Lifted Probabilistic Inference: A Guide for the Database Researcher

Eric Gribkoff Dan Suciu Guy Van den Broeck

1 Introduction

Modern knowledge bases such as Yago [14], DeepDive [19], and Google's Knowledge Vault [6] are constructed from large corpora of text by using some form of supervised information extraction. The extracted data usually starts as a large probabilistic database, then its accuracy is improved by adding domain knowledge expressed as hard or soft constraints. Finally, the knowledge base can be queried using some general-purpose query language (SQL, or Sparql).

A key technical challenge during the construction, refinement, and querying of knowledge bases is probabilistic reasoning. Because of the size of the data involved, probabilistic reasoning in knowledge bases becomes a central data management problem. The number of random variables is very large, typically one for each fact in the knowledge base. Most systems today perform inference by using Markov Chain Monte Carlo (MCMC) methods; for example, DeepDive uses Gibbs Sampling, a form of MCMC. While MCMC methods are broadly applicable, probabilistic inference techniques based on MCMC have no polynomial time convergence guarantees; this is unavoidable, as exact probabilistic inference is provably intractable. Nevertheless, many classes of queries can be evaluated efficiently using an alternate, quite promising approach: *lifted inference*. While traditional methods first ground the knowledge base and run MCMC over the resulting large probabilistic space, lifted inference performs the inference directly on the first-order expression, which is much smaller than the entire probabilistic space. Lifted inference has evolved separately in the AI community [21] and in the database community [28]. When applicable, lifted inference is extremely efficient, in theory and in practice. Evaluating a query using lifted inference is equivalent to computing a SQL query with aggregate operators. Today's database engines have a large toolbox for computing aggregate queries, and such queries can be computed quite efficiently both on a single server and on a distributed system. But lifted inference has an important limitation: it only applies to some first-order expressions. In this paper we review lifted inference, both from the AI and from the database perspective, and describe some recent techniques that have expanded the applicability of lifted inference.

2 Probabilistic Databases

A Knowledge Base starts as a large collection of facts with probabilities, called a *probabilistic database* in the DB community. We review here the basic concepts following [28]. A probabilistic database is a relational database

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Tweeter

Name	Topic	P
Alice	Transactions	0.7
Alice	SocialNetworks	0.8
Carol	SocialNetworks	0.9

Follows

Follower	Followee	P
Alice	J.Bieber	0.3
Alice	G.Keillor	0.4
Carol	J.Bieber	0.5

Celebrity

Name	P
J.Bieber	0.99
M.Jordan	0.66
G.Keillor	0.33

Figure 1: A probabilistic database

where each tuple is a random Boolean variable. Each relation has one extra attribute, P, representing the probability that the record is present. For now, we will assume that all tuples are independent, and will revisit this assumption in Sect. 5. The main research challenge in probabilistic databases is to compute the output probabilities for query answers.

A simple example is in Fig. 1, which shows a tuple-independent probabilistic database with three tables. Consider the following SQL query:

```
select distinct Tweeter.topic
from Tweeter, Follows
where Tweeter.name = Follows.follower
```

Equivalently, the query written in First-Order Logic (FOL) is:

$$Q_1(t) = \exists x \exists y \, \text{Tweeter}(x, t) \wedge \text{Follows}(x, y)$$

The query retrieves all topics tweeted by people who are followers (occur in the `Follows` table). The answers to the query are uncertain, and the probabilistic database system needs to compute the output probability for each answer:

Topic	P
Transactions	$0.7 \cdot [1 - (1 - 0.3) \cdot (1 - 0.4)] = 0.406$
SocialNetworks	$1 - \{1 - 0.8 \cdot [1 - (1 - 0.3) \cdot (1 - 0.4)]\} \cdot \{1 - 0.9 \cdot 0.5\} = 0.7052$

In general, if a database has n tuples, then it defines a probability space with 2^n possible outcomes. While some queries, like the query above, can be computed efficiently, for other queries the complexity of computing the output probabilities is #P-hard in the size of the database. A simple example is the following query, whose complexity is #P-hard:

$$Q_2(t) = \exists x \exists y \, \text{Tweeter}(x, t) \wedge \text{Follows}(x, y) \wedge \text{Celebrity}(y)$$

One common misconception about probabilistic databases is that the independence assumption limits their usefulness. This is wrong: correlations can be modeled using constraints, and the latter folded back inside query evaluation over tuple-independent probabilistic databases; we will discuss this in Sect. 5.

3 Weighted Model Counting (WMC)

Probabilistic inference in Knowledge Bases can be reduced to the Weighted Model Counting (WMC) problem, which is a very well studied problem in the theory community. In Model Counting, or #SAT [11], one has to compute the number of models $\#F$ of a Boolean formula F. In WMC, each model has a weight and we have to compute the *sum of their weights*. Several state-of-the-art probabilistic inference algorithms for Bayesian networks [5, 24, 2], relational Bayesian networks [3] and probabilistic programs [8] perform a reduction to WMC. We give here a brief overview of WMC.

Propositional WMC Consider a Boolean Formula F over n Boolean variables $X_1, \ldots, X_n$, and two weight functions, $\bar{w}, w$, that associates to each Boolean variable X_i two weights: $\bar{w}(X_i), w(X_i) \in \mathbb{R}$. The weight of a truth assignment, $\theta : \{X_1, \ldots, X_n\} \to \{0, 1\}$, is defined as:

$$\text{WM}(\theta) = \prod_{i=1,n:\theta(X_i)=0} \bar{w}(X_i) \times \prod_{i=1,n:\theta(X_i)=1} w(X_i)$$

In other words, when $X_i = 0$ then we multiply with $\bar{w}(X_i)$, and when $X_i = 1$ then we multiply with $w(X_i)$. The *weight of the Boolean formula*, F, is the sum of weights of all assignments that satisfy F:

$$\text{WMC}(F, \bar{w}, w) = \sum_{\theta:\{X_1,\ldots,X_n\}\to\{0,1\},\theta(F)=1} \text{WM}(\theta)$$

For a simple illustration, if $F = X_1 \wedge X_2$ then $\text{WMC}(F, \bar{w}, w) = w(X_1) \cdot w(X_2) \cdot \prod_{i=3}^{n}(w(X_i) + \bar{w}(X_i))$.

Often, the weight $\bar{w}(X_i)$ is omitted from the specification of the WMC and assumed by default to be equal to 1. In Sect. 6 we will use values for $\bar{w}$ that are different from 1. There are three special cases of WMC of particular interest.

1. In the *Standard* WMC, $\bar{w}(X_i) = 1$ for all i. In that case we write $\text{WMC}(F, w)$ instead of $\text{WMC}(F, 1, w)$.
2. In *Model Counting*, #SAT, we set $w(X_i) = \bar{w}(X_i) = 1$ for all i, then $WMC(F, w) = \#F$, the number of satisfying assignments of F.
3. In *Probability Computation*, each variable X_i is set to true with some known probability $p(X_i) \in [0, 1]$, and we want to compute $\Pr(F, p)$, the probability that F is true. Then we set $\bar{w}(X_i) = 1 - p(X_i)$, $w(X_i) = p(X_i)$, in other words we have $\Pr(F, p) = \text{WMC}(F, p, 1 - p)$.

The WMC, the standard WMC, and the probability computation problems are equivalent, under the mild assumption that none of the following denominators are zero:

$$\text{WMC}(F, \bar{w}, w) = \text{WMC}(F, 1, w/\bar{w}) \times \prod_i \bar{w}(X_i) \tag{1}$$

$$\text{WMC}(F, \bar{w}, w) = \Pr(F, w/(w + \bar{w})) \times \prod_i (w(X_i) + \bar{w}(X_i))$$

Each of the three problems is reducible to the others. As such, we shall use them interchangeably. Probabilities are most convenient to describe lifted inference in Sect. 4, while weights are most convenient to describe the soft constraints in Sect. 5. Notice that weighted model count and probability computation strictly generalize #SAT.

In the original paper where Valiant introduced the class #P [29] he already noted that #SAT is #P-complete; he proved in a companion paper [30] that the problem remains #P-hard even if F is restricted to positive 2CNF formulas, and similarly for positive 2DNF formulas. Provan and Ball [22] proved that F can be further restricted to be a *Partitioned* Positive 2CNF (or 2DNF), meaning that every clause is of the form $X_i \vee Y_j$, where $X_1, X_2, \ldots$, and $Y_1, Y_2, \ldots$, are disjoint sets of variables, but it is possible that $X_i = X_j$ or $Y_i = Y_j$ for $i \neq j$. Of course, these classes of formulas remain #P-hard for WMC and for probabilistic computation.

First-Order WMC Next, we move from propositional formulas to sentences in first-order logic. Consider a relational vocabulary (database schema): $R_1, \ldots, R_k$. Assume a domain of constants D of size n, and denote $\mathcal{L}$ the set of all grounded atoms that can be built using the relations R_j and the values in the domain D. Let $\bar{w}, w : \mathcal{L} \to \mathbb{R}^+$ be two weight functions. For any first-order sentence Φ, its *grounding*, or *lineage* is the Boolean function $F_{\Phi,D}$ over the grounded atoms $\mathcal{L}$ (viewed as propositional Boolean variables), obtained by replacing quantifiers $\exists$ and $\forall$ with disjunctions or conjunctions over the domain D: $F_{\exists x\Phi,D} = \bigvee_{a \in D} F_{\Phi[a/x],D}$, and $F_{\forall x\Phi,D} = \bigwedge_{a \in D} F_{\Phi[a/x],D}$; furthermore, $F_{\Phi_1 \text{ op } \Phi_2,D} = F_{\Phi_1,D}$ op $F_{\Phi_2,D}$ for op $\in \{\vee, \wedge\}$, $F_{\neg\Phi,D} = \neg F_{\Phi,D}$ and finally $F_t = t$ for a ground atom $t \in \mathcal{L}$. The grounding of even a simple FO sentence can be quite large. Consider the sentence $\Phi = \exists x, y, z, w,\ R_1(x, y) \wedge R_2(y, z) \wedge R_3(z, w)$. If the domain D has size 1000, the grounding of Φ contains 10^{12} distinct formulas. The terms "grounding" and "lineage" were introduced independently in the AI literature and in the DB literature; we will use them interchangeably in this paper.

The weighted *first-order* model count of sentence Φ for domain D is the weighted model count of its grounding, that is, $\text{WFOMC}(\Phi, D, \bar{w}, w) = \text{WMC}(F_{\Phi,D}, \bar{w}, w)$.

Definition 1 (The WFOMC Problem:): Given a domain D, weight functions $w, \bar{w}$, and first-order sentence Φ, compute $\text{WFOMC}(\Phi, D, \bar{w}, w)$.

One can view the triple $(D, w, \bar{w})$ as a probabilistic database, where every tuple t has probability $w(t)/(w(t) + \bar{w}(t))$, and the FO-WMC is equivalent to the query computation problem in a probabilistic database.

4 Lifted Inference

The term *lifted inference* was coined by Poole [21] to mean (informally) inference techniques that can solve WFOMC without grounding the FO sentence. Subsequently different classes of techniques have been called

"lifted", even with grounding. A more rigorous definition was proposed by Van den Broeck [31]; an algorithm is called *domain lifted* if it runs in time polynomial in n, the size of the domain. Independently, the database community searched for algorithms whose data complexity is in PTIME. Following Vardi [36], in *data complexity* the query expression is fixed, and one measures the complexity as a function of the size of the input database. Dalvi and Suciu [4] showed that some queries have a PTIME data complexity, while others are #P-hard. For the WFOMC task, these two notions developed in the AI and DB communities coincide:[1] liftability, defined as an algorithm that runs in PTIME in the size of the domain, and PTIME data complexity. We briefly review here lifted inference algorithms for WFOMC.

Lifted inference algorithms compute $\Pr(\Phi)$ by repeatedly applying one of the following rules to a FO sentence Φ, until it becomes a ground tuple t, in which case $\Pr(t)$ can be simply looked up in the database:

Independent (decomposable) AND $\Pr(\Phi_1 \wedge \Phi_2) = \Pr(\Phi_1) \cdot \Pr(\Phi_2)$, when the lineages of Φ_1, Φ_2 have disjoint sets of atoms. This pre-condition is checked by analysis of the first-order expressions and implies Φ_1, Φ_2 are independent.

Independent (decomposable) FORALL $\Pr(\forall x \Phi) = \prod_{a \in D} \Pr(\Phi[a/x])$ if x is a "separator variable" in Φ (for any two constants $a \neq b$, $\Phi[a/x]$ is independent from $\Phi[b/x]$).

Inclusion/exclusion $\Pr(\Phi_1 \vee \Phi_2) = \Pr(\Phi_1) + \Pr(\Phi_2) - \Pr(\Phi_1 \wedge \Phi_2)$.

Negation $\Pr(\neg \Phi) = 1 - \Pr(\Phi)$

Exclusive (deterministic) OR $\Pr(\Phi_1 \vee \Phi_2) = \Pr(\Phi_1) + \Pr(\Phi_2)$ if Φ_1, Φ_2 are exclusive ($\Phi_1 \wedge \Phi_2 \equiv$ false).

By duality we also have $\Pr(\Phi_1 \vee \Phi_2) = 1 - (1 - \Pr(\Phi_1) \cdot (1 - \Pr(\Phi_2)))$ when Φ_1, Φ_2 are independent, and $\Pr(\exists x \Phi) = 1 - \prod_{a \in D}(1 - \Pr(\Phi[a/x]))$ when x is a separator variable in Φ. The rules are effective, in the sense that one can check their preconditions using a simple syntactic analysis on the FOL expressions, and applying the rules can be done in time polynomial in the size of the domain D [28]. For a simple example, to compute the probability of $\exists x \exists y \text{Tweeter}(x, \text{SocialNetworks}) \wedge \text{Follows}(x, y)$ on the database in Fig. 1 one applies the separator variable rules to $\exists x$, followed by an independent AND, arriving at the formula in Sect. 2.

A sequence of lifted inference rules is naturally represented as a tree, which can be interpreted either as a *query plan*, or as a *circuit*. Fig. 2 shows the Relational Query plan [28], and the equivalent FO-d-DNNF circuit [35, 32] for computing the query Q_1 in Sect. 2.

For some queries one cannot compute their probabilities using lifted inference alone; for example this must happen when the query has a data complexity that is #P-hard. For example, consider

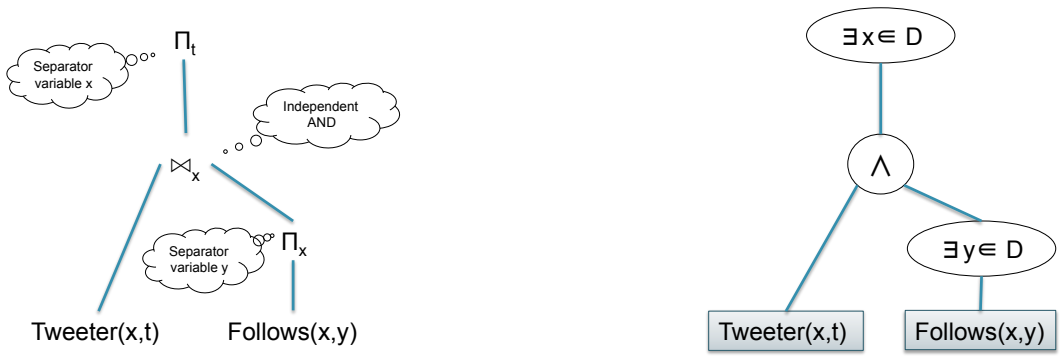
$$\Phi_2 = \exists x \exists y \text{Tweeter}(x, \text{SocialNetworks}) \wedge \text{Follows}(x, y) \wedge \text{Celebrity}(y) \tag{2}$$

which corresponds to Q_2 in Sect. 2. Its lineage is the following (where we abbreviate the constants in the domain by their first letter, e.g. writing A instead of Alice, etc):

$$\begin{aligned} F = &\text{Tweeter}(A, S) \wedge \text{Follows}(A, J) \wedge \text{Celebrity}(J) \\ &\vee \text{Tweeter}(A, S) \wedge \text{Follows}(A, G) \wedge \text{Celebrity}(G) \\ &\vee \text{Tweeter}(C, S) \wedge \text{Follows}(A, J) \wedge \text{Celebrity}(J) \end{aligned}$$

It is not hard to see that computing $\Pr(\Phi_2)$ is #P-hard in the size of D: indeed, any PP2DNF formula $\bigvee X_i \wedge Y_j$ is equivalent to a lineage above, if we choose Tweeter to represent the variables X_i, Celebrity to represent Y_j, and Follows to represent all prime implicants $X_i \wedge Y_j$ (by setting the probabilities to 1 or 0). In other words, computing the answer to Q_2 is #P-hard, and therefore (under standard complexity assumptions) its probability cannot be computed using lifted inference.

[1] In other contexts (e.g., Markov logic), one may want to distinguish between the data and domain, resulting in distinct notions.



(a) (b)

Figure 2: A Query Plan (a) and the equivalent FO-d-DNNF circuit (b); both compute the query Q_1 in Sect. 2. The Appendix shows how to express the query directly in Postgres.

Completeness One open question is whether the lifted inference rules are complete for exact probabilistic inference: one wonders if one should search for new rules, or whether our list is exhaustive in some formal sense. The criteria for "completeness" is to be able to compute the probability of any query computable in PTIME. It has been shown in [4] that the rules are complete to evaluate all Unions of Conjunctive Queries (UCQ) that are in PTIME. It follows immediately that the rules are complete both for UCQ, and for their dual counterpart, the positive, universal fragment of First Order Logic. However, these rules are *not* complete for languages with negation: as the following example shows, there are PTIME queries with negation where none of the above rules apply, and an algorithm incorporating only these rules is immediately stuck. Applying logical resolution can derive equivalent queries such that the above rules apply, but even then it is open if this gives a complete algorithm for lifted inference [13]. For example, consider this query:

$$\forall x \forall y (\text{Tweeter}(x) \vee \neg \text{Follows}(x, y)) \wedge (\text{Follows}(x, y) \vee \neg \text{Celebrity}(y))$$

(Written as $\text{Follows}(x, y) \Rightarrow \text{Tweeter}(x)$ and $\text{Celebrity}(y) \Rightarrow \text{Follows}(x, y)$, it says that every follower is a tweeter, and that every celebrity is followed by everyone.) Our list of rules are stuck on this query: in fact, if we remove all negations, then the query is known to be #P-hard, hence to get unstuck we need to use the negation in an essential way. We do this by applying resolution between the two clauses then we obtain the equivalent query:

$$\begin{aligned}
\Phi =& \forall x \forall y (\text{Tweeter}(x) \vee \neg \text{Follows}(x, y)) \wedge (\text{Follows}(x, y) \vee \neg \text{Celebrity}(y)) \wedge (\text{Tweeter}(x) \vee \neg \text{Celebrity}(y)) \\
\equiv& \forall x \forall y (\text{Tweeter}(x) \vee \neg \text{Follows}(x, y)) \wedge (\text{Follows}(x, y) \vee \neg \text{Celebrity}(y)) \wedge \text{Tweeter}(x) \\
\vee& \forall x \forall y (\text{Tweeter}(x) \vee \neg \text{Follows}(x, y)) \wedge (\text{Follows}(x, y) \vee \neg \text{Celebrity}(y)) \wedge \neg \text{Celebrity}(y) \\
\equiv& \forall x \forall y \wedge (\text{Follows}(x, y) \vee \neg \text{Celebrity}(y)) \wedge \text{Tweeter}(x) \\
\vee& \forall x \forall y (\text{Tweeter}(x) \vee \neg \text{Follows}(x, y)) \wedge \neg \text{Celebrity}(y)
\end{aligned}$$

Now we apply inclusion/exclusion and obtain:

$$\begin{aligned}
\Pr(\Phi) = & \Pr(\forall x \forall y (\text{Follows}(x, y) \vee \neg \text{Celebrity}(y)) \wedge \text{Tweeter}(x)) \\
& + \Pr(\forall x \forall y (\text{Tweeter}(x) \vee \neg \text{Follows}(x, y)) \wedge \neg \text{Celebrity}(y)) \\
& - \Pr(\forall x \text{Tweeter}(x) \wedge \forall y \neg \text{Celebrity}(y))
\end{aligned}$$

and all three resulting expressions can be computed using lifted inference.

Symmetric Probabilities A special case is when all ground tuples belonging to the same relation have the same probabilities. These types of weights occur in the #SAT model counting setting, and are of particular interest in machine learning. Van den Broeck [34] has established recently that, if one adds one other rule (to eliminate unary predicates) then the rules are complete when all probabilities are symmetric, and the sentence Φ has at most two logical variables. In that case, for example, query (2) is computable in PTIME.

Rule independence Are all rules necessary? In other words, can we remove some rules form the list because we can express them using the other rules? For example, exclusive OR can be expressed using inclusion/exclusion. An interesting open question is whether inclusion/exclusion can be substituted by exclusive OR.

Approximate Lifted Inference Lifted inference has been extended beyond exact inference to handle approximations. In the database community, Dylla [7] reasons on the FOL formula to determine lower and upper bounds on the output probability, extending the techniques in [20] developed for propositional inference. The AI community has a rich literature on approximate lifted inference with symmetric probabilities (e.g., [18, 16, 27, 33, 10, 26]).

Other Lifted Tasks Next to the task of computing marginal probabilities, the AI community considers several other inference and learning tasks. This includes the *Maximum a Posteriori* (MAP) problem, that is, to determine the most likely state of all variables in the distribution. For recent work on lifting the MAP task with symmetric probabilities, see [17, 1, 25]. Because the standard probabilistic database setting assumes tuple independence, the MAP task has received little attention in the probabilistic database literature: without any additional constraints, the probability distribution completely factorizes, and computing the MAP state reduces to setting every tuple with probability greater than 0.5 to true and the rest to false. Recently, [12] proposed the *Most Probable Database* (MPD) problem, which is variation on the MAP task that has several database applications. Nevertheless, exact lifted inference for the MAP and MPD tasks is still poorly understood. Other tasks one may consider include lifted marginal MAP, a harder generalization of the MAP task where the most likely state of only some of the variables is sought, and lifted structure and parameter learning. These are largely unexplored problem areas.

5 Hard and Soft Constraints

Conceptually, the construction of a large Knowledge Base has two major steps. The first step generates a database of facts, by using some information extraction systems. The result is a database where each fact has some confidence, or probability, and, thus, it looks much like a probabilistic database. Second, one adds constraints representing common knowledge; these constraints, also called rules, are FOL formulas and may be either hard, or soft.

We illustrate soft rules by adapting the syntax of Markov Logic Networks (MLNs), introduced by Richardson and Domingos [23]. Consider the following three soft rules:

$$\begin{aligned} 5.38: &\quad \Phi_1(x) = \text{Tweeter}(x, \text{MachineLearning}) \Rightarrow \text{Follows}(x, \text{M.Jordan}) \\ 1.2: &\quad \Phi_2(x) = \forall y(\text{Follows}(x, y) \Rightarrow \text{Celebrity}(y)) \\ 0.8: &\quad \Phi_3(x) = \forall y \forall z(\text{Tweeter}(x, y) \wedge \text{Tweeter}(x, z) \wedge y \neq z) \end{aligned}$$

The first rule says that tweeters on MachineLearning are likely to follow M.Jordan. The second rule says that people who are followed are typically celebrities. Note that both these rules have a weight > 1, and therefore we assert that they are more likely to happen. The last rule has a weight < 1, saying that for any tweeter x, it is unlikely that x tweets on two different topics $y \neq z$. Each formula has a free variable x.

An MLN is a set of FOL formulas $\text{MLN} = \{\Phi_1, \ldots, \Phi_m\}$ and two weight functions $\bar{w}(\Phi_i), w(\Phi_i)$; the weight function $\bar{w}(\Phi_i)$ is often equal to 1, and omitted from the specification of the MLN. The formulas may have free variables, in other words they are not necessarily sentences; we write $\Phi_i(\mathbf{x}_i)$ to denote that $\mathbf{x}_i$ are the free variables in Φ_i.

Given a domain D, we will describe now how the MLN defines a probability space on the subsets $\mathcal{L}$. Recall that $\mathcal{L}$ represents the set of grounded atoms over the domain D. A subset of $\mathcal{L}$ is called a *possible world*, and we

will define it equivalently as a truth assignment $\theta : \mathcal{L} \to \{0, 1\}$. Its *weight* is:

$$\mathrm{WM}(\theta) = \prod_{\Phi_i \in \mathrm{MLN}, \mathbf{a} \in D^{|\mathbf{x}_i|} : \theta \not\models \Phi_i[\mathbf{a}/\mathbf{x}_i]} \bar{w}(\Phi_i) \times \prod_{\Phi_i \in \mathrm{MLN}, \mathbf{a} \in D^{|\mathbf{x}_i|} : \theta \models \Phi_i[\mathbf{a}/\mathbf{x}_i]} w(\Phi_i)$$

In other words, to compute the weight of a possible world θ, we take every formula $\Phi_i(\mathbf{x}_i)$ in the MLN, substitute $\mathbf{x}_i$ in all possible ways with constants $\mathbf{a}$ from the domain to obtain a sentence $\Phi_i[\mathbf{a}/\mathbf{x}_i]$, and we multiply the weight either by $\bar{w}(\Phi_i)$, or by $w(\Phi_i)$, depending on whether the sentence $\Phi_i[\mathbf{a}/\mathbf{x}_i]$ is false or true in the world θ. For example, consider a simple MLN with the single rule $\Phi(x) = \exists y R(x, y)$, weight functions $w(\Phi) = 2$ and $\bar{w}(\Phi) = 0.5$, with domain $x, y \in \{A, B, C\}$. Let θ be a world where $R(A, A), R(A, B), R(B, A)$ are true and $R(A, C), R(B, B), R(B, C), R(C, A), R(C, B), R(C, C)$ are false. Then $\Phi(A), \Phi(B)$ are true and $\Phi(C)$ is false, hence $\mathrm{WM}(\theta) = 2 \cdot 2 \cdot 0.5 = 2$.

The weight of a query (sentence) Φ is defined as before:

$$\mathrm{WMC}(\Phi, \mathrm{MLN}, \bar{w}, w) = \sum_{\theta : \mathcal{L} \to \{0,1\} : \theta(F_\Phi) = 1} \mathrm{WM}(\theta) \tag{3}$$

and its probability is obtained by dividing by a normalization factor Z:

$$\Pr(\Phi) = \frac{\mathrm{WMC}(\Phi, \mathrm{MLN}, \bar{w}, w)}{Z}$$
$$\text{where } Z = \sum_{\theta : \mathcal{L} \to \{0,1\}} \mathrm{WM}(\theta)$$

In a typical Knowledge Base one starts with a tuple-independent probabilistic database, then adds soft rules that represent common knowledge. Once we add rules, the tuples in the database are no longer independent. In fact, every grounding for some formula in the MLN creates a new factor, and correlates all the tuples that occur in that grounding.

In the standard definition of an MLN, every formula has $\bar{w}(\Phi_i) = 1$, in other words we have only the positive weight function $w(\Phi_i)$. A "non-standard" MLN, which has explicit negated weights $\bar{w}(\Phi_i)$, can be easily converted into a standard MLN by using equation (1). The negated weights $\bar{w}$, however, make it easier to represent hard constraints by setting $\bar{w}(\Phi_i) = 0$ and $w(\Phi_i) = 1$: in this case the formula Φ_i acts as a hard constraint, because only worlds where Φ_i is true contribute to the sum (3). In the standard formulation, a hard constraint is represented by setting $w(\Phi_i) = \infty$, requiring the system to handle hard constraints separately.

6 Recent Developments: Transformations and Negative Weights

In a series of recent papers [15, 34] some researchers (including the authors) have found simple transformations that convert between MLNs and WFOMC problems, and use clever expressions on the weights in order to simplify the logical formulas. The new weights may be negative, which may correspond to either negative probabilities, or probabilities > 1. (Recall that the connection between weights w, $\bar{w}$ and the probability p is $p = w/(w + \bar{w})$.) We give here a brief overview of these rewritings.

First we show a well known fact, that every MLN can be converted into an equivalent tuple-independent probabilistic database, plus hard constraints [35, 9]. If $\mathrm{MLN} = \{\Phi_1(\mathbf{x}_1), \ldots, \Phi_m(\mathbf{x}_m)\}$, then for each $i = 1, m$ we create a new relational symbol T_i of arity $|\mathbf{x}_i|$, add the hard constraint:

$$\Gamma_i = \forall \mathbf{x}_i (\Phi_i(\mathbf{x}_i) \Leftrightarrow T(\mathbf{x}_i)) \tag{4}$$

and create a new probabilistic relation T_i where all tuples $T_i(\mathbf{a})$ have the same weights:

$$\bar{w}(T_i) = \bar{w}(\Phi_i) \qquad\qquad w(T_i) = w(\Phi_i)$$

(We write $w(T_i)$ instead of $w(T_i(\mathbf{a}))$ since all tuples in T_i have exactly the same weights.) In other words, we move the weights from the formula Φ_i to the new relation T_i, and add a hard constraint asserting that the formula and the relation are equivalent.

A disadvantage of Eq.(4) is that it introduces negations, because it expands to $(\neg\Phi_i \vee T_i) \wedge (\Phi_i \vee \neg T_i)$. As we explained earlier, negations are more difficult to handle in lifted inference. A way around this is to modify the hard constraint to:

$$\Gamma_i = \forall \mathbf{x}_i(\Phi_i(\mathbf{x}_i) \vee T_i(\mathbf{x}_i))$$

and set the weights of each tuple $T_i(\mathbf{a})$ with the objective of achieving the equalities:

$$\bar{w}(\Phi_i) = w(T_i) \qquad\qquad w(\Phi_i) = \bar{w}(T_i) + w(T_i)$$

Recall that $\bar{w}(\Phi_i)$ is often 1, and in that case $w(T_i) = 1$. These expressions are justified by the following argument. Consider some values $\mathbf{a}$ for the variables $\mathbf{x}_i$. If $\Phi_i[\mathbf{a}/\mathbf{x}_i]$ is false in a world, then $T_i[\mathbf{a}/\mathbf{x}_i]$ must be true (because $\Phi_i[\mathbf{a}/\mathbf{x}_i] \vee T_i[\mathbf{a}/\mathbf{x}_i]$ is a hard constraint), and the same factor $\bar{w}(\Phi_i) = w(T_i)$ is applied to this world in the original MLN as in the new MLN. If $\Phi_i[\mathbf{a}/\mathbf{x}_i]$ is true in a world, then $T_i[\mathbf{a}/\mathbf{x}_i]$ can be either true or false, and the factors of the two possible worlds add up to $\bar{w}(T_i) + w(T_i) = w(\Phi_i)$. Next, we use these two expressions to solve for $\bar{w}(T_i)$ and $w(T_i)$:

$$\bar{w}(T_i) = w(\Phi_i) - \bar{w}(\Phi_i) \qquad\qquad w(T_i) = \bar{w}(\Phi_i)$$

In other words, we have replaced the constraint (4) with a simpler constraint, without negations, by using more clever expressions for the weights. Notice that $\bar{w}(T_i)$ may be negative.

In fact, negation can be completely removed from the formula during lifted inference, through the following simple rewriting. For every relational symbol R that occurs negated in a formula Φ, create two new relational symbols of the same arities, N, T, replace all atoms of the form $\neg R(\mathbf{x})$ with $N(\mathbf{x})$ and add the following hard constraints:

$$\Gamma = \forall \mathbf{x}(R(\mathbf{x}) \vee N(\mathbf{x})) \wedge (R(\mathbf{x}) \vee T(\mathbf{x})) \wedge (N(\mathbf{x}) \vee T(\mathbf{x}))$$

with the following weights:

$$\bar{w}(N) = w(N) = w(T) = 1 \qquad\qquad \bar{w}(T) = -1$$

To see why this works, call a possible world "good" if, for every tuple $\mathbf{a}$, exactly one of the tuples $R(\mathbf{a})$ and $N(\mathbf{a})$ is true. In other words, in a good world N denotes $\neg R$. The hard constraint Γ further ensures that all tuples $T(\mathbf{a})$ are true in a good world, hence good worlds have the same weights in the original and in the new MLN. Consider now a "bad" world that satisfies Γ: then there exists a tuple $\mathbf{a}$ such that both $R(\mathbf{a})$ and $N(\mathbf{a})$ are true in that world. However, in that case $T(\mathbf{a})$ can be set to either true or false, and the weights of the two corresponding worlds cancel each other out. Therefore, the weights of all bad worlds cancel each other out.

Finally, another source of complexity in lifted inference comes from formulas that use a mixture of existential and universal variables. These can also be removed by using clever expressions for the weights. For example, consider a hard constraint:

$$\Phi = \forall x \exists y \varphi(x, y)$$

Create a new relational symbol $T(x)$, and replace Φ with the following constraint:

$$\Phi' = \forall x \forall y(T(x) \vee \neg\varphi(x, y))$$

and weights:

$$\bar{w}(T) = -1 \qquad\qquad w(T) = 1$$

We invite the reader to verify the correctness of this rewriting.

7 Open Problems

Many inference problems in AI are symmetric, where all tuples in a single relation have identical probabilities. As discussed in Sect. 4, the complexity of inference with symmetric probabilities is different (strictly easier) than in the asymmetric setting. In particular, many of the intractability results for probabilistic databases do not apply directly to the symmetric setting. It is of theoretical and practical interest to further characterize the complexity of queries with symmetric probabilities.

An open question is whether the inclusion/exclusion rule is necessary for lifted inference, or if some sequence of applications of the deterministic OR and other simpler rules is sufficient to compute any liftable query. Lifted inference rules for probabilistic databases use inclusion/exclusion, but in the AI community, lifted inference relies on deterministic OR. It is conjectured that inclusion/exclusion is strictly more powerful than deterministic OR: that is, there exist queries which cannot be computed in PTIME unless inclusion/exclusion is applied to identify intractable subqueries that cancel out and hence need not be computed. If the conjecture is correct, approaches that rely solely on deterministic OR can not be complete for lifted inference. However, deterministic OR is simpler to express; if it is equally as powerful, it may be preferred in practical systems.

As we have presented, there have been many recent advances in lifted inference techniques. There is a need for these new techniques to be incorporated into a robust, state of the art WFOMC tool for lifted inference that operates equally well in the database and AI settings. Effort in this direction is ongoing. For a prototype that implements many of the techniques discussed above, see `http://dtai.cs.kuleuven.be/wfomc`.

References

[1] H. Bui, T. Huynh, and S. Riedel. Automorphism groups of graphical models and lifted variational inference. In *Proceedings of UAI*, 2013.

[2] M. Chavira and A. Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6-7):772–799, Apr. 2008.

[3] M. Chavira, A. Darwiche, and M. Jaeger. Compiling relational Bayesian networks for exact inference. *International Journal of Approximate Reasoning*, 42(1-2):4–20, May 2006.

[4] N. Dalvi and D. Suciu. The dichotomy of probabilistic inference for unions of conjunctive queries. *Journal of the ACM (JACM)*, 59(6):30, 2012.

[5] A. Darwiche. A logical approach to factoring belief networks. *Proceedings of KR*, pages 409–420, 2002.

[6] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *KDD*, 2014.

[7] M. Dylla, I. Miliaraki, and M. Theobald. Top-k query processing in probabilistic databases with non-materialized views. In *ICDE*, pages 122–133, 2013.

[8] D. Fierens, G. Van den Broeck, I. Thon, B. Gutmann, and L. De Raedt. Inference in probabilistic logic programs using weighted CNF's. In *Proceedings of UAI*, pages 211–220, July 2011.

[9] V. Gogate and P. Domingos. Probabilistic theorem proving. In *UAI*, pages 256–265, 2011.

[10] V. Gogate, A. K. Jha, and D. Venugopal. Advances in lifted importance sampling. In *AAAI*, 2012.

[11] C. P. Gomes, A. Sabharwal, and B. Selman. Model counting. *Handbook of Satisfiability*, 185:633–654, 2009.

[12] E. Gribkoff, G. Van den Broeck, and D. Suciu. The most probable database problem. *Big Uncertain Data Workshop*, 2014.

[13] E. Gribkoff, G. Van den Broeck, and D. Suciu. Understanding the complexity of lifted inference and asymmetric weighted model counting. In *UAI*, pages 280–289, 2014.

[14] J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum. Yago2: A spatially and temporally enhanced knowledge base from wikipedia. *Artif. Intell.*, 194:28–61, 2013.

[15] A. K. Jha and D. Suciu. Probabilistic databases with markoviews. *PVLDB*, 5(11):1160–1171, 2012.

[16] K. Kersting, B. Ahmadi, and S. Natarajan. Counting belief propagation. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 277–284. AUAI Press, 2009.

[17] M. Mladenov, B. Ahmadi, and K. Kersting. Lifted linear programming. In *International Conference on Artificial Intelligence and Statistics*, pages 788–797, 2012.

[18] M. Niepert. Symmetry-aware marginal density estimation. In *AAAI*, 2013.

[19] F. Niu, C. Zhang, C. Ré, and J. W. Shavlik. Deepdive: Web-scale knowledge-base construction using statistical learning and inference. In *VLDS*, pages 25–28, 2012.

[20] D. Olteanu and H. Wen. Ranking query answers in probabilistic databases: Complexity and efficient algorithms. In *ICDE*, pages 282–293, 2012.

[21] D. Poole. First-order probabilistic inference. In *Proceedings of IJCAI*, pages 985–991, 2003.

[22] J. S. Provan and M. O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM Journal on Computing*, 12(4):777–788, 1983.

[23] M. Richardson and P. Domingos. Markov logic networks. *Machine learning*, 62(1-2):107–136, 2006.

[24] T. Sang, P. Beame, and H. Kautz. Solving Bayesian networks by weighted model counting. In *Proceedings of AAAI*, volume 1, pages 475–482, 2005.

[25] S. Sarkhel, D. Venugopal, P. Singla, and V. Gogate. Lifted map inference for markov logic networks. In *AISTATS*, pages 859–867, 2014.

[26] P. Sen, A. Deshpande, and L. Getoor. Bisimulation-based approximate lifted inference. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 496–505. AUAI Press, 2009.

[27] P. Singla and P. Domingos. Lifted first-order belief propagation. In *AAAI*, volume 8, pages 1094–1099, 2008.

[28] D. Suciu, D. Olteanu, C. Ré, and C. Koch. Probabilistic databases. *Synthesis Lectures on Data Management*, 3(2):1–180, 2011.

[29] L. G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

[30] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[31] G. Van den Broeck. On the completeness of first-order knowledge compilation for lifted probabilistic inference. In *NIPS*, pages 1386–1394, 2011.

[32] G. Van den Broeck. *Lifted Inference and Learning in Statistical Relational Models*. PhD thesis, KU Leuven, 2013.

[33] G. Van den Broeck, A. Choi, and A. Darwiche. Lifted relax, compensate and then recover: From approximate to exact lifted probabilistic inference. In *Proceedings of UAI*, 2012.

[34] G. Van den Broeck, W. Meert, and A. Darwiche. Skolemization for weighted first-order model counting. In *Proceedings of KR*, 2014.

[35] G. Van den Broeck, N. Taghipour, W. Meert, J. Davis, and L. De Raedt. Lifted probabilistic inference by first-order knowledge compilation. In *Proceedings of IJCAI*, pages 2178–2185, 2011.

[36] M. Y. Vardi. The complexity of relational query languages (extended abstract). In *STOC*, pages 137–146, 1982.

A Appendix

Below is the SQL code for running query Q_1 on the probabilistic database in Fig. 1. The SQL query below implements the plan in Fig. 2 and has been tested on PostgreSQL 9.2.3.

```
-- define an aggregate function to compute the product
create or replace function combine_prod (float, float) returns float as 'select $1 * $2' language SQL;
create or replace function final_prod (float) returns float as 'select $1' language SQL;
drop aggregate if exists prod (float);
create aggregate prod (float)
(  sfunc = combine_prod,
   stype = float,
   finalfunc = final_prod,
   initcond = '1.0'
);

-----------------------------------------------------
-- create/populate tables
create table Tweeter(name text, topic text, p float);
create table Follows(follower text, followee text, p float);

insert into Tweeter values('Alice', 'Transactions',   0.7);
insert into Tweeter values('Alice', 'SocialNetworks', 0.8);
insert into Tweeter values('Carol', 'SocialNetworks', 0.9);

insert into Follows values('Alice', 'J.Bieber', 0.3);
insert into Follows values('Alice', 'G.Keillor', 0.4);
insert into Follows values('Carol', 'J.Bieber', 0.5);


-----------------------------------------------------
-- run the query
with Temp as
  (select Follows.follower, 1.0-prod(1.0-p) as p
   from Follows
   group by Follows.follower)
select Tweeter.topic, 1.0-prod(1-Tweeter.p*Temp.p)
from Tweeter, Temp
where Tweeter.name=Temp.follower
group by Tweeter.topic;
```