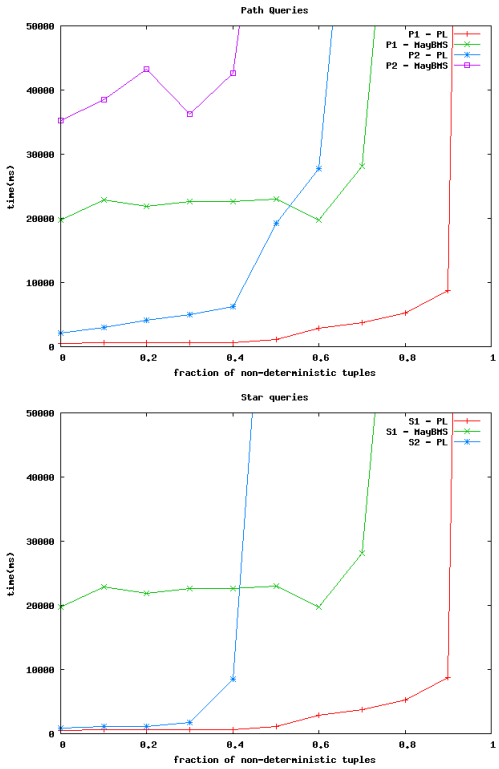
5.3 Effect of FDT

This is the same experiment as the last one, except that now we set $r_f = 1$, so every tuple violates FDs, but we vary r_d from 0 to 1. When $r_d = 1$, then the queries are very hard to compute and both systems will fare poorly. We want to observe the cases when r_d is small, making the problem tractable. And we observe that in these cases, our system performs very well for low values of r_d, as illustrated in Figure 5.

6 Related Work

Query Evaluation on Probabilistic Databases has been well studied in past years and many different approaches have been proposed. In exact evaluation, [5] showed that *safe* queries can be evaluated completely extensionally by reducing inference to database operators. But they restricted themselves to only *safe* plans. [11] proposed a way to

Figure 5: Effect of FDT



evaluate *safe* queries via any plan; the evaluation being still inside the database, but they augment the tables with random variables, like c-tables. In contrast, the other approaches, which are more general and can evaluate any query, tend to have a disconnect between the evaluation of the tuple and computing their probability. [16] first compute the tuples along with their lineage, and then evaluate the probability using the lineage. [17] construct a Bayesian Network, instead of lineage, and evalaute the answer probability of tuples by doing inference over these networks. Our approach breaks this disconnect by leveraging the database to do some probability computation even in case of unsafe queries.

Exact Evaluation is not always feasible for unsafe queries, and hence many approximation strategies[14, 6] have also been proposed, based on sampling. There are other approximate inference approaches in graphical models[7, 3, 18] that can also be leveraged. Note that these approximation strategies can be used on the And-Or Networks as well; hence our method complements the existing inference algorithms. 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.

7 Conclusion and Future Work

We showed how one can leverage the characteristics of the data to push probability computations inside the database, even for #P-hard unsafe queries. The existing methods could either evaluate only *safe* queries or they did the entire probability computation in an intensional way outside the database. We have shown a way to bridge the two approaches, showing that one doesn't have to choose one or the other paradigm. Even unsafe queries can be evaluated efficiently in a scalable way, and extensional methods don't always have to output the exact probability as output. Another way to think of our approach is that it's a way of reducing the scale of problem, while still preserving the complexity. Hence it complements the exisiting works on top-k query evaluations, approximate/exact computations. We also study the parametrized complexity of our algorithm and give the best theoretical guarantees proposed so far for this problem.
But we leave many questions unanswered. Its open how to choose a query plan, that (i)minimizes the size of output network or (ii)minimizes the treewidth of the output network. Note that answering (ii) is very crucial, since our algorithm being exponential in treewidth is very sensitive to it. We don't know the query complexity of finding the optimal query plan(minimum treewidth). For the simple queries that we considered, it is same as the optimal query plans in the traditional sense, but its not clear thats true in general. Also we need to extend the approach to solve self-joins and evaluate queries over more complicated models. With more complicated models, we may not be able to do many computations extensionally, and hence this raises the question whether the second stage symbolic evaluation that we do outside the database can be converted to database operators. When the scale of the data is huge and treewidth is very small, there would clearly be an advantage in doing this.

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] 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.

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

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

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

[6] 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.

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

[8] C. Koch. Maybms: A system for managing large uncertain and probabilistic databases. In C. Aggarwal, editor, *Managing and Mining Uncertain Data*, chapter 6. Springer-Verlag, 2009.

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

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

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

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

[13] C. Re, N. N. Dalvi, and D. Suciu. Query evaluation on probabilistic databases. *IEEE Data Eng. Bull*, 2006.

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

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

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

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

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