Probabilistic Databases for All

Dan Suciu
University of Washington

ABSTRACT

In probabilistic databases the data is uncertain and is modeled by a probability distribution. The central problem in probabilistic databases is query evaluation, which requires performing not only traditional data processing such as joins, projections, unions, but also probabilistic inference in order to compute the probability of each item in the answer. At their core, probabilistic databases are a proposal to integrate logic with probability theory. This paper accompanies a talk given as part of the *Gems of PODS* series, and describes several results in probabilistic databases, explaining their significance in the broader context of model counting, probabilistic inference, and Statistical Relational Models.

KEYWORDS

Query processing; probabilistic inference; model counting; knowledge compilation

ACM Reference Format:
Dan Suciu. 2020. Probabilistic Databases for All. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3375395.3389129

1 INTRODUCTION

In probabilistic databases the data items are probabilistic events: a tuple is present only with some probability, or the value of an attribute is a random variable. Conceptually, this defines a probability distribution over possible database instances, called *possible worlds*. Probabilistic databases are motivated by a variety of applications such as modeling uncertain data [72], missing values [73], data cleaning [70], database repair [67], deduplication [10], knowledge base construction [81], and approximate query processing [80].

The central problem is query evaluation: given a query Q and probabilistic database D, compute the answer of Q on D. The problem is denoted *PQE*, for *probabilistic query evaluation.* In addition to standard data processing, like joining tables, or removing duplicates, *PQE* also requires *probabilistic inference*, in order to compute the probability of each item in the answer. The latter has been the key focus of the research in probabilistic databases, leading to several interesting findings. This short survey paper accompanies a talk on Probabilistic Databases given as part of the *Gems of PODS* series, and is a high level overview of the most interesting results and findings in this space.[1] The significance of these results is best understood in the broader context of probabilistic graphical models, weighted model counting, and statistical relational learning.

Background Probabilistic Graphical Models (PGM) study multivariate probability distributions by describing their independence relationships using a graph. PGMs have a long history, predating Computer Science, and have been successful in many applications; see e.g. the historical notes in [52]. In a PGM, a distribution is represented as a product of factors, and probabilistic inference exploits the independence relations captured by the factorized expression, for example through the belief propagation algorithm. The complexity of the inference problem in PGMs is exponential in the tree-width [13, 20, 62].

Closer in spirit to probabilistic databases than inference in PGMs is the *model counting problem.* Here we are given a Boolean formula and want to compute the number of satisfying assignments. Equivalently, we are asking for the probability of the Boolean formula, when each Boolean variable is set to *true* randomly and independently, with probability 1/2. In the *weighted model counting* variant, each Boolean variable may have a different probability, not necessarily 1/2. Valiant proved that model counting is #P-complete [78]. *PQE* is precisely a weighted model counting problem, where the Boolean formula is the *lineage* of the query on the database. General inference techniques for PGMs are known to be too weak for weighted model counting, and instead, new approaches have been proposed in the literature. One approach consists of extensions of the Davis-Putnam-Logemann-Loveland (DPLL) family of algorithms [22, 23], originally designed for the SAT problem; a survey of DPLL-style model counting algorithms can be found in [35]. A second major approach, known as *knowledge compilation,* is to convert the input formula into a circuit, from which the model count can be computed efficiently [18, 19, 42, 59]. These two are tightly related, in the sense that the trace of a DPLL-style algorithm is a circuit [41, 42], hence both approaches have similar asymptotic complexities. As we shall see, probabilistic databases highlight a new limitations of these approaches, and offer a solution.

At a conceptual level, probabilistic databases represent an integration of logic and probability theory. The quest to integrate logic and probability theory has a long history. Nowhere is this more imperative than in AI, where the need to integrate formal reasoning on one hand, with statistical relational learning and statistical inference on the other hand, has been long recognized as a major goal and challenge. Today, it remains one of the top priorities in AI research, see for example [47, 65]. One line of research that aimed explicitly at addressing this challenge are *Statistical Relational Models*, SRM [26, 33, 66, 69]. Here a (large) statistical model, such as a PGM, is represented through a concise Knowledge Base, consisting of a small number of first order sentences. The actual model is obtained by *grounding* the formulas in the knowledge base with all constants in some given domain. The main vision of

Suciu was partially supported by NSF grants III-1703281, III-1614738, AitF 1535565.

[1] Biased, by necessity, by the author's own taste and preferences.

SRMs is that traditional tasks, such as parameter learning and probabilistic inference, should be performed on the concise, first order representation, and not on the much larger grounded model. This approach is called *lifted inference* in SRMs [49, 63], to be contrasted with *grounded inference* where the statistical model is computed first, then followed by the application of some standard inference method. While initial work on probabilistic databases evolved independently of that in SRMs, they share the same goals and informed each other.

Probabilistic Databases: A New Angle Probabilistic databases bring a new, sometimes surprising perspective, to this rich background. The simplest probabilistic data model is one where each tuple is an independent probabilistic event; this is called a tuple-independent database, TID. While several alternatives have also been studied, like block-disjoint-independent [16] or attributes as random variables [4], TIDs are the best understood to date. TIDs are already used in some interesting applications, like relational embeddings [29], and correlations can still be added to TIDs, namely by conditioning on a database constraint, as we review in Sec. 3.

A first major finding in probabilistic databases is that the *data complexity* [79] of the query evaluation problem over TIDs admits a dichotomy into polynomial time and #P-hard. In other words, for every fixed query, one can either compute its probability in polynomial time in the size of the input database, or this problem is #P-hard. No query has some intermediate complexity between polynomial time and #P-hard. This result is known to hold only for some classes of queries, which include Unions of Conjunctive Queries (UCQs) and some restricted classes of queries with negation; it is currently open whether this dichotomy holds for all first order queries. Viewed through the angle of the model counting problem, this result is a statement about the complexity of families of Boolean functions defined by a query. For each fixed query Q, consider the set of all lineages of Q, over all finite databases: then weighted model counting for this class of Boolean formulas is either in polynomial time, or is #P-hard. We review this result in Sec. 4.

A second surprising finding is the apparent need for the *inclusion/exclusion rule* in lifted inference. As we explained, the term *lifted inference* originates in statistical relational models, and refers to any algorithm that performs inference directly on structure of the first order sentence. Lifted inference always runs in polynomial time in the size of the database, thus we have a *complete* lifted inference if we can compute *all* queries that are in polynomial time. It turns out that, for completeness, we need to add the inclusion/exclusion rule to more basic lifted inference rules. Inclusion/exclusion is never needed in either PGM or weighted model counting, hence its key role in lifted inference comes as a surprise. We describe lifted inference and the role of the inclusion/exclusion rule in Sec. 5. An open question to date is whether the *disjointness* rule could replace inclusion/exclusion and still be able to compute all polynomial time queries [58].

In addition to probabilistic inference, *PQE* also needs to perform standard data processing, such as computing joins or unions or duplicate elimination. Data processing in modern SQL engines is done by first converting the query into a query plan, optimizing the plan, and finally executing it. Given any plan, it is possible to modify each of its operators to compute the probabilities of the output tuples, by performing simple operations (multiplication, addition, subtraction) over the input tuples' probabilities. The question is whether the final probability returned by the plan has any relationship with the correct probability required by *PQE*. Somewhat surprisingly, for a conjunctive query without self-joins, each such plan computes an upper bound of the correct probability, and can also be modified to compute a lower bound. This means that it is possible to compute, inside the SQL engine, upper and lower bounds on the query's probability, even when the corresponding *PQE* problem is #P-hard. We describe this in Sec. 6.

Probabilistic databases also gave an answer to the question whether lifted inference is more efficient than grounded inference: the answer is yes, at least when grounded inference is performed using a DPLL-style algorithm. More precisely, there exists an infinite set of UCQs such that (a) each such query can be computed using lifted inference (and, thus, its complexity is in polynomial time), and (b) every decision-DNNF for its lineage has size that is exponential in the size of the database. The decision-DNNF is the type of circuit that represents the trace of any DPLL-style algorithm, hence this implies that DPLL-style algorithms will run in exponential time. We describe this result in Sec. 7.

Finally, probabilistic databases shed some important light on the question whether symmetries in the data can help speed up probabilistic inference. Statistical Relational Models define a probability distribution that is invariant under any permutation of the domain, and, thus, are *partially exchangeable* according to an appropriately chosen set of statistics [49, 60]. A *symmetric probabilistic database* is any probabilistic database that is invariant under permutations of the domain; equivalently, for any relation name, all tuples of that relation have the same probability. The question is whether *PQE* becomes easier on symmetric databases. In fact, early work on lifted inference almost identified "lifted" with "exploiting symmetries". A major result[2] in this space is that, for every query in FO^2, the *PQE* problem over symmetric databases is in polynomial time [24]. Recall that FO^2 denotes first order logic with two variables [54]. However, it turns out that the good news stops at 2 variables: with three variables one can already construct a query that is hard even on symmetric databases. We describe symmetric databases in Sec. 8.

Terminology In one of the early works on probabilistic databases, Fuhr and Rölleke [30] extended the operators of the relational algebra with simple formulas to manipulate the tuple probabilities, and called this process *extensional semantics*; they also called *intensional semantics* the algebra modified to compute event expressions rather than probabilities. These two terms were used in many early papers on probabilistic database, including the survey [74]. They are completely equivalent to *lifted inference* and *grounded inference* respectively, which are standard terms in use today, and will also be used in this paper.

2 THE BASICS

Fix a relational vocabulary, in other words, fix a database schema. If `DOM` is a finite domain, then we denote by Tup(`DOM`), or simply Tup when the domain is clear from the context, the set of all possible tuples over the given schema whose constants are in `DOM`. A traditional database instance is any subset of Tup. In the context of

[2]This result came from the SRM community, but intellectually it is in perfect alignment with the goals of probabilistic databases.

probabilistic databases, we call such a subset $W \subseteq \text{Tup}$ a *possible world*. A *probabilistic database* is a probability distribution over all possible worlds with a given domain DOM. More precisely, a probabilistic database is $D = (\texttt{DOM}, p_D)$, where $p_D : 2^{\text{Tup}} \rightarrow [0, 1]$ and $\sum_{W \subseteq \text{Tup}} p_D(W) = 1$. Consider a Boolean Query Q, and recall that we write $W \models Q$ if Q is true in a world W. The *marginal probability* of Q is:

$$p_D(Q) = \sum_{W \subseteq \text{Tup}: W \models Q} p_D(W) \tag{1}$$

Given a tuple $t \in \text{Tup}$, its marginal probability is:

$$p_D(t) = \sum_{W \subseteq \text{Tup}: t \in W} p_D(W) \tag{2}$$

In general Tup is very large, for example it may have millions or billions of tuples, and the number of possible worlds is exponential in this number. This makes it impossible to represent explicitly the probability distribution p_D. The most common approach in probabilistic databases is to assume that the tuples t are independent probabilistic events; then, the database is called a *tuple independent datbase*, or TID. In a TID a possible world W is generated by including in W randomly and independently each tuple $t \in \text{Tup}$, and its probability is:

$$p_D(W) = \prod_{t \in W} p_D(t) \times \prod_{t \in \text{Tup}-W} (1 - p_D(t)) \tag{3}$$

In order to represent the TID, we only need to list the marginal probability of each tuple, $p_D(t)$. It is common to represent these probabilities in a standard relational database, where each relation R has one additional attribute P, such that, for every tuple $t \in R$, its probability is $p_D(t) \stackrel{\text{def}}{=} t.P$, and for every tuple $t \notin R$, $p_D(t) = 0$.

Example 2.1. Fig. 1 shows a simple TID with 9 tuples. Strictly speaking there are more than 2^9 possible worlds, because the set of possible tuples Tup includes tuples not shown in the database, for example $R(b_3)$ or $S(b_1, b_1)$, but any possible world that includes such tuples has probability 0, hence, w.l.o.g. we may consider only the 2^9 possible worlds obtained by taking subsets of the database in the Figure. Consider now the sentence:

$$Q = \forall x \forall y (S(x, y) \Rightarrow R(x)) \tag{4}$$

Once can think of Q as an inclusion constraint, stating that every value x that occurs in S also occurs in R. We want to compute the probability that Q holds, when the world W is chosen at random. Since the tuples are independent, we can derive a simple formula for this probability. Consider every value of x, for example $x = a_1$. A possible world that satisfies Q must either contain the tuple $R(a_1)$, an event with probability p_1, or *not* contain any of the tuples $S(a_1, b_1), S(a_1, b_2)$, an event with probability $(1 - q_1)(1 - q_2)$; the same applies to the other values $x = a_2$, $x = a_3$, leading to:

$$\begin{aligned} p_D(Q) =& (p_1 + (1 - p_1)(1 - q_1)(1 - q_2)) \\ & \times (p_2 + (1 - p_2)(1 - q_3)(1 - q_4)(1 - q_5)) \\ & \times (1 - q_6) \end{aligned}$$

The Probabilistic Query Evaluation Problem, *PQE* The key problem in probabilistic databases is query evaluation: given a query Q and a probabilistic database D, compute $p_D(Q)$. We denote this problem by *PQE*. The query is usually assumed to be in some logic, for example in some restriction of First Order Logic [15], or a logic program like ProbLog [51], or a datalog program [6], or a query in monadic second order logic [1], or some tree pattern [50]. The probabilistic database is most often assumed to be a TID, where all probabilities are given as rational numbers. Not surprisingly, query evaluation is hard:

Theorem 2.2. *Fix the query $H_0 = \forall x \forall y (R(x) \vee S(x, y) \vee T(y))$. Then, computing $p_D(H_0)$ is #P-hard in the size of the database D.*

The proof is by a reduction from the *Positive, Partitioned, 2CNF counting problem*, which was proven to be #P-complete by Provan and Ball [64].

The Dual Query Fix any FO sentence Q, and assume it contains only the connectives $\wedge, \vee, \neg, \exists, \forall$ (in other words it does not contain $\Rightarrow$). We define the *dual* of Q to be the sentence obtained by switching the quantifiers $\exists$ and $\forall$, and switching the connectives $\wedge$ and $\vee$. It is not hard to check that the query evaluation problems for a query and its dual are polynomial time equivalent. For example, the dual of the query $\forall x \forall y (R(x) \vee S(x, y) \vee T(y))$ is $\exists x \exists y (R(x) \wedge S(x, y) \wedge T(y))$ and therefore both have the same complexity, namely #P-hard by Theorem 2.2.

3 CORRELATIONS THROUGH CONSTRAINTS

The AI literature has described numerous applications where logic and probability theory are naturally combined [26, 33, 66, 69]. In all these applications it is important to represent correlations between atomic events, and this is usually done using Graphical Models, such as Bayesian Networks or Markov Networks [20, 52]. In contrast, most of the work on probabilistic databases has focused on tuple-independent databases (TIDs). Thus, we are led naturally to this question:

Question 3.1. How can we represent complex correlations between tuples in a probabilistic database?

It turns out that correlations can be naturally represented using *database constraints*. Constraints in probabilistic databases play the same role as factors in graphical models, and allow us to represent arbitrarily complex correlations by using TIDs and constraints. We will illustrate the basic ideas by showing how a Markov Logic Network (MLN) [26] can be represented in this way.

An MLN consists of a set of *soft constraints*. Each soft constraint is a pair (w, Δ), where $w \geq 0$ is a real number called the *weight* of the soft constraint, and Δ is a First Order formula. Intuitively, the soft constraint asserts that the formula Δ typically holds in the data, but is it not a hard requirement, and the weight w represents the degree to which Δ holds. For a simple illustration, consider the following soft constraint.[3]

$$3.9 \qquad \textit{Manager}(M, E) \Rightarrow \textit{HighlyCompensated}(M) \tag{5}$$

Here M, E are free variables, representing a manager and an employee respectively. The soft constraint says that, typically, managers are highly compensated. The weight $w = 3.9$ represents our confidence in this soft constraint: in general $w > 1$ means that we believe the constraint is more likely than not, and $w = \infty$ is a hard constraint. As a guiding intuition, a weight w can be converted into

[3]Our presentation follows [25] and is a slight departure from the original definition, for reasons discussed in [25].

R

x	P
a_1	p_1
a_2	p_2
a_3	p_3

S

x	y	P
a_1	b_1	q_1
a_1	b_2	q_2
a_2	b_3	q_3
a_2	b_4	q_4
a_2	b_5	q_5
a_4	b_6	q_6

(a)

p_D (

R

x
a_1
a_3

S

x	y
a_1	b_1
a_2	b_4
a_2	b_5
a_4	b_6

) $=p_1(1-p_2)p_3q_1(1-q_2)(1-q_3)q_4q_5q_6$

(b)

Figure 1: (a) A simple Tuple Independent Database (TID) D with 9 tuples. The values $p_1, p_2, p_3, q_1, \ldots, q_6$ are probabilities in $[0, 1]$. A possible world is obtained by randomly sampling each tuple with that probability. There are 2^9 possible worlds, and one is shown in (b).

a probability by the formula $p = w/(1 + w)$, thus $w = p/(1 - p)$ represents the "odds" of p, but this correspondence is only a guiding intuition and does not hold for MLN's; see the Appendix, and also the discussion in [25].

The semantics of an MLN is given by a traditional Markov Network [52]. The random variables are all possible tuples, Tup, and each soft constraint defines a set of factors, as follows. A *grounding* of the soft constraint (w, Δ) is a pair (w, F) where F is a sentence obtained by substituting the free variables in Δ with constants in the domain. Let *ground(MLN)* denote the set of all groundings of the MLN. Each grounding represents one factor of the Markov Network. If a possible world satisfies the sentence F, then it contributes a factor w; otherwise it contributes a factor 1. Formally, the *weight* of a possible world $W \subseteq$ Tup is defined as the product of the weights of all factors that hold in W:

$$weight(W) = \prod_{(w,F)\in ground(MLN): W \models F} w$$

Finally, the probability of a world is $p_{MLN}(W) \overset{\text{def}}{=} weight(W)/Z$, where Z is the normalizing factor, $Z \overset{\text{def}}{=} \sum_W weight(W)$.

In our example (5), the weight of a world W is $(3.9)^n$, where n is the number of pairs $(m, e) \in \mathtt{DOM}\times\mathtt{DOM}$ such $W \models \neg Manager(m, e) \vee HighlyCompensated(m)$. Its probability is this weight divided by the normalization factor Z. The MLN represents complex correlations between tuples, for example, if m is a manager of some employee e, then the probability that m is highly compensated increases and, in fact, the more employees m manages the higher the probability of her/him being highly compensated. In general, MLN's are as expressive as standard Markov Networks (see the Appendix), yet can be significantly more concise.

We explain now how an MLN can be represented using a TID and a constraint, by illustrating on the MLN in (5); for the general case we refer to [25, 37, 45]. Let R be a fresh relational symbol. We define the TID D, over the vocabulary R, *Manager*, *HighlyCompensated*, and with the following probabilities. For every two constants $m, e \in$ DOM:

$$p_D(Manager(m,e)) = 1/2 \qquad p_D(HighlyCompensated(m)) = 1/2$$
$$p_D(R(m,e)) = 1/(w-1)$$

In other words, all possible tuples in *Manager* and *HighlyCompensated* have probabilities $1/2$, and all possible tuples in R have probability $1/2.9 \approx 0.345$. We invite the reader to check the following statement:

Proposition 3.1. *[25, 37, 45] Consider the MLN in* (5)*, D be the probabilistic database above, and denote by Γ the following sentence:*

$$\Gamma = \forall m \forall e (R(m,e) \vee \neg Manager(m,e) \vee HighlyCompensated(m))$$

Then, for any Boolean query, Q, over the vocabulary consisting of Manager and HighlyCompensated, it holds: $p_{MLN}(Q) = p_D(Q|\Gamma)$. The latter is the conditional probability, $p_D(Q|\Gamma) \overset{\text{def}}{=} p_D(Q \wedge \Gamma)/p_D(\Gamma)$.

As a consequence, lifted inference evaluation techniques developed for probabilistic databases can be carried over to inference in MLN's [37]. When coupled with constraints, TIDs have the same representation power as MLN's, and, thus, are not confined to independence only, as suggested occasionally in the literature [69]; they simply replace traditional factors used in graphical models, with constraints.

4 DATA COMPLEXITY AND A DICHOTOMY THEOREM

One of the important findings of probabilistic databases is a dichotomy of the complexity of the *PQE* problem, which we discuss here. Probabilistic inference in graphical models is #P-hard in general.[4] Since probabilistic databases can represent graphical models, one doesn't expect the probabilistic query evaluation problem, *PQE*, to be any easier. However, the database perspective brings a new and powerful tool, through the notion of *data complexity*. Introduced by Vardi [79] for traditional databases, data complexity defines the evaluation problem by fixing the query Q and considering as input only the database D. In this light, each query Q defines a new problem, denoted $PQE(Q)$, raising the following question.

Question 4.1 (Data Complexity). Given a query Q, what is the complexity of the problem: *given D, compute $p_D(Q)$*? We denote this problem by $PQE(Q)$.

[4]This follows from the fact that model counting for 2CNF is #P-hard, and the fact that any 2CNF (and in Boolean formula in general) can be represented as a Bayesian Network.

If Q is a first order sentence, then $PQE(Q)$ is in[5] #P, and for some queries it may be lower. The question is whether we can establish the complexity of $PQE(Q)$ for any query Q. The answer, of course, depends on the logic $\mathcal{L}$ from which Q is drawn and, for some logics we do have a complete characterization of the complexity $PQE(Q)$ of all queries in that logic. We will describe here the complexity of Unions of Conjunctive Queries (UCQ), where the complexity of $PQE(Q)$ is either polynomial time or #P-hard, thus forms a dichotomy. We refer the reader to [16, 27, 36, 58] for dichotomies in other settings.

The dichotomy result for UCQs immediately generalizes to a richer logic, which we describe here. Recall that a FO sentence is in *prenex normal form* if it is written as a string of quantifiers, called the *prefix*, followed by quantifier-free formula, called the *matrix*. We call an FO sentence *unate* if it is in prenex normal form and for every relational symbol R_i in the vocabulary, either all its occurrences are in positive positions, or all its occurrences are in negated positions. For example, $\forall x(R(x) \Rightarrow S(x)) \wedge (R(x) \Rightarrow T(x))$ is a unate sentence, because both occurrences of R are in negated positions. In contrast, $\forall x(R(x) \Rightarrow S(x)) \wedge (S(x) \Rightarrow T(x))$ is not unate, because S occurs both in a positive and a negated position. In particular, every monotone FO sentence is unate. The term "unate" comes from the study of read-once Boolean formulas [34].

Theorem 4.1 (Dichotomy Theorem). *[17] Let $\mathcal{L}$ be the set of unate FO sentence whose quantifier prefix is $\forall^*$, or $\exists^*$. Then, for every $Q \in \mathcal{L}$, the probabilistic query evaluation problem, $PQE(Q)$, is either in polynomial time, or is #P-complete.*

For example, if Q is the query in Example 2.1, then $PQE(Q)$ is in polynomial time, while $PQE(H_0)$ is #P-hard where H_0 is the query defined in Theorem 2.2. Notice that any UCQ query is, in particular, a monotone FO sentence with quantifier prefix $\exists^*$, hence the theorem holds for all UCQ queries. In fact the result in [17] is stated only for UCQ queries, but it is not hard to see that this implies the more general result in Theorem 4.1. Indeed, any unate FO query can be transformed into a monotone query: replace all negated symbols $\neg R(x, y, \ldots)$ with fresh symbols $R'(x, y, \ldots)$, and define the probabilities of the new tuples $t' \in R'$ as $t'.P = 1 - t.P$, where t is the same tuple in R. The reader may check that the probability $p_D(Q)$ remains unchanged. Thus, every query satisfying the assumptions of Theorem 4.1 is equivalent to either a monotone FO sentence with quantifier prefix $\exists^*$, which is equivalent to a UCQ, or to a monotone FO sentence with quantifier prefix $\forall^*$, whose dual is equivalent to a UCQ; thus the dichotomy theorem for UCQ's in [17] implies Theorem 4.1.

Ladner [53] has proven the existence of decision problems that are in NP, but are neither NP-hard nor in polynomial time. This opens up the possibility that some query might exist such that $PQE(Q)$ is neither in polynomial time nor #P-hard: the theorem rules out this possibility for the restricted logic $\mathcal{L}$ of the theorem.

Naturally, at this point we would like to study decision problem for the complexity of $PQE(Q)$: *"given Q, find the complexity of the PQE(Q) problem"*. This problem depends on the choice of the language $\mathcal{L}$ from which Q is drawn, leading us to the next question:

[5]This claim requires some clarification, because #P is a class of counting problems [77], while $p_D(Q)$ is a rational number. Denoting by N the least common denominator of all probability values in the input database D, then $N \cdot p_D(Q)$ is a natural number, and computing it is in #P [16].

Question 4.2 (Deciding the Complexity)**.** Fix a query language $\mathcal{L}$. Find a decision procedure that, for each query $Q \in \mathcal{L}$, decides the complexity of $PQE(Q)$.

For the logic $\mathcal{L}$ defined in Theorem 4.1, we only know that it is decidable whether $PQE(Q)$ is in polynomial time or is #P-hard; the exact complexity is unknown. The same applies to UCQs: while we can decide whether a query is in polynomial time or #P-hard, we don't know the complexity of this decision problem. However, we have a surprisingly simple answer for a restricted language, namely that of Conjunctive Queries without self-joins. We briefly review this class of queries here. A *Conjunctive Query* is a formula of the form:

$$Q = \exists \boldsymbol{x}\, (R_1(\boldsymbol{x}_1) \wedge \cdots \wedge R_m(\boldsymbol{x}_m)) \tag{6}$$

where each expression $R_i(\boldsymbol{x}_i)$ is called a *atom*. As usual, the boldface notations $\boldsymbol{x}, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$ denote sets of variables. We say that Q is *without self-joins* if the symbols $R_1, R_2, \ldots$ are distinct. We are concerned here only with Boolean queries, hence we assume that all variables are existentially quantified. Fix a single variable x of the query, and denote by $at(x)$ the set of atoms that contain x: formally, $at(x) = \{R_i(\boldsymbol{x}_i) \mid x \in \boldsymbol{x}_i\}$.

Definition 4.2. A conjunctive query Q is called *hierarchical* if, for any two variables x, y, one of the following conditions hold: $at(x) \subseteq at(y)$ or $at(x) \supseteq at(y)$ or $at(x) \cap at(y) = \emptyset$.

The following provides a simple characterization of the complexity of conjunctive queries without self-joins.

Theorem 4.3. *[16] Let Q be a conjunctive query without self-joins. (1) If Q is hierarchical then PQE(Q) is in polynomial time. (2) If Q is not hierarchical then PQE(Q) is #P-hard. Moreover, the decision problem* "given Q, decide the complexity of $PQE(Q)$" *is in AC^0.*

We illustrate with two simple examples: $\exists x \exists y(R(x) \wedge S(x, y))$ is hierarchical, hence it is in polynomial time, while $\exists x \exists y(R(x) \wedge S(x, y) \wedge T(y))$ is non-hierarchical, because $at(x) = \{R, S\}$ and $at(y) = \{S, T\}$, and thus is #P-hard. For membership in AC^0 we refer to [16]. Recently, Amarilli and Kimelfeld [3] have strengthened case (2) of the theorem by proving that the query remains #P-hard even if all probabilities in the database are 1/2.

If Q has self-joins, then the criterion in Theorem 4.3 no longer holds. A simple counterexample is $\exists x \exists y \exists z(R(x, y) \wedge R(y, z))$, which is hierarchical, yet is #P-hard [17].

Dichotomy results for other logics Fink and Olteanu [27] considered a language that includes conjunctive queries without self-joins, and also has set difference (a form of negation) and showed, somewhat surprisingly, that being hierarchical is again a necessary and sufficient condition for its complexity to be in polynomial time. Another restricted subclass of queries with negation is considered by Gribkoff et al. [36]. In contrast, no non-trivial dichotomy results are known for logics that allow sentences with both $\forall$ and $\exists$.

Deciding the Complexity FO For every query in First Order Logic, $PQE(Q)$ is in #P [16], however, we do not know if FO admits a dichotomy into polynomial time and #P-hard. However, we can prove that it is not possible to separate these two classes:

Theorem 4.4. *Assuming FP ≠ #P, the following problem is undecidable:* "given Q in FO, decide whether $PQE(Q)$ is #P-hard".

The theorem implies that, if FO had a dichotomy into polynomial time and #P-hard, then we will not be able to decided between these two classes, unless FP=#P. The proof follows by reduction from the satisfiability problem for finite models, *"given a sentence* Γ *in FO, check whether it admits a finite model"*, which was proven by Trakhtenbrot to be undecidable [76]. The reduction is the following. Given a sentence Γ, let R, S, T be three relation symbols that do not occur in Γ, and let $H_0 = \forall x \forall y (R(x) \vee S(x, y) \vee T(y))$. Then Γ is finitely satisfiable iff the query $Q \overset{\text{def}}{=} \Gamma \wedge H_0$ is #P-hard.

5 THE INCLUSION/EXCLUSION FORMULA

An interesting aspect of the probabilistic query evaluation problem (*PQE*) for probabilistic databases is the central role played by the inclusion/exclusion formula. We explain it here.

Our question here is an algorithmic question: how do we compute $p_D(Q)$? Our focus here will be only on queries where *PQE*(Q) is in polynomial time, and we naturally expect our algorithm to also run in polynomial time. In this setting the query Q is fixed, and the complexity is measured in the size of the input database. One possibility is to *ground* the query on the database, to obtain a large Boolean formula called *lineage*, then perform probabilistic inference on the lineage. However, this approach may run in exponential time, even if the query Q is in polynomial time, as we explain later. The alternative is to compute $p_D(Q)$ by inspecting only the First Order syntax of the query expression, an approach that is called *lifted inference* [25]. Much of the research in probabilistic databases, and also in Statistical Relational Learning (SRL) [33, 66] has focused on lifted inference.

In general, probabilistic inference is based on simple inference primitives, like conditional independence, $p(X, Y|Z) = p(X|Z)p(Y|Z)$ implicitly used by the belief propagation algorithm, or the Shannon expansion, $p(F) = p(F[X = 0])p(\neg X) + p(F[X = 1])p(X)$ used in weighted model counting. Similarly, lifted inference is also based on simple rules, which can be applied recursively on the structure of the first order sentence Q:

$$p_D(Q_1 \wedge Q_2) = p_D(Q_1) \cdot p_D(Q_2) \qquad \text{if "}Q_1, Q_2 \text{ are inependent"} \quad (7)$$

$$p_D(\forall x Q) = \prod_{a \in \mathtt{DOM}} p_D(Q[a/x]) \qquad \text{if "}x \text{ is separator variable"} \quad (8)$$

$$p_D(Q \wedge \neg Q') = p_D(Q) - p_D(Q \wedge Q') \quad (9)$$

The side condition "Q_1, Q_2 are independent" checks if Q_1 and Q_2 have disjoint sets of relational symbols, which ensures that they are independent events, since D is a TID. Similarly, the condition "x is a separator variable" checks whether x occurs in all atoms, and, moreover, for every relational symbol R_i, x occurs on the same position in all atoms using the symbol R_i; this ensures that the events $Q[a_1/x], Q[a_2/x], \ldots$ are independent. Let's call the three rules above, plus their dual rules [6] the *basic rules*. *Lifted inference* is the algorithm that, given Q and D computes $p_D(Q)$ by repeated applications of these rules. It always runs in polynomial time in the size of the database D, however it may fail on some queries, namely when the syntactic conditions required by the rules do not apply. A simple example of lifted inference is $p_D(\forall x \forall y (R(x) \wedge S(y))) = p_D(\forall x R(x)) p_D(\forall y S(y)) = \prod_{a \in \mathtt{DOM}} p_D(R(a)) \cdot \prod_{b \in \mathtt{DOM}} p_D(S(b))$; this takes linear time in the size of relations R, S. A simple example where they fail is H_0 in Theorem 2.2.

[6]The dual rules are: $p_D(Q_1 \vee Q_2) = 1 - (1 - p_D(Q_1))(1 - p_D(Q_2))$, $p_D(\exists x Q) = 1 - \prod_{a \in \mathtt{DOM}} (1 - p_D(Q[a/x]))$, and $p_D(Q \vee \neg Q') = p_D(Q) + (1 - p_D(Q \vee Q'))$.

If lifted inference succeeds on a query Q, then *PQE*(Q) is in polynomial time. What about the converse? Are the basic rules sufficient to compute *any* query whose complexity is in polynomial time? Of course, the answer depends on the language $\mathcal{L}$ from where the query Q is drawn. For example, the basic rules turn out to be complete for the set of Conjunctive Queries without self-joins, yet are incomplete for Conjunctive Queries: we invite the reader to check that the basic rules fail to apply to the query[7] $Q_J \overset{\text{def}}{=} \exists x \exists y \exists u \exists v (R(x) \wedge S(x, y) \wedge T(u) \wedge S(u, v))$, yet, we will prove below that this query is in polynomial time. This leads to a natural question:

Question 5.1 (Probabilistic Inference Rules)**.** Consider a logic $\mathcal{L}$ that admits a dichotomy into polynomial time and #P-hard. Find a set of probabilistic inference rules that is *complete* for $\mathcal{L}$, meaning that, for any $Q \in \mathcal{L}$, if *PQE*(Q) is in polynomial time, then we should be able to compute $p_D(Q)$ using the lifted inference rules.

It turns out that we need to add the *inclusion/exclusion formula* to the basic rules for completeness. In its simplest form, the inclusion/exclusion formula is:

$$p_D(Q_1 \vee Q_2) = p_D(Q_1) + p_D(Q_2) - p_D(Q_1 \wedge Q_2) \quad (10)$$

As before, we need to add the dual formula, which, in this case, expresses $\wedge$ in terms of $\vee$. For a simple illustration, we show how to use this rule to compute the dual of Q_J:

$$\begin{aligned} &p_D\left(\forall x \forall y (R(x) \vee S(x, y)) \vee \forall u \forall v (T(u) \vee S(u, v))\right) = \\ &p_D\left(\forall x \forall y (R(x) \vee S(x, y))\right) + p_D\left(\forall u \forall v (T(u) \vee S(u, v))\right) \\ &- p_D\left(\forall x \forall y (R(x) \vee S(x, y)) \wedge \forall u \forall v (T(u) \vee S(u, v))\right) \end{aligned}$$

The first two expressions can be computed in similar way to Example 2.1, while for the third expression we notice that the query is equivalent to $\forall x \forall y ((R(x) \vee S(x, y)) \wedge (T(x) \vee S(x, y)))$ and x is now a separator variable, allowing us to apply Rule (8).

The basic rules plus the dual of (10) were proven in [17] to be complete for the class of Unions of Conjunctive Queries. In our setting that result becomes:

Theorem 5.1 (Complete Set of Inference Rules). *Let* $\mathcal{L}$ *be the set of unate FO sentences whose quantifier prefix is either* $\forall^*$ *or* $\exists^*$ *(same as in Theorem 4.1). Then, the basic rules plus the inclusion/exclusion rule are complete for* $\mathcal{L}$*. In other words, if* $Q \in \mathcal{L}$ *and* PQE(Q) *is in polynomial time, then we can compute* $p_D(Q)$ *by using lifted inference.*

To keep the presentation at a high level, we have omitted several technical details, like the need to perform shattering and ranking on a query before applying the rules, and the important role of cancellations; we refer the reader to [74] for a detailed exposition.

Discussion We end this section with a discussion about the surprising need to use the inclusion/exclusion formula. This formula is never used in other settings of probabilistic inference, but instead it is replaced by the *disjointness rule*: $p(U \vee V) = p(U) + p(V)$, if the

[7]The subscript J stands for "join". Q_J is the first in a progression of queries that illustrate the applicability of various lifted inference rules, see [74].

events U, V are *disjoint*, meaning $U \wedge V \equiv$ *false*. Indeed, by using the disjointness rule one can derive $p(A \vee B) = p(A \vee (\neg A \wedge B)) = p(A) + p(\neg A \wedge B) = p(A) + p(B) - p(A \wedge B)$, making inclusion/exclusion unnecessary. For that reason inclusion/exclusion is not used in either graphical models or weighted model counting. This leads to the question whether we can replace inclusion/exclusion with the disjointness rule and still have a complete set of lifted inference rules. The answer is currently unknown. The difficulty lies in the fact that the inclusion/exclusion rule exposes the possibility to cancel terms, and cancellation is a critical step in lifted inference. For a high level illustration, consider a query of the form $AB \vee BC \vee CD$, where $A, B, \ldots$ are sentences, and AB abbreviates $A \wedge B$. The inclusion/exclusion formula expands into 7 terms, but two of them are equal to $p_D(ABCD)$ and cancel out, and we obtain: $p_D(AB \vee BC \vee CD) = p_D(AB) + p_D(BC) + p_D(CD) - p_D(ABC) - p_D(BCD)$. If the query $ABCD$ is #P-hard and all others are in polynomial time, then the cancellation is absolutely necessary in order to avoid trying to compute $p_D(ABCD)$. It remains open whether any application of the inclusion/exclusion formula followed by cancellations can be expressed as a sequence of applications of the disjointness rule.[8] An important progress was made recently by Monet [58], who has answered this question in the affirmative for a significant special case; the general case still remains open.

6 QUERY PLANS

An important aspect of probabilistic databases is the need to perform both probabilistic inference *and* traditional query processing. Modern database engines perform query processing by first converting the query into a query plan, optimizing it, then executing that plan. Probabilistic inference can be performed on top of that plan, by modifying each operator to also compute the probabilities of their output tuples. Every lifted inference rule corresponds to some query operator that performs simple operations on the probabilities; we refer the reader to [31, 32] for details. Therefore, if the query is "liftable", in particular *PQE*(Q) is in polynomial time, then, with the right plan, the query's probability can be computed during standard query processing of the plan.

What if the query is not liftable, e.g. because *PQE*(Q) is #P-hard? We can still use any query plan for Q and modify its operators to compute some probabilities, but does the resulting probability have any meaning at all? Surprisingly, for a conjunctive query without self-joins, this probability is guaranteed to be an upper bound of $p_D(Q)$. This means that we can always compute an upper bound on $p_D(Q)$ during standard query processing, thus benefiting from the performance of modern database engines. A lower bound can also be computed in similar ways. We will describe here the main intuition and refer the reader to [31, 32] for details.

We consider only Conjunctive Queries without self-joins and assume that the probabilistic database is represented in a standard relational database, where each relation has an additional probability attribute P. Thus, a relation $R(x, y)$ becomes $R(x, y, P)$ where P stores the probability of the tuple. We need two operators. (1) Natural join $\bowtie$, modified to multiply the probabilities of two arguments, and (2) Group-by/aggregate γ, where the aggregate operator is $u \oplus v \overset{\text{def}}{=} 1 - (1-u)(1-v)$. We illustrate the result of both operators on the database shown in Figure 1(a):

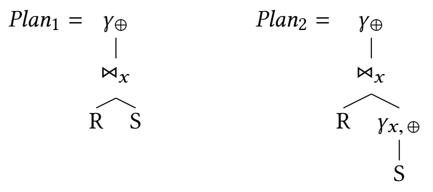
$R \bowtie_x S =$

x	y	P
a_1	b_1	p_1q_1
a_1	b_2	p_1q_2
a_2	b_3	p_2q_3
a_2	b_4	p_2q_4
a_2	b_5	p_2q_5

$\gamma_{x,\oplus}(S) =$

x	P
a_1	$1-(1-q_1)(1-q_2)$
a_2	$1-(1-q_3)(1-q_4)(1-q_5)$
a_4	q_6

A conjunctive query typically admits several such plans. However, not all plans lead to correct probability computations. For example, consider the query $\exists x \exists y (R(x) \wedge S(x, y))$ and the two plans below:



If we ignored the probability field, then these two plans are equivalent, as they both simply check if the join $R \bowtie S$ is non-empty, but they return different probabilities, and only the second plan returns the correct [9] probability $p_D(Q)$.

A query plan that returns the correct probability of $p_D(Q)$ is called a *safe query plan*. In general, a query may admit zero or more safe plans, and several unsafe plans, and simple criteria exists for checking if a plan is safe [32].

It turns out that the result of any plan, safe or unsafe, is an upper bound on $p_D(Q)$ [32]:

Theorem 6.1. *Let Q be a Boolean conjunctive query without self-joins, and Plan be a query plan for Q. Let $Plan_D$ denote the result of the plan when executed on a database D. (When Q is a Boolean query then this is a single number representing the probability field P.) Then, for every TID D, $Plan_D$ is an upper bound of $p_D(Q)$ [32]. Moreover, there exists a simple modification of the probabilities in the database D, such that, denoting D_1 the resulting database, the plan executed on D_1 is a lower bound of $p_D(Q)$ [31]. In summary:*

$$Plan_{D_1} \leq p_D(Q) \leq Plan_D$$

This leads to the following strategy for computing an upper bound of $p_D(Q)$: generate *all* plans, compute their probabilities, return the minimum value. This is a guaranteed upper bound of $p_D(Q)$. Naively computing all query plans leads to significant performance degradation (two orders of magnitude) but several optimizations, such as pruning some plans that are dominated by others, and reusing common subexpressions among plans, brings the performance close to that of standard query processing [32]. As stated in the theorem, one can also compute a lower bound of

[8] When using the disjointness rule we must also ensure that the canceled terms do not show up again when we eliminate negation. Continuing our example, a wrong way to apply disjointness is $AB \vee BC \vee CD = AB \vee \bar{A}BC \vee A\bar{B}CD \vee \bar{A}\bar{B}CD$ because to in order to compute $p_D(A\bar{B}CD)$ we need again the term $ABCD$. To avoid this term, we can write $AB \vee BC \vee CD = AB\bar{C} \vee BC \vee \bar{B}CD$, and now we can compute $p_D(AB\bar{C}) = p_D(AB) - p_D(ABC)$ etc.

[9] $Plan_1 = 1 - (1 - p_1q_1)(1 - p_1q_2)(1 - p_2q_3)(1 - p_2q_4)(1 - p_2q_5)$.
$Plan_2 = 1 - (1 - p_1(1 - (1 - q_1)(1 - q_2)))(1 - p_2(1 - (1 - q_3)(1 - q_4)(1 - q_5)))$.

$p_D(Q)$, however this requires that we modify the probability $t.P$ of each tuple t to $1 - (1 - t.P)^{1/k}$, where k is the number of times t occurs in the lineage DNF of Q on the database [31]. Computing the counts k could be done using a *group-by-count(*)* query in SQL (at additional performance cost).

We note that all results discussed in this section are limited to conjunctive queries without self-joins. It is open how to extend these results beyond this class of queries.

7 QUERY COMPILATION

One important finding in probabilistic databases is that lifted inference is provably more efficient than grounded inference. We have seen that lifted inference refers to solving *PQE*(Q) by reasoning only on the first order syntax of the query. In contrast, in *grounded inference* we first compute its lineage, then apply some weighted model counting algorithm to the lineage. It turns out that there exists queries for which lifted inference is in polynomial time, and any grounded inference method takes exponential time. We describe this result here, after a brief review of the necessary background.

In *model counting* we are given a Boolean formula F over variables $X_1, \ldots, X_n$, and ask for the number of truth assignments of F, denoted by $\#F$. This is one of classic #P-hard problems introduced by Valiant [78], who proved that the problem remains #P-hard even if the formula is restricted to positive 2CNF (or positive 2DNF, by duality). In the *weighted model counting* version, each variable X_i is associated with a weight w_i. This is equivalent to the following formulation (see the Appendix): given a probability $p_i \in [0, 1]$ for each Boolean variable X_i, $i = 1, n$, compute the probability that F is true, $p(F)$, when each variable X_i is set to true independently, with probability p_i. This problem has been studied extensively in the literature, see [35] for a survey.

The *lineage* of a query Q over a domain $\texttt{DOM}$ is defined as follows. Associate to each tuple $t_i \in \text{Tup}(\texttt{DOM})$ a Boolean variable X_i. Then, each truth assignment $\theta : \{X_1, \ldots, X_n\} \to \{0, 1\}$ corresponds to a possible world $W \subseteq \text{Tup}(\texttt{DOM})$, consisting of those tuples t_i for which the corresponding variable X_i is set to true. The lineage of Q is the Boolean function $F_{Q,\texttt{DOM}}$ defined as follows: $F_{Q,\texttt{DOM}}$ is true on an assignment θ iff the possible world W corresponding to θ satisfies the query: $F_{Q,\texttt{DOM}}[\theta] = 1$ iff $W \models Q$. If Q is a first order query, then the lineage can be computed inductively on its structure and its size is polynomial in the size of the domain; we review this in the appendix.

Grounded inference first computes the lineage $F_{Q,\texttt{DOM}}$, then uses some weighted model counting algorithm, call it $\mathcal{A}$, to compute the probability of the lineage. On one hand, this works for any query and any database, while lifted inference works only on queries where *PQE*(Q) is in polynomial time. On the other hand, lifted inference always runs in polynomial time, hence, naturally, we would like $\mathcal{A}$ to also run in polynomial time on the lineage $F_{Q,\texttt{DOM}}$ whenever Q is liftable. This leads to a natural question.

Question 7.1. Fix a weighted model counting algorithm $\mathcal{A}$. Does $\mathcal{A}$ run in polynomial time for every liftable query Q?

If *PQE*(Q) is #P-hard, then, of course, we don't expect $\mathcal{A}$ to run in polynomial time. The question only asks whether $\mathcal{A}$ runs in polynomial time for liftable queries, hence when *PQE*(Q) is in polynomial time. The answer, of course, depends on the algorithm $\mathcal{A}$, and on the language from which Q is drawn.

Modern exact model counting algorithms such as Cachet [71] and sharpSAT [75] are based on full backtracking search using the *DPLL* family of algorithms ([22, 23]), extended with caching [5, 57] and components (Relsat [7]). A survey can be found in [35]. All model counting algorithms and knowledge representations (discussed below), are based on three simple probabilistic inference primitives:

$$\begin{aligned} p(F) = &\quad p(F[X = 0])(1 - p(X)) + p(F[X = 1])p(X) && (11) \\ p(F_1 \wedge F_2) = &\quad p(F_1)p(F_2) \quad \text{if "}F_1, F_2 \text{ are independent"} && (12) \\ p(F_1 \vee F_2) = &\quad p(F_1) + p(F_2) \quad \text{if "}F_1, F_2 \text{ are disjoint events"} && (13) \end{aligned}$$

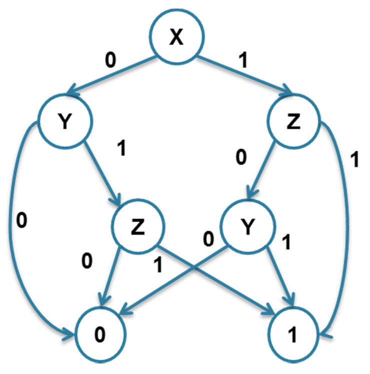
A *DPLL-style algorithm* for computing $p(F)$ maintains a cache of previously computed probabilities, and computes the probability $p(F)$ of a Boolean expression F by applying one of the rules (11) or (12). Rule (11) is called a *Shannon expansion*. The choice of the Boolean variable X does not affect correctness, but affects performance dramatically. Some DPLL-style algorithm also apply Rule (12), which is called *components*. For that they need to write the formula as $F = F_1 \wedge F_2$ such that F_1, F_2 do not share any common Boolean variables. If F is a CNF expression, then this can be done by computing the connected components of the primal graph, hence the name of the rule.

Rule (13) is applied only when F_1, F_2 are *disjoint events*, meaning $F_1 \wedge F_2 \equiv$ *false*. Testing disjointness is co-NP hard, and therefore DPLL-style algorithm do not use this rule. We mention it here only briefly, because it appears in d-DNNFs described below.

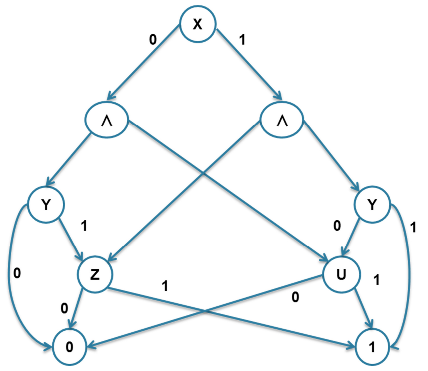
An alternative approach to DPLL-style algorithms is *knowledge compilation*, which converts the input Boolean formula into a representation (usually a circuit) from which the model count can be computed efficiently in the size of the representation [18, 19, 42, 59]. We describe here three such representations. A *Free Binary Decision Diagram*, FBDD, is a rooted DAG with two leaf nodes labeled 0 and 1 respectively, such that, every internal node is labeled with a Boolean variables X_i, has two outgoing edges, labeled 0 and 1, and every path from the root to a leaf node contains every variable X_i at most once. An *Ordered Binary Decision Diagram*, OBDD, is an FBDD such that every path visits the variables in the same order. A *decision-DNNF* is an FBDD extended with independent-$\wedge$ nodes, i.e. restricted to have subtrees with disjoint sets of variables. Figure 2 illustrates a simple FBDD and decision-DNNF; see also the discussion in [9]. Finally, a *d-DNNF* (disjoint-Deterministic-Negation-Normal-Form[10]) is a circuit whose leaf nodes are labeled with variables and whose internal nodes are labeled with $\vee$ nodes whose children are disjoint events, $\wedge$ nodes whose children are independent events, and $\neg$ nodes are applied only directly to variables.

Huang and Darwiche noted the following strong connection between knowledge compilation and DPLL-style algorithms: the trace of any DPLL-based algorithm is a type of knowledge representation [19, 21]. More precisely: (a) The trace of a DPLL-style algorithm

[10]The terminology used in d-DNNF is this: $\wedge$-nodes are called "disjoint" and $\vee$-nodes are called "deterministic". We prefer to use the terms "independent" and "disjoint" instead, which are common in probability theory. Also, in d-DNNF the $\neg$ operator is only allowed to occur above a leaf nodes, raising the question whether F and $\neg F$ admit d-DNNF's whose sizes are polynomially related (this question is open). This restriction on $\neg$ is removed when studying the circuit complexity; Monet [58] coined the term d-D for a d-DNNF with this restriction removed.



(a)



(b)

Figure 2: Illustration of an FBDD and a decision-DNNF from [9]. (a) An FBDD representing the Boolean formula $(\neg X)YZ \lor XY \lor XZ$. (b) A decision-DNNF representing the Boolean formula $(\neg X)YZU \lor XYZ \lor XZU$.

with caching and with a fixed variable order is an OBDD. (b) The trace of a DPLL-style algorithm with caching (without restriction on the variable order) is an FBDD. (c) The trace of a DPLL-style algorithm with caching and components is a decision-DNNF.

Results on Query Compilation With this background in mind, Question 7.1 can be restated by asking for the performance of DPLL-style algorithms, with or without components, on a query's lineage.

THEOREM 7.1. *Let* DOM *be a domain of size n.*

(i) Let Q be a conjunctive query without self-joins. (a) If Q is hierarchical, then the lineage $F_{Q,\texttt{DOM}}$ admits an OBDD whose size is linear in n [46, 61]. (b) If Q is non-hierarchical, then every OBDD has size $\geq (2^n - 1)/n$ [9].

(ii) There exists an infinite set of Unions of Conjunctive Queries Q such that PQE(Q) is in polynomial time, but every decision-DNNF of the lineage $F_{Q,\texttt{DOM}}$ has size $2^{\Omega(\sqrt{n})}$ [9].

The first part of the theorem strengthens the dichotomy for conjunctive queries without self-joins. In one class queries are hierarchical, their OBDD have linear size, and their complexity is in polynomial time. In the other class queries are non-hierarchical, their OBDD is exponentially large, and their complexity is #P-hard. This result extends to the following dichotomy for Unions of Conjunctive Queries [9, 46]: in one class all UCQs queries are *inversion-free* (a syntactic notion) and admit OBDDs of linear size, and in the other class all queries have inversions, and their OBDD's have size $\geq (2^n - 1)/n$. However, this dichotomy is no longer the same as the dichotomy into polynomial-time and #P-hard: there exists UCQ queries for which *PQE*(*Q*) is in polynomial time, yet their smallest OBDD is exponentially large.

The second part of the theorem implies that lifted inference is strictly more efficient than any grounded inference using a DPLL-style algorithm. This is independent of what heuristics is used to choose the variable order, or the caching policy, or whether it implements components or not. Indeed, if we ran such an algorithm on a database instance with a domain of size *n*, then its runtime is given by the size of the trace which, by the theorem, is $2^{\Omega(\sqrt{n})}$. In contrast, lifted inference computes these queries in polynomial time.

8 SYMMETRIC DATABASES

The vision of *lifted inference* in Statistical Relational Models [66] is to exploit symmetries in the graphical model obtained after grounding a knowledge base. In fact, the term "lifted inference" is sometimes used to mean only exploiting symmetries, and not to refer to the inference rules discussed in Sec. 5. A *symmetric probabilistic database* is a database where, for every relation symbol *R* in the vocabulary, all *R*-tuples in Tup have the same probability, p_R. A natural question to ask is how does the complexity of *PQE* change if we assume that the input database is symmetric.

Notice that a symmetric database is very restrictive, since every *possible* tuple of a given relation must have the same probability, it is not sufficient to assign the same probability to all tuples in a database. For example, even assuming all probabilities in Fig.1 are equal, $p_1 = p_2 = \cdots = q_6$, the database is not symmetric, because the possible tuples that are not in the database have probability zero. Symmetric databases are motivated by Markov Logic Networks, since their translation to a probabilistic database is symmetric, e.g. the database over the vocabulary *Manager*, *HighlyCompensated*, *R* defined in Sec. 3 is symmetric, because every tuple in *Manager* has probability 1/2, every tuple in *HighlyCompensated* has probability 1/2, and every tuple in *R* has probability 1/2.9.

Surprisingly, symmetric databases can lower the complexity of query evaluation, as observed in [44]. For example, consider the query $H_0 = \forall x \forall y (R(x) \lor S(x, y) \lor T(y))$, and assume that the input is a symmetric database over a domain of size *n*. Fix two numbers $0 \leq k, \ell \leq n$, and condition on the event $|R| = k$ and $|T| = \ell$: the probability that H_0 is true is $p_S^{n^2 - k\ell}$ because all n^2 tuples (i, j) in *S* must be present, except the $k\ell$ tuples where $i \in R$ and $j \in T$. This leads to the following expression, which is computable in polynomial time in *n*:

$$p_D(H_0) = \sum_{k,\ell=0,n} \binom{n}{k}\binom{n}{\ell} p_R^k (1 - p_R)^{n-k} p_T^\ell (1 - p_T)^{n-\ell} p_S^{n^2 - k\ell}$$

Here p_R, p_S, p_T are numbers in $[0, 1]$ representing the probability of tuples in the relations R, S, T respectively.

Van den Broeck et al. proved the following surprising result:

Theorem 8.1. *[24] For every query Q in FO^2, the complexity of $PQE(Q)$ over symmetric databases is in PTIME*

This is an important step towards realizing the original vision of lifted inference in statistical relational models: exploit symmetries of lifted models in order to speed up probabilistic inference. The next natural question is whether we can generalize this result beyond FO^2. The answer is, mostly no:

Theorem 8.2. *[8] (a) There exists a sentence $Q \in FO^3$ such that $PQE(Q)$ over symmetric databases is $\#P_1$-hard. (b) There exists a conjunctive query Q such that $PQE(Q)$ over symmetric databases is $\#P_1$-hard. (c) For every γ-acyclic conjunctive query without self-joins, $PQE(Q)$ is in polynomial time.*

The class $\#P_1$ consists of #P problems where the input is given in unary. When the database is symmetric, then $PQE(Q)$ is in $\#P_1$, because the input consists only of the number n representing the size of the domain,[11] which is given as $111 \cdots 1$. Very little is known about the class $\#P_1$, in particular no natural complete problems are known for this calss. The $\#P_1$-complete queries mentioned in the theorem are not "natural", i.e. the theorem proves that they exists (and could be constructed), without giving their expression explicitly.

9 OTHER RELATED WORK

The challenge of query evaluation on probabilistic databases has lead to many innovative ideas that broaden our understanding of probabilistic inference in general. We briefly mention here some of them.

Amarilli et al. [1] study the *PQE* problem by restricting the database to have a bounded tree-width. While most of the work on probabilistic databases has fixed the query and allowed the database to be arbitrary, this work takes the opposite view, by restricting only the database. They show that, for every query in Monadic Second Order logic, $PQE(Q)$ is in polynomial time, when the input database is restricted to have a tree width $\leq k$, for some fixed k. This is a very powerful result, which should be followed up by efforts to identify applications where the database has bounded tree width.

Several probabilistic database systems have been built in the last decade or so. The most successful system built on top of an existing RDBMS is MayBMS [40], which implements a form of weighted model counting inside postgres' query plans. When the query is liftable, then MayBMS ensures an execution plan that is in polynomial time, otherwise it does a full DPLL-style search. ProbLog [51] supports a datalog-style query language with probabilistic primitive, and has found numerous applications in machine learning. ProbLog is developed from scratch (i.e. it is not extending a database system); during query execution it first grounds the query, then compiles the lineage into an OBDD or an SDD, then performs probabilistic inference on the compiled representation.

Many extensions of the basic tuple independent databases have been considered in the literature. Ceylan et al. [12] study open world probabilistic databases, where each tuple not explicitly listed in the database is associated with some small probability of being present, while Friedman and van den Broeck [28] add constraints over the missing tuples. Also motivated by open world databases, Grohe and Lindner study TIDs over *infinite* probabilistic databases, where the set of possible tuples is infinite [38, 39], leading to subtle and difficult semantic questions.

Recursive queries and infinitary logics over probabilistic databases are studied in [6] and [2] respectively.

A closely related area is that of *incomplete databases*. An incomplete database is simply a collection of possible worlds, without probabilities. In other words it is a probabilistic database without the probabilities. Incomplete databases were originally motivated by the need to model correctly the treatment of NULLs in SQL [43], but today they find numerous applications in query answering using views, data integration and exchange, inconsistency management, see the survey by Libkin [55]. Since they lack probabilities, the query answering is defined in terms of *certain answers*: an answer is certain if it is an answer in any possible world of the incomplete database. In other words an answer is certain if, for any probability distribution on the possible worlds of the incomplete database, its probability is 1. Sometimes requiring certainty is too stringent, since it will reject many answers just because they are missing from one or a few possible worlds. One approach to relax this strict requirement is to return answers whose *asymptotic* probability is equal to 1 [14, 56]: more precisely, we endow the possible worlds with some uniform distribution, similar to symmetric databases, then let the domain size tend to ∞, and return those answer whose limit probability is 1.

10 CONCLUSIONS

Research on probabilistic database was conducted on a rich background of probabilistic graphical models, weighted model counting, and statistical relational models. They bring a new perspective to these areas, by adopting tools specific in databases and database theory: the separation of query and data in *data complexity*, the use of constraints, the translation of a query into a query plan. At a conceptual level, probabilistic databases represent one approach to the integration of logic and probability theory, putting most emphasis on the complexity of query answering.

Despite the collection of theoretical results on the tractable queries, probabilistic databases have yet to lead to commercial systems. The major limitation is the lack of techniques for computing "the other" queries, namely those whose complexity is #P-hard. This limitation becomes particularly severe when modeling correlations through database constraints, because constraints are typically universally quantified first order sentences for which even approximating the output probability is NP-hard in general [25, 68]. A new approach is needed for progress in this space, one that is likely to use special properties of both queries and data. With the notable exception of [1], most of the work on probabilistic databases has imposed restrictions only the query, and assumed the worst case for the database. Recent work creates hope for this direction by identifying restrictions on the structure of the Boolean expressions that are sufficient for model counting to be in polynomial time [11, 48].

[11] We assume here that the relation probabilities $p_{R_1}, p_{R_2}, \ldots$ are known and fixed.

Further reading This paper is not meant as a comprehensive survey and omits many results on probabilistic databases (including by the author!), as well as many technical details. Readers interested in detailed surveys are referred to [25, 74].

ACKNOWLEDGMENTS

The author thanks Batya Kenig and Guy van den Broeck for their useful comments on early drafts of the paper. This work was supported by NSF IIS 1907997, NSF III 1703281, NSF III-1614738, and NSF 1535565.

REFERENCES

[1] Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. 2015. Provenance Circuits for Trees and Treelike Instances. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*. 56–68. https://doi.org/10.1007/978-3-662-47666-6_5

[2] Antoine Amarilli and İsmail İlkan Ceylan. 2020. A Dichotomy for Homomorphism-Closed Queries on Probabilistic Graphs. In *23rd International Conference on Database Theory, ICDT 2020, March 30-April 2, 2020, Copenhagen, Denmark*. 5:1–5:20. https://doi.org/10.4230/LIPIcs.ICDT.2020.5

[3] Antoine Amarilli and Benny Kimelfeld. 2019. Uniform Reliability of Self-Join-Free Conjunctive Queries. arXiv:1908.07093 [cs.DB]

[4] Marcelo Arenas, Pablo Barceló, and Mikaël Monet. 2019. Counting Problems over Incomplete Databases. *CoRR* abs/1912.11064 (2019). arXiv:1912.11064 http://arxiv.org/abs/1912.11064

[5] Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. 2003. Algorithms and Complexity Results for #SAT and Bayesian Inference. In *FOCS*. 340–351.

[6] Vince Bárány, Balder ten Cate, Benny Kimelfeld, Dan Olteanu, and Zografoula Vagena. 2017. Declarative Probabilistic Programming with Datalog. *ACM Trans. Database Syst.* 42, 4 (2017), 22:1–22:35. https://doi.org/10.1145/3132700

[7] Roberto J. Bayardo, Jr., and J. D. Pehoushek. 2000. Counting Models using Connected Components. In *AAAI*. 157–162.

[8] Paul Beame, Guy Van den Broeck, Eric Gribkoff, and Dan Suciu. 2015. Symmetric Weighted First-Order Model Counting. In *Proceedings of the 34th ACM Symposium on Principles of Database Systems, PODS 2015, Melbourne, Victoria, Australia, May 31 - June 4, 2015*. 313–328. https://doi.org/10.1145/2745754.2745760

[9] Paul Beame, Jerry Li, Sudeepa Roy, and Dan Suciu. 2017. Exact Model Counting of Query Expressions: Limitations of Propositional Methods. *ACM Trans. Database Syst.* 42, 1 (2017), 1:1–1:46. https://doi.org/10.1145/2984632

[10] George Beskales, Mohamed A. Soliman, Ihab F. Ilyas, Shai Ben-David, and Yubin Kim. 2010. ProbClean: A probabilistic duplicate detection system. In *Proceedings of the 26th International Conference on Data Engineering, ICDE 2010, March 1-6, 2010, Long Beach, California, USA*. 1193–1196. https://doi.org/10.1109/ICDE.2010.5447744

[11] Johann Brault-Baron, Florent Capelli, and Stefan Mengel. 2015. Understanding Model Counting for beta-acyclic CNF-formulas. In *32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015) (Leibniz International Proceedings in Informatics (LIPIcs))*, Ernst W. Mayr and Nicolas Ollinger (Eds.), Vol. 30. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 143–156. https://doi.org/10.4230/LIPIcs.STACS.2015.143

[12] İsmail İlkan Ceylan, Adnan Darwiche, and Guy Van den Broeck. 2016. Open-World Probabilistic Databases. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016*. 339–348. http://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12908

[13] Robert G. Cowell, Steffen L. Lauritzen, A. Philip David, and David J. Spiegelhalter. 1999. *Probabilistic Networks and Expert Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[14] Nilesh N. Dalvi, Gerome Miklau, and Dan Suciu. 2005. Asymptotic Conditional Probabilities for Conjunctive Queries. In *Database Theory - ICDT 2005, 10th International Conference, Edinburgh, UK, January 5-7, 2005, Proceedings*. 289–305. https://doi.org/10.1007/978-3-540-30570-5_20

[15] Nilesh N. Dalvi and Dan Suciu. 2004. Efficient Query Evaluation on Probabilistic Databases. In *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, VLDB 2004, Toronto, Canada, August 31 - September 3 2004*. 864–875. https://doi.org/10.1016/B978-012088469-8.50076-0

[16] Nilesh N. Dalvi and Dan Suciu. 2007. Management of probabilistic data: foundations and challenges. In *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 11-13, 2007, Beijing, China*. 1–12. https://doi.org/10.1145/1265530.1265531

[17] Nilesh N. Dalvi and Dan Suciu. 2012. The dichotomy of probabilistic inference for unions of conjunctive queries. *J. ACM* 59, 6 (2012), 30:1–30:87. https://doi.org/10.1145/2395116.2395119

[18] Adnan Darwiche. 2001. Decomposable negation normal form. *J. ACM* 48, 4 (2001), 608–647.

[19] Adnan Darwiche. 2001. On the Tractable Counting of Theory Models and its Application to Truth Maintenance and Belief Revision. *Journal of Applied Non-Classical Logics* 11, 1-2 (2001), 11–34.

[20] Adnan Darwiche. 2009. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press.

[21] Adnan Darwiche and Pierre Marquis. 2002. A knowledge compilation map. *J. Artif. Int. Res.* 17, 1 (Sept. 2002), 229–264.

[22] Martin Davis, George Logemann, and Donald Loveland. 1962. A machine program for theorem-proving. *Commun. ACM* 5, 7 (1962), 394–397.

[23] Martin Davis and Hilary Putnam. 1960. A Computing Procedure for Quantification Theory. *J. ACM* 7, 3 (1960), 201–215.

[24] Guy Van den Broeck, Wannes Meert, and Adnan Darwiche. 2014. Skolemization for Weighted First-Order Model Counting. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014*. http://www.aaai.org/ocs/index.php/KR/KR14/paper/view/8012

[25] Guy Van den Broeck and Dan Suciu. 2017. Query Processing on Probabilistic Data: A Survey. *Foundations and Trends in Databases* 7, 3-4 (2017), 197–341. https://doi.org/10.1561/1900000052

[26] Pedro M. Domingos and Daniel Lowd. 2009. *Markov Logic: An Interface Layer for Artificial Intelligence*. Morgan & Claypool Publishers. https://doi.org/10.2200/S00206ED1V01Y200907AIM007

[27] Robert Fink and Dan Olteanu. 2016. Dichotomies for Queries with Negation in Probabilistic Databases. *ACM Trans. Database Syst.* 41, 1 (2016), 4:1–4:47. https://doi.org/10.1145/2877203

[28] Tal Friedman and Guy Van den Broeck. 2019. On Constrained Open-World Probabilistic Databases. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*. 5722–5729. https://doi.org/10.24963/ijcai.2019/793

[29] Tal Friedman and Guy Van den Broeck. 2020. Symbolic Querying of Vector Spaces: Probabilistic Databases Meets Relational Embeddings. *CoRR* abs/2002.10029 (2020). arXiv:2002.10029 https://arxiv.org/abs/2002.10029

[30] Norbert Fuhr and Thomas Rölleke. 1997. A Probabilistic Relational Algebra for the Integration of Information Retrieval and Database Systems. *ACM Trans. Inf. Syst.* 15, 1 (1997), 32–66. https://doi.org/10.1145/239041.239045

[31] Wolfgang Gatterbauer and Dan Suciu. 2014. Oblivious bounds on the probability of boolean functions. *ACM Trans. Database Syst.* 39, 1 (2014), 5:1–5:34. https://doi.org/10.1145/2532641

[32] Wolfgang Gatterbauer and Dan Suciu. 2015. Approximate Lifted Inference with Probabilistic Databases. *PVLDB* 8, 5 (2015), 629–640. https://doi.org/10.14778/2735479.2735494

[33] Ben Taskar Lise Getoor (Ed.). 2007. *Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning)*.

[34] Martin Charles Golumbic, Aviad Mintz, and Udi Rotics. 2006. Factoring and recognition of read-once functions using cographs and normality and the readability of functions associated with partial k-trees. *Discret. Appl. Math.* 154, 10 (2006), 1465–1477. https://doi.org/10.1016/j.dam.2005.09.016

[35] Carla P. Gomes, Ashish Sabharwal, and Bart Selman. 2009. Model Counting. In *Handbook of Satisfiability*. IOS Press, 633–654.

[36] Eric Gribkoff, Guy Van den Broeck, and Dan Suciu. 2014. Understanding the Complexity of Lifted Inference and Asymmetric Weighted Model Counting. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence, UAI 2014, Quebec City, Quebec, Canada, July 23-27, 2014*. 280–289. https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=2463&proceeding_id=30

[37] Eric Gribkoff and Dan Suciu. 2016. SlimShot: In-Database Probabilistic Inference for Knowledge Bases. *PVLDB* 9, 7 (2016), 552–563. https://doi.org/10.14778/2904483.2904487

[38] Martin Grohe and Peter Lindner. 2019. Probabilistic Databases with an Infinite Open-World Assumption. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*. 17–31. https://doi.org/10.1145/3294052.3319681

[39] Martin Grohe and Peter Lindner. 2020. Infinite Probabilistic Databases. In *23rd International Conference on Database Theory, ICDT 2020, March 30-April 2, 2020, Copenhagen, Denmark*. 16:1–16:20. https://doi.org/10.4230/LIPIcs.ICDT.2020.16

[40] Jiewen Huang, Lyublena Antova, Christoph Koch, and Dan Olteanu. 2009. MayBMS: a probabilistic database management system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009*. 1071–1074. https://doi.org/10.1145/1559845.1559984

[41] Jinbo Huang and Adnan Darwiche. 2005. DPLL with a Trace: From SAT to Knowledge Compilation. In *IJCAI*. 156–162.

[42] Jinbo Huang and Adnan Darwiche. 2007. The Language of Search. *JAIR* 29 (2007), 191–219.

[43] Tomasz Imielinski and Witold Lipski Jr. 1984. Incomplete Information in Relational Databases. *J. ACM* 31, 4 (1984), 761–791. https://doi.org/10.1145/1634.1886

[44] Abhay Kumar Jha, Vibhav Gogate, Alexandra Meliou, and Dan Suciu. 2010. Lifted Inference Seen from the Other Side : The Tractable Features. In *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010. Proceedings of a meeting held 6-9 December 2010, Vancouver, British Columbia, Canada.* 973–981. http://papers.nips.cc/paper/4067-lifted-inference-seen-from-the-other-side-the-tractable-features

[45] Abhay Kumar Jha and Dan Suciu. 2012. Probabilistic Databases with MarkoViews. *PVLDB* 5, 11 (2012), 1160–1171. https://doi.org/10.14778/2350229.2350236

[46] Abhay Kumar Jha and Dan Suciu. 2013. Knowledge Compilation Meets Database Theory: Compiling Queries to Decision Diagrams. *Theory Comput. Syst.* 52, 3 (2013), 403–440. https://doi.org/10.1007/s00224-012-9392-5

[47] Henry A. Kautz. 2020. The Third AI Summer. https://www.cs.rochester.edu/u/kautz/talks/Kautz%20Engelmore%20Lecture.pdf. The "Robert S. Engelmore Memorial Award Lecture" at AAAI.

[48] Batya Kenig and Avigdor Gal. 2015. On the Impact of Junction-Tree Topology on Weighted Model Counting. In *Scalable Uncertainty Management - 9th International Conference, SUM 2015, Québec City, QC, Canada, September 16-18, 2015. Proceedings.* 83–98. https://doi.org/10.1007/978-3-319-23540-0_6

[49] Kristian Kersting. 2012. Lifted Probabilistic Inference. In *ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27-31 , 2012.* 33–38. https://doi.org/10.3233/978-1-61499-098-7-33

[50] Benny Kimelfeld and Yehoshua Sagiv. 2008. Modeling and querying probabilistic XML data. *SIGMOD Rec.* 37, 4 (2008), 69–77. https://doi.org/10.1145/1519103.1519115

[51] Angelika Kimmig, Bart Demoen, Luc De Raedt, Vítor Santos Costa, and Ricardo Rocha. 2011. On the implementation of the probabilistic logic programming language ProbLog. *Theory Pract. Log. Program.* 11, 2-3 (2011), 235–262. https://doi.org/10.1017/S1471068410000566

[52] Daphne Koller and Nir Friedman. 2009. *Probabilistic Graphical Models - Principles and Techniques.* MIT Press.

[53] Richard E. Ladner. 1975. On the Structure of Polynomial Time Reducibility. *J. ACM* 22, 1 (1975), 155–171. https://doi.org/10.1145/321864.321877

[54] Leonid Libkin. 2004. *Elements of Finite Model Theory.* Springer. https://doi.org/10.1007/978-3-662-07003-1

[55] Leonid Libkin. 2014. Incomplete data: what went wrong, and how to fix it. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'14, Snowbird, UT, USA, June 22-27, 2014.* 1–13. https://doi.org/10.1145/2594538.2594561

[56] Leonid Libkin. 2018. Certain Answers Meet Zero-One Laws. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018.* 195–207. https://doi.org/10.1145/3196959.3196983

[57] Stephen M. Majercik and Michael L. Littman. 1998. Using caching to solve larger probabilistic planning problems. In *AAAI* (Madison, Wisconsin, USA). 954–959.

[58] Mikaël Monet. 2019. Solving a Special Case of the Intensional vs Extensional Conjecture in Probabilistic Databases. *CoRR* abs/1912.11864 (2019). arXiv:1912.11864 http://arxiv.org/abs/1912.11864

[59] Christian Muise, Sheila A. McIlraith, J. Christopher Beck, and Eric I. Hsu. 2012. Dsharp: fast d-DNNF compilation with sharpSAT. In *Canadian AI* (Toronto, ON, Canada). 356–361.

[60] Mathias Niepert and Guy Van den Broeck. 2014. Tractability through Exchangeability: A New Perspective on Efficient Probabilistic Inference. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.* 2467–2475. http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8615

[61] Dan Olteanu and Jiewen Huang. 2008. Using OBDDs for Efficient Query Evaluation on Probabilistic Databases. In *Scalable Uncertainty Management, Second International Conference, SUM 2008, Naples, Italy, October 1-3, 2008. Proceedings.* 326–340. https://doi.org/10.1007/978-3-540-87993-0_26

[62] Judea Pearl. 1988. *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference.* Morgan Kaufmann. http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20{&}path=ASIN/1558604790

[63] David Poole. 2003. First-order probabilistic inference. In *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003.* 985–991. http://ijcai.org/Proceedings/03/Papers/142.pdf

[64] J. Scott Provan and Michael O. Ball. 1983. The Complexity of Counting Cuts and of Computing the Probability that a Graph is Connected. *SIAM J. Comput.* 12, 4 (1983), 777–788.

[65] Luc De Raedt, Sebastijan Dumančić, Robin Manhaeve, and Giuseppe Marra. 2020. From Statistical Relational to Neuro-Symbolic Artificial Intelligence. *CoRR* abs/2003.08316v1 (2020). arXiv:2003.08316 https://arxiv.org/abs/2003.08316

[66] Luc De Raedt, Kristian Kersting, Sriraam Natarajan, and David Poole. 2016. *Statistical Relational Artificial Intelligence: Logic, Probability, and Computation.* Morgan & Claypool Publishers. https://doi.org/10.2200/S00692ED1V01Y201601AIM032

[67] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *PVLDB* 10, 11 (2017), 1190–1201. https://doi.org/10.14778/3137628.3137631

[68] Dan Roth. 1996. On the Hardness of Approximate Reasoning. *Artif. Intell.* 82, 1-2 (1996), 273–302.

[69] Stuart J. Russell. 2015. Unifying logic and probability. *Commun. ACM* 58, 7 (2015), 88–97. https://doi.org/10.1145/2699411

[70] Christopher De Sa, Ihab F. Ilyas, Benny Kimelfeld, Christopher Ré, and Theodoros Rekatsinas. 2019. A Formal Framework for Probabilistic Unclean Databases. In *22nd International Conference on Database Theory, ICDT 2019, March 26-28, 2019, Lisbon, Portugal.* 6:1–6:18. https://doi.org/10.4230/LIPIcs.ICDT.2019.6

[71] Tian Sang, Fahiem Bacchus, Paul Beame, Henry A. Kautz, and Toniann Pitassi. 2004. Combining Component Caching and Clause Learning for Effective Model Counting. In *SAT.*

[72] Anish Das Sarma, Omar Benjelloun, Alon Y. Halevy, Shubha U. Nabar, and Jennifer Widom. 2009. Representing uncertain data: models, properties, and algorithms. *VLDB J.* 18, 5 (2009), 989–1019. https://doi.org/10.1007/s00778-009-0147-0

[73] Julia Stoyanovich, Susan B. Davidson, Tova Milo, and Val Tannen. 2011. Deriving probabilistic databases with inference ensembles. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany.* 303–314. https://doi.org/10.1109/ICDE.2011.5767854

[74] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. 2011. *Probabilistic Databases.* Morgan & Claypool Publishers. https://doi.org/10.2200/S00362ED1V01Y201105DTM016

[75] Marc Thurley. 2006. sharpSAT: counting models with advanced component caching and implicit BCP. In *SAT* (Seattle, WA). 424–429.

[76] BA Trakhtenbrot. 1950. The impossibility of an algorithm for the decidability problem on finite classes. In *Doklady AN SSR*, Vol. 70. 569–572.

[77] Leslie G. Valiant. 1979. The Complexity of Computing the Permanent. *Theor. Comput. Sci.* 8 (1979), 189–201. https://doi.org/10.1016/0304-3975(79)90044-6

[78] Leslie G. Valiant. 1979. The Complexity of Enumeration and Reliability Problems. *SIAM J. Comput.* 8, 3 (1979), 410–421. https://doi.org/10.1137/0208032

[79] Moshe Y. Vardi. 1982. The Complexity of Relational Query Languages (Extended Abstract). In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA.* 137–146. https://doi.org/10.1145/800070.802186

[80] Kai Zeng, Shi Gao, Barzan Mozafari, and Carlo Zaniolo. 2014. The analytical bootstrap: a new method for fast error estimation in approximate query processing. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014.* 277–288. https://doi.org/10.1145/2588555.2588579

[81] Ce Zhang, Christopher Ré, Michael J. Cafarella, Jaeho Shin, Feiran Wang, and Sen Wu. 2017. DeepDive: declarative knowledge base construction. *Commun. ACM* 60, 5 (2017), 93–102. https://doi.org/10.1145/3060586

WEIGHTED MODEL COUNTING

We review here the connection between weights, probabilities, and factors. Our running example is the Boolean formula

$$F = (X_1 \vee X_2) \wedge (X_1 \vee X_3) \wedge (X_2 \vee X_3) \qquad (14)$$

and Figure 3.

Let $X_1, \ldots, X_n$ be Boolean variables. A *truth assignment* or a *model* is a function $\theta : \{X_1, \ldots, X_n\} \rightarrow \{0, 1\}$, or, equivalently, $\theta \in \{0, 1\}^n$. Given a Boolean formula F, *a model of* F is an assignment θ that satisfies F.

Let $p_1, \ldots, p_n \in \mathbb{R}$. We interpret these numbers as the probabilities of $X_1, \ldots, X_n$ being set to 1, and define:

$$p(\theta) \overset{\text{def}}{=} \prod_{i:\theta(X_i)=0} (1 - p_i) \cdot \prod_{i:\theta(X_i)=1} p_i \qquad (15)$$

When p_i takes standard values, $p_i \in [0, 1]$, then this is precisely the probability of the assignment θ if we assign each variable X_i independently the value 1, with probability p_i.

Alternatively, let w_i be a number $\in \mathbb{R} \cup \{\infty\}$, called the weight of the Boolean variable X_i, and define the weight of a model:

$$weight(\theta) \overset{\text{def}}{=} \prod_{i:\theta(X_i)=1} w_i \qquad (16)$$

θ :							
X_1	X_2	X_3	Formula F	$p(\theta)$	*weight*(θ)	Feature G	*weight*$'(\theta)$
0	0	0	0	$(1-p_1)(1-p_2)(1-p_3)$	1	1	w_4
0	0	1	0	$(1-p_1)(1-p_2)p_3$	w_3	1	w_3w_4
0	1	0	0	$(1-p_1)p_2(1-p_3)$	w_2	1	w_2w_4
0	1	1	1	$(1-p_1)p_2p_3$	w_2w_3	1	$w_2w_3w_4$
1	0	0	0	$p_1(1-p_2)(1-p_3)$	w_1	0	w_1
1	0	1	1	$p_1(1-p_2)p_3$	w_1w_3	0	w_1w_3
1	1	0	1	$p_1p_2(1-p_3)$	w_1w_2	1	$w_1w_2w_4$
1	1	1	1	$p_1p_2p_3$	$w_1w_2w_3$	1	$w_1w_2w_3w_4$

Figure 3: Probabilities and weights

Denote Z the sum of the weights of all models. It is easy to verify that Z has a simple closed form:

$$Z \overset{\text{def}}{=} \sum_{\theta} weight(\theta) = \prod_i (1 + w_i)$$

We define:

$$p(\theta) \overset{\text{def}}{=} weight(\theta)/Z \tag{17}$$

It is easy to check that, when $p_i = w_i/(1+w_i)$, or, equivalently $w_i = p_i/(1-p_i)$ for all $i = 1, n$, then the formulas (15) and (17) are equal. Standard probability values $p_i \in [0,1]$ are mapped to standard weights $w_i \in [0, \infty]$, but the equivalence of (15) and (17) holds even for non-standard values.[12]

The weight, and the probability of a Boolean formula F are:

$$weight(F) \overset{\text{def}}{=} \sum_{\theta:\theta \models F} weight(\theta)$$

$$p(F) \overset{\text{def}}{=} weight(F)/Z$$

For our running example Eq.(14), Figure 3 shows the eight assignments, four of which are models of F and we derive:

$$weight(F) = w_2w_3 + w_1w_3 + w_2w_3 + w_1w_2w_3$$

A Markov Network (NN) defines a multivariate probability distribution as a product of factors [52]. In our setting, we define a factor as either a single-variable factor (w_i, X_i), or as a pair (w, G), where $w \in \mathbb{R}$ and G is a Boolean formula. The value of the factor is w when G is true, and 1 otherwise. The *factorized probability distribution*, p', defined by a set of factors $\mathcal{F}$ is:

$$weight'(\theta) \overset{\text{def}}{=} \prod_{i:\theta(X_i)=1} w_i \times \prod_{(w,G)\in\mathcal{F}:\theta\models G} w$$

$$Z' \overset{\text{def}}{=} \sum_{\theta} weight'(\theta)$$

$$p'(\theta) \overset{\text{def}}{=} weight'(\theta)/Z'$$

Continuing our running example in Figure 3, suppose we add the factor $(w_4, (X_1 \Rightarrow X_2))$. This modifies the weight to *weight*$'(\theta)$ shown in the last column in Figure 3, and the new weight of F is:

$$weight'(F) = w_2w_3w_4 + w_1w_3 + w_2w_3w_4 + w_1w_2w_3w_4$$

[12]One has to require $w_i \neq -1$, to ensure that $p_i \in (-\infty, \infty)$ and that $Z \neq 0$.

Thus, in an MN the normalization factor Z' no longer has a simple closed form expression, and there is no longer a simple mapping from weights to probabilities.

Readers familiar with MN's may notice that in our setting the factor (w, G) takes only values 1 and w, while, in a general MN, a factor over k variables make take 2^k values, $w_1, \ldots, w_{2^k}$. However, this can be converted into a product of 2^k factors, where each takes only values 1 and w_i respectively, hence our definition is w.l.o.g.

Finally, we show now how to convert an MN into an independent model conditioned on a constraint, by replacing the factor (w_4, G) with a new independent variable X_4 and a constraint Γ. We show two approaches. In the first, *weight*$(X_4) \overset{\text{def}}{=} w_4$ and $\Gamma \overset{\text{def}}{=} (X_4 \Leftrightarrow G)$. Let p'' denote the probability distribution defined by the 4 independent random variables $X_1, \ldots, X_4$. Then $p'(\theta) = p''(\theta|\Gamma)$; indeed, while p'' is a distribution over 16 outcomes (since we have 4 variables), only 8 of them satisfy the constraint Γ, hence our claim is an easily verified identity about two distributions over eight outcomes.

The second approach defines *weight*$(X_4) = 1/(w_4 - 1)$ and $\Gamma \overset{\text{def}}{=} X_4 \vee G$. We claim that here, too, $p'(\theta) = p''(\theta|\Gamma)$. The main idea in the proof is the following observation. In the distribution p' the factor G contributes either a weight 1 or w_4, depending on whether G is false or true under the assignment θ: importantly, their ratio is $1 : w_4$. Consider now the weights contributed by X_4 in the new distribution conditioned on Γ. When G is false, then X_4 must be true and it contributes the weight $1/(w_4 - 1)$. When G is true, then X_4 can be either false or true, and the sum of the two weights is $1 + 1/(w_4 - 1) = w_4/(w_4 - 1)$. The ratio of these two factors is also $1 : w_4$. We invite the reader to complete the proof of $p'(\theta) = p''(\theta|\Gamma)$. Finally, we notice that, when $w_4 < 1$, then *weight*$(X_4) < 0$. In particular, the probability of X_4 is a non-standard value, either < 0 or > 1. However, any conditional probability $p''(F|\Gamma)$ is still a standard value in $[0, 1]$.

LINEAGE OF AN FO SENTENCE

We briefly review the standard, inductive definition of the lineage of an FO sentence Q:

$$F_{Q_1 \wedge Q_2, \mathtt{DOM}} = F_{Q_1,\mathtt{DOM}} \wedge F_{Q_2,\mathtt{DOM}} \qquad F_{Q_1 \vee Q_2, \mathtt{DOM}} = F_{Q_1,\mathtt{DOM}} \vee F_{Q_2,\mathtt{DOM}}$$

$$F_{\forall x Q, \mathtt{DOM}} = \bigwedge_{a \in \mathtt{DOM}} F_{Q[a/x],\mathtt{DOM}} \qquad F_{\exists x Q, \mathtt{DOM}} = \bigvee_{a \in \mathtt{DOM}} F_{Q[a/x],\mathtt{DOM}}$$

$$F_{\neg Q, \mathtt{DOM}} = \neg F_{Q,\mathtt{DOM}} \qquad F_{t_i, \mathtt{DOM}} = X_i$$