A New Class of Lineage Expressions over Probabilistic Databases computable in P-time

Batya Kenig, Avigdor Gal, and Ofer Strichman

Technion, Israel Instutute of Technology, Haifa, Israel

Abstract. We study the problem of query evaluation over tuple-independent probabilistic databases. We define a new characterization of lineage expressions called *disjoint branch acyclic*, and show this class to be computed in P-time. Specifically, this work extends the class of lineage expressions for which evaluation can be performed in PTIME. We achieve this extension with a novel usage of junction trees to compute the probability of these lineage expressions.

1 Introduction

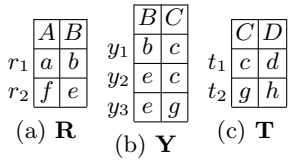
Applications in many areas such as data cleaning, data integration and event monitoring produce large volumes of uncertain data. Probabilistic databases in which the tuples' presence is uncertain, and known only with some probability, enable modeling and processing such uncertain data.

Answering queries over probabilistic databases has drawn much attention in the database community in recent years. A model of tuple-independent (or tuple-level semantics) probabilistic databases was introduced by Cavallo and Pittarelli [3] and was extensively discussed in the literature, *e.g.*, [9, 6]. According to this model, each tuple t is annotated by an existence probability $p_t > 0$, meaning it appears in a possible world with probability p_t, independently of other tuples. This defines a probability distribution over all possible database instances.

Query evaluation over tuple-independent probabilistic databases is $\#P$-hard in general, even for simple conjunctive queries without self-joins [6]. Dalvi and Suciu have introduced a dichotomy classification of queries over tuple-independent probabilistic databases, where any query with a *safe plan* can be computed *extensionally* by extending the query operators to enable an efficient computation of the result's probability [6, 5]. The extensional approach is very efficient, but may be applied to a limited set of queries [13, 6].

Even for queries without a safe plan there are database instances for which probabilities could be computed in PTIME. An intensional approach to evaluate queries over tuple-independent probabilistic databases considers both the query and the database instance. The query result is first computed and represented as a Boolean formula, termed a *lineage expression* [2], defined over Boolean variables corresponding to tuples in the database. The lineage describes how the answer was derived from the tuples in the database (see Table 1).

Various inference algorithms can be used to compute the result tuple probabilities, either exactly [16] or approximately [17]. Roy et al. [18] and Sen et



(a) **R** (b) **Y** (c) **T**

Fig. 1: Probabilistic Database with variables

al. [19] showed a polynomial time algorithm for recognizing lineage expressions that can be transformed to a read-once form, and computing their probability. Their algorithm is applicable for lineages resulting from conjunctive queries without self joins.

Consider the tuple-independent probabilistic database of Figure 1 and the Boolean conjunctive query

$$Q1() \text{:-} R(x,y), Y(y,z) \tag{1}$$

$Q1$ is a conjunctive query without self-joins that has a safe plan [6]. Olteanu and Huang [16] showed that the lineage resulting from conjunctive queries without self-joins, that have a safe plan, always have a read-once equivalent. Indeed, the lineage expression of $Q1$ over the database in Fig. 1 is $r_1y_1 + r_2y_2 + r_2y_3$, which has an equivalent read-once form, $r_1y_1 + r_2(y_2 + y_3)$.
$Q2$ is an example of a query that does not have a safe plan:

$$Q2() \text{:-} R(x,y), Y(y,z), T(z,w) \tag{2}$$

The lineage expression of $Q2$ over the database is

$$r_1y_1t_1 + r_2y_2t_1 + r_2y_3t_2 \tag{3}$$

It was shown [18, 12] that Expression 3 does not have an equivalent read-once form.

In this work we introduce a new class of lineage expressions called *disjoint branch acyclic lineage expressions*. Such lineage expressions are defined using restrictions on their respective hypergraph. Going back to Example 1, the lineage expression in Eq. 3 is disjoint branch acyclic, possessing a special structure that can be exploited for efficient computation.

We characterize disjoint branch acyclic lineage expressions and present, as part of the proof of the class computation time, an algorithm to compute the probability of this form in time that is polynomial in the size of the formula.

The rest of the paper is organized as follows: Section 2 introduces background on a specific class of chordal graphs, probability computation using probabilistic graphical models, and hypergraph acyclicity. Lineage acyclicity is presented in Section 3. Section 4 presents the main theorem of this work, proving it by showing an algorithm for probability computation of disjoint branch acyclic lineage expressions. We conclude in Section 5.

2 Preliminaries

At the heart of the proposed method for computing the probability of Boolean lineage expressions lies a graph with a specific structure termed *rooted directed path graph*. This class of graphs and its PTIME recognition algorithm were first introduced by Gavril [11]. This section discusses this class of graphs and other notions significant to our proposed approach. We present a class of chordal graphs, namely *rooted directed path graphs* (Section 2.1) and discuss probability computation using probabilistic graph models (Section 2.2). We conclude with the introduction of hypergraph acyclicity (Section 2.3).

A clique C of a graph $G(V, E)$ is a subset of V where every pair of nodes is adjacent. We denote by K_G the set of maximal cliques in G. For a vertex $v \in V$ we denote by K_v the set of maximal cliques in K_G that contain v. We use $T(V, E)$ to denote a tree. A subtree is a connected subgraph of a tree. In particular, a path in a tree can be viewed as a subtree. Whenever a subtree is induced from a subset of nodes $V' \subseteq V$ of a tree $T(V, E)$, we do not explicitly state its set of edges, but rather denote it using $T(V')$.

2.1 Classes of chordal graphs

A *chord* is an edge connecting two non-consecutive nodes in a cycle or path. G is *chordal* or *triangulated* if it does not contain any chordless cycles. Discovering whether G is chordal can be performed in time $O(|V| + |E|)$ [20].

A $P4$ denotes a chordless path with four vertices and three edges. A graph is considered to be $P4$-*free* if it does not contain a $P4$.

An *intersection graph* of a finite family of non-empty sets is obtained by representing each set by a vertex, and connecting two vertices if their corresponding sets intersect. Gavril [10] characterizes the connection between chordal graphs and intersection graphs, as follows.

Theorem 1 ([10]). *Let $G(V, E)$ be an undirected graph. The following statements are equivalent:*

1. *G is chordal.*
2. *There exists a tree $T(K_G)$ such that for every $v \in V$ the subgraph induced by K_v is a subtree $T(K_v)$.*
3. *G is the intersection graph of a family of subtrees of some tree T'.*

$T(K_G)$ is called a *junction tree* possessing the following *running intersection property*: for every pair of cliques $C_1, C_2 \in K_v$ every clique on the path from C_1 to C_2 in $T(K_G)$ belongs to K_v.

Let T' be a rooted directed tree, and consider a group of directed paths in T'. Let G be the intersection graph of directed paths in T'. Then G is a *Rooted Directed Path Graph*, and T' is called the *host tree* of G.

The following property (Theorem 2 [11]) defines a characteristic tree, associated with a rooted directed path graph. This tree is of prime concern in this work.

Theorem 2 ([11]). *A graph $G(V, E)$ is a rooted directed path graph (rdpg) iff there exists a rooted directed tree T_r whose vertex set is K_G, so that for every vertex $v \in V$, $T_r(K_v)$ is a directed path of T_r.*

Constructing the characteristic tree of a rooted directed path graph $G(V)$, if one exists, takes $O(|V|^4)$ [11].[1] The characteristic tree T_r of an rdpg $G(V, E)$ is, in fact, a special form of a junction tree (Definition 1), where every vertex $v \in V$ appears in exactly one branch of T_r. Such a junction tree is known as a *disjoint branch junction tree* (dbjt) [8].

Definition 1. *Let T_r be a junction tree with root r and children $r_1, r_2, ..., r_l$ roots of subtrees $T_{r_1}, ..., T_{r_l}$, respectively. A junction tree T_r is a* dbjt *if:*

1. *T_r contains a single node, r,* i.e., *$|T_r| = 1$, or*
2. *The following two conditions jointly hold: (a) $\forall r_i \neq r_j, C_{r_i} \cap C_{r_j} = \emptyset$; and (b) $\forall T_{r_i} \in \{T_{r_1}, T_{r_2}, ..., T_{r_l}\}$, T_{r_i} recursively complies with the conditions 1 and 2.*

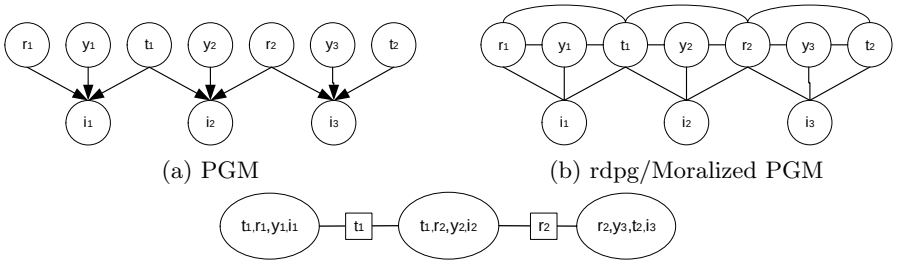
An example of a rooted directed path graph and its corresponding characteristic tree (or dbjt) are presented in Figures 2b and 2c, respectively. Definition 1 characterizes the dbjt properties that enable the efficient computation we show in this work.

2.2 Probability Computation using Probabilistic Graph Models

Probabilistic Graphical Models (PGMs) refer to a set of approaches for representing and reasoning about large joint probability distributions [15]. A PGM is a graph in which nodes represent random variables and edges represent direct dependencies between them. An example of a directed PGM is given in Figure 2a, representing the lineage expression of Eq. 3.

Inference in PGMs is the task of answering queries over the probability distribution described by the graph and is, in general, #P-complete [15]. One of the well-known inference algorithms is the junction tree algorithm [15]. The algorithm is designed for undirected PGMs in which for every maximal clique C in the PGM, there exists a factor F_C that is a function from the set of assignments of C to the set of non-negative reals. The algorithm consists of two parts, *compilation* and *message passing*. The compilation part includes three steps, namely *moralization, triangulation* and *construction*, as follows. *Moralization*, in the case of a directed PGM, involves connecting all parents of a given node and dropping the direction of edges (*e.g.*, Figure 2b). *Triangulation* adds extra edges to create a chordal graph. The example graph obtained after moralization in Figure 2b is already chordal and therefore no edges need to be added. We note that this example is also a rooted directed path graph (see Section 2.1). *Construction* forms

[1] To date, this is the most efficient published recognition algorithm for rooted directed path graphs [4]. There is also an unpublished linear time algorithm [7] for this class of graphs.



(a) PGM (b) rdpg/Moralized PGM

(c) Characteristic tree/dbjt

Fig. 2: PGM, moralization and junction tree

a junction tree over the maximal cliques in the resulting graph $G(V)$. Figure 2c shows the junction tree of the graph in Figure 2b. Note that since the graph is a rooted directed path graph, its junction tree is in fact a dbjt.

The message-passing part has two steps. First, for each edge of the tree $(C_1, C_2) \in T$ a factor, defined over the variables in the intersection $S = C_1 \cap C_2$, is defined. The factor entries are initialized to 1. Then, neighboring nodes C_1, C_2 exchange messages through the factor defined on their intersection F_S, $S = C_1 \cap C_2$. The message from C_1 to C_2 is:

$$\mu_{C_1,C_2}(S) = \sum_{x \in C_1 \setminus S} F_1(C_1)$$

and the message from C_2 to C_1 is:

$$\mu_{C_2,C_1}(S) = \frac{\sum_{x \in C_2 \setminus S} F_2(C_2)}{\mu_{C_1,C_2}(S)}$$

After every pair of adjacent nodes in the junction tree have exchanged messages, each factor holds the marginal of the joint probability distribution of the entire variable set. The message-passing protocol is such that every edge in the tree is processed once in each direction. Therefore, the runtime of the message passing algorithm is $O(N \cdot D^k)$ where N is the number of nodes in the tree, D is the domain of the variables, and k is the size of the largest clique. $k-1$ is referred to as the *width* of the associated PGM and clearly, the inference algorithm is exponential in the PGM's width so bounded width implies tractability in graphical models. PGMs may have several different triangulations, affecting the size of the largest clique in the graph. The smallest width that can be obtained for a PGM is its *treewidth*, where the treewidth of a chordal graph is simply its width. Finding an optimal triangulation is known to be NP-complete. However, in this work we introduce a method to compute the probability of a family of lineage expressions in time that is polynomial in their treewidth.

2.3 Hypergraphs and Acyclicity

A *hypergraph* $H = (V, E)$ is a generalization of a graph where V is the set of nodes and the set of edges E is a set of non-empty subsets of V. Edges in a hypergraph are termed *hyperedges*. The *primal graph* $G(H) = (V, E^G)$ corresponding to a hypergraph H is the graph whose vertices are those of H and whose edges are the set of all pairs of nodes that occur together in some hyperedge of H ($E^G = \{(u, v) : \{u, v\} \subseteq V, \exists e \in E, \{u, v\} \subseteq e\}$). A hypergraph H is *conformal* if every clique in its primal graph $G(H)$ is contained in a hyperedge of H.
Acyclicity in a hypergraph is defined as follows.

Definition 2 (acyclicity [1]). *A hypergraph H is acyclic (or α-acyclic) if H is conformal and its primal graph $G(H)$ is chordal.*

Beeri et. al [1] showed that a hypergraph is acyclic iff it has a junction tree. Duris [8] also showed that for a restricted form of acyclic hypergraphs (called γ-acyclic) there exists a dbjt rooted at every node. An algorithm that constructs a dbjt in time $O(|V|^2)$ for γ-acyclic hypergraphs was also introduced there.

3 Disjoint Branch Acyclic Lineage (DBAL) Expressions

We now introduce a class of Boolean lineage expressions, connecting it to rooted directed path graphs. Let $f(V)$ denote a lineage expression of a set of literals V, resulting from a query q, as derived by the query engine. Lineage expressions of conjunctive queries are monotone formulas, where all literals are positive and only conjunctions and disjunctions are used. An *implicant* $p \subseteq V$ of f is a set of literals such that whenever they are true, f is true as well. An implicant of f is called a *prime implicant* if it cannot be reduced. We denote by f_{IDNF} f's DNF form containing only prime implicants. f_{IDNF} can be modeled as a hypergraph $H_f(V, E)$, where each literal corresponds to a node in the graph and each prime implicant corresponds to a hyperedge.

f_{IDNF} is not always available, and expanding f to its DNF form may result in an exponentially larger formula. Therefore, we now propose the construction of an alternative graph $G(f)$, built over $f(V)$. The set of literals V is the set of nodes of $G(f)$ and two nodes are connected iff they belong to a common prime implicant. $G(f)$ is exactly H_f's primal graph, *i.e.*, $G(H_f) = G(f)$. For a restricted set of queries, $G(f)$ can be built directly from f by using a method proposed by Roy et al. [18]. We say that f is *conformal* if every maximal clique in $G(f)$ is contained in a prime implicant of f.

Definition 1 (Lineage expression acyclicity). *A lineage expression f is acyclic if $G(f)$ is chordal and f is conformal.*

Definition 2 (Disjoint Branch Acyclic Lineage Expressions). *A lineage expression f is a Disjoint Branch Acyclic Lineage Expression or DBAL if f is acyclic and $G(f)$ is a rooted directed path graph.*

Following the discussion in Section 2.1, DBAL expressions have a dbjt. The lineage expression in Eq. 3 (Section 1) is an example of a DBAL. Its corresponding dbjt is presented in Figure 2c.

4 DBAL Expression Probability Computation

In this section we prove that the probability of disjoint branch acyclic lineage (DBAL) expressions over tuple independent probabilistic databases can be computed in PTIME.

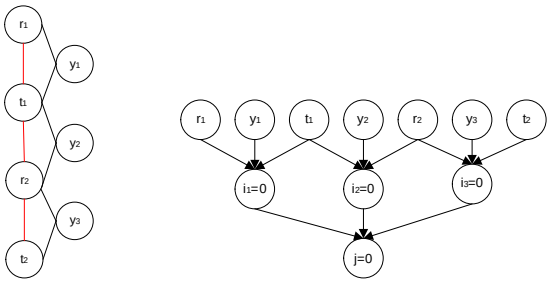
Theorem 1. *Let $f(V)$ be a DBAL expression. The probability $Pr(f = 1)$ can be computed in time $O(nk^2)$ where $n = |V|$ and k is the size of the largest clique in $G(f)$.*

At the heart of the proof is an algorithm for computing the probability of DBAL expressions in time that is quadratic in the size of the treewidth. This solution is unique since, to the best of our knowledge, it is the first time an algorithm that runs in time polynomial (quadratic) of the treewidth, as opposed to exponential, is introduced in the context of tuple independent probabilistic databases.

Section 4.1 introduces an example that will be used to demonstrate the algorithm and Section 4.2 discusses factor representation in our setting. Finally, Section 4.3 details the algorithm and presents lemmas 1 and 2 that argue for the correctness of the algorithm and its complexity, respectively, which together proves Theorem 1 above.

4.1 Illustrating Example

We first motivate and explain the algorithm approach using a simple example.



(a) Primal graph

(b) Bayesian net

Fig. 3: Illustration for Example 1

Example 1. Consider query $Q2() : -R(x, y), Y(y, z), T(z, w)$, presented earlier over the instance in Table 1. The lineage of the query is $j = r_1y_1t_1 + r_2y_2t_1 + r_2y_3t_2$. The primal graph corresponding to the query is given in Figure 3a. It

is easy to see that this lineage expression is not read-once since it has a P4: (r_1, t_1, r_2, t_2). Let us denote by $i_1 = r_1 y_1 t_1$, $i_2 = r_2 y_2 t_1$, and $i_3 = r_2 y_3 t_2$. We are ultimately interested in calculating the probability: $Pr(j = 1) = 1 - Pr(j = 0)$. If $j = 0$ then we know that $i_1 = i_2 = i_3 = 0$. These values can be seen as introduction of evidence in a Bayesian network, illustrated in Figure 3b. After moralization (see Section 2.2), the network is chordal and conformal and therefore has a junction tree depicted in Figure 4. In the Junction-Tree algorithm, any node may be selected as root. For this example let us select node $\{r_2, y_3, t_2\}$ as the root. □

In the classic junction tree algorithm, where factors are represented in tabular form, each entry in the factor table represents a single assignment, leading to a representation that is exponential in the number of variables in the table. We present linear sized factors, where each entry represents multiple assignments. See, for example, Figure 4, where asterisks represent wildcard assignments. Here, the number of entries in each factor is exactly the node's cardinality. The message passing is illustrated in Figure 4, and will be demonstrated in detail in Section 4.3. After the completion of the algorithm, the root node contains the marginal probability of its entries (see Section 2.2). Therefore, all the entries of the root node's factor are added in order to obtain the required probability.

4.2 Factor Representation and Projection

Hereinafter we shall use a tabular notation to represent a factor, where columns represent random variables, and rows correspond to a set of mutual exclusive value assignments. The notation introduced below is illustrated in Example 2. Given a factor over the set of random variables $X = (X_1, X_2, ..., X_n)$ we denote by $F_X[j, k]$ (or simply $F[j, k]$ whenever the variable set is clear from the context) the value of X_k in the jth entry. X_k's value may be the wildcard,'*', indicating that it can be either 0 or 1. The assignments represented by the jth entry are denoted by $F[j]$ and their overall probability is denoted by $Pr(F[j])$. Let $X' \subset X$ be a subset of the variables of factor F, we denote by $F[j, X']$ the values of variables X' in the jth entry of the factor. Finally, given an assignment $X = x$, we denote by $Pr(F[x])$ the probability corresponding to this entry in factor F.

Example 2. Consider the factor F in Table 2, representing the joint distribution of independent boolean random variables X_1, X_2, X_3. Using our notation, $F[2, 3] = *$ and $F[2] = [1, 0, *]$. Also, $Pr(F[3]) = p_{X_1} \cdot p_{\overline{X_3}}$ and $F[3, \{X_1, X_3\}] = [1, 0]$. Finally, $Pr(F[\{X_1 = 1, X_2 = 0, X_3 = *\}]) = p_{X_1}$. □

Each maximal clique in the primal graph of a DBAL expression corresponds to exactly one prime implicant of the lineage's IDNF form. As a result, each node in the corresponding junction tree contains a factor that represents a single DNF prime implicant of the lineage. We will refer to these as *DNF factors*. For each variable X in the expression we define a *base factor*, F^b_X. Base factors contain exactly two entries, with values $0, 1$ and their appropriate probabilities $p_{\overline{X}}, p_X$,

respectively. Each base factor is assigned to exactly one node in the junction tree.

Consider some DNF prime implicant d, containing k literals, $d = X_1 \cdot X_2 \cdot ... \cdot X_k$. The probability of $d = 0$ is computed as follows:

$$Pr(d=0) = Pr(X_1 = 0) + Pr(X_1 = 1, X_2 = 0) + ... + Pr(X_1 = 1, ..., X_{k-1} = 1, X_k = 0). \tag{4}$$

The k summands in Eq. 4 create a mutually exclusive and exhaustive set of configurations.

For illustration, consider Table 1 over $X_1, ..., X_k$. The "Pr" values are initialized to 1.

The asterisks in the table represent wildcard assignments, as follows:

X_1	X_2	...	...	X_k	Pr
0	*	*	*	*	1
1	0	*	*	*	1
1	1	0	*	*	1
1	...	...	...	*	1
1	1	1	...	0	1

Table 1: Factor Table

$$\begin{aligned}
&Pr(X_1 = 1, ..., X_{i-1} = 1, X_i = 0, X_{i+1} = *, ..., X_k = *) = \\
&\sum_{x_{i+1},...,x_k \in \{0,1\}} Pr(X_1 = 1, ...X_{i-1} = 1, X_i = 0, X_{i+1} = x_{i+1}, ..., X_k = x_k) = \\
&\sum_{x_{i+1},...,x_k \in \{0,1\}} Pr(X_{i+1} = x_{i+1}, ..., X_k = x_k | X_1 = 1, ..., X_{i-1} = 1, X_i = 0) \cdot Pr(X_1 = 1, ..., X_{i-1} = 1, X_i = 0) \\
&= Pr(X_1 = 1) \cdot ... \cdot Pr(X_{i-1} = 1) \cdot Pr(X_i = 0) \cdot \sum_{x_{i+1},...,x_k \in \{0,1\}} Pr(X_{i+1} = x_{i+1}, ..., X_k = x_k)
\end{aligned} \tag{5}$$

using the tuple independence assumption in the transition from the third to the fourth line of the equation. Informally, once we know that $X_j = 0$, then the implicant's value is false regardless of the values of its other literals.

At the beginning of the algorithm, the values in the "Pr" column of the factors depend on the assignment of the base factors to the nodes in the tree. For example, a factor over variables X_1, X_2, X_3 at the beginning of the algorithm is given in Table 2. For the sake of illustration, we assume that the base factor $F^b_{X_2}$ is assigned to a different node (DNF factor).

The proposed algorithm is actually a series of projections (defined below) over the linear-sized factors of the junction tree. In the general message passing algorithm [15], in which each entry in the factor represents a single configuration of the variables (and therefore the size of the factor is exponential in the number of variables), the probabilities of the entries with common values in the projected variables are simply added. This is not the case for the linear sized factors used in our setting. Definition 1 formalizes this notion of projection in our setting, and Example 3 demonstrates it.

X_1	X_2	X_3	Pr
0	*	*	$p_{\overline{X_1}}$
1	0	*	p_{X_1}
1	1	0	$p_{X_1} \cdot p_{\overline{X_3}}$

Table 2: Factor Table with partial base factors

Definition 1 (factor projection). *Let $F_{X \cup X'}$ be a factor over variables $X \cup X'$ where $X = \{X_1, ..., X_m\}$ and $X' = \{X'_1, ..., X'_l\}$. The projection of F over*

the variables in X, denoted $F_X = \prod_X F_{X \cup X'}$, is a new factor containing only variables X. The probability column in F_X is computed as follows:

$$Pr(F_X[j]) = \sum_{i \in [1,|X \cup X'|]: F_{X \cup X'}[i,X] = F_X[j]} Pr(F_{X \cup X'}[i])$$

The projection $\prod_X F_{X \cup X'}$ may be applied to $F_{X \cup X'}$ under the following conditions:

1. *The variables X', projected out of the factor $F_{X \cup X'}$, appear after (referring to column order) the variables X.*
2. *The base factors corresponding to the variables X' are included in factor $F_{X \cup X'}$ before the projection operation can be applied.*

Example 3. Consider Table 2 and the factor F_{X_1,X_2,X_3} over the variable set $\{X_1, X_2, X_3\}$. We start by projecting out the variable X_3. Condition 1 of Definition 1 is satisfied. As for Condition 2, X_3's probability, p_{X_3}, is already available in the factor, and therefore

$$F_{X_1,X_2} = \prod_{X_1,X_2} F_{X_1,X_2,X_3} = \begin{array}{cc|c} X_1 & X_2 & Pr \\ \hline 0 & * & p_{\overline{X_1}} \\ 1 & 0 & p_{X_1} \\ 1 & 1 & p_{X_1} \cdot p_{\overline{X_3}} \end{array}$$

Projecting out X_2 from F_{X_1,X_2} requires multiplying in X_2's base factor to satisfy Condition 2. Therefore,

$$F_{X_1} = \prod_{X_1} F_{X_1,X_2} = \begin{array}{c|c} X_1 & Pr \\ \hline 0 & p_{\overline{X_1}} \\ 1 & p_{X_1}(p_{\overline{X_2}} + p_{X_2} \cdot p_{\overline{X_3}}) \end{array}$$

□

4.3 Algorithm Description

Let C_i denote the set of variables in node i of the junction tree, $|C_i|$ its cardinality, and F_i its factor. In this section we use the factor notation defined in Section 4.2. B_r denotes the set of variables in node r, for which base factors have been assigned, i.e., $B_r = \{X : X \in C_r,\ F_X^b \textit{ is assigned to } r\}$. We denote by $children(i)$ and $p(i)$ the children and parent of node i in the junction tree, respectively. A message between node i and node j, $\mu_{i,j}(C_i \cap C_j)$ is a factor over the intersection of the two nodes. The number of entries in $\mu_{i,j}(C_i \cap C_j)$ is $|C_i \cap C_j| + 1$ (including the entry containing all ones).

The algorithm uses a partial order $\preccurlyeq$ over the variables in the junction tree T. We denote by $vars(\preccurlyeq)$ the set of variables over which $\preccurlyeq$ is defined.

The pseudocode of the algorithm over linear sized factors is given in algorithms 1 and 2. After the initial call to Algorithm 2 (Line 1 of Algorithm 1), the

Algorithm 1: Message Passing: Initial Call

Input: dbjt (see Definition 1) $T_{r'}$ with root r' corresponding to a lineage expression f.
Output: $Pr(f = 0)$

1: Call Algorithm 2 with parameters: $T_{r'}$ and $\preccurlyeq \leftarrow \emptyset$.
2: **Return** $\sum_{j=1}^{|C_{r'}|} Pr(F_{r'}[j])$.

algorithm performs a series of recursive calls to update the probabilities in the node factors of the junction tree.

Each message from a node i to its parent, $p(i)$, is a projection on factor F_i over the variables $C_i \cap C_{p(i)}$. According to the definition of projection (Definition 1), variables $C_i \cap C_{p(i)}$ should appear before $C_i \setminus C_{p(i)}$ in the factor table representation. Therefore, lines 1-5 of Algorithm 2 define an order over the variables in the root node r that was given as a parameter (C_r), such that projection over variables $C_r \cap C_{p(r)}$ is made possible. In Example 1, Figure 4, the root node contains ordered variables $\{r_2, y_3, t_2\}$. In the factor for the child node with variables $\{t_1, r_2, y_2\}$, r_2 appears before t_1 and y_2 because the message between this node and its parent is over variable r_2. Likewise, in the factor with variables $\{t_1, r_1, y_1\}$, t_1 appears before r_1 and y_1. The order is updated in line 5.

Lines 6-10 initialize a factor for node r based on the ordering $\preccurlyeq$ that was updated in lines 1-5, and according to the base factors assigned to this node. In Example 1 (Figure 4), the base factors for variables y_3 and t_2 are assigned to the root node, while the base factor for r_2 is assigned to the middle node (with variables $\{t_1, r_2, y_2\}$).

Lines 11-21 initiate a recursive call on the children of r. In Line 13, the messages from all children of r are collected. Each one of the messages, $\mu_{i,r}(C_{r_i} \cap C_r)$, received by the node in line 13 contains $|C_{r_i} \cap C_r| + 1$ entries, which form an exhaustive and mutual exclusive set of configurations. For example, consider the message $\mu_{1,2}(t_1)$ from node $\{t_1, r_1, y_1\}$ to node $\{t_1, r_2, y_2\}$ (Figure 4).

As in the case of projection, in order to add the probabilities of the entries in the messages, the appropriate base factors need to be multiplied in before the addition can take place. For example, in Figure 4, the base factor for t_1, $F^b_{t_1}$, is not part of the factor of node $\{t_1, r_1, y_1\}$, and therefore not part of the original message, $\mu_{1,2}(t_1)$, (containing only entries where $t_1 = 0$ and $t_1 = 1$). However, in order to augment the factor with the entry $t_1 = *$, the values in the probability column of the entries corresponding to $t_1 = 0$ and $t_1 = 1$ need to be added. In order for the resulting probability to be correct, the probabilities of the entries corresponding to $t_1 = 0$ and $t_1 = 1$ are multiplied by $p_{\overline{t_1}}$ and p_{t_1} respectively. The entry where $t_1 = *$ in Figure 4 was appended to the message because it will be used by node $\{t_1, r_2, y_2\}$. Such entries are calculated in lines 14-21 of the algorithm. There are exactly $|C_{r_i} \cap C_r|$ such entries added to the message which correspond to partial sums over the entries in the original message $\mu_{i,r}(C_{r_i} \cap C_r)$.

Algorithm 2: Message Passing: Main Procedure

Input: dbjt T_r with root r and a partial order $\preccurlyeq$.
Output: Factor F_r with correct probabilities

1: **if** $r \neq r'$ **then**
2: Define an order $\preccurlyeq_r$ over C_r s.t.: 1. $C_r \cap C_{p(r)}$ appear before $C_r \backslash C_{p(r)}$ 2. The order of variables $C_r \cap vars(\preccurlyeq)$ complies with $\preccurlyeq$.
3: **else**
4: define an arbitrary order over variables C_r
5: Update $\preccurlyeq$ according to the steps above.
{*Initialize node's factor based on* $\preccurlyeq$}
6: Define a linear-sized factor F_r based on $\preccurlyeq$.
7: **for** $j \leftarrow 1$ to $|C_r|$ **do**
8: $Pr(F_r[j]) \leftarrow 1.0$ {initialize factor entries}
9: **for** $X \in B_r$ **do**
10: $Pr(F_r[j]) \leftarrow Pr(F_r[j]) \cdot Pr(X = F_r[j, \{X\}])$
{*apply projection on subtrees*}
11: **for all** $r_i \in children(r)$ **do**
12: Recursively call the algorithm on subtree T_{r_i} with root r_i and (updated) ordering $\preccurlyeq$.
13: $\mu_{i,r}(C_r \cap C_{r_i}) \leftarrow \prod_{C_r \cap C_{r_i}}(F_{r_i})$ [project on the children's factor to get the message]
14: **for** $j \leftarrow 1$ to $|C_r \cap C_{r_i}| + 1$ **do**
15: $M_{i,r}[j] \leftarrow 1.0$ [iterate over entries in the message]
16: **for** $X \in ((C_r \cap C_{r_i}) \setminus B_{r_i})$ **do**
17: $M_{i,r}[j] \leftarrow M_{i,r}[j] \cdot Pr(X = \mu_{i,r}[j, \{X\}])$ $[Pr(X = *) = 1.0]$
18: $prob \leftarrow Pr(\mu_{i,r}[|C_{r_i} \cap C_r| + 1) \cdot M_{i,r}[|C_{r_i} \cap C_r| + 1]$ [initialize $prob$ according to entry [1,1,...,1]]
19: **for** $k \leftarrow |C_{r_i} \cap C_r|$ to 1 **do**
20: $prob \leftarrow prob + Pr(\mu_{i,r}[k]) \cdot M_{i,r}[k]$ [update $prob$]
21: $Pr(\mu_{i,r}[X_1 = 1, ..., X_{k-1} = 1, X_k = *, ..., X_{|C_{r_i} \cap C_r|} = *]) \leftarrow prob$
{*update factor using children's projected factors*}
22: **for** $j \leftarrow 1$ to $|C_r|$ **do**
23: **for all** $r_i \in children(r)$ **do**
24: $Pr(F_r[j]) \leftarrow Pr(F_r[j]) \cdot Pr(\mu_{i,r}[F_r[j, C_{r_i} \cap C_r]])$

Finally, lines 22-24 update F_r according to the messages received from its children. The correctness of the algorithm for disjoint branch junction trees is given in Lemma 1. The proof is omitted due to space considerations. It is available in the full version of this paper [14].

Lemma 1. *Let $T_{r'}$ be a dbjt with root r', corresponding to lineage expression f. After running Algorithm 1 on $T_{r'}$, $Pr(F_{r'}[j])$, $j \in [1, |C_{r'}|]$ contains the marginal probability corresponding to the configurations represented by the jth entry of this factor.*

Lemma 2. *The complexity of algorithms 1 and 2 on a disjoint branch junction tree of size n is $O(n \cdot k_{MAX}^2)$ where $k_{MAX} = MAX_{i=1..n}|C_i|$.*

Proof. The loop in lines 6-10 is performed in $O(|C_r|^2)$ since $|B_r| \leq |C_r|$. Similarly, the loop in lines 14-17 is performed in $O((|C_r \cap C_{r_i}| + 1)^2)$, but since subsumption cannot occur in the junction tree, $|C_r| > |C_r \cap C_{r_i}|$, we arrive again at runtime of $O(|C_r|^2)$. The loop in lines 19-21 takes time $O(|C_r \cap C_{r_i}|)$.

A node r in the tree receives messages from all of its neighbors, except its parent in the algorithm. Since the children create a partition of a subset of the variables in the node, then the number of children can be at most $|C_r|$. The number of entries for which the probability is updated is exactly $|C_r|$, therefore the total runtime is $\sum_{i=1}^{n} O(|C_r|^2) = O(n \cdot (k_{MAX}^2))$.

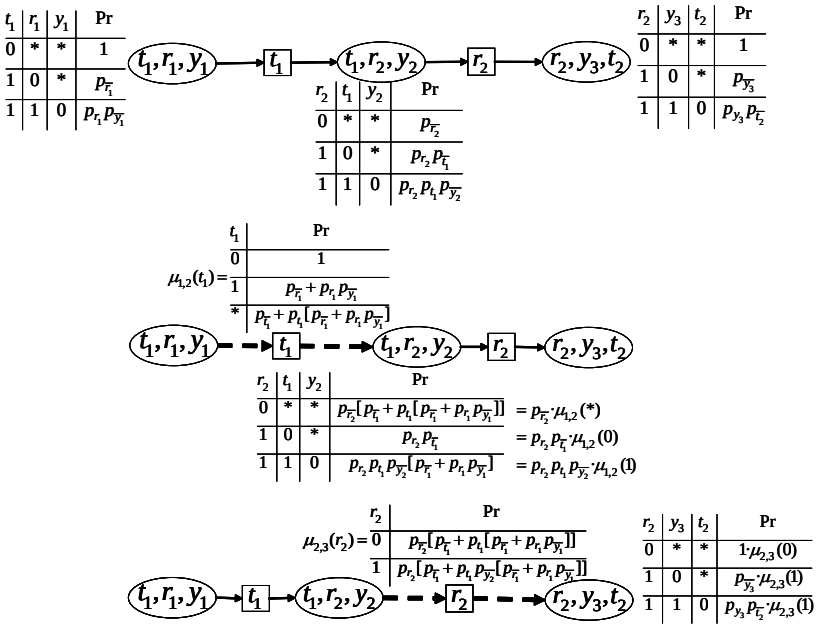


Fig. 4: Message Passing using Alg. 2 over $f = t_1 r_1 y_1 + t_1 r_2 y_2 + r_2 y_3 t_2$

Algorithms 1 and 2 along with Lemmas 1 and 2 complete the proof for the main theorem of this section, Theorem 1. Overall, we have shown that DBAL expressions, having a dbjt, can be evaluated in polynomial time.

5 Conclusions

We have presented *disjoint branch acyclic lineage expressions*, a new class of lineage expressions of queries over tuple independent probabilistic databases, and shown that probability computation over this class can be done in low polynomial data complexity.

As part of future research we plan to investigate queries and database instances that induce junction trees with structural properties that enable efficient probability calculation. Furthermore, we plan to explore how such queries relate to existing characterizations of tractability [13]. Since correlations between tuples can naturally arise in many applications, we intend to investigate how to extend the proposed approach to models without the tuple-independence assumption.

Acknowledgments

The work was carried out in and partially supported by the Technion–Microsoft Electronic Commerce research center.

References

1. C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30:479–513, July 1983.
2. O. Benjelloun, A. Sarma, A. Halevy, M. Theobald, and J. Widom. Databases with uncertainty and lineage. *VLDB Journal*, 17(2):243–264, 2008.
3. R. Cavallo and M. Pittarelli. The theory of probabilistic databases. In *VLDB*, pages 71–81, 1987.
4. S. Chaplick. *Path Graphs and PR-trees*. PhD thesis, Charles University, Prague, Jan. 2012.
5. N. Dalvi, K. Schnaitter, and D. Suciu. Computing query probability with incidence algebras. In *PODS*, pages 203–214. ACM, 2010.
6. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDB Journal*, 16:523–544, October 2007.
7. P. F. Dietz. *Intersection Graph Algorithms*. PhD thesis, Cornell University, Aug. 1984.
8. D. Duris. Some characterizations of γ and β-acyclicity of hypergraphs. *Inf. Process. Lett.*, 112(16):617–620, 2012.
9. N. Fuhr and T. Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Transactions on Information Systems*, 15:32–66, 1994.
10. F. Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B*, 16(1):47 – 56, 1974.
11. F. Gavril. A recignition algorithm for the intersection graphs of directed paths in directed trees. *Discrete Mathematics*, 13:237 – 249, 1975.
12. M. Golumbic, A. Mintz, and U. Rotics. Factoring and recognition of read-once functions using cographs and normality. In *DAC*, June 2001.
13. A. K. Jha and D. Suciu. Knowledge compilation meets database theory: compiling queries to decision diagrams. In *ICDT*, pages 162–173, 2011.
14. B. Kenig, A. Gal, and O. Strichman. A new class of lineage expressions over probabilistic databases computable in p-time. Technical Report IE/IS-2013-01, Technion – Israel Institute of Technology, Jan. 2013. http://ie.technion.ac.il/tech_reports/1365582056_TechReportSUM2013.pdf.
15. D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press, August 2009.
16. D. Olteanu and J. Huang. Using OBDDs for efficient query evaluation on probabilistic databases. *Scalable Uncertainty Management*, pages 326–340, 2008.
17. D. Olteanu, J. Huang, and C. Koch. Approximate confidence computation in probabilistic databases. In *ICDE*, pages 145–156, 2010.
18. S. Roy, V. Perduca, and V. Tannen. Faster query answering in probabilistic databases using read-once functions. In *ICDT*, pages 232–243, 2011.
19. P. Sen, A. Deshpande, and L. Getoor. Read-once functions and query evaluation in probabilistic databases. *PVLDB*, 3(1):1068–1079, 2010.
20. R. E. Tarjan and M. Yannakakis. Addendum: Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 14(1):254–255, 1985.