Computing Query Probability with Incidence Algebras

Nilesh Dalvi
Yahoo Research
Santa Clara, CA, USA
ndalvi@yahoo-inc.com

Karl Schnaitter
UC Santa Cruz
Santa Cruz, CA, USA
karlsch@soe.ucsc.edu

Dan Suciu
University of Washington
Seattle, WA, USA
suciu@cs.washington.edu

ABSTRACT

We describe an algorithm that evaluates queries over probabilistic databases using Mobius' inversion formula in incidence algebras. The queries we consider are unions of conjunctive queries (equivalently: existential, positive First Order sentences), and the probabilistic databases are tuple-independent structures. Our algorithm runs in PTIME on a subset of queries called "safe" queries, and is complete, in the sense that every unsafe query is hard for the class $FP^{\#P}$. The algorithm is very simple and easy to implement in practice, yet it is non-obvious. Mobius' inversion formula, which is in essence inclusion-exclusion, plays a key role for completeness, by allowing the algorithm to compute the probability of some safe queries even when they have some subqueries that are unsafe. We also apply the same lattice-theoretic techniques to analyze an algorithm based on lifted conditioning, and prove that it is incomplete.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query Processing*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic

General Terms

Algorithms, Theory

Keywords

Mobius inversion, incidence algebra, probabilistic database

1. INTRODUCTION

In this paper we show how to use *incidence algebras* to evaluate *unions of conjunctive queries* over probabilistic databases. These queries correspond to the select-project-join-union fragment of the relational algebra, and they also correspond to *existential positive formulas* of First Order Logic. A probabilistic database, also referred to as a *probabilistic structure*, is a pair $(\mathbf{A}, P)$ where $\mathbf{A} = (A, R_1^A, \ldots, R_k^A)$ is first order structure over vocabulary $R_1, \ldots, R_k$, and P is a function that associates to each tuple t in $\mathbf{A}$ a number $P(t) \in [0, 1]$. A probabilistic structure defines a probability distribution on the set of substructures $\mathbf{B}$ of $\mathbf{A}$ by:

$$P_{\mathbf{A}}(\mathbf{B}) = \prod_{i=1}^{k}\Big(\prod_{t \in R_i^B} P(t) \times \prod_{t \in R_i^A - R_i^B} (1 - P(t))\Big) \quad (1)$$

We describe a simple, yet quite non-obvious algorithm for computing the probability of an existential, positive FO sentence Φ, $P_{\mathbf{A}}(\Phi)$[1], based on Mobius' inversion formula in incidence algebras. The algorithm runs in polynomial time in the size of $\mathbf{A}$. The algorithm only applies to certain sentences, called *safe* sentences, and is sound and complete in the following way. It is sound, in that it computes correctly the probability for each safe sentence, and it is complete in that, for every fixed unsafe sentence Φ, the data complexity of computing Φ is $FP^{\#P}$-hard. This establishes a dichotomy for the complexity of unions of conjunctive queries over probabilistic structures. The algorithm is more general than, and significantly simpler than a previous algorithm for conjunctive sentences [5].

The existence of $FP^{\#P}$-hard queries on probabilistic structures was observed by Grädel et al. [8] in the context of query reliability. In the following years, several studies [4, 6, 11, 10], sought to identify classes of tractable queries. These works provided conditions for tractability only for conjunctive queries without self-joins. The only exception is [5], which considers conjunctive queries with self-joins. We extend those results to a larger class of queries, and at the same time provide a very simple algorithm. Some other prior work is complimentary to ours, e.g., the results that consider the effects of functional dependencies [11].

Our results have applications to probabilistic inference on positive Boolean expressions [7]. For every tuple t in a structure $\mathbf{A}$, let X_t be a distinct Boolean variable. Every existential positive FO sentence Φ defines a positive DNF Boolean expression over the variables X_t, sometimes called *lineage expression*, whose probability is the same as $P_{\mathbf{A}}(\Phi)$. Our result can be used to classify the complexity of computing the probability of Positive DNF formulas defined by a fixed sentence Φ. For example, the two sentences[2]

$$\Phi_1 = R(x), S(x,y) \vee S(x,y), T(y) \vee R(x), T(y)$$
$$\Phi_2 = R(x), S(x,y) \vee S(x,y), T(y)$$

define two classes of positive Boolean DNF expressions (lineages):

$$F_1 = \bigvee_{a \in R, (a,b) \in S} X_a Y_{a,b} \vee \bigvee_{(a,b) \in S, b \in T} Y_{a,b}, Z_b \vee \bigvee_{a \in R, b \in S} X_a Y_b$$
$$F_2 = \bigvee_{a \in R, (a,b) \in S} X_a Y_{a,b} \vee \bigvee_{(a,b) \in S, b \in T} Y_{a,b}, Z_b$$

[1]This is the *marginal probability* $P_{\mathbf{A}}(\Phi) = \sum_{\mathbf{B}:\mathbf{B} \models \Phi} P_{\mathbf{A}}(\mathbf{B})$.

[2]We omit quantifiers and drop the conjunct they are clear from the context, e.g. $\Phi_2 = \exists x \exists y (R(x) \wedge S(x,y) \vee S(x,y) \wedge T(y))$.

PODS'10, June 6–11, 2010, Indianapolis, Indiana, USA.

Our result implies that, for each such class of Boolean formulas, either all formulas in that class can be evaluated in PTIME in the size of the formula, or the complexity for that class is hard for $FP^{\#P}$; e.g. F_1 can be evaluated in PTIME using our algorithm, while F_2 is hard.

The PTIME algorithm we present here relies in a critical way on an interesting connection between existential positive FO sentences and incidence algebras [16]. By using the Mobius inversion formula in incidence algebras we resolve a major difficulty of the evaluation problem: a sentence that is in PTIME may have a subexpression that is hard. This is illustrated by Φ_1 above, which is in PTIME, but has Φ_2 as a subexpression, which is hard; to evaluate Φ_1 one must avoid trying to evaluate Φ_2. Our solution is to express $P(\Phi)$ using Mobius' inversion formula: subexpressions of Φ that have a Mobius value of zero do not contribute to $P(\Phi)$, and this allows us to compute $P(\Phi)$ without computing its hard subexpressions. The Mobius inversion formula corresponds to the inclusion/exclusion principle, which is ubiquitous in probabilistic inference: the connection between the two in the context of probabilistic inference has already been recognized in [9]. However, to the best of our knowledge, ours is the first application that exploits the full power of Mobius inversion to remove hard subexpressions from a computation of probability.

Another distinguishing, and quite non-obvious aspect of our approach is that we apply our algorithm on the CNF, rather than the more commonly used DNF representation of existential, positive FO sentences. This departure from the common representation of existential, positive FO is necessary in order to handle correctly existential quantifiers.

We call sentences on which our algorithm works *safe*; those on which the algorithm fails we call *unsafe*. We prove a theorem stating that the evaluation problem of a safe query is in PTIME, and of an unsafe query is hard for $FP^{\#P}$: this establishes both the completeness of our algorithm and a dichotomy of all existential, positive FO sentences. The proof of the theorem is in two steps. First, we define a simple class of sentences called *forbidden sentences*, where each atom has at most two variables, and a set of simple *rewrite rules* on existential, positive FO sentences; we prove that the safe sentences can be characterized as those that cannot be rewritten into a forbidden sentence. Second, we prove that every forbidden sentence is hard for $FP^{\#P}$, using a direct, and rather difficult proof which we include in [3]. Together, these two results prove that every unsafe sentence is hard for $FP^{\#P}$, establishing the dichotomy. Notice that our characterization of safe queries is reminiscent of minors in graph theory. There, a graph H is called a minor of a graph G if H can be obtained from G through a sequence of edge contractions. "Being a minor of" defines a partial order on graphs: Robertson and Seymour's celebrated result states that any minor-closed family is characterized by a finite set of forbidden minors. Our characterization of safe queries is also done in terms of forbidden minors, however the order relation is more complex and the set of forbidden minors is infinite.

In the last part of the paper, we make a strong claim: that using Mobius' inversion formula is a *necessary* technique for completeness. Today's approaches to general probabilistic inference for Boolean expressions rely on combining (using some advanced heuristics) a few basic techniques: independence, disjointness, and conditioning. In conditioning, one chooses a Boolean variable X, then computes $P(F) = P(F \mid X)P(X) + P(F \mid \neg X)(1 - P(X))$. We extended these techniques to unions of conjunctive queries, an approach that is generally known as *lifted inference* [12, 15, 14] and given a PTIME algorithm based on these three techniques. The algorithm performs conditioning on subformulas of Φ instead of Boolean variables. We prove that this algorithm is not complete, by showing a formula Φ (Fig. 2) that is computable in PTIME, but for which it is not possible to compute using lifted inference that combines conditioning, independence, and disjointness on subformulas. On the other hand, we note that conditioning has certain practical advantages that are lost by Mobius' inversion formula: by repeated conditioning on Boolean variables, one can construct a Free Binary Decision Diagram [17], which has further applications beyond probabilistic inference. There seems to be no procedure to convert Mobius' inversion formula into FBDDs; in fact, we conjecture that the formula in Fig. 2 does not have an FBDD whose size is polynomial in that of the input structure.

Finally, we mention that a different way to define classes of Boolean formulas has been studied in the context of the *constraint satisfaction problem* (CSP). Creignou et al. [2, 1] showed that the counting version of the CSP problem has a dichotomy into PTIME and $FP^{\#P}$-hard. These results are orthogonal to ours: they define the class of formulas by specifying the set of Boolean operators, such as and/or/not/majority/parity etc, and do not restrict the shape of the Boolean formula otherwise. As a consequence, the only class where counting is in PTIME is defined by affine operators: all classes of monotone formulas are hard. In contrast, in our classification there exist classes of formulas that are in PTIME, for example the class defined by Φ_1 above.

2. BACKGROUND AND OVERVIEW

Prior Results A very simple PTIME algorithm for conjunctive queries without self-joins is discussed in [4, 6]. When the conjunctive query is connected, the algorithm chooses a variable that occurs in all atoms (called a *root variable*) and projects it out, computing recursively the probabilities of the sub-queries; if no root variable exists, then the query is $FP^{\#P}$-hard. When the conjunctive query is disconnected, then the algorithm computes the probabilities of the connected components, then multiples them. Thus, the algorithm alternates between two steps, called independent projection, and independent join. For example, consider the conjunctive query[3]:

$$\varphi \quad = \quad R(x,y), S(x,z)$$

The algorithm computes its probability by performing the following steps:

$$\begin{aligned} P(\varphi) &= 1 - \prod_{a \in A}(1 - P(R(a,y), S(a,z))) \\ P(R(a,y), S(a,z)) &= P(R(a,y)) \cdot P(S(a,z)) \\ P(R(a,y)) &= 1 - \prod_{b \in A}(1 - P(R(a,b))) \\ P(S(a,z)) &= 1 - \prod_{c \in A}(1 - P(S(a,c))) \end{aligned}$$

The first line projects out the root variable x, where A is the active domain of the probabilistic structure: it is based on fact that, in $\varphi \equiv \bigvee_{a \in A}(R(a,y), S(a,z))$, the sub-queries $R(a,y), S(a,z)$ are independent for distinct values of the constant a. The second line applies independent join; and the third and fourth lines apply independent project again.

This simple algorithm, however, cannot be applied to a query with self-joins because both the projection and the join step are incorrect. For a simple example, consider $R(x,y), R(y,z)$. Here y is a root variable, but the queries $R(x,a), R(a,z)$ and $R(x,b), R(b,z)$

[3]All queries are Boolean and quantifiers are dropped; in complete notation, φ is $\exists x.\exists y.\exists z.R(x,y), S(x,z)$.

are dependent (both depend on $R(a,b)$ and $R(b,a)$). Hence, it is not possible to do an independent projection on y. In fact, this query is $FP^{\#P}$-hard.

Queries with self-joins were analyzed in [5] based on the notion of an *inversion*. In a restricted form, an inversion consists of two atoms, over the same relational symbol, and two positions in those atoms, such that the first position contains a root variable in the first atom and a non-root variable in the second atom, and the second position contains a non-root / root pair of variables. In our example above, the atoms $R(x,y)$ and $R(y,z)$ and the positions 1 and 2 form an inversion: position 1 has variables x and y (non-root / root) and position 2 has variables y and z (root / non-root). The paper describes a first PTIME algorithm for queries without inversions, by expressing its probability in terms of several sums, each of which can be reduced to a polynomial size expression. Then, the paper notices that some queries with inversion can also be computed in polynomial time, and describes a second PTIME algorithm that uses one sum (called *eraser*) to cancel the effect of a another, exponentially sized sum. The algorithm succeeds if it can erase all exponentially sized sums (corresponding to sub-queries with inversions).

Our approach The algorithm that we describe in this paper is both more general (it applies to unions of conjunctive queries), and significantly simpler than either of the two algorithms in [5]. We illustrate it here on a conjunctive query with a self-join (S occurs twice):

$$\varphi \quad = \quad R(x_1), S(x_1,y_1), S(x_2,y_2), T(x_2)$$

Our algorithm starts by applying the inclusion-exclusion formula:

$$\begin{aligned} P(R(x_1), S(x_1,y_1), S(x_2,y_2), T(x_2)) = \\ P(R(x_1), S(x_1,y_1)) + P(S(x_2,y_2), T(y_2)) \\ -P(R(x_1), S(x_1,y_1) \vee S(x_2,y_2), T(x_2)) \end{aligned}$$

This is the dual of the more popular inclusion-exclusion formula for disjunctions; we describe it formally in the framework of incidence algebras in Sec. 3. The first two queries are without self-joins and can be evaluated as before. To evaluate the query on the last line, we simultaneously project out both variables x_1, x_2, writing the query as:

$$\psi \quad = \quad \bigvee_{a \in A} (R(a), S(a,y_1) \vee S(a,y_2), T(a))$$

The variables x_1, x_2 are chosen because they satisfy the following conditions: they occur in all atoms, and for the atoms with the same relation name (S in our case) they occur in the same position. We call such a set of variables *separator variables* (Sec. 4). As a consequence, sub-queries $R(a), S(a,y_1) \vee S(a,y_2), T(a)$ corresponding to distinct constants a are independent. We use this independence, then rewrite the sub-query into CNF and apply the inclusion/exclusion formula again:

$$\begin{aligned} &P(\psi) = 1 - \prod_{a \in A} (1 - P(R(a), S(a,y_1) \vee S(a,y_2), T(a))) \\ &R(a), S(a,y_1) \vee S(a,y_2), T(a) \equiv (R(a) \vee T(a)) \wedge S(a,y) \\ &P((R(a) \vee T(a)) \wedge S(a,y)) \\ &\quad = \quad P(R(a) \vee T(a)) + P(S(a,y)) - P(R(a) \vee T(a) \vee S(a,y)) \\ &\quad = \quad P(R(a)) + P(T(a)) - P(R(a)) \cdot P(T(a)) \\ &\qquad -1 + (1 - P(R(a)))(1 - P(T(a))) \prod_{b \in A} (1 - P(S(a,b))) \end{aligned}$$

In summary, the algorithm alternates between applying the inclusion/exclusion formula, and performing a simultaneous projection on separator variables: when no separator variables exists, then the query is $FP^{\#P}$-hard. The two steps can be seen as generalizations of the independent join, and the independent projection for conjunctive queries without self-joins.

Ranking Before running the algorithm, a rewriting of the query is necessary. Consider $R(x,y), R(y,x)$: it has no separator variable because neither x nor y occurs in both atoms on the same position. After a simple rewriting, however, the query can be evaluated by our algorithm: partition the relation $R(x,y)$ into three sets, according to $x < y$, $x = y$, $x > y$, call them $R^<, R^=, R^>$, and rewrite the query as $R^<(x,y), R^>(y,x) \vee R^=(z)$. Now x, z is a separator, because the three relational symbols are distinct. We call this rewriting *ranking* (Sec. 5). It needs to be done only once, before running the algorithm, since all sub-queries of a ranked queries are ranked. A similar but more general rewriting called *coverage* was introduced in [5]: ranking corresponds to the canonical coverage.

Incidence Algebras An immediate consequence of using the inclusion-exclusion formula is that sub-queries that happen to cancel out do not have to be evaluated. This turns out to be a fundamental property of the algorithm that allows it to be complete since, as we have explained, some queries are in PTIME but may have sub-queries that are hard. This cancellation is described by the Mobius inversion formula, which groups equal terms in the inclusion-exclusion expansion under coefficients called the *Mobius function*. Using this notion, it is easy to state when a query is PTIME: this happens if and only if all its sub-queries that have a non-zero Mobius function are in PTIME. Thus, while the algorithm itself could be described without any reference to the Mobius inversion formula, by simply using inclusion-exclusion, the Mobius function gives a key insight into what the algorithm does: it recurses only on sub-queries whose Mobius function is non-zero. In fact, we prove the following result (Theorem 6.6): for every finite lattice, there exists a query whose sub-queries generate precisely that lattice, such that all sub-queries are in PTIME except that corresponding to the bottom of the lattice. Thus, the query is in PTIME iff the Mobius function of the lattice bottom is zero. In other words, any formulation of the algorithm must identify, in some way, the elements with a zero Mobius function in an arbitrary lattice: queries are as general as any lattice. For that reason we prefer to expose the Mobius function in the algorithm rather than hide it under the inclusion/exclusion formula.

Lifted Inference At a deeper level, lattices and their associated Mobius function help us understand the limitations of alternative query evaluation algorithms. In Sec. 7 we study an evaluation algorithm based on lifted conditioning and disjointness. We show that conditioning is equivalent to replacing the lattice of sub-queries with a certain sub-lattice. By repeated conditioning one it is sometimes possible to simplify the lattice sufficiently to remove all hard sub-queries whose Mobius function is zero. However, we given an example of a lattice with 9 elements (Fig 2) whose bottom element has the Mobius function equal to zero, but where no conditioning can further restrict the lattice. Thus, the algorithm based on lifted conditioning makes no progress on this lattice, and cannot evaluate the corresponding query. By contrast, our algorithm based on Mobius' inversion formula will easily evaluate the query by skipping the bottom element (since its Mobius function is zero). Thus, our new algorithm based on Mobius' inversion formula is more general than existing techniques based on lifted inference. Finally, we comment on the implications for the completeness of the algorithm in [5].

In the rest of the paper we will refer to conjunctive queries and unions of conjunctive queries as conjunctive sentences, and exis-

tential positive FO sentences (or just positive FO sentences) respectively.

3. EXISTENTIAL POSITIVE FO AND INCIDENCE ALGEBRAS

We describe here the connection between positive FO and incidence algebras. We start with basic notations.

3.1 Existential Positive FO

Fix a vocabulary $\bar{R} = \{R_1, R_2, \ldots\}$. A *conjunctive* sentence φ is a first-order logical formula obtained from positive relational atoms using $\wedge$ and $\exists$:

$$\varphi \quad = \quad \exists \bar{x}.(r_1 \wedge \ldots \wedge r_k) \tag{2}$$

We allow the use of constants. $Var(\varphi) = \bar{x}$ denotes the set of variables in φ, and $Atoms(\varphi) = \{r_1, \ldots, r_k\}$ the set of atoms. Consider the undirected graph where the nodes are $Atoms(\varphi)$ and edges are pairs (r_i, r_j) s.t. r_i, r_j have a common variable. A *component* of φ is a connected component in this graph. Each conjunctive sentence φ can be written as:

$$\varphi \quad = \quad \gamma_1 \wedge \ldots \wedge \gamma_p$$

where each γ_i is a component; in particular, γ_i and γ_j do not share any common variables, when $i \neq j$.

A *disjunctive* sentence is an expression of the form:

$$\varphi' \quad = \quad \gamma'_1 \vee \ldots \vee \gamma'_q$$

where each γ'_i is a single component.

An *existential, positive* sentence Φ is obtained from positive atoms using $\wedge$, $\exists$ and $\vee$; we will refer to it briefly as *positive sentence*. We write a positive sentence either in DNF or in CNF:

$$\Phi \quad = \quad \varphi_1 \vee \ldots \vee \varphi_m \tag{3}$$
$$\Phi \quad = \quad \varphi'_1 \wedge \ldots \wedge \varphi'_M \tag{4}$$

where φ_i are conjunctive sentences in DNF (3), and φ'_i are disjunctive sentences in CNF (4). The DNF can be rewritten into the CNF by:

$$\Phi \quad = \quad \bigvee_{i=1,m} \bigwedge_{j=1,p_i} \gamma_{ij} = \bigwedge_{f} \bigvee_{i} \gamma_{if(i)}$$

where f ranges over functions with domain $[m]$ s.t. $\forall i \in [m]$, $f(i) \in [p_i]$. This rewriting can increase the size of the sentence exponentially[4]. Finally, we will often drop $\exists$ and $\wedge$ when clear from the context.

A classic result by Sagiv and Yannakakis [13] gives a necessary and sufficient condition for a logical implication of positive sentences written in DNF: if $\Phi = \bigvee_i \varphi_i$ and $\Phi' = \bigvee_j \varphi'_j$, then:

$$\Phi \Rightarrow \Phi' \quad \text{iff} \quad \forall i.\exists j.\varphi_i \Rightarrow \varphi'_j \tag{5}$$

No analogous property holds for CNF: $R(x,a), S(a,z)$ logically implies $R(x,y), S(y,z)$ (where a is a constant), but $R(x,a) \not\Rightarrow R(x,y), S(y,z)$ and $S(a,z) \not\Rightarrow R(x,y), S(y,z)$. We show in Sec. 5 a rewriting technique that enforces such a property.

3.2 Incidence Algebras

Next, we review the basic notions in incidence algebras following Stanley [16]. A finite *lattice* is a finite ordered set $(\hat{L}, \leq)$ where every two elements $u, v \in \hat{L}$ have a least upper bound $u \vee v$ and a greatest lower bound $u \wedge v$, usually called *join* and *meet*. Since it is finite, it has a minimum and a maximum element, denoted $\hat{0}, \hat{1}$. We denote $L = \hat{L} - \{\hat{1}\}$ (departing from [16], where L denotes $\hat{L} - \{\hat{0}, \hat{1}\}$). L is a meet-semi-lattice. The *incidence algebra* $I(\hat{L})$ is the algebra[5] of real (or complex) matrices t of dimension $|\hat{L}| \times |\hat{L}|$, where the only non-zero elements t_{uv} (denoted $t(u,v)$) are for $u \leq v$; alternatively, a matrix can be seen as a linear function $t : \mathbf{R}^{\hat{L}} \to \mathbf{R}^{\hat{L}}$. Two matrices are of key importance in incidence algebras: $\zeta_{\hat{L}} \in I(\hat{L})$, defined as $\zeta_{\hat{L}}(u,v) = 1$ forall $u \leq v$; and its inverse, the Mobius function $\mu_{\hat{L}} : \{(u,v) \mid u, v \in \hat{L}, u \leq v\} \to Z$, defined by:

$$\mu_{\hat{L}}(u,u) \quad = \quad 1$$
$$\mu_{\hat{L}}(u,v) \quad = \quad - \sum_{w:u<w\leq v} \mu_{\hat{L}}(w,v)$$

We drop the subscript and write μ when $\hat{L}$ is clear from the context.

The fact that μ is the inverse of ζ means the following thing. Let $f : \hat{L} \to \mathbf{R}$ be a real function defined on the lattice. Define a new function g as $g(v) = \sum_{u\leq v} f(u)$. Then $f(v) = \sum_{u\leq v} \mu(u,v)g(u)$. This is called Mobius' inversion formula, and is a key piece of our algorithm. Note that it simply expresses the fact that $g = \zeta(f)$ implies $f = \mu(g)$.

3.3 Their Connection

A *labeled lattice* is a triple $\hat{\mathbf{L}} = (\hat{L}, \leq, \lambda)$ where $(\hat{L}, \leq)$ is a lattice and λ assigns to each element in $u \in \hat{L}$ a positive FO sentence $\lambda(u)$ s.t. $\lambda(u) \equiv \lambda(v)$ iff $u = v$.

DEFINITION 3.1. *A D-lattice is a labeled lattice $\hat{\mathbf{L}}$ where, forall $u \neq \hat{1}$, $\lambda(u)$ is conjunctive, forall u, v, $\lambda(u \wedge v)$ is logically equivalent to $\lambda(u) \wedge \lambda(v)$, and $\lambda(\hat{1}) \equiv \bigvee_{u<\hat{1}} \lambda(u)$.*

A C-lattice is a labeled lattice $\hat{\mathbf{L}}$ where, forall $u \neq \hat{1}$, $\lambda(u)$ is disjunctive, forall u, v, $\lambda(u \wedge v)$ is logically equivalent to $\lambda(u) \vee \lambda(v)$, and $\lambda(\hat{1}) = \bigwedge_{u<\hat{1}} \lambda(u)$.

In a D-lattice, $u \leq v$ iff $\lambda(u) \Rightarrow \lambda(v)$. This is because $\lambda(u) = \lambda(u \wedge v)$ is logically equivalent to $\lambda(u) \wedge \lambda(v)$. Similarly, in a C-lattice, $u \leq v$ iff $\lambda(v) \Rightarrow \lambda(u)$. If $\hat{\mathbf{L}}$ is a D- or C-lattice, we say $\hat{\mathbf{L}}$ *represents* $\Phi = \lambda(\hat{1})$.

PROPOSITION 3.2 (INVERSION FORMULA FOR POSITIVE FO). *Fix a probabilistic structure $(\mathbf{A}, P)$ and a positive sentence Φ; denote $P_{\mathbf{A}}$ as P. Let $\hat{\mathbf{L}}$ be either a D-lattice or a C-lattice representing Φ. Then:*

$$P(\Phi) = P(\lambda(\hat{1})) \quad = \quad -\sum_{v<\hat{1}} \mu_L(v,\hat{1}) P(\lambda(v)) \tag{6}$$

PROOF. The proof for the D-lattice is from [16]. Denote $f(u) = P(\lambda(u) \wedge \neg(\bigvee_{v<u} \lambda(v)))$. Then:

$$P(\lambda(u)) = \sum_{v\leq u} f(v) \quad \Rightarrow \quad f(u) = \sum_{v\leq u} \mu(v,u) P(\lambda(v))$$

The claim follows by setting $u = \hat{1}$ and noting $f(\hat{1}) = 0$. For a C-lattice, write $\lambda'(u) = \neg\lambda(u)$. Then $P(\lambda(\hat{1})) = 1 - P(\lambda'(\hat{1})) = 1 + \sum_{v<\hat{1}} \mu(v,\hat{1}) P(\lambda'(v))$ and the claim follows from the fact that $\sum_{v\in\hat{L}} \mu(v,\hat{1}) = 0$. □

[4]Our algorithm runs in PTIME *data complexity*; we do not address the expression complexity in this paper.

[5]An *algebra* is a vector space plus a multiplication operation [16].

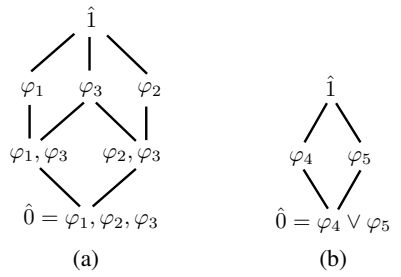


Figure 1: The D-lattice (a) and the C-lattice (b) for Φ (Ex. 3.3).

The proposition generalizes the well known inclusion/exclusion formula (for D-lattices), and its less well known dual (for C-lattices):

$$\begin{aligned} P(a \vee b \vee c) &= P(a) + P(b) + P(c) \\ &\quad -P(a \wedge b) - P(a \wedge c) - P(b \wedge c) + P(a \wedge b \wedge c) \\ P(a \wedge b \wedge c) &= P(a) + P(b) + P(c) \\ &\quad -P(a \vee b) - P(a \vee c) - P(b \vee c) + P(a \vee b \vee c) \end{aligned}$$

We show how to construct a canonical D-lattice, $\hat{\mathbf{L}}_D(\Phi)$ that represents a positive sentence Φ. Start from the DNF in Eq.(3), and for each subset $s \subseteq [m]$ denote $\varphi_s = \bigwedge_{i \in s} \varphi_i$. Let $\hat{L}$ be the set of these conjunctive sentences, up to logical equivalence, and ordered by logical implication (hence, $|L| \leq 2^m$). Label each element $u \in \hat{L}$, $u \neq \hat{1}$, with its corresponding φ_s (choose any, if there are multiple equivalent ones), and label $\hat{1}$ with $\bigvee_{s \neq \emptyset} \varphi_s$ ($\equiv \Phi$). We denote the resulting D-lattice $\hat{\mathbf{L}}_D(\Phi)$. Similarly, $\hat{\mathbf{L}}_C(\Phi)$ is the C-lattice that represents Φ, obtained from the CNF of Φ in Eq.(4), setting $\varphi'_s = \bigvee_{i \in s} \varphi'_i$.

The first main technique of our algorithm is this. Given Φ, compute its C-lattice, then use Eq.(6) to compute $P(\Phi)$; we explain later why we use the C-lattice instead of the D-lattice. It remains to compute the probability of disjunctive sentences $P(\lambda(u))$: we show this in the next section. The power of this technique comes from the fact that, whenever $\mu(u, \hat{1}) = 0$, then we do not need to compute the corresponding $P(\lambda(u))$. As we explain in Sec. 7 this is strictly more powerful than the current techniques used in probabilistic inference, such as lifted conditioning.

Example 3.3 Consider the following positive sentence:

$$\begin{aligned} \Phi &= R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(y_2) \vee R(x_3), T(y_3) \\ &= \varphi_1 \vee \varphi_2 \vee \varphi_3 \end{aligned}$$

The Hasse diagram of the D-lattice $\mathbf{L}_D(\Phi)$ is shown in Fig. 1 (a). There are eight subsets $s \subseteq [3]$, but only seven up to logical equivalence, because[6] $\varphi_1, \varphi_2 \equiv \varphi_1, \varphi_2, \varphi_3$. The values of the Mobius function are, from top to bottom: $1, -1, -1, -1, 1, 1, 0$, hence the inversion formula is[7]:

$$P(\Phi) = P(\varphi_1) + P(\varphi_2) + P(\varphi_3) - P(\varphi_1\varphi_3) - P(\varphi_2\varphi_3)$$

The Hasse diagram of the C-lattice $\mathbf{L}_C(\Phi)$ is shown in Fig. 1 (b). To see this, first express Φ in CNF:

$$\begin{aligned} \Phi &= (R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(y_2) \vee R(x_3)) \wedge \\ &\quad (R(x_4), S(x_4, y_4) \vee S(x_5, y_5), T(y_5) \vee T(y_6)) \\ &= (R(x_3) \vee S(x_2, y_2), T(y_2)) \wedge (R(x_4), S(x_4, y_4) \vee T(y_6)) \\ &= \varphi_4 \wedge \varphi_5 \end{aligned}$$

Note that $\hat{0}$ is labeled with $\varphi_4 \vee \varphi_5 \equiv R(x_3) \vee T(y_6)$. The inversion formula here is:

$$P(\Phi) = P(\varphi_4) + P(\varphi_5) - P(\varphi_4 \vee \varphi_5)$$

where $\varphi_4 \vee \varphi_5 \equiv R(x_3) \vee T(y_6)$.

3.4 Minimization

By *minimizing* a conjunctive sentence φ we mean replacing it with an equivalent sentence φ_0 that has the smallest number of atoms. A disjunctive sentence $\Phi = \bigvee \gamma_i$ is *minimized* if every conjunctive sentence is minimized and there is no homomorphism $\varphi_i \Rightarrow \varphi_j$ for $i \neq j$. If such a homomorphism exists, then we call φ_j redundant: clearly we can remove it from the expression Φ without affecting its semantics.

For the purpose of D-lattices, it doesn't matter if we minimize the sentence or not: if the sentence is not minimized, then one can show that all lattice elements corresponding to redundant terms have the Mobius function equal to zero. More precisely, any two D-lattices that represent the same sentence have the same set of elements with a non-zero Mobius function: we state this fact precisely in the remainder of this section. A similar fact does not hold in general for C-lattices, but it holds over ranked structures (Sec. 5.2).

An element u in a lattice *covers* v if $u > v$ and there is no w s.t. $u > w > v$. An *atom*[8] is an element that covers $\hat{0}$; a *co-atom* is an element covered by $\hat{1}$. An element u is called *co-atomic* if it is a meet of coatoms. Let L_0 denote the set of co-atomic elements: L_0 is a meet semilattice, and $\hat{L}_0 = L_0 \cup \{\hat{1}\}$ is a lattice. We prove the following in the full paper [3]:

PROPOSITION 3.4. *(1) If $u \in L$ and $\mu_{\hat{L}}(u, \hat{1}) \neq 0$ then u is co-atomic. (2) Forall $u \in L_0$, $\mu_{\hat{L}}(u, \hat{1}) = \mu_{\hat{L}_0}(u, \hat{1})$.*

Let $\hat{\mathbf{L}}$ and $\hat{\mathbf{L}}'$ be D-lattices representing the sentences Φ and Φ'. If $\Phi \equiv \Phi'$, then $\hat{\mathbf{L}}$ and $\hat{\mathbf{L}}'$ have the same co-atoms, up to logical equivalence. Indeed, we can write Φ as the disjunction of co-atom labels in $\hat{\mathbf{L}}$, and one co-atom cannot imply another. Thus, by applying Eq.(5) in both directions, we get a one-to-one correspondence between the co-atoms of $\hat{\mathbf{L}}$ and $\hat{\mathbf{L}}'$, indicating logical equivalence. It follows from Prop. 3.4 that, when D-lattices represent equivalent formulas, the set of labels $\lambda(u)$ where $\mu(u, \hat{1}) \neq 0$ are equivalent. Thus, an algorithm that inspects only these labels is independent of the particular representation of a sentence.

A similar property fails on C-lattices, because Eq.(5) does not extend to CNF. For example, $\Phi = R(x, a), S(a, z)$ and $\Phi' = R(x, a), S(a, z), R(x', y'), S(y', z')$ are logically equivalent, but have different co-atoms. The co-atoms of Φ are $R(x, a)$ and $S(a, z)$ (the C-lattice is V-shaped, as in Fig. 1 (b)), and the co-atoms of Φ' are $R(x, a)$, $(R(x', y'), S(y', z'))$, and $S(a, z)$ (the C-lattice is W-shaped, as in Fig. 1 (a)).

4. INDEPENDENCE AND SEPARATORS

Next, we show how to compute the probability of a disjunctive sentence $\bigvee_i \gamma_i$; this is the second technique used in our algorithm,

[6]There exists a homomorphism $\varphi_1, \varphi_2, \varphi_3 \rightarrow \varphi_1, \varphi_2$ that maps $R(x_3)$ to $R(x_1)$ and $T(y_3)$ to $T(y_2)$.

[7]One can arrive at the same expression by using inclusion-exclusion instead of Mobius' inversion formula, and noting that $\varphi_1, \varphi_2 \equiv \varphi_1, \varphi_2\varphi_3$, hence these two terms cancel out in the inclusion-exclusion expression.

[8]Not to be confused with a relational atom r_i in (2).

and consists of eliminating, simultaneously, one existential variable from each γ_i, by exploiting independence.

Let φ be a conjunctive sentence. A *valuation* h is a substitution of its variables with constants; $h(\varphi)$, is a set of ground tuples. We call two conjunctive sentences φ_1, φ_2 *tuple-independent* if for all valuations h_1, h_2, we have $h_1(\varphi_1) \cap h_2(\varphi_2) = \emptyset$. Two positive sentences Φ, Φ' are *tuple-independent* if, after expressing them in DNF, $\Phi = \bigvee_i \varphi_i$, $\Phi' = \bigvee_j \varphi'_j$, all pairs φ_i, φ'_j are independent.

Let $\Phi_1, \ldots, \Phi_m$ be positive sentences s.t. any two are tuple-independent. Then:

$$P(\bigvee_i \Phi_i) \quad = \quad 1 - \prod_i (1 - P(\Phi_i))$$

This is because the m lineage expressions for Φ_i depend on disjoint sets of Boolean variables, and therefore they are independent probabilistic events. In other words, tuple-independence is a sufficient condition for independence in the probabilistic sense. Although it is only a sufficient condition, we will abbreviate tuple-independence with independence in this section.

Let φ be a positive sentence, $V = \{x_1, \ldots, x_m\} \subseteq Vars(\varphi)$, and a a constant. Denote $\varphi[a/V] = \varphi[a/x_1, \ldots, a/x_m]$ (all variables in V are substituted with a).

DEFINITION 4.1. *Let $\varphi = \bigvee_{i=1,m} \gamma_i$ be a disjunctive sentence. A* separator *is a set of variables $V = \{x_1, \ldots, x_m\}$, $x_i \in Var(\gamma_i)$, such that for all $a \neq b$, $\varphi[a/V]$, $\varphi[b/V]$ are independent.*

PROPOSITION 4.2. *Let φ be a disjunctive sentence with a separator V, and $(\mathbf{A}, P)$ a probabilistic structure with active domain D. Then:*

$$P(\varphi) \quad = \quad 1 - \prod_{a \in D} (1 - P(\varphi[a/V])) \qquad (7)$$

The claim follows from the fact that $\varphi \equiv \bigvee_{a \in D} \varphi[a/V]$ on all structures whose active domain is included in D.

In summary, to compute the probability of a disjunctive sentence, we find a separator, then apply Eq.(7): each expression $\varphi[a/V]$ is a positive sentence, simpler than the original one (it has strictly fewer variables in each atom) and we apply again the inversion formula. This technique, by itself, is not complete: we need to "rank" the relations in order to make it complete, as we show in the next section. Before that, we illustrate with an example.

Example 4.3 Consider $\varphi = R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(x_2)$. Here $\{x_1, x_2\}$ is a separator. To see this, note that for any constants $a \neq b$, the sentences $\varphi[a] = R(a), S(a, y_1) \vee S(a, y_2), T(a)$ and $\varphi[b] = R(b), S(b, y_1) \vee S(b, y_2), T(b)$ are independent, because the former only looks at tuples that start with a, while the latter only looks at tuples that start with b.

Consider $\varphi = R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(y_2)$. This sentence has no separator. For example, $\{x_1, x_2\}$ is not a separator because both sentences $\varphi[a]$ and $\varphi[b]$ have the atom $T(y_2)$ in common: if two homomorphisms h_1, h_2 map y_2 to some constant c, then $T(c) \in h_1(\varphi[a]) \cap h_2(\varphi[b])$, hence they are dependent. The set $\{x_1, y_2\}$ is also not a separator, because $\varphi[a]$ contains the atom $S(a, y_1)$, $\varphi[b]$ contains the atom $S(x_2, b)$, and these two can be mapped to the common ground tuple $S(a, b)$.

We end with a necessary condition for V to be a separator.

DEFINITION 4.4. *If γ is a component, a variable of γ is called a* root variable *if it occurs in all atoms of γ.*

Note that components do not necessarily have root variables, e.g., $R(x), S(x, y), T(y)$. We have:

PROPOSITION 4.5. *If V is a separator of $\bigvee_i \gamma_i$, then each separator variable $x_i \in Vars(\gamma_i)$ is a root variable for γ_i.*

The claim follows from the fact that, if r is any atom in φ_i that does not contain x_i: then r is unchanged in $\gamma_i[a]$ and in $\gamma_i[b]$, hence they are not independent.

5. RANKING

In this section, we define a simple restriction on all formulas and structures that simplifies our later analysis: we require that, in each relation, the attributes may be strictly ordered $A_1 < A_2 < \ldots$ We show how to alter any positive sentence and probabilistic structure to satisfy this constraint, without changing the sentence probability. This is a necessary preprocessing step for our algorithm to work, and a very convenient technique in the proofs.

5.1 Ranked Structures

Throughout this section, we use $<$ to denote a total order on the active domain of a probabilistic structure (such an order always exists). In our examples, we assume that $<$ is the natural ordering on integers, but the order may be chosen arbitrarily in general.

DEFINITION 5.1. *A relation instance R is* ranked *if every tuple $R(a_1, \ldots, a_k)$ is such that $a_1 < \cdots < a_k$. A probabilistic structure is* ranked *if all its relations are ranked.*

To motivate ranked structures, we observe that the techniques given in previous sections do not directly lead to a complete algorithm. We illustrate by reviewing the example in Sec. 2: $\gamma = R(x, y), R(y, x)$. This component is connected, so we cannot use Mobius inversion to simplify it, and we also cannot apply Eq.(7) because there is no separator: indeed, $\{x\}$ is not a separator because $R(a, y), R(y, a)$ and $R(b, y), R(y, b)$ are not independent (they share the tuple $R(a, b)$), and by symmetry neither is $\{y\}$. However, consider a structure with a unary relation R_{12} and binary relations $R_{1<2}$, $R_{2<1}$ defined as:

$$R_{12} = \pi_{X_1}(\sigma_{X_1 = X_2}(R)) \qquad R_{2<1} = \pi_{X_2 X_1}(\sigma_{X_2 < X_1}(R))$$
$$R_{1<2} = \sigma_{X_1 < X_2}(R)$$

Here, we use X_i to refer to the i-th attribute of R. This is a ranked structure: in both relations $R_{1<2}$ and $R_{2<1}$ the first attribute is less than the second. Moreover: $\gamma \equiv R_{12}(z) \vee R_{1<2}(x, y), R_{2<1}(x, y)$ and now $\{z, x\}$ is a separator, because $R_{1<2}(a, y), R_{2<1}(a, y)$ and $R_{1<2}(b, y), R_{2<1}(b, y)$ are independent. Thus, Eq.(7) applies to the formula over the ranked structure, and we can compute the probability of γ in polynomial time.

DEFINITION 5.2. *A positive sentence is in* reduced form *if each atom $R(x_1, \ldots, x_k)$ is such that (a) each x_i is a variable (i.e. not a constant), and (b) the variables $x_1, \ldots, x_k$ are distinct.*

We now prove that the evaluation of any sentence can be reduced to an equivalent sentence over a ranked structure, and we further guarantee that the resulting sentence is in reduced form.

PROPOSITION 5.3. *Let Φ_0 be positive sentence. Then, there exists a sentence Φ in reduced form such that for any structure $\mathbf{A}_0$, one can construct in polynomial time a ranked structure $\mathbf{A}$ such that $P_{\mathbf{A}_0}(\Phi_0) = P_{\mathbf{A}}(\Phi)$.*

PROOF. Let $R(X_1, \ldots, X_k)$ be a relation symbol and let ρ be a maximal, consistent conjunction of order predicates involving attributes of R and the constants occurring in Φ_0. Thus, for any pair of attributes names or constants y, z, ρ implies exactly one of

$y < z$, $y = z$, $y > z$. Before describing Φ, we show to construct the ranked structure $\mathbf{A}$ from an unranked one $\mathbf{A}_0$. We say X_j is *unbound* if $\rho \not\Rightarrow X_j = c$ for any constant c. The ranked structure will have one symbol R^ρ for every symbol R in the unranked structure and every maximal consistent predicate ρ. The instance $\mathbf{A}$ is computed as $R^\rho = \pi_{\bar{X}}(\sigma_\rho(R))$ where $\bar{X}$ contains one X_j in each class of unbound attributes that are equivalent under ρ, listed in increasing order according to ρ. Clearly $\mathbf{A}$ can be computed in PTIME from $\mathbf{A}_0$.

We show now how to rewrite any positive sentence Φ_0 into an equivalent, reduced sentence Φ over ranked structures s.t. $P_{\mathbf{A}_0}(\Phi_0) = P_{\mathbf{A}}(\Phi)$. We start with a conjunctive sentence $\varphi = r_1, \ldots, r_n$ and let R_i denote the relation symbol of r_i. Consider a maximally consistent predicate ρ_i on the attributes of R_i, for each $i = 1, n$, and let $\rho \equiv \rho_1, \ldots, \rho_n$ be the conjunction. We say that ρ is *consistent* if there is a valuation h such that $h(\varphi) \models \rho$. Given a consistent ρ, divide the variables into equivalence classes of variables that ρ requires to be equal, and choose one representative variable from each class. Let $r_i^{\rho_i}$ be the result of changing $R_i(x_1, \ldots, x_k)$ to $R_i^{\rho_i}(y_1, \ldots, y_m)$, where $y_1, \ldots, y_m$ are chosen as follows. Consider the unbound attribute classes in R_i, in increasing order according to ρ_i. Choose y_p to be the representative of a variable that occurs in the position of an attribute in the p-th class of unbound attributes. This works because the position of any unbound attribute X must have a variable: if there is a constant a, then $h(r_i) \models X = a$ for all valuations h. But $\rho_i \Rightarrow X \neq a$ so this contradicts the assumption that ρ is consistent. Using a similar argument, we can show that each y_i is distinct, so $r_i^{\rho_i}$ is in reduced form. Furthermore, $\varphi \equiv \bigwedge_\rho r_1^{\rho_1}, \ldots, r_n^{\rho_n}$ where the disjunction ranges over all maximal ρ_i such that ρ is consistent. For a positive sentence Φ_0, we apply the above procedure to each conjunctive sentence in the DNF of Φ_0 to yield a sentence in reduced form on the ranked relations R^ρ. $\square$

Example 5.4 Let $\varphi = R(x, a), R(a, x)$. If we define the ranked relations $R_1 = \pi_{X_2}(\sigma_{X_1=a}(R))$, $R_2 = \pi_{X_1}(\sigma_{X_2=a}(R))$, and $R_{12} = \pi_\emptyset(\sigma_{X_1=X_2=a}(R))$, we have $\varphi \equiv R_1(x), R_2(x) \vee R_{12}()$.

Next, consider $\varphi = R(x), S(x, x, y), S(u, v, v)$. Define

$$\begin{aligned} S_{123} &= \pi_{X_1}(\sigma_{X_1=X_2=X_3}(S)) \\ S_{23<1} &= \pi_{X_2X_1}(\sigma_{X_2=X_3<X_1}((S)) \end{aligned}$$

and so on. We can rewrite φ as:

$$\begin{aligned} \varphi \equiv\ & R(x), S_{123}(x) \\ \vee\ & R(x), S_{12<3}(x, y), S_{1<23}(u, v) \vee R(x), S_{12<3}(x, y), S_{23<1}(v, u) \\ \vee\ & R(x), S_{3<12}(y, x), S_{1<23}(u, v) \vee R(x), S_{3<12}(y, x), S_{23<1}(v, u) \end{aligned}$$

and note that these relations are ranked. $\square$

Thus, when computing $P_{\mathbf{A}}(\Phi)$, we may conveniently assume w.l.o.g. that $\mathbf{A}$ is ranked and Φ is in reduced form. When we replace separator variables with a constant as in Eq.(7), we can easily restore the formula to reduced form. Given a disjunctive sentence φ in reduced form and a separator V, we remove a from $\varphi[a/V]$ as follows. For each relation R, suppose the separator variables occur at position X_i of R. Then we remove all rows from R where $X_i \neq a$, reduce the arity of R by removing column i, and remove $x_i = a$ from all atoms $R(x_1, \ldots, x_k)$ in $\varphi[a/V]$.

We end this section with two applications of ranking. The first shows a homomorphism theorem for CNF sentences.

PROPOSITION 5.5. *Assume all structures to be ranked, and all sentences to be in reduced form.*

- *If φ, φ' are conjunctive sentences, and φ is satisfiable over ranked structures*[9] *then $\varphi \Rightarrow \varphi'$ iff there exists a homomorphism $h : \varphi' \to \varphi$.*
- *Formula (5) holds for positive sentences in DNF.*
- *The dual of (5) holds for positive sentences in CNF:*

$$\textstyle\bigwedge_i \varphi_i \Rightarrow \bigwedge_j \varphi'_j \quad \textit{iff} \quad \forall j.\exists i.\varphi_i \Rightarrow \varphi'_j$$

The proof is the full version of the paper [3]. The first two items are known to fail for conjunctive sentences with order predicates: for example $R(x, y), R(y, x)$ logically implies $R(x, y), x \leq y$, but there is no homomorphism from the latter to the former. They hold for ranked structures because there is a strict total order on the attributes of each relation. The last item implies the following. If $\hat{\mathbf{L}}$ and $\hat{\mathbf{L}}'$ are two C-lattices representing equivalent sentences, then they have the same co-atoms. In conjunction with Prop. 3.4, this implies that an algorithm that ignores lattice elements where $\mu(u, \hat{1}) = 0$ does not depend on the representation of the positive sentence. This completes our discussion at the end of Sec. 3.

The second result shows how to handle atoms without variables.

PROPOSITION 5.6. *Let γ_0, γ_1 be components in reduced form s.t. $Var(\gamma_0) = \emptyset$, $Var(\gamma_1) \neq \emptyset$. Then γ_0, γ_1 are independent.*

PROOF. Note that γ_0 contains a single atom $R()$; if it had two atoms then it is not a component. Since γ_1 is connected, each atom must have at least one variable, hence it cannot have the same relation symbol $R()$. $\square$

Let $\varphi = \bigvee \gamma_i$ be a disjunctive sentence, $\varphi_0 = \bigvee_{i:Var(\gamma_i)=\emptyset} \gamma_i$ and $\varphi_1 = \bigvee_{i:Var(\gamma_i)\neq\emptyset} \gamma_i$. It follows that:

$$P(\varphi) = 1 - (1 - P(\varphi_0))(1 - P(\varphi_1)) \tag{8}$$

5.2 Finding a Separator

Assuming structures to be ranked, we give here a necessary and sufficient condition for a disjunctive sentence in reduced form to have a separator, which we use both in the algorithm and to prove hardness for $FP^{\#P}$. We need some definitions first.

Let $\varphi = \gamma_1 \vee \ldots \vee \gamma_m$ be a disjunctive sentence, in reduced form. Throughout this section we assume that φ is minimized and that $Var(\gamma_i) \cap Var(\gamma_j) = \emptyset$ for all $i \neq j$ (if not, then rename the variables). Two atoms $r \in Atoms(\gamma_i)$ and $r' \in Atoms(\gamma_j)$ are called *unifiable* if they have the same relational symbol. We may also say r, r' *unify*. It is easy to see that γ_i and γ_j contain two unifiable atoms iff they are not tuple-independent. Two variables x, x' are *unifiable* if there exist two unifiable atoms r, r' such that x occurs in r at the same position that x' occurs in r'. This relationship is reflexive and symmetric. We also say that x, x' are *recursively unifiable* if either x, x' are unifiable, or there exists a variable x'' such that x, x'' and x', x'' are recursively unifiable.

A variable x is *maximal* if it is only recursively unifiable with root variables. Hence all maximal variables are root variables. The following are canonical examples of sentences where each component has a root variable, but there are no maximal variables:

$$\begin{aligned} h_0 &= R(x_0), S_1(x_0, y_0), T(y_0) \\ h_1 &= R(x_0), S_1(x_0, y_0) \vee S_1(x_1, y_1), T(y_1) \\ h_2 &= R(x_0), S_1(x_0, y_0) \vee S_1(x_1, y_1), S_2(x_1, y_1) \vee S_2(x_2, y_2), T(y_2) \\ &\ldots \\ h_k &= R(x_0), S_1(x_0, y_0) \vee S_1(x_1, y_1), S_2(x_1, y_1) \vee \\ &\quad \ldots \vee S_{k-1}(x_{k-1}, y_{k-1}), S_k(x_{k-1}, y_{k-1}) \vee (S_k(x_k, y_k), T(y_k) \end{aligned}$$

[9] Meaning: it is satisfied by at least one (ranked) structure.

In each h_k, $k \geq 1$, the root variables are x_{i-1}, y_i for $i = 1, k-1$, and there are no maximal variables.

Maximality propagates during unification: if x is maximal and x, x' unify, then x' must be maximal because otherwise x would recursively unify with a non-root variable.

Let W_i be the set of maximal variables occurring in γ_i. If an atom in γ_i unifies with an atom in γ_j, then $|W_i| = |W_j|$ because the two atoms contain all maximal variables in each component, and maximality propagates through unification. Since the structures are ranked, for every i there exists a total order on the maximal variables in W_i: $x_{i1} < x_{i2} < \ldots$ The *rank* of a variable $x \in W_i$ is the position where it occurs in this order. The following result gives us a means to find a separator if it exists:

PROPOSITION 5.7. *A disjunctive sentence has a separator iff every component has a maximal variable. In that case, the set comprising maximal variables with rank* 1 *forms a separator.*

PROOF. Consider the disjunctive sentence $\varphi = \bigvee_{i=1}^{m} \gamma_i$ and set of variables $V = \{x_1, \ldots, x_m\}$ s.t. $x_i \in Vars(\gamma_i)$, $i = 1, m$. It is straightforward to show that V is a separator iff any pair of unifiable atoms have a member of V in the same position. Hence, if V is a separator, then each $x_i \in V$ can only (recursively) unify with another $x_j \in V$. Since x_j is a root variable (Prop. 4.5), each $x_i \in Vars(\gamma_i)$ is maximal, as desired.

Now suppose every component has a maximal variable. Choose V such that x_i is the maximal variable in γ_i with rank 1. If two atoms r, r' unify, then they have maximal variables occurring in the same positions. In particular, the first maximal variable has rank 1, and thus is in V. We conclude that V is a separator. □

For a trivial illustration of this result, consider the disjunctive sentence $R(x,y), S(x,y) \vee S(x',y'), T(x',y')$. All variables are root variables, and the sets of maximal variables are $W_1 = \{x, y\}$, $W_2 = \{x', y'\}$. We break the tie by using the ranking: choosing arbitrarily rank 1, we obtain the separator $\{x, x'\}$. (Rank 2 would gives us the separator $\{y, y'\}$). A more interesting example is:

Example 5.8 In φ, not all root variables are maximal:

$$\varphi \quad = \quad R(z_1, x_1), S(z_1, x_1, y_1) \vee S(z_2, x_2, y_2), T(z_2, y_2) \vee R(z_3, x_3), T(z_3, y_3)$$

The root variables are z_1, x_1, z_2, y_2, z_3. The sets of maximal variables in each component are $W_1 = \{z_1\}$, $W_2 = \{z_2\}$, $W_3 = \{z_3\}$, and the set $\{z_1, z_2, z_3\}$ is a separator.

6. THE ALGORITHM

Algorithm 6.1 takes as input a ranked probabilistic structure $\mathbf{A}$ and a positive sentence Φ in reduced form (Def 5.2), and computes the probability $P(\Phi)$, or fails. The algorithm proceeds recursively on the structure of the sentence Φ. The first step applies the Mobius inversion formula Eq.(6) to the C-lattice for Φ, expressing $P(\Phi)$ as a sum of several $P(\varphi)$, where each φ is a disjunctive sentence. Skipping those φ's where the Mobius function is zero, for all others it proceeds with the second step. Here, the algorithm first minimizes $\varphi = \bigvee \gamma_i$, then computes $P(\bigvee \gamma_i)$, by using Eq.(8), and Eq.(7). For the latter, the algorithm needs to find a separator first, as described in Sec. 5.2: if none exists, then the algorithm fails.

The expression $P(\varphi_0)$ represents the base case of the algorithm: this is when the recursion stops, when all variables have been substituted with constants from the structure $\mathbf{A}$. Notice that φ_0 is of the form $\bigvee r_i$, where each r_i is a ground atom. Its probability is $1 - \prod_i (1 - P(r_i))$, where P is the probability function of the probabilistic structure $(\mathbf{A}, P)$. We illustrate the algorithm with two examples.

Algorithm 6.1 Algorithm for Computing $P(\Phi)$
`Input`: Positive sentence Φ in reduced form;
Ranked structure $(\mathbf{A}, p)$ with active domain D
`Output`: $P(\Phi)$

Function MobiusStep(Φ) /* Φ = positive sentence */
  Let $\hat{\mathbf{L}} = \hat{\mathbf{L}}_C(\Phi)$ be a C-lattice representing Φ
  Return $\sum_{u<\hat{1}} \mu_{\hat{L}}(u, \hat{1}) *$ **IndepStep**($\lambda(u)$)
□
Function IndepStep(φ) /* $\varphi = \bigvee_i \gamma_i$ */
  Minimize φ (Sec. 3.4)
  Let $\varphi = \varphi_0 \vee \varphi_1$
    where: $\varphi_0 = \bigvee_{i:Var(\gamma_i)=\emptyset} \gamma_i$, $\varphi_1 = \bigvee_{i:Var(\gamma_i)\neq\emptyset} \gamma_i$
  Let V = a separator for φ_1 (Sec. 5.2)
  If (no separator exists) **then FAIL (UNSAFE)**
  Let $p_0 = P(\varphi_0)$
  Let $p_1 = 1 - \prod_{a \in D}(1 -$ **MobiusStep**$(\varphi_1[a/V]))$
    /* Note: assume $\varphi_1[a/V]$ is reduced (Sec.5) */
  Return $1 - (1 - p_0)(1 - p_1)$.
□

Example 6.1 Let $\Phi = R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(y_2) \vee R(x_3), T(y_3)$. This example is interesting because, as we will show, the subexpression $R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(y_2)$ is hard (it has no separator), but the entire sentence is in PTIME. The algorithm computes the C-lattice, shown in Fig. 1 (b), then expresses $P(\Phi) = P(\varphi_4) + P(\varphi_5) - P(\varphi_6)$ where $\varphi_6 = R(x) \vee T(y)$ (see Example 3.3 for notations). Next, the algorithm applies the independence step to each of φ_4, φ_5, φ_6; we illustrate here for $\varphi_4 = R(x_3) \vee S(x_2, y_2), T(y_2)$ only; the other expressions are similar. Here, $\{x_3, y_2\}$ is a set of separator variables, hence:

$$P(\varphi_4) \quad = \quad 1 - \prod_{a \in A} (1 - P(R(a) \vee S(x_2, a), T(a)))$$

Next, we apply the algorithm recursively on $R(a) \vee S(x_2, a), T(a)$. In CNF it becomes[10] $(R(a) \vee S(x_2, a))(R(a) \vee T(a))$, and the algorithm returns $P(R(a) \vee S(x_2, a)) + P(R(a) \vee T(a)) - P(R(a) \vee S(x_2, a) \vee T(a))$. Consider the last of the three expressions (the other two are similar): its probability is

$$1 - (1 - P(R(a) \vee T(a))) \prod_{b \in A} (1 - P(S(b, a)))$$

Now we have finally reached the base case, where we compute the probabilities of sentences without variables: $P(R(a) \vee T(a)) = 1 - (1 - P(R(a)))(1 - P(T(a)))$, and similarly for the others.

Example 6.2 Consider the sentence φ in Example 5.8. Since this is already CNF (it is a disjunctive sentence), the algorithm proceeds directly to the second step. The separator is $V = \{z_1, z_2, z_3\}$ (see Ex. 5.8), and therefore:

$$P(\varphi) \quad = \quad 1 - \prod_{a \in A} (1 - P(\varphi[a/V])$$

where $\varphi[a/V]$ is:

$$R(a, x_1), S(a, x_1, y_1) \vee S(a, x_2, y_2), T(a, y_2) \vee R(a, x_3), T(a, y_3)$$

After reducing the sentence (i.e. removing the constant a), it becomes identical to Example 6.1.

[10] Strictly speaking, we would have had to rewrite the sentence into a reduce form first, by rewriting $S(x_2, a)$ into $S_{2<a}(x_2)$, etc.

In the rest of this section we show that the algorithm is complete, meaning that, if it fails on a positive sentence Φ, then Φ is $FP^{\#P}$-hard.

6.1 Safe Sentences

The sentences on which the algorithm terminates (and thus are in PTIME) admit characterization as a minor-closed family, for a partial order that we define below.

Let φ be a disjunctive sentence. A *level* is a non-empty set of variables[11] W such that every atom in φ contains at most one variable in W and for any unifiable variables x, x', if $x \in W$ then $x' \in W$. In particular, a separator is a level W that has one variable in common with each atom; in general, a level does not need to be a separator. For a variable $x \in W$, let n_x be the number of atoms that contain x; let $n = \max_x n_x$. Let $A = \{a_1, \ldots, a_k\}$ be a set of constants not occurring in φ s.t. $k \leq n$. Denote $\varphi[A/W]$ the sentence obtained as follows: substitute each variable $x \in W$ with some constant $a_i \in A$ and take the union of all such substitutions:

$$\varphi[A/W] \quad = \quad \bigvee_{\theta: W \to A} \varphi[\theta]$$

Note that $\varphi[A/W]$ is not necessarily a disjunctive sentence, since some components γ_i may become disconnected in $\varphi[A/W]$.

DEFINITION 6.3. *Define the following rewrite rule $\Phi \to \Phi_0$ on positive sentences. Below, $\varphi, \varphi_0, \varphi_1$, denote disjunctive sentences:*

$$\begin{array}{rcll} \varphi & \to & \varphi[A/W] & W \text{ is a level, } A \text{ is a set of constants} \\ \varphi_0 \vee \varphi_1 & \to & \varphi_1 & \text{if } Vars(\varphi_0) = \emptyset \\ \Phi & \to & \varphi & \exists u \in \mathbf{L}_C(\Phi).\mu(u, \hat{1}) \neq 0, \varphi = \lambda(u) \end{array}$$

The second and third rules are called simple *rules. The first rule is also simple if W is a separator and $|A| = 1$.*

The first rewrite rule allows us to substitute variables with constants; the second to get rid of disjuncts without any variables; the last rule allows us to replace a CNF sentence Φ with one element of its C-lattice, provided its Mobius value is non-zero. The transitive closure $\stackrel{*}{\to}$ defines a partial order on positive sentences.

DEFINITION 6.4. *A positive sentence Φ is called* unsafe *if there exists a sequence of simple rewritings $\Phi \stackrel{*}{\to} \varphi$ s.t. φ is a disjunctive sentence without separators. Otherwise it is called* safe.

Thus, the set of safe sentences can be defined as the downwards closed family (under the partial order defined by simple rewritings) that does not contain any disjunctive sentence without separators. The main result in this paper is:

THEOREM 6.5 (SOUNDNESS AND COMPLETENESS). *Fix a positive sentence Φ.*

Soundness *If Φ is safe then, for any probabilistic structure, Algorithm 6.1 terminates successfully (i.e. doesn't fail), computes correctly $P(\Phi)$, and runs in time $O(n^k)$, where n is the size of the active domain of the structure, and k the largest arity of any symbol in the vocabulary.*

Completeness *If Φ is unsafe then it is hard for $FP^{\#P}$.*

Soundness follows immediately, by induction: if the algorithms starts with Φ, then for any sentence Φ_0 processed recursively, it is the case that $\Phi \stackrel{*}{\to} \Phi_0$, where all rewrites are simple. Thus, if the algorithm ever gets stuck, Φ is unsafe; conversely, if Φ is safe, then the algorithm will succeed in evaluating it on any probabilistic structure. The complexity follows from the fact that each recursive step of the algorithm removes one variable from every atom, and traverses the domain D once, at a cost $O(n)$. Completeness is harder to prove, and we discuss it in Sec. 6.3.

For a simple illustration, consider the sentence:

$$\varphi \quad = \quad R(z_1, x_1), S(z_1, x_1, y_1) \vee S(z_2, x_2, y_2), T(z_2, y_2)$$

To show that it is hard, we substitute the separator variables z_1, z_2 with a constant a, and obtain

$$\varphi \quad \to \quad R(a, x_1), S(a, x_1, y_1) \vee S(a, x_2, y_2), T(a, y_2)$$

Since the latter is a disjunctive sentence without a separator, it follows that φ is hard.

6.2 Discussion

An Optimization The first step of the algorithm can be optimized, as follows. If the DNF sentence $\Phi = \bigwedge \gamma_i$ is such that the relational symbols appearing in γ_i are distinct for different i, then the first step of the algorithm can be optimized to compute $P(\Phi) = \prod_i P(\gamma_i)$ instead of using Mobius' inversion formula. To see an example, consider the sentence $\Phi = R(x), S(y)$, which can be computed as

$$P(\Phi) \quad = \quad (1 - \prod_a (1 - P(R(a))))(1 - \prod_a (1 - P(S(a))))$$

Without this optimization, the algorithm would apply Mobius' inversion formula first:

$$\begin{array}{rcl} P(\Phi) & = & P(R(x)) + P(S(y)) - P(R(x) \vee S(y)) \\ & = & 1 - \prod_a (1 - P(R(a))) + 1 - \prod_a (1 - S(a)) \\ & & - 1 + \prod_a (1 - P(R(a)) - P(S(a)) + P(R(a)) \cdot P(S(a))) \end{array}$$

The two expressions are equal, but the former is easier to compute.

A Justification We justify here two major choices we made in the algorithm: using the C-lattice instead of the D-lattice, and relying on the inversion formula with the Mobius function instead of some simpler method to eliminate unsafe subexpressions.

To see the need for the C-lattice, let's examine a possible dual algorithm, which applies the Mobius step to the D-lattice. Such an algorithm fails on Ex. 5.8, because here the D-lattice is $2^{[3]}$, and the Mobius function is $+1$ or -1 for every element of the lattice. The lattice contains $R(z_1, x_1), S(z_1, x_1, y_1), S(z_2, x_2, y_2), T(z_2, y_2)$, which is unsafe[12]. Thus, the dual algorithm fails.

To see the need of the Mobius inversion, we prove that an existential, positive FO sentence can be "as hard as any lattice".

THEOREM 6.6 (REPRESENTATION THEOREM). *Let $(\hat{L}, \leq)$ be any lattice. There exists a positive sentence Φ such that: $\mathbf{L}_D(\Phi) = (\hat{L}, \leq, \lambda)$, $\lambda(\hat{0})$ is unsafe, and for all $u \neq \hat{0}$, $\lambda(u)$ is safe. The dual statement holds for the C-lattice.*

PROOF. Call an element $r \in L$ *join irreducible* if whenever $v_1 \vee v_2 = r$, then either $v_1 = r$ or $v_2 = r$. (Every atom is join irreducible, but the converse is not true in general.) Let $R = \{r_0, r_1, \ldots, r_k\}$ be all join irreducible elements in L. For every

[11] No connection to the maximal sets W_i in Sec. 5.2.

[12] It rewrites to $R(a, x_1), S(a, x_1, y_1), S(a, x_2, y_2), T(a, y_2) \longrightarrow R(a, x_1), S(a, x_1, y_1) \vee S(a, x_2, y_2), T(a, y_2)$.

$u \in L$ denote $R_u = \{r \mid r \in R, r \leq u\}$, and note that $R_{u \wedge v} = R_u \cup R_v$. Define the following components[13]:

$$\begin{aligned}\gamma_0 &= R(x_1), S_1(x_1, y_1)\\ \gamma_i &= S_i(x_{i+1}, y_{i+1}), S_{i+1}(x_{i+1}, y_{i+1}) \;\; i = 1, k-1\\ \gamma_k &= S_k(x_k, y_k), T(y_k)\end{aligned}$$

Consider the sentences Φ and Ψ below:

$$\Phi = \bigvee_{u<\hat{1}} \bigwedge_{r_i \in R_u} \gamma_i \qquad \Psi = \bigwedge_{u<\hat{1}} \bigvee_{r_i \in R_u} \gamma_i$$

Then both $\hat{\mathbf{L}}_D(\Phi)$ and $\hat{\mathbf{L}}_C(\Psi)$ satisfy the theorem. $\square$

The theorem says that the lattice associated to a sentence can be as complex as any lattice. There is no substitute for checking if the Mobius function of a sub-query is zero: for any complex lattice $\hat{L}$ one can construct a sentence Φ that generates that lattice and where the only unsafe sentence is at $\hat{0}$: then Φ is safe iff $\mu_{\hat{L}}(\hat{0}, \hat{1}) = 0$.

6.3 Outline of the Completeness Proof

In this section we give an outline of the completeness proof and defer details to the full version [3]. We have seen that Φ is unsafe iff there exists a rewriting $\Phi \stackrel{*}{\rightarrow} \varphi$ where φ has no separators. Call a rewriting *maximal* if every instance of the third rule in Def. 6.3, $\Phi \rightarrow \lambda(u)$, is such that for all lattice elements $v > u$, $\lambda(v)$ is safe: that is u is a maximal unsafe element in the CNF lattice. Clearly, if Φ is unsafe then there exists a maximal rewriting $\Phi \stackrel{*}{\rightarrow} \varphi$ where φ has no separators. We prove the following:

LEMMA 6.7. *If $\Phi \stackrel{*}{\rightarrow} \varphi$ is a maximal rewriting, then there exists a PTIME algorithm for evaluating $P_{\mathbf{A}}(\varphi)$ on probabilistic structure $\mathbf{A}$, with a single access to an oracle for computing $P_{\mathbf{B}}(\Phi)$ on probabilistic structures $\mathbf{B}$.*

Thus, to prove that every unsafe sentence is hard, it suffices to prove that every sentence without separators is hard. To prove the latter, we will continue to apply the same rewrite rules to further simplify the sentence, until we reach an unsafe sentence where each atom has at most two variables: we call it a forbidden sentence. Then, we prove that all forbidden sentences are hard.

However, there is a problem with this plan. We may get stuck during rewriting before reaching a sentence with two variables per atom. This happens when a disjunctive sentence has no level, which prevents us from applying any rewrite rule. We illustrate here a simple sentence without a level:

$$\varphi = R(x, y), S(y, z) \vee R(x', y'), S(x', y')$$

Each consecutive pair of variables in the sequence x, x', y, y', z is unifiable. This indicates that no level exists, because it would have to include all variables, while by definition a level may have at most one variable from each atom; hence, this sentence does not have any level. While this sentence already has only two variables per atom, it illustrates where we may get stuck in trying to apply a rewriting.

To circumvent this, we transform the sentence (with two variables or more) as follows. Let $V = Vars(\varphi)$. A *leveling* is a function $l : V \rightarrow [L]$, where $L > 0$, s.t. for all $i \in [L]$, $l^{-1}(i)$ is a level. Conceptually, l partitions the variables into levels, and assigns an integer to each level. This, in turn, associates exactly one level to each relation attribute, since unifiable variables must be in the same level. We also call φ an *L-leveled sentence*, or simply leveled sentence. Clearly, a leveled sentence has a level: in fact it has L disjoint levels. We show that each sentence is equivalent to a leveled sentence, on some restricted structures.

Call a structure $\mathbf{A}$ *L-leveled* if there exists a function $l : A \rightarrow [L]$ s.t. if two constants $a \neq b$ appear in the same tuple then $l(a) \neq l(b)$, and if they appear in the same column of a relation then $l(a) = l(b)$. For example, consider a single binary relation $R(A, B)$. An instance of R represents a graph. There are no 1-leveled instances, because for every tuple (a, b) we must have $l(a) \neq l(b)$. A 2-leveled instance is a bipartite graph. There are no 3-leveled structures, except if one level is empty. For a second example, consider two relations $R(A, B)$, $S(A, B)$. (Recall that our structures are ranked, hence for every tuple $R(a, b)$ or $S(a, b)$ we have $a < b$). An example of a 3-leveled structure is a 3-partite graph where the R-edges go from partition 1 to partition 2 and the S-edges go from partition 2 to partition 3.

PROPOSITION 6.8. *Let φ be a disjunctive sentence that has no separators. Then there exists $L > 0$ and an L-leveled sentence φ^L s.t. that φ^L has no separator and the evaluation problem of φ^L over L-leveled structures can be reduced in PTIME to the evaluation problem of φ.*

The proof is in the full version [3]. We illustrate the main idea on the example above. We choose $L = 4$ and the leveled sentence φ becomes:

$$\begin{aligned}\varphi^L = {} & R_{23}(x_2, y_3), S_{34}(y_3, u_4) \vee R_{12}(x_1, y_2), S_{23}(y_2, z_3) \vee\\ & R_{23}(x'_2, y'_3), S_{23}(x'_2, y'_3)\end{aligned}$$

Here φ^L is leveled, and still does not have a separator. It is also easy to see that if a probabilistic structure $\mathbf{A}$ is 4-leveled, then φ and φ^L are equivalent over that structure. Thus, it suffices to prove hardness of φ^L on 4-leveled structures: this implies hardness of φ.

To summarize, our hardness proof is as follows. Start with a disjunctive sentence without separators, and apply the leveling construct once. Then continue to apply the rewritings 6.3: it is easy to see that, whenever $\varphi \rightarrow \varphi'$ and φ is leveled, then φ' is also leveled; in other words we only need to level once.

DEFINITION 6.9. *A* forbidden sentence *is a disjunctive sentence φ that has no separator, and is 2-leveled; in particular, every atom has at most two variables.*

We prove the following in the full version [3]:

THEOREM 6.10. *Suppose φ is leveled, and has no separator. Then there exists a rewriting $\varphi \stackrel{*}{\rightarrow} \varphi'$ s.t. φ' is a forbidden sentence.*

The level L may decrease after rewriting. In this theorem we must be allowed to use non-simple rewritings $\varphi \rightarrow \varphi[A/W]$, where W is not a separator (of course) and A has more than one constant. We show in [3] examples were the theorem fails if one restricts A to have size 1.

Finally, the completeness of the algorithm follows from the following theorem, which is technically the hardest result of this work. The proof is given in [3]:

THEOREM 6.11. *If φ is a forbidden sentence then it is hard for $FP^{\#P}$ over 2-leveled structures.*

7. LIFTED INFERENCE

Conditioning and disjointness are two important techniques in probabilistic inference. The first expresses the probability of some

[13] That is, $\bigvee \gamma_i = h_k$.

Boolean expression Φ as $P(\Phi) = P(\Phi \mid X)*P(X) + P(\Phi \mid \neg X)*(1 - P(X))$ where X is a Boolean variable. Disjointness allows us to write $P(\Phi \vee \Psi) = P(\Phi) + P(\Psi)$ when Φ and Ψ are exclusive probabilistic events. Recently, a complementary set of techniques called *lifted inference* has been shown to be very effective in probabilistic inference [12, 15, 14], by doing inference at the logic formula level instead of at the Boolean expression level. In the case of conditioning, *lifted conditioning* uses a sentence rather than a variable to condition.

We give an algorithm that uses lifted conditioning and disjointness in place of the Mobius step of Algorithm 6.1. When the algorithm succeeds, it runs in PTIME in the size of the probabilistic structure. However, we also show that the algorithm is incomplete; in fact, we claim that no algorithm based on these two techniques only can be complete.

Given $\Phi = \bigvee \varphi_i$, our goal is to compute $P(\Phi)$ in a sequence of conditioning/disjointness steps, without Mobius' inversion formula. The second step of Algorithm 6.1 (existential quantification based on independence) remains the same and is not repeated here. For reasons discussed earlier, that step requires that we have a CNF representation of the sentence, $\Psi = \bigwedge \varphi_i$, but both conditioning and disjointness operate on disjunctions, so we apply De Morgan's laws $P(\bigwedge \varphi_i) = 1 - P(\bigvee \neg\varphi_i)$. Thus, with some abuse of terminology we assume that our input is a D-lattice, although its elements are labeled with negations of disjunctive sentences.

We illustrate first with an example.

Example 7.1 Consider the sentence in Example 6.1:

$$\begin{aligned}\Phi &= R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(y_2) \vee R(x_3), T(y_3) \\ &= \varphi_1 \vee \varphi_2 \vee \varphi_3\end{aligned}$$

We illustrate here directly on the DNF lattice, without using the negation. (This works in our simple example, but in general one must start from the CNF, then negate.) The Hasse diagram of the DNF lattice is shown in Fig. 1. First, let's revisit Mobius' inversion formula:

$$P(\Phi) = P(\varphi_1) + P(\varphi_2) + P(\varphi_3) - P(\varphi_1\varphi_3) - P(\varphi_2\varphi_3)$$

The only unsafe sentence in the lattice is the bottom element of the lattice, where φ_1 and φ_2 occur together, but that disappears from the sum because $\mu(\hat{0}, \hat{1}) = 0$. We show how to compute Φ by conditioning on φ_3. We denote $\bar{\varphi} = \neg\varphi$ for a formula φ:

$$\begin{aligned}P(\Phi) &= P(\varphi_3) + P((\varphi_1 \vee \varphi_2) \wedge \bar{\varphi}_3) \\ &= P(\varphi_3) + P((\varphi_1, \bar{\varphi}_3) \vee (\varphi_2, \bar{\varphi}_3)) \\ &= P(\varphi_3) + P(\varphi_1, \bar{\varphi}_3) + P(\varphi_2, \bar{\varphi}_3) \\ &= P(\varphi_3) + (P(\varphi_1) - P(\varphi_1, \varphi_3)) + (P(\varphi_2) - P(\varphi_2, \varphi_3))\end{aligned}$$

The first line is conditioning (φ_3 is either true, or false), and the third line is based on mutual exclusion: φ_1 and φ_2 become mutually exclusive when φ_3 is false. We expand one more step, because our algorithm operates only on positive sentences: this is the fourth line. This last expansion may be replaced with a different usage, e.g. the construction of a BDD, not addressed in this paper. All sentences in the last line are safe, and the algorithm can proceed recursively.

For lifted conditioning to work, it is of key importance that we choose the correct subformula to condition. Consider what would happen if we conditioned on φ_1 instead:

$$\begin{aligned}P(\Phi) &= P(\varphi_1) + P((\varphi_2 \vee \varphi_3) \wedge \bar{\varphi}_1) \\ &= P(\varphi_1) + P(\varphi_2 \wedge \bar{\varphi}_1) + P(\varphi_3 \wedge \bar{\varphi}_1)\end{aligned}$$

Now we are stuck, because the expression $\varphi_2 \wedge \bar{\varphi}_1$ is hard.

Algorithm 7.1 computes the probability of a DNF formula using conditionals and disjointness. The algorithm operates on a DNF lattice (the negation of a CNF sentence). The algorithm starts by minimizing the expression $\bigvee_i \varphi_i$, which corresponds to removing all elements that are not co-atomic from the DNF lattice L (Prop 3.4). Recall that $\Phi = \bigvee_{u<\hat{1}} \lambda(u)$.

Next, the algorithm chooses a particular sub-lattice E, called the *cond-lattice*, and conditions on the disjunction of all sentences in the lattice. We define E below: first we show how to use it. Denote $u_1, \ldots, u_k$ the minimal elements of $L - E$. For any subset $S \subseteq L$, denote $\Phi_S = \bigvee_{u \in S, u<\hat{1}} \lambda(u)$; in particular, $\Phi_L = \Phi$.

The *conditioning* and the *disjointness* rules give us:

$$\begin{aligned}P(\Phi_L) &= P(\Phi_E) + P(\Phi_{L-E} \wedge (\neg\Phi_E)) \\ &= P(\Phi_E) + \sum_{i=1,k} (\Phi_{[u_i,\hat{1}]} \wedge (\neg\Phi_E))\end{aligned}$$

We have used here the fact that, for $i \neq j$, the sentences $\Phi_{[u_i,\hat{1}]}$ and $\Phi_{[u_j,\hat{1}]}$ are disjoint given $\neg\Phi_E$. Finally, we do this:

$$P(\Phi_{[u_i,\hat{1}]} \wedge (\neg\Phi_E)) = P(\Phi_{[u_i,\hat{1}]}) - P(\Phi_{[u_i,\hat{1}]\wedge E})$$

where $[u_i, \hat{1}] \wedge E = \{u \wedge v \mid u \geq u_i, v \in E - \{\hat{1}\}\}$

This completes the high level description of the algorithm. We show now how to choose the cond-lattice, then show that the algorithm is incomplete.

7.1 Computing the Cond-Lattice

Fix a lattice $(\hat{L}, \leq)$. The set of zero elements, Z, and the set of z-atoms ZA are defined as follows[14]:

$$\begin{aligned}Z &= \{z \mid \mu_L(z, \hat{1}) = 0\} \\ ZA &= \{a \mid a \text{ covers some element } z \in Z\}\end{aligned}$$

The algorithm reduces the problem of computing $P(\Phi_L)$ for the entire lattice L to computing $P(\Phi_K)$ for three kinds of sub-lattices K: E, $[u_i, \hat{1}] \wedge E$, and $[u_i, \hat{1}]$. The goal is to choose E to avoid computing unsafe sentences. We assume the worse: that every zero element $z \in Z$ is unsafe (if a non-zero element is unsafe then the sentence is hard). So our goal is: choose E s.t. for any sub-lattice K above, if z is a zero element and $z \in K$, then $\mu_K(z, \hat{1}) = 0$. That is, we can't necessarily remove the zeros in one conditioning step, but if we ensure that they continue to be zeroes, they will eventually be eliminated.

The *join closure* of $S \subseteq L$ is $cl(S) = \{\bigvee_{u \in s} u \mid s \subseteq S\}$. Note that $\hat{0} \in cl(S)$. The join closure is a join-semilattice and is made into a lattice by adding $\hat{1}$.

DEFINITION 7.2. *Let L be a lattice. The cond-lattice $E \subseteq L$ is $E = \{\hat{1}\} \cup cl(Z \cup ZA)$.*

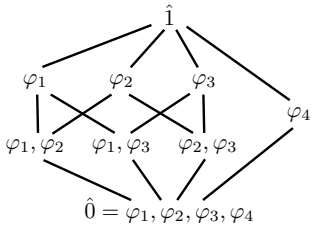
The following three propositions, proved in [3], show that E has our required properties.

PROPOSITION 7.3. *If $u \in L$ and $w \in [u, \hat{1}]$ then $\mu_{[u,\hat{1}]}(u, \hat{1}) = \mu_L(u, \hat{1})$.*

PROPOSITION 7.4. *For all $z \in Z$, $\mu_E(z, \hat{1}) = 0$.*

PROPOSITION 7.5. *Assume $\hat{0} \in Z$. Then, for every zero element $z \in L$ and for every $w \in L - E$ we have $\mu_{[\hat{0},w]}(z, w) = 0$.*

[14] "Covers" is defined in Sec. 3.



$$\gamma_1 = R(x_1), S_1(x_1, y_1) \qquad \varphi_1 = \gamma_3, \gamma_4$$
$$\gamma_2 = S_1(x_2, y_2), S_2(x_2, y_2) \qquad \varphi_2 = \gamma_2, \gamma_4$$
$$\gamma_3 = S_2(x_3, y_3), S_3(x_3, y_3) \qquad \varphi_3 = \gamma_1, \gamma_4$$
$$\gamma_4 = S_3(x_4, y_4), T(y_4) \qquad \varphi_4 = \gamma_1, \gamma_2, \gamma_3$$
$$\Phi = \varphi_1 \vee \varphi_2 \vee \varphi_3 \vee \varphi_4$$

Figure 2: A lattice that is atomic, coatomic, and $\mu(\hat{0}, \hat{1}) = 0$. Its sentence Φ is given by Th. 6.6 (compare to h_3 in Sec. 5.2).

Example 7.6 Consider Example 7.1. The cond-lattice for Fig. 1 (a) is

$$\begin{aligned} E &= cl(\{\hat{0}, (\varphi_1, \varphi_3), (\varphi_3, \varphi_2)\}) \\ &= \{\hat{0}, (\varphi_1, \varphi_3), (\varphi_3, \varphi_2), \varphi_3, \hat{1}\} \end{aligned}$$

Notice that this set is not co-atomic: in other words, when viewed as a sentence, it minimizes to φ_3, and thus we have gotten rid of $\hat{0}$.

To get a better intuition on how conditioning works from a lattice-theoretic perspective, consider the case when $Z = \{\hat{0}\}$. In this case ZA is the set of atoms, and E is simply the set of all atomic elements; usually this is a strict subset of L, and conditioning partitions the lattice into E, $[u_i, \hat{1}] \wedge E$, and $[u_1, \hat{1}]$. When processing E recursively, the algorithm retains only co-atomic elements. Thus, conditioning works by repeatedly removing elements that are not atomic, then elements that are not co-atomic, until $\hat{0}$ is removed, in which case we have removed the unsafe sentence and we can proceed arbitrarily.

7.2 Incompleteness

Assume $\hat{0}$ is an unsafe sentence, and all other sentences in the lattice are safe. Lifted conditioning proceeds by repeatedly replacing the lattice L with a smaller lattice E, obtained as follows: first retain only the atomic elements ($= cl(Z \cup ZA)$), then retain only the co-atomic elements (this is minimization of the resulting formula). Conditioning on any formula other than E is a bad idea, because then we get stuck having to evaluate the unsafe formula at $\hat{0}$. Thus, lifted conditioning repeatedly trims the lattice to the atomic-, then to the co-atomic-elements, until, hopefully, $\hat{0}$ is removed. Proposition 3.4 implies that, if $\hat{0}$ is eventually removed this way, then $\mu(\hat{0}, \hat{1}) = 0$. But does the converse hold ?

Fig.2 shows a lattice where this process fails. Here $\mu(\hat{0}, \hat{1}) = 0$; by Th. 6.6 there exists a sentence Φ that generates this lattice, where $\hat{0}$ is unsafe and all other elements are safe (Φ is shown in the Figure). Yet the lattice is both atomic and co-atomic. Hence cond-lattice is the entire lattice $E = L$. We cannot condition on any formula and still have $\mu(\hat{0}, \hat{1}) = 0$ in the new lattice. In other words, no matter what formula we condition on, we will eventually get stuck having to evaluate the sentence at $\hat{0}$. On the other hand, Mobius' inversion formula easily computes the probability of this sentence, by exploiting directly the fact that $\mu(\hat{0}, \hat{1}) = 0$.

8. CONCLUSIONS

We have proposed a simple, yet non-obvious algorithm for computing the probability of an existential, positive sentence over a probabilistic structure. For every *safe* sentence, the algorithm runs in PTIME in the size of the input structure; every *unsafe* sentence is hard. Our algorithm relies in a critical way on Mobius' inversion formula, which allows it to avoid attempting to compute the probability of sub-sentences that are hard. We have also discussed the limitations of an alternative approach to computing probabilities, based on conditioning and independence.

Algorithm 7.1 Compute $P\Phi$) using lifted conditional
Input: $\Phi = \bigvee_{i=1,m} \varphi_i$, $L = L_{DNF}(\Phi)$
Output $P(\Phi)$

Function Cond(L)
If L has a single co-atom **Then** proceed with `IndepStep`
Remove from L all elements that are not co-atomic (Prop 3.4)
Let $Z = \{u \mid u \in L, \mu_L(u, \hat{1}) = 0\}$
Let $ZA = \{u \mid u \in L, u \text{ covers some } z \in Z\}$
If $Z = \emptyset$ **Then** $E := [u, \hat{1}]$ for arbitrary u
  Else $E := cl(Z \cup ZA)$
If $E = L$ **then FAIL (unable to proceed)**
Let $u_1, \ldots, u_k$ be the minimal elements of $L - E$
Return $\mathbf{Cond}(E) + \sum_{i=1,k} \mathbf{Cond}(u_i) - \mathbf{Cond}([u_i, \hat{1}] \wedge E)$

Acknowledgments We thank Christoph Koch and Paul Beame for pointing us (independently) to incidence algebras, and the anonymous reviewers for their comments. This work was partially supported by NSF IIS-0713576.

9. REFERENCES

[1] N. Creignou and M. Hermann. Complexity of generalized satisfiability counting problems. *Inf. Comput*, 125(1):1–12, 1996.
[2] Nadia Creignou. A dichotomy theorem for maximum generalized satisfiability problems. *J. Comput. Syst. Sci.*, 51(3):511–522, 1995.
[3] N. Dalvi, K. Schnaitter, and D. Suciu. Computing query probability with incidence algebras. Technical Report UW-CSE-10-03-02, University of Washington, March 2010.
[4] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, Toronto, Canada, 2004.
[5] N. Dalvi and D. Suciu. The dichotomy of conjunctive queries on probabilistic structures. In *PODS*, pages 293–302, 2007.
[6] N. Dalvi and D. Suciu. Management of probabilistic data: Foundations and challenges. In *PODS*, pages 1–12, Beijing, China, 2007. (invited talk).
[7] Adnan Darwiche. A differential approach to inference in bayesian networks. *Journal of the ACM*, 50(3):280–305, 2003.
[8] Erich Grädel, Yuri Gurevich, and Colin Hirsch. The complexity of query reliability. In *PODS*, pages 227–234, 1998.
[9] Kevin H. Knuth. Lattice duality: The origin of probability and entropy. *Neurocomputing*, 67:245–274, 2005.
[10] Dan Olteanu and Jiewen Huang. Secondary-storage confidence computation for conjunctive queries with inequalities. In *SIGMOD*, pages 389–402, 2009.
[11] Dan Olteanu, Jiewen Huang, and Christoph Koch. Sprout: Lazy vs. eager query plans for tuple-independent probabilistic databases. In *ICDE*, pages 640–651, 2009.
[12] D. Poole. First-order probabilistic inference. In *IJCAI*, 2003.
[13] Yehoushua Sagiv and Mihalis Yannakakis. Equivalences among relational expressions with the union and difference operators. *Journal of the ACM*, 27:633–655, 1980.
[14] P. Sen, A.Deshpande, and L. Getoor. Bisimulation-based approximate lifted inference. In *UAI*, 2009.
[15] Parag Singla and Pedro Domingos. Lifted first-order belief propagation. In *AAAI*, pages 1094–1099, 2008.
[16] Richard P. Stanley. *Enumerative Combinatorics*. Cambridge University Press, 1997.
[17] Ingo Wegener. BDDs–design, analysis, complexity, and applications. *Discrete Applied Mathematics*, 138(1-2):229–251, 2004.