Chapter 1

GRAPHICAL MODELS FOR UNCERTAIN DATA

Amol Deshpande
University of Maryland, College Park, MD
amol@cs.umd.edu

Lise Getoor
University of Maryland, College Park, MD
getoor@cs.umd.edu

Prithviraj Sen
University of Maryland, College Park, MD
sen@cs.umd.edu

Abstract Graphical models are a popular and well-studied framework for compact representation of a joint probability distribution over a large number of interdependent variables, and for efficient reasoning about such a distribution. They have been proven useful in a wide range of domains from natural language processing to computer vision to bioinformatics. In this chapter, we present an approach to using graphical models for managing and querying large-scale uncertain databases. We present a unified framework based on the concepts from graphical models that can model not only tuple-level and attribute-level uncertainties, but can also handle arbitrary correlations that may be present among the data; our framework can also naturally capture *shared correlations* where the same uncertainties and correlations occur repeatedly in the data. We develop an efficient strategy for query evaluation over such probabilistic databases by casting the query processing problem as an *inference* problem in an appropriately constructed graphical model, and present optimizations specific to probabilistic databases that enable efficient query evaluation. We conclude the chapter with a discussion of related and future work on these topics.

Keywords: Graphical models; probabilistic databases; inference; first-order probabilistic models.

1. Introduction

An increasing number of real-world applications are demanding support for managing, storing, and querying uncertain data in relational database systems. Examples include data integration [14], sensor network applications [22], information extraction systems [34], mobile object tracking systems [11] and others. Traditional relational database management systems are not suited for storing or querying uncertain data, or for reasoning about the uncertainty itself – commonly desired in these applications. As a result, numerous approaches have been proposed to handle uncertainty in databases over the years [32, 10, 24, 26, 4, 39, 11, 22, 14, 8, 6]. However, most of these approaches make simplistic and restrictive assumptions concerning the types of uncertainties that can be represented. In particular, many of the proposed models can only capture and reason about tuple-level existence uncertainties, and cannot be easily extended to handle uncertain attribute values which occur naturally in many domains. Second, they make highly restrictive independence assumptions and cannot easily model correlations among the tuples or attribute values.

Consider a simple car advertisement database (Figure 1.1) containing information regarding pre-owned cars for sale, culled from various sources on the Internet. By its very nature, the data in such a database contains various types of uncertainties that interact in complex ways. First off, we may have uncertainty about the validity of a tuple – older ads are likely to correspond to cars that have already been sold. We may represent such uncertainty by associating an *existence probability* (denoted $prob_e$) with each tuple. Second, many of the attribute values may not be known precisely. In some cases, we may have an explicit probability distribution over an attribute value instead (e.g. the *SellerID* attribute for Ad 103 in Figure 1.1(a)). More typically, we may have a joint probability distribution over the attributes, and the uncertainty in the attribute values for a specific tuple may be computed using the known attribute values for that tuple. Figure 1.1(d) shows such a joint probability distribution over the attributes *make, model* and *mpg*; this can then be used to compute a distribution over the *mpg* attribute for a specific tuple (given the tuple's *make* and/or *model* information). Finally, the data may exhibit complex attribute-level or tuple-level correlations. For instance, since the ads 101 and 102 are both entered by the same seller, their validity is expected to be highly correlated; such a correlation may be represented using a joint probability distribution as shown in Figure 1.1(c).

Many other application domains naturally produce correlated data as well [52]. For instance, data integration may result in relations containing duplicate tuples that refer to the same *entity*; such tuples must be modeled as *mutually exclusive* [10, 1]. Real-world datasets such as the Christmas Bird Count [16] naturally contain complex correlations among tuples. Data generated by sen-

Ad	SellerID	Date	Type	Model	mpg	Price	$prob_e$
101	201	1/1	Sedan	Civic(EX)	?	$6000	0.5
102	201	1/10	Sedan	Civic(DX)	?	$4000	0.45
103	{201: 0.6, 202: 0.4}	1/15	{Sedan: 0.3, Hybrid: 0.7}	Civic	?	$12000	0.8
104	202	1/1	Hybrid	Civic	?	$20000	0.2
105	202	1/1	Hybrid	Civic	?	$20000	0.2

(a) Advertisements

SellerID	Reputation
201	Shady
202	Good

(b) Sellers

Ad 101	Ad 102	*prob*
valid	valid	0.4
valid	invalid	0.1
invalid	valid	0.05
invalid	invalid	0.45

(c)

Type	Model	mpg	*prob*
Sedan	Civic(EX)	26	0.2
		28	0.6
		30	0.2
	Civic(DX)	32	0.1
		35	0.7
		37	0.2
	Civic	28	0.4
		35	0.6
Hybrid	Civic	45	0.4
		50	0.6

(d)

Figure 1.1. (a,b) A simple car advertisement database with two relations, one containing uncertain data; (c) A joint probability function (*factor*) that represents the correlation between the validity of two of the ads ($prob_e$ for the corresponding tuples in the *Advertisements* table can be computed from this); (d) A *shared* factor that captures the correlations between several attributes in *Advertisements* – this can be used to obtain a probability distribution over missing attribute values for any tuple.

sor networks is typically highly correlated, both in time and space [22]. Finally, data generated through the application of a machine learning technique (e.g. classification labels) typically exhibits complex correlation patterns. Furthermore, the problem of handling correlations among tuples arises naturally during query evaluation *even when one assumes that the base data tuples are independent*. In other words, the independence assumption is not closed under the relational operators, specifically *join* [26, 14].

In this chapter, we present a framework built on the foundations of probabilistic graphical models that allows us to uniformly handle uncertainties and correlations in the data, while keeping the basic probabilistic framework simple and intuitive. The salient features of our proposed framework are as follows:

- Our framework enables us to uniformly represent both tuple-level and attribute-level uncertainties and correlations through the use of *conditional probability distributions* and *joint probability factors*. Our proposed model is based on the commonly-used *possible world semantics* [26, 14], and as

a result, every relational algebra query has precise and clear semantics on uncertain data.

- Our framework can represent and exploit recurring correlation patterns (called *shared factors*) that are common in many application domains and are also manifested during the query evaluation process itself (Figure 1.1(d) shows one such shared factor).
- We show how to cast query evaluation on probabilistic databases as an *inference* problem in probabilistic graphical models and develop techniques for efficiently constructing such models during query processing. This equivalence not only aids in our understanding of query evaluation on uncertain databases, but also enables transparent technology transfer by allowing us to draw upon the prior work on inference in the probabilistic reasoning community. In fact several of the novel inference algorithms we develop for query evaluation over probabilistic databases are of interest to the probabilistic reasoning community as well.

Our focus in this chapter is on management of large-scale uncertain data using probabilistic graphical models. We differentiate this from the dual problem of casting *inference in probabilistic graphical models* as *query evaluation* in an appropriately designed database (where the conditional probability distributions are stored as database relations) [9]. We revisit this issue in Section 5, along with several other topics such as probabilistic relational models and the relationship between our approach and other probabilistic query evaluation approaches.

The rest of the paper is organized as follows. We begin with a brief overview of graphical models (Section 2); we focus on representation and inference, and refer the reader to several texts on machine learning [44, 13, 35, 27] for learning and other advanced issues. We then present our framework for representing uncertain data using these concepts (Section 3). Next we develop an approach to cast query processing in probabilistic databases as an inference problem, and present several techniques for efficient inference (Section 4). We conclude with a discussion of related topics such as probabilistic relational models, safe plans, and lineage-based approaches (Section 5).

2. Graphical Models: Overview

Probabilistic graphical models (PGMs) comprise a powerful class of approaches that enable us to compactly represent and efficiently reason about very large joint probability distributions [44, 13]. They provide a principled approach to dealing with the uncertainty in many application domains through the use of probability theory, while effectively coping with the computational and representational complexity through the use of graph theory. They have

been proven useful in a wide range of domains including natural language processing, computer vision, social networks, bioinformatics, code design, sensor networks, and unstructured data integration to name a few. Techniques from graphical models literature have also been applied to many topics directly of interest to the database community including information extraction, sensor data analysis, imprecise data representation and querying, selectivity estimation for query optimization, and data privacy.

At a high level, our goal is to efficiently represent and operate upon a joint distribution Pr over a set of random variables $\mathcal{X} = \{X_1, \ldots, X_n\}$. Even if these variables are binary-valued, a naive representation of the joint distribution requires the specification of 2^n numbers (the probabilities of the 2^n different assignments to the variables), which would be infeasible except for very small n. Fortunately, most real-world application domains exhibit a high degree of structure in this joint distribution that allows us to factor the representation of the distribution into modular components. More specifically, PGMs exploit *conditional independences* among the variables:

Definition 2.1. *Let* $\mathbf{X}$*,* $\mathbf{Y}$*, and* $\mathbf{Z}$ *be sets of random variables.* $\mathbf{X}$ *is* conditionally independent *of* $\mathbf{Y}$ *given* $\mathbf{Z}$ *(denoted* $\mathbf{X} \perp \mathbf{Y} | \mathbf{Z}$*) in distribution* $\Pr$ *if:*

$$\Pr(\mathbf{X} = \mathbf{x}, \mathbf{Y} = \mathbf{y} | \mathbf{Z} = \mathbf{z}) = \Pr(\mathbf{X} = \mathbf{x} | \mathbf{Z} = \mathbf{z}) \Pr(\mathbf{Y} = \mathbf{y} | \mathbf{Z} = \mathbf{z})$$

for all values $\mathbf{x} \in dom(\mathbf{X})$*,* $\mathbf{y} \in dom(\mathbf{Y})$ *and* $\mathbf{z} \in dom(\mathbf{Z})$*.*

A graphical model consists of two components: (1) A graph whose nodes are the random variables and whose edges connect variables that interact directly; variables that are not directly connected are conditionally independent given some combination of the other variables. (2) A set of small functions called *factors* each over a subset of the random variables.

Definition 2.2. *A* factor $f(\mathbf{X})$ *is a function over a (small) set of random variables* $\mathbf{X} = \{X_1, \ldots, X_k\}$ *such that* $f(\mathbf{x}) \geq 0 \ \forall \mathbf{x} \in dom(X_1) \times \ldots \times dom(X_k)$*.*

The set of factors that can be associated with a graphical model is constrained by the nature (undirected vs directed) and the structure of the graph as we will see later. Note that it is not required that $f(\mathbf{x})$ be ≤ 1; in other words, factors are not required to be (but can be) probability distributions.

Definition 2.3. *A* probabilistic graphical model *(PGM)* $\mathcal{P} = \langle \mathcal{F}, \mathcal{X} \rangle$ *defines a joint distribution over the set of random variables* $\mathcal{X}$ *via a set of factors* $\mathcal{F}$*, each defined over a subset of* $\mathcal{X}$*. Given a complete joint assignment* $\mathbf{x} \in dom(X_1) \times \cdots \times dom(X_n)$ *to the variables in* $\mathcal{X}$*, the joint distribution is defined by:*

$$\Pr(\mathbf{x}) = \frac{1}{\mathcal{Z}} \prod_{f \in \mathcal{F}} f(\mathbf{x}_f)$$

where $\mathbf{x}_f$ denotes the assignments restricted to the arguments of f and $\mathcal{Z} = \sum_{\mathbf{x}'} \prod_{f \in \mathcal{F}} f(\mathbf{x}'_f)$ is a normalization constant.

The power of graphical models comes from the graphical representation of factors that makes it easy to understand, reason about, and operate upon them. Depending on the nature of the interactions between the variables, there are two popular classes of graphical models, *Bayesian networks (directed models)*, and *Markov networks (undirected models)*. These differ in the family of probability distributions they can represent, the set of factorizations they allow, and the way in which the interactions are quantified along the edges. We discuss these briefly in turn.

Directed Graphical Models: Bayesian Networks

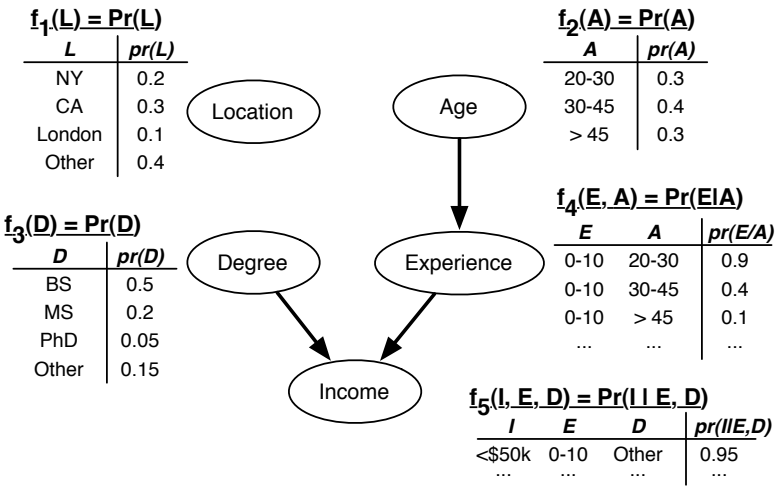
Directed graphical models, popularly known as Bayesian networks, are typically used to represent causal or asymmetric interactions amongst a set of random variables. A directed edge from variable X_i to variable X_j in the graph (which must be *acyclic*) is used to indicate that X_i directly influences X_j. A canonical set of conditional independences encoded by a directed graphical model is obtained as follows: a node X_j is independent of all its non-descendants given the values of its parents. In other words, if X_i is not a descendant or a parent of X_j, then $X_i \perp X_j | parents(X_j)$. The rest of the conditional independences encoded by the model can be derived from these.

The probability distribution that a directed graphical model represents can be factorized as follows:

$$\Pr(X_1, \ldots, X_n) = \prod_{i=1}^{n} \Pr(X_i | \text{parents}(X_i))$$

In other words, each of the factors associated with a Bayesian network is a conditional probability distribution (CPD) over a node given its parents in the graph.

Figure 1.2 shows a simple example Bayesian network that models the *location, age, degree, experience*, and *income* of a person. In this application domain, we might model the *location* to be independent from the rest of the variables (as captured by not having any edges to or from the corresponding node in the graph). For simplicity, we also model the *age* and *degree* to be independent from each other if no other information about the person is known. Although *income* is influenced by *degree, age*, and *experience*, in most cases, the influence from *age* will be indirect, and will disappear given the *experience* of the person; in other words, once the *experience* level of a person is known, the *age* does not provide any additional information about the *income*. This is modeled by not having any direct edge from *age* to *income*. The figure also shows the factors that will be associated with such a Bayesian network



$$Pr(L, A, D, E, I) = f_1(L)\ f_2(A)\ f_3(D)\ f_4(E, A)\ f_5(I, E, D))$$
$$= Pr(L)\ Pr(A)\ Pr(D)\ Pr(E \mid A)\ Pr(I \mid E, D))$$

Examples of conditional independences captured:

Location $\perp$ {Age, Degree, Experience, Income}

Degree $\perp$ {Age, Experience}

Income $\perp$ Age | Experience

Figure 1.2. Example of a directed model for a domain with 5 random variables

(one CPD each corresponding to each node), and the expression for the joint probability distribution as a product of the factors.

A domain expert typically chooses the edges to be added to the model, although the graph could also be learned from a training dataset. A sparse graph with few edges leads to more compact representation and (typically) more efficient inference, but a denser graph might be required to capture all the interactions between the variables faithfully.

The compactness of representing a joint probability distribution using a Bayesian network is evident from the above example. If each of the variables has domain of size 10, the size of the joint pdf will be 10^5, whereas the number of probabilities required to store the factors as shown in the figure is only about 1000, an order of magnitude reduction.

Since Bayesian networks are easy to design, interpret and reason about, they are extensively used in practice. Some popular examples of Bayesian networks include Hidden Markov Models [47, 56], Kalman Filters [37, 57], and QMR networks [40, 33].

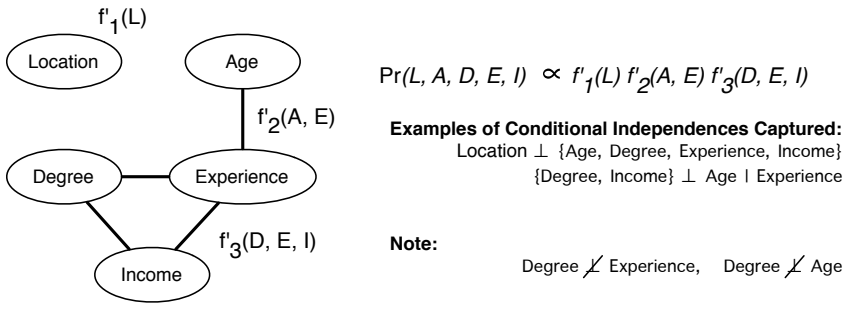


Figure 1.3. Example of an undirected model for a domain with 5 random variables

Undirected Graphical Models: Markov Networks

Undirected graphical models, or Markov Networks, are useful for representing distributions over variables where there is no natural directionality to the influence of one variable over another and where the interactions are more symmetric. Examples include the interactions between atoms in a molecular structure, the dependencies between the labels of pixels of an image, or the interactions between environmental properties sensed by geographically co-located sensors [22]. Markov networks are sometimes preferred over Bayesian networks because they provide a simpler model of independences between variables.

The probability distribution represented by a Markov network factorizes in a somewhat less intuitive manner than Bayesian networks; in many cases, the factors may only indicate the relative compatibility of different assignments to the variables, but may not have any straightforward probabilistic interpretation. Let G be the undirected graph over the random variables $\mathcal{X} = \{X_1, \ldots, X_n\}$ corresponding to a Markov network, and let $\mathcal{C}$ denote the set of cliques (complete subgraphs) of G. Then the probability distribution represented by the Markov network factorizes as follows:

$$\Pr(X_1, \ldots, X_n) = \frac{1}{Z} \prod_{C \in \mathcal{C}} f_C(X_C)$$

where $f_C(X_C)$ are the factors (also called *potential functions*) each over a complete subgraph of G. $Z = \sum_X \prod_{C \in \mathcal{C}} f_C(X_C)$ is the normalization constant.

Figure 1.3 shows an example Markov network over the same set of random variables as above. The maximal complete subgraphs of the network are $\{Location\}, \{Degree, Experience, Income\}, \{Age, Experience\}$ and factors may be defined over any of these sets of random variables, or their subsets.

The conditional independences captured by a Markov network are determined as follows: if a set of nodes $\mathbf{X}$ separates sets of nodes $\mathbf{Y}$ and $\mathbf{Z}$ (i.e., if by removing the nodes in $\mathbf{X}$, there are no paths between a node in $\mathbf{Y}$ and a node in $\mathbf{Z}$), then $\mathbf{Y}$ and $\mathbf{Z}$ are conditionally independent given $\mathbf{X}$. Figure 1.3 also shows the conditional independences captured by our example network.

An important subclass of undirected models is the class of *decomposable models* [20]. In a decomposable model, the graph is constrained to be *chordal* (*triangulated*) and the factors are the joint probability distributions over the maximal cliques of the graph. These types of models have many desirable properties such as closed product form factorizations that are easy to compute and reason about [21]. Further, these bear many similarities to the notion of *acyclic database schemas* [5].

Inference Queries

Next we consider the main types of tasks (queries) that are commonly performed over the model. The most common query type is the *conditional probability query*, $\Pr(\mathbf{Y} \mid \mathbf{E} = \mathbf{e})$. Such a query consists of two parts: (1) the *evidence*, a subset $\mathbf{E}$ of random variables in the network, and an instantiation $\mathbf{e}$ to these variables; and (2) the *query*, a subset $\mathbf{Y}$ of random variables in the network. Our task is to compute $\Pr(\mathbf{Y} \mid \mathbf{E} = \mathbf{e}) = \frac{\Pr(\mathbf{Y},\mathbf{e})}{\Pr(\mathbf{e})}$, i.e., the probability distribution over the values $\mathbf{y}$ of $\mathbf{Y}$, conditioned on the fact that $\mathbf{E} = \mathbf{e}$.

A special case of conditional probability queries is simply *marginal computation queries*, where we are asked to compute the marginal probability distribution $\Pr(\mathbf{Y})$ over a subset of variables $\mathbf{Y}$.

Another type of query that often arises, called *maximum a posteriori (MAP)*, is finding the most probable assignment to some subset of variables. As with conditional probability queries, we are usually given evidence $\mathbf{E} = \mathbf{e}$, and a set of query variables, $\mathbf{Y}$. In this case, however, our goal is to compute the *most likely assignment* to $\mathbf{Y}$ given the evidence $\mathbf{E} = \mathbf{e}$, i.e.:

$$argmax_{\mathbf{y}} \Pr(\mathbf{y}, \mathbf{e})$$

where, in general, $argmax_x f(x)$ represents the value of x for which $f(x)$ is maximal. Note that there might be more than one assignment that has the highest posterior probability. In this case, we can either decide that the MAP task is to return the set of possible assignments, or to return an arbitrary member of that set.

A special variant of this class of queries is the *most probable explanation (MPE)* queries. An MPE query tries to find the most likely assignment to all of the (non-evidence) variables, i.e., $\mathbf{Y} = \mathbf{X} - \mathbf{E}$. MPE queries are somewhat easier than MAP queries, which are much harder to answer than the other tasks; this is because MAP queries contain both summations and maximiza-

tions, thus combining the elements of both conditional probabilities queries and MPE queries.

The simplest way to use the graphical model to answer any of these queries is: (1) generate the joint probability distribution over all the variables, (2) condition it using the evidence (generating another joint pdf), and then (3) sum over the unneeded variables (in the case of a conditional probability query) or search for the most likely entry (in the case of an MPE query). For example, consider the example shown in Figure 1.2, and lets say we want to compute the marginal probability distribution corresponding to $income\ (I)$. This distribution can be obtained from the full joint distribution by summing out the rest of the variables:

$$\begin{aligned} \Pr(I) &= \Sigma_{L,A,D,E}\ \Pr(I,L,A,D,E) \\ &= \Sigma_{L,A,D,E}\ f_1(L)f_2(A)f_3(D)f_4(E,A)f_5(I,E,D) \end{aligned}$$

However, this approach is not very satisfactory and is likely to be infeasible in most cases, since it results in an exponential space and computational blowup that the graphical model representation was designed to avoid. In general, the exact computation of either of the inference tasks is #P-complete. However, many graphical models that arise in practice have certain properties that allow efficient probabilistic computation [59]. More specifically, the problem can be solved in polynomial time for graphical models with bounded tree-width [50].

Variable elimination (VE) [59, 19], also known as *bucket elimination*, is an exact inference algorithm that has the ability to exploit this structure. Intuitively variable elimination specifies the order in which the variables are summed out (eliminated) from the above expression; eliminating a variable requires multiplying all factors that contain the variable, and then summing out the variable. Say we chose the order: L, A, D, E, then the computation is as follows (the expression evaluated in each step is underlined, and its result is bold-faced in the next step):

$$\begin{aligned} \Pr(I) &= \Sigma_{L,A,D,E} f_1(L)f_2(A)f_3(D)f_4(E,A)f_5(I,E,D) \\ &= \Sigma_E(\Sigma_D f_5(I,E,D)f_3(D)\ \ (\Sigma_A f_2(A)f_4(E,A)\ \ \underline{(\Sigma_L f_1(L))})) \\ &= \Sigma_E(\Sigma_D f_5(I,E,D)f_3(D)\ \ \underline{(\Sigma_A f_2(A)f_4(E,A))}) \\ &= \Sigma_E \underline{(\Sigma_D f_5(I,E,D)f_3(D))}\ \mathbf{g_1}(\mathbf{E}) \\ &= \underline{\Sigma_E\ \mathbf{g_2}(\mathbf{I},\mathbf{E})g_1(E)} \\ &= \mathbf{g_3}(\mathbf{I}) \end{aligned}$$

The order in which the variables are summed out is known as the *elimination order*, and the cost of running VE depends on the choice of the elimination order. Even though finding the optimal ordering is NP-hard [2] (this is closely

related to the problem of finding the optimal triangulation of a graph), good heuristics are available [7, 18].

Another popular algorithm for exact inference is the *junction tree* algorithm [13, 30]. A junction tree is an efficient data structure for reusing work for several inference queries on the same graph. Once a junction tree is constructed, we can provide exact answers to inference queries over any subset of variables in the same clique by running the sum-product message passing or belief propagation algorithms. The message passing algorithm runs in time that is linear in the number of cliques in the tree and exponential in the size of the largest clique in the tree (which is same as the tree-width of the model).

However, many real-life graphical models give rise to graphs with large tree-widths, and the design of good approximation schemes in such cases is an active topic of research in the statistics and probabilistic reasoning communities. The most commonly used techniques include methods based on belief propagation (e.g. loopy belief propagation [42]), sampling-based techniques (e.g. Gibbs sampling, particle filters [3, 38]) and variational approximation methods [36] to name a few. We refer the reader to [35] for further details.

3. Representing Uncertainty using Graphical Models

We are now ready to define a probabilistic database in terms of a PGM. The basic idea is to use random variables to depict the uncertain attribute values and factors to represent the uncertainty and the correlations. Let R denote a probabilistic relation or simply, relation, and let $attr(R)$ denote the set of attributes of R. A relation R consists of a set of probabilistic tuples or simply, tuples, each of which is a mapping from $attr(R)$ to random variables. Let $t.a$ denote the random variable corresponding to tuple $t \in R$ and attribute $a \in attr(R)$. Besides mapping each attribute to a random variable, every tuple t is also associated with a boolean-valued random variable which captures the existence uncertainty of t and we denote this by $t.e$.

Definition 3.1. *A* probabilistic database *or simply, a* database*, $\mathcal{D}$ is a pair $\langle \mathcal{R}, \mathcal{P} \rangle$ where $\mathcal{R}$ is a set of relations and $\mathcal{P}$ denotes a PGM defined over the set of random variables associated with the tuples in $\mathcal{R}$.*

Figure 1.4(a) shows a small two-relation database that we use as a running example. In this database, every tuple has an uncertain attribute (the **B** attributes) and these are indicated in Figure 1.4(a) by specifying the probabilities with which each attribute takes the assignments from its domain. In our proposed framework, we represent this uncertainty by associating a random variable with each of the uncertain attributes, and by using factors to capture the corresponding probability distributions and correlations if present.

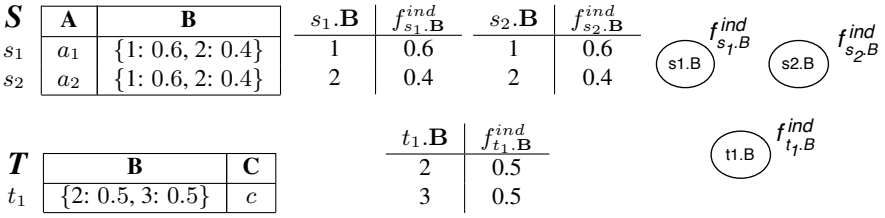


Figure 1.4. (a) A small database with uncertain attributes. For ease of exposition, we show the marginal pdfs over the attribute values in the table; this information can be derived from the factors. (b) Factors corresponding to the database assuming complete independence. (c) Graphical representation of the factors.

For instance, $s_2.\mathbf{B}$ can be assigned the value 1 with probability 0.6 and the value 2 with probability 0.4 and we would represent this using the factor $f_{s_2.\mathbf{B}}$ shown in Figure 1.4(b). We show all three required factors $f_{s_1.\mathbf{B}}(s_1.\mathbf{B})$, $f_{s_2.\mathbf{B}}(s_2.\mathbf{B})$ and $f_{t_1.\mathbf{B}}(t_1.\mathbf{B})$ in Figure 1.4(b). Here we assume that the attributes are independent of each other. If, for instance, $s_2.\mathbf{B}$ and $t_1.\mathbf{B}$ were correlated, we would capture that using a factor $f_{t_1.\mathbf{B},s_2.\mathbf{B}}(t_1.\mathbf{B}, s_2.\mathbf{B})$ (detailed example below).

In addition to the random variables which denote uncertain attribute values, we can introduce tuple *existence* random variables $s_1.e$, $s_2.e$, and $t_1.e$, to capture tuple uncertainty. These are boolean-valued random variables and can have associated factors. In Figure 1.4, we assume the tuples are certain, so we don't show the existence random variables for the base tuples.

Possible World Semantics

We now define the semantics for our formulation of a probabilistic database. Let $\mathcal{X}$ denote the set of random variables associated with database $\mathcal{D} = \langle \mathcal{R}, \mathcal{P} \rangle$. Possible world semantics define a probabilistic database $\mathcal{D}$ as a probability distribution over deterministic databases (possible worlds) [14] each of which is obtained by assigning $\mathcal{X}$ a joint assignment $\mathbf{x} \in \times_{X \in \mathcal{X}} dom(X)$. The probability associated with the possible world obtained from the joint assignment $\mathbf{x}$ is given by the distribution defined by the PGM $\mathcal{P}$ (Definition 2.3).

For the example shown in Figure 1.4, each possible world is obtained by assigning all three random variables $s_1.\mathbf{B}$, $s_2.\mathbf{B}$ and $t_1.\mathbf{B}$ assignments from their respective domains. Since each of the attributes can take 2 values, there are $2^3 = 8$ possible worlds. Figure 1.5 shows all 8 possible worlds with the corresponding probabilities listed under the column "prob.(ind.)" (indicating the independence assumption). The probability associated with each possible world is obtained by multiplying the appropriate numbers returned by the

possible world	prob (ind.)	prob. (implies)	prob. (diff.)	prob. (pos.corr.)
$D_1 : S = \{(a_1, 1), (a_2, 1)\}, T = \{(2, c)\}$	0.18	0.50	0.30	0.06
$D_2 : S = \{(a_1, 1), (a_2, 1)\}, T = \{(3, c)\}$	0.18	0.02	0.06	0.30
$D_3 : S = \{(a_1, 1), (a_2, 2)\}, T = \{(2, c)\}$	0.12	0	0.20	0.04
$D_4 : S = \{(a_1, 1), (a_2, 1)\}, T = \{(3, c)\}$	0.12	0.08	0.04	0.20
$D_5 : S = \{(a_1, 2), (a_2, 1)\}, T = \{(2, c)\}$	0.12	0	0	0.24
$D_6 : S = \{(a_1, 2), (a_2, 1)\}, T = \{(3, c)\}$	0.12	0.08	0.24	0
$D_7 : S = \{(a_1, 2), (a_2, 2)\}, T = \{(2, c)\}$	0.08	0	0	0.16
$D_8 : S = \{(a_1, 2), (a_2, 2)\}, T = \{(3, c)\}$	0.08	0.32	0.16	0

Figure 1.5. Possible worlds for example in Figure 1.4(a) and three other different types of correlations.

$$\overset{implies}{\Pr}(s_1.\mathbf{B}, s_2.\mathbf{B}, t_1.\mathbf{B}) = f^{implies}_{t_1.\mathbf{B}}(t_1.\mathbf{B}) f^{implies}_{t_1.\mathbf{B}, s_1.\mathbf{B}}(t_1.\mathbf{B}, s_1.\mathbf{B}) f^{implies}_{t_1.\mathbf{B}, s_2.\mathbf{B}}(t_1.\mathbf{B}, s_2.\mathbf{B})$$

$t_1.\mathbf{B}$	$f^{implies}_{t_1.\mathbf{B}}$
2	0.5
3	0.5

$t_1.\mathbf{B}$	$s_1.\mathbf{B}$	$f^{implies}_{t_1.\mathbf{B}, s_1.\mathbf{B}}$
2	1	1
2	2	0
3	1	0.2
3	2	0.8

$t_1.\mathbf{B}$	$s_2.\mathbf{B}$	$f^{implies}_{t_1.\mathbf{B}, s_2.\mathbf{B}}$
2	1	1
2	2	0
3	1	0.2
3	2	0.8

Figure 1.6. Factors for the probabilistic databases with "implies" correlations (we have omitted the normalization constant $\mathcal{Z}$ because the numbers are such that distribution is already normalized)

factors and normalizing if necessary. For instance, for the possible world obtained by the assignment $s_1.\mathbf{B} = 1$, $s_2.\mathbf{B} = 2$, $t_1.\mathbf{B} = 2$ (D_3 in Figure 1.5) the probability is $0.6 \times 0.4 \times 0.5 = 0.12$.

Let us now try to modify our example to illustrate how to represent correlations in a probabilistic database. In particular, we will try to construct three different databases containing the following dependencies:

- *implies*: $t_1.\mathbf{B} = 2$ implies $s_1.\mathbf{B} \neq 2$ and $s_2.\mathbf{B} \neq 2$, in other words, $(t_1.\mathbf{B} = 2) \implies (s_1.\mathbf{B} = 1) \wedge (s_2.\mathbf{B} = 1)$.
- *different*: $t_1\mathbf{B}$ and $s_1.\mathbf{B}$ cannot have the same assignment, in other words, $(t_1.\mathbf{B} = 2) \Leftrightarrow (s_1.\mathbf{B} = 1)$ or $(s_1.\mathbf{B} = 2) \Leftrightarrow (t_1.\mathbf{B} = 3)$.
- *positive correlation*: High positive correlation between $t_1.\mathbf{B}$ and $s_1.\mathbf{B}$ – if one is assigned 2 then the other is also assigned the same value with high probability.

Figure 1.5 shows four distributions over the possible worlds that each satisfy one of the above correlations (the columns are labeled with abbreviations of the names of the correlations, e.g., the column for positive correlation is labeled "pos. corr.").

To represent the possible worlds of our example database with the new correlations, we simply redefine the factors in the database appropriately. For example, Figure 1.6 represents the factors for the first case (*implies*). In this case, we use a factor on $t_1.B$ and $s_1.B$ to encode the correlation that $(t_1.\mathbf{B} = 2) \implies (s_1.\mathbf{B} = 1)$. Similarly, a factor on $t_1.B$ and $s_2.B$ is used to encode the other correlation.

Note that in Definition 3.1, we make no restrictions as to which random variables appear as arguments in a factor. Thus, if the user wishes, she may define a factor containing random variables from the same tuple, different tuples, tuples from different relations or tuple existence and attribute value random variables; thus, in our formulation we can express any kind of correlation that one might think of representing in a probabilistic database.

Shared Factors

In many cases, the uncertainty in the data is defined using general statistics that *do not* vary on a per-tuple basis, and this leads to significant duplication of factors in the probabilistic database. For instance, when combining data from different sources in a data integration scenario, the sources may be assigned data quality values, which may be translated into tuple existence probabilities [1]; all tuples from the same source are then expected to have the same factor associated with them. If the uncertainties are derived from an attribute-level joint probability distribution (as shown in our earlier example in Figure 1.1), then many of the factors are expected to be identical.

Another source of shared correlations in probabilistic databases is the query evaluation approach itself. As we will see in the next section, while evaluating queries we first build an augmented PGM on the fly by introducing small factors involving the base tuples and the intermediate tuples. For instance, if tuples t and t' join to produce intermediate tuple r, we introduce a factor that encodes the correlation that r exists iff both t and t exist (an $\wedge$-factor). More importantly, such a factor is introduced whenever any pair of tuples join, thus leading to repeated copies of the same $\wedge$-factor.

We call such factors *shared factors* and explicitly capture them in our framework; furthermore, our inference algorithms actively identify and exploit such commonalities to reduce the query processing time [53].

Representing Probabilistic Relations

Earlier approaches represented probabilistic relations by storing uncertainty with each tuple in isolation. This is inadequate for our purpose since the same tuple can be involved in multiple factors, and the same factor can be associated with different sets of random variables. This necessitates an approach where the data and the uncertainty parts are stored separately. Figure 1.7 shows how

id	A	B
s_1	a_1	$\perp$
s_2	a_2	$\perp$

id	B	C
t_1	$\perp$	c

(a) Base Tables

fid	args	probs
f_1	1	"2,0.5;3,0.5"
f_2	2	"2,1,1;2,2,0 ..."
f_3	2	"2,1,1;2,2,0 ..."

(b) *factors* table

RV	fid	pos
$t_1.B$	$f_1 = f^{implies}_{t_1.B}$	1
$t_1.B$	$f_2 = f^{implies}_{t_1.B,s_1.B}$	1
$s_1.B$	$f_2 = f^{implies}_{t_1.B,s_1.B}$	2
$t_1.B$	$f_3 = f^{implies}_{t_1.B,s_2.B}$	1
$s_1.B$	$f_3 = f^{implies}_{t_1.B,s_2.B}$	2

(c) *factor-rvs* table

Figure 1.7. Representing the factors from Figure 1.6 using a relational database; shared factors can be represented by using an additional level of indirection.

we store the factors and associate them with the tuples in our current prototype implementation. We use an internal *id* attribute for each relation that is automatically generated when tuples are inserted into the relation; this attribute is used to identify the random variables corresponding to a tuple uniquely. We use $\perp$ to indicate uncertain attribute values (Figure 1.7(a)). Two additional tables are used to store the factors and their associations with the tuple variables:

- *factors:* This table stores a serialized representation of the factor along with some auxiliary information such as the number of arguments.
- *factor-rvs:* This normalized relation stores the association between factors and random variables; the random variables can be of two types: (1) attribute value random variables (e.g. $t_1.B$), or (2) existence random variables (e.g. $t_1.e$). Each row in this table indicates the participation of a random variable in a factor. Since the table is normalized, we also need to store the "position" of the random variable in the factor.

Note that this schema does not exploit shared factors (factors f_2 and f_3 are identical in the above example); they can be easily handled by adding one additional table.

4. Query Evaluation over Uncertain Data

Having defined our representation scheme, we now move our discussion to query evaluation. The main advantage of associating possible world semantics with a probabilistic database is that it lends precise semantics to the query evaluation problem. Given a user-submitted query q (expressed in some standard query language such as relational algebra) and a database $\mathcal{D}$, the result of evaluating q against $\mathcal{D}$ is defined to be the set of results obtained by evaluating q against each possible world of $\mathcal{D}$, augmented with the probabilities of the possible worlds. Relating back to our earlier examples, suppose we want to run the query $q = \prod_{\mathbf{C}}(S \bowtie_{\mathbf{B}} T)$. Figure 1.8(a) shows the set of results obtained from each set of possible worlds, augmented by the corresponding probabilities depending on which database we ran the query against.

possible world	query result	prob. (ind.)	prob. (implies)	prob. (diff.)	prob. (pos.corr.)
D_1	$\emptyset$	0.18	0.50	0.30	0.06
D_2	$\emptyset$	0.18	0.02	0.06	0.30
D_3	$\{c\}$	0.12	0	0.20	0.04
D_4	$\emptyset$	0.12	0.08	0.04	0.20
D_5	$\{c\}$	0.12	0	0	0.24
D_6	$\emptyset$	0.12	0.08	0.24	0
D_7	$\{c\}$	0.08	0	0	0.16
D_8	$\emptyset$	0.08	0.32	0.16	0

(a)

query result	$\Pr(D_3) + \Pr(D_5) + \Pr(D_7)$			
	ind.	implies	diff.	pos.corr.
$\{c\}$	0.32	0	0.20	0.40

(b)

Figure 1.8. Results of running the query $\prod_{\mathbf{C}}(S \bowtie_{\mathbf{B}} T)$ on example probabilistic databases (Figures 1.4 and 1.5). The query returns a non-empty (and identical) result in possible worlds D_3, D_5, and D_7, and the final result probability is obtained by adding up the probabilities of those worlds.

Now, even though query evaluation under possible world semantics is clear and intuitive, it is typically not feasible to evaluate a query directly using these semantics. First and foremost among these issues is the size of the result. Since the number of possible worlds is exponential in the number of random variables in the database (to be more precise, it is equal to the product of the domain sizes of all random variables), if every possible world returns a different result, the result size itself will be very large. To get around this issue, it is traditional to compress the result before returning it to the user. One way of doing this is to collect all tuples from the set of results returned by possible world semantics and return these along with the sum of probabilities of the possible worlds that return the tuple as a result [14]. In Figure 1.8(a), there is only one tuple that is returned as a result and this tuple is returned by possible worlds D_3, D_5 and D_7. In Figure 1.8(b), we show the resulting probabilities obtained by summing across these three possible worlds for each example database.

The second issue is related to the complexity of computing the results of a query from these first principles. Since the number of possible worlds is very large for any non-trivial probabilistic database, evaluating results directly by enumerating all of its possible worlds is going to be infeasible.

To solve this problem, we first make the connection between computing query results for a probabilistic database and the marginal probability computation problem for probabilistic graphical models.

Definition 4.1. *Given a PGM $\mathcal{P} = \langle \mathcal{F}, \mathcal{X} \rangle$ and a random variable $X \in \mathcal{X}$, the* marginal probability *associated with the assignment $X = x$, where $x \in dom(X)$, is defined as $\mu(x) = \sum_{\mathbf{x} \sim x} \Pr(\mathbf{x})$, where $\Pr(\mathbf{x})$ denotes the distribution defined by the PGM and $\mathbf{x} \sim x$ denotes a joint assignment to $\mathcal{X}$ where X is assigned x.*

Since each possible world is obtained by a joint assignment to all random variables in the probabilistic database, there is an intuitive connection between computing marginal probabilities and computing result tuple probabilities by summing over all possible worlds. In the rest of this section, we make this connection more precise. We first show how to augment the PGM underlying the database such that the augmented PGM contains random variables representing result tuples. We can then express the probability computation associated with evaluating the query as a standard marginal probability computation problem; this allows us to use standard probabilistic inference algorithms to evaluate queries over probabilistic databases.

We first present an example to illustrate the basic ideas underlying our approach to augmenting the PGM underlying the database given a query, after that we discuss how to augment the PGM in the general case given any relational algebra query.

Example

Consider running the query $\prod_{\mathbf{C}}(S \bowtie_{\mathbf{B}} T)$ on the database presented in Figure 1.4(a). Our query evaluation approach is very similar to query evaluation in traditional database systems and is depicted in Figure 1.9. Just as in traditional database query processing, in Figure 1.9, we introduce intermediates tuples produced by the join (i_1 and i_2) and produce a result tuple (r_1) from the projection operation. What makes query processing for probabilistic databases different from traditional database query processing is the fact that we need to preserve the correlations among the random variables representing the intermediate and result tuples and the random variables representing the tuples they were produced from. In our example, there are three such correlations that we need to take care of:

- i_1 (produced by the join between s_1 and t_1) exists or $i_1.e$ is `true` only in those possible worlds where both $s_1.\mathbf{B}$ and $t_1.\mathbf{B}$ are assigned the value 2.
- Similarly, $i_2.e$ is `true` only in those possible worlds where both $s_2.\mathbf{B}$ and $t_1.\mathbf{B}$ are assigned the value 2.
- Finally, r_1 (the result tuple produced by the projection) exists or $r_1.e$ is `true` only in those possible worlds that produce at least one of i_1 or i_2 or both.

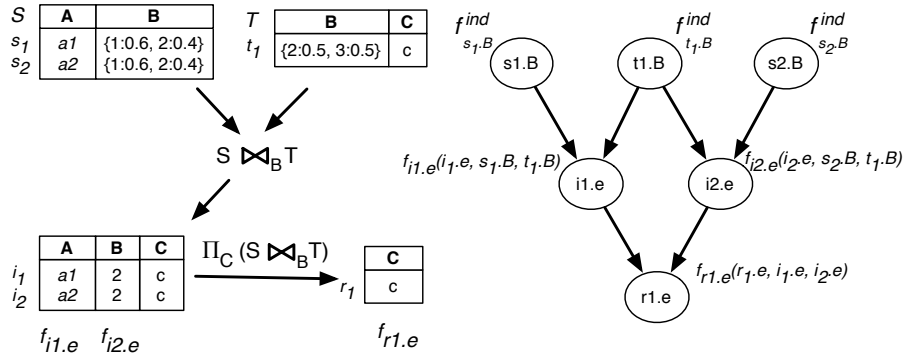


Figure 1.9. Evaluating $\prod_{\mathbf{C}}(S \bowtie_{\mathbf{B}} T)$ on the database from Figure 1.4(a).

To enforce these correlations, during query evaluation we introduce intermediate factors defined over appropriate random variables. For our example, we introduce the following three correlations:

- For the correlation among $i_1.e$, $s_1.\mathbf{B}$ and $t_1.\mathbf{B}$ we introduce the factor $f_{i_1.e}$ which is defined as:

$$f_{i_1.e}(i_1.e, s_1.\mathbf{B}, t_1.\mathbf{B}) = \begin{cases} 1 & \text{if } i_1.e \Leftrightarrow ((s_1.\mathbf{B} == 2) \wedge (t_1.\mathbf{B} == 2)) \\ 0 & \text{otherwise} \end{cases}$$

- Similarly, for the correlation among $i_2.e$, $s_2.\mathbf{B}$ and $t_1.\mathbf{B}$ we introduce the factor $f_{i_2.e}$ which is defined as:

$$f_{i_2.e}(i_2.e, s_2.\mathbf{B}, t_1.\mathbf{B}) = \begin{cases} 1 & \text{if } i_2.e \Leftrightarrow ((s_2.\mathbf{B} == 2) \wedge (t_1.\mathbf{B} == 2)) \\ 0 & \text{otherwise} \end{cases}$$

- For the correlation among $r_1.e$, $i_1.e$ and $i_2.e$, we introduce a factor $f_{r_1.e}$ capturing the `or` semantics:

$$f_{r_1.e}(r_1.e, i_1.e, i_2.e) = \begin{cases} 1 & \text{if } r_1.e \Leftrightarrow (i_1.e \vee i_2.e) \\ 0 & \text{otherwise} \end{cases}$$

Figure 1.9 depicts the full run of the query along with the introduced factors.

Now, to compute the probability of existence of r_1 (which is what we did in Figure 1.8 by enumerating over all possible worlds), we simply need to compute the marginal probability associated with the assignment $r_1.e = \texttt{true}$ from PGM formed by the set of factors in the base data and the factors introduced during query evaluation. For instance, for the example where we assumed complete independence among all uncertain attribute values (Figure 1.4(b)) our augmented PGM is given by the collection $f_{s_1.\mathbf{B}}, f_{s_2.\mathbf{B}}, f_{t_1.\mathbf{B}}, f_{i_1.e}, f_{i_2.e}$

and $f_{r_1.e}$, and to compute the marginal probability we can simply use any of the exact inference algorithms available in the probabilistic reasoning literature such as variable elimination [59, 19] or the junction tree algorithm [30].

Generating Factors during Query Evaluation

Query evaluation for general relational algebra also follows the same basic ideas. In what follows, we modify the traditional relational algebra operators so that they not only generate intermediate tuples but also introduce intermediate factors which, combined with the factors on the base data, provide a PGM that can then be used to compute marginal probabilities of the random variables associated with result tuples of interest. We next describe the modified σ, $\times$, $\prod$, δ, $\cup$, $-$ and γ (aggregation) operators where we use $\emptyset$ to denote a special "null" symbol.

Select: Let $\sigma_c(R)$ denote the query we are interested in, where c denotes the predicate of the select operation. Every tuple $t \in R$ can be jointly instantiated with values from $\times_{a \in attr(R)} dom(t.a)$. If none of these instantiations satisfy c then t does not give rise to any result tuple. If even a single instantiation satisfies c, then we generate an intermediate tuple r that maps attributes from R to random variables, besides being associated with a tuple existence random variable $r.e$. We then introduce factors encoding the correlations among the random variables for r and the random variables for t. The first factor we introduce is $f^{\sigma}_{r.e}$, which encodes the correlations for $r.e$:

$$f^{\sigma}_{r.e}(r.e, t.e, \{t.a\}_{a \in attr(R)}) = \begin{cases} 1 & \text{if } t.e \wedge c(\{t.a\}_{a \in attr(R)}) \Leftrightarrow r.e \\ 0 & \text{otherwise} \end{cases}$$

where $c(\{t.a\}_{a \in attrR})$ is `true` if a joint assignment to the attribute value random variables of t satisfies the predicate c and `false` otherwise.

We also introduce a factor for $r.a$, $\forall a \in attr(R)$ (where $dom(r.A) = dom(t.A)$), denoted by $f^{\sigma}_{r.a}$. $f^{\sigma}_{r.a}$ takes $t.a, r.e$ and $r.a$ as arguments and can be defined as:

$$f^{\sigma}_{r.a}(r.a, r.e, t.a) = \begin{cases} 1 & \text{if } r.e \wedge (t.a == r.a) \\ 1 & \text{if } \overline{r.e} \wedge (r.a == \emptyset) \\ 0 & \text{otherwise} \end{cases}$$

Cartesian Product: Suppose R_1 and R_2 are the two relations involved in the Cartesian product operation. Let r denote the join result of two tuples $t_1 \in R_1$ and $t_2 \in R_2$. Thus r maps every attribute from $attr(R_1) \cup attr(R_2)$ to a random variable, besides being associated with a tuple existence random variable $r.e$. The factor for $r.e$, denoted by $f^{\times}_{r.e}$, takes $t_1.e$, $t_2.e$ and $r.e$ as

arguments, and is defined as:

$$f^{\times}_{r.e}(r.e, t_1.e, t_2.e) = \begin{cases} 1 & \text{if } t_1.e \wedge t_2.e \Leftrightarrow r.e \\ 0 & \text{otherwise} \end{cases}$$

We also introduce a factor $f^{\times}_{r.a}$ for each $a \in attr(R_1) \cup attr(R_2)$, and this is defined exactly in the same fashion as $f^{\sigma}_{r.a}$. Basically, for $a \in attr(R_1)$ ($a \in attr(R_2)$), it returns 1 if $r.e \wedge (t_1.a == r.a)$ ($r.e \wedge (t_2.a == r.a)$) holds or if $\overline{r.e} \wedge (r.a == \emptyset)$ holds, and 0 otherwise.

Project (without duplicate elimination): Let $\prod_{\mathbf{a}}(R)$ denote the operation we are interested in where $\mathbf{a} \subseteq attr(R)$ denotes the set of attributes we want to project onto. Let r denote the result of projecting $t \in R$. Thus r maps each attribute $a \in \mathbf{a}$ to a random variable, besides being associated with $r.e$. The factor for $r.e$, denoted by $f^{\prod}_{r.e}$, takes $t.e$ and $r.e$ as arguments, and is defined as follows:

$$f^{\prod}_{r.e}(r.e, t.e) = \begin{cases} 1 & \text{if } t.e \Leftrightarrow r.e \\ 0 & \text{otherwise} \end{cases}$$

Each factor $f^{\prod}_{r.a}$, introduced for $r.a,\ \forall a \in \mathbf{a}$, is defined exactly as $f^{\sigma}_{r.a}$, in other words, $f^{\prod}_{r.a}(r.a, r.e, t.a) = f^{\sigma}_{r.a}(r.a, r.e, t.a)$.

Duplicate Elimination: Duplicate elimination is a slightly more complex operation because it can give rise to multiple intermediate tuples even if there was only one input tuple to begin with. Let R denote the relation from which we want to eliminate duplicates, then the resulting relation after duplicate elimination will contain tuples whose existence is uncertain, more precisely the resulting tuples' attribute values are known. Any element from $\bigcup_{t \in R} \times_{a \in attr(R)} dom(t.a)$ may correspond to the values of a possible result tuple. Let r denote any such result tuple whose attribute values are known, only $r.e$ is not `true` with certainty. Denote by r_a the value of attribute a in r. We only need to introduce the factor $f^{\delta}_{r.e}$ for $r.e$. To do this we compute the set of tuples from R that may give rise to r. Any tuple t that satisfies $\bigwedge_{a \in attr(R)}(r_a \in dom(t.a))$ may give rise to r. Let y^r_t be an intermediate random variable with $dom(y^r_t) = \{\texttt{true}, \texttt{false}\}$ such that y^r_t is `true` iff t gives rise to r and `false` otherwise. This is easily done by introducing a factor $f^{\delta}_{y^r_t}$ that takes $\{t.a\}_{a \in attr(R)}$, $t.e$ and y^r_t as arguments and is defined as:

$$f^{\delta}_{y^r_t}(y^r_t, \{t.a\}_{a \in attr(R)}, t.e) = \begin{cases} 1 & \text{if } t.e \wedge \bigwedge_a (t.a == r_a) \Leftrightarrow y^r_t \\ 0 & \text{otherwise} \end{cases}$$

where $\{t.a\}_{a \in attr(R)}$ denotes all attribute value random variables of t. We can then define $f^{\delta}_{r.e}$ in terms of y^r_t. $f^{\delta}_{r.e}$ takes as arguments $\{y^r_t\}_{t \in T_r}$, where

T_r denotes the set of tuples that may give rise to r (contains the assignment $\{r_a\}_{a \in attr(R)}$ in its joint domain), and $r.e$, and is defined as:

$$f^{\delta}_{r.e}(r.e, \{y^r_t\}_{t \in T_r}) = \begin{cases} 1 & \text{if } \bigvee_{t \in T_r} y^r_t \Leftrightarrow r.e \\ 0 & \text{otherwise} \end{cases}$$

Union and set difference: These operators require set semantics. Let R_1 and R_2 denote the relations on which we want to apply one of these two operators, either $R_1 \cup R_2$ or $R_1 - R_2$. We will assume that both R_1 and R_2 are sets of tuples such that every tuple contained in them have their attribute values fixed and the only uncertainty associated with these tuples are with their existence (if not then we can apply a δ operation to convert them to this form). Now, consider result tuple r and sets of tuples T^1_r, containing all tuples from R_1 that match r's attribute values, and T^2_r, containing all tuples from R_2 that match r's attribute values. The required factors for $r.e$ can now be defined as follows:

$$f^{\cup}_{r.e}(r.e, \{t_1.e\}_{t_1 \in T^1_r}, \{t_2.e\}_{t_2 \in T^2_r}) = \begin{cases} 1 & \text{if } (\bigvee_{t \in T^1_r \cup T^2_r} t.e) \Leftrightarrow r.e \\ 0 & \text{otherwise} \end{cases}$$

$$f^{-}_{r.e}(r.e, \{t_1.e\}_{t_1 \in T^1_r}, \{t_2.e\}_{t_2 \in T^2_r}) = \begin{cases} 1 & \text{if } ((\bigvee_{t \in T^1_r} t.e) \wedge \neg(\bigvee_{t \in T^2_r} t.e)) \Leftrightarrow r.e \\ 0 & \text{otherwise} \end{cases}$$

Aggregation operators: Aggregation operators are also easily handled using factors. Suppose we want to compute the `sum` aggregate on attribute a of relation R, then we simply define a random variable $r.a$ for the result and introduce a factor that takes as arguments $\{t.a\}_{t \in attr(R)}$ and $r.a$, and define the factor so that it returns 1 if $r.a == (\sum_{t \in R} t.a)$ and 0 otherwise. Thus for any aggregate operator γ and result tuple random variable $r.a$, we can define the following factor:

$$f^{\gamma}_{r.a}(r.a, \{t.a\}_{t \in R}) = \begin{cases} 1 & \text{if } r.a == \gamma_{t \in R} t.a \\ 1 & \text{if } (r.a == \emptyset) \Leftrightarrow \bigwedge_{t \in R} (t.a == \emptyset) \\ 0 & \text{otherwise} \end{cases}$$

Query Evaluation as Inference

Given a query and a probabilistic database (and the corresponding PGM), we can use the procedures described in the previous section to construct an augmented PGM that contains random variables corresponding to the result tuples. Computing the result probabilities is simply a matter of evaluating

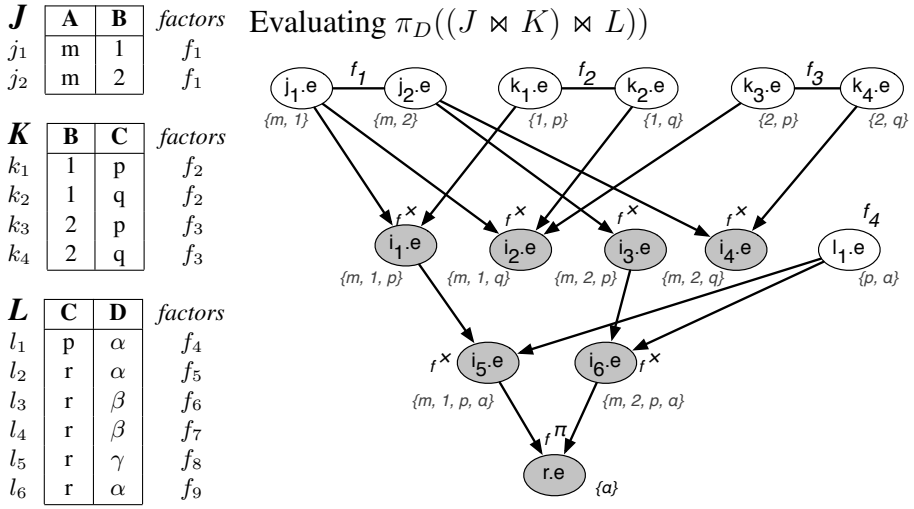


Figure 1.10. An example query evaluation over a 3-relation database with only tuple uncertainty but many correlations (tuples associated with the same factor are correlated with each other). The intermediate tuples are shown alongside the corresponding random variables. Tuples $l_2, \ldots, l_6$ do not participate in the query.

marginal probability queries over this PGM. We can use any standard exact or approximate inference algorithm developed in the probabilistic reasoning community for this purpose, depending on our requirements of accuracy and speed. Note that the resulting PGM, and hence the complexity of inference, will depend on the query plan used for executing the query. We revisit this issue in Section 5.

Figure 1.10 shows the PGM generated when evaluating a multi-way join query over 3 relations; computing the result tuple probability is equivalent to computing the marginal probability distribution over the random variable $r.e$. Similarly, Figure 1.11 shows the PGM constructed in response to an aggregate query (details below).

Optimizations

For the above operator modifications, we have attempted to be completely general and hence the factors introduced may look slightly more complicated than need be. For example, it is not necessary that $f^{\sigma}_{r.E}$ take as arguments all random variables $\{t.a\}_{a \in attr(R)}$ (as defined above), it only needs to take those $t.a$ random variables as arguments which are involved in the predicate c of the σ operation. Also, given a theta-join we do not need to implement this as a Cartesian product followed by a select operation. It is straightforward to

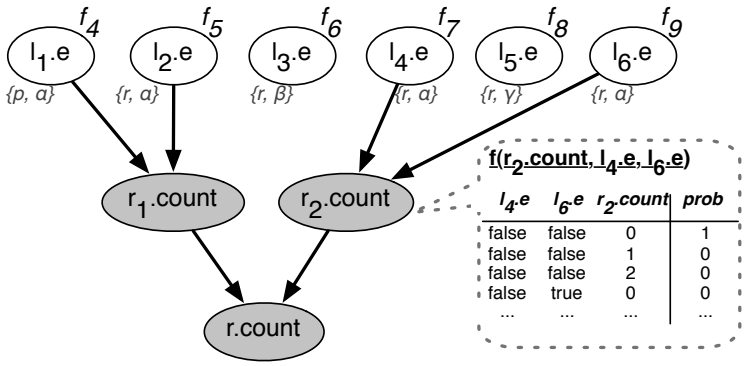


Figure 1.11. PGM constructed during the evaluation of ${}_{count}G(\sigma_{D=\alpha}(L))$ over the probabilistic database from Figure 1.10. By exploiting decomposability of *count*, we can limit the maximum size of the newly introduced factors to 3 (the naive implementation would have constructed a 5-variable factor).

push the select operation into the Cartesian product factors and implement the theta-join directly by modifying $f^{\times}_{r.E}$ appropriately using c.

Another type of optimization that is extremely useful for aggregate computation, duplicate elimination and the set-theoretic operations ($\cup$ and $-$) is to exploit decomposable functions. A decomposable function is one whose result does not depend on the order in which the inputs are presented to it. For instance, $\vee$ is a decomposable function, and so are most of the aggregation operators including `sum`, `count`, `max` and `min`. The problem with some of the redefined relational algebra operators is that, if implemented naively, they may lead to large intermediate factors. For instance, while running a δ operation, if T_r contains n tuples for some r, then the factor $f^{\delta}_{r.e}$ will be of size 2^{n+1}. By exploiting decomposability of $\vee$ we can implement the same factor using a linear number of constant sized (3-argument) factors which may lead to significant speedups. We refer the interested reader to [50, 60] for more details. The only aggregation operator that is not decomposable is `avg`, but even in this case we can exploit the same ideas by implementing `avg` in terms of `sum` and `count` both of which are decomposable. Figure 1.11 shows the PGM constructed for an example aggregate query over the database from Figure 1.10.

Finally, one of the key ways we can reduce the complexity of query evaluation is by exploiting recurring (shared) factors. In recent work [53], we developed a general-purpose inference algorithm that can exploit such shared factors. Our algorithm identifies and exploits the symmetry present in the augmented PGM to significantly speed up query evaluation in most cases. We

omit the details due to space constraints and refer the reader to [53] for further details.

5. Related Work and Discussion

Next we briefly discuss some of the closely related concepts in query evaluation over probabilistic databases, namely *safe plans* and *lineage*. We then briefly discuss the relationship of our approach to probabilistic relational models, lifted inference, and scalable inference using databases. We believe most of these represent rich opportunities for future research.

Safe Plans

One of the key results in query evaluation over probabilistic databases is the dichotomy of conjunctive query evaluation on tuple-independent probabilistic databases by Dalvi and Suciu [14, 15]. Briefly the result states that the complexity of evaluating a conjunctive query over tuple-independent probabilistic databases is either PTIME or #P-complete. For the former case, Dalvi and Suciu [14] also present an algorithm to find what are called *safe query plans*, that permit correct *extensional* evaluation of the query. We relate the notion of safe plans to our approach through the following theorem:

Theorem 5.1. *When executing a query over a tuple-independent probabilistic database using a safe query plan, the resulting probabilistic graphical model is tree-structured (for which inference can be done in PTIME).*

Note that the dichotomy result presented in [15] reflects a worst-case scenario over all possible instances of a probabilistic database. In other words, even if a query does not have safe plan, for a specific probabilistic database instance, query evaluation may still be reasonably efficient. Our approach can easily capture this because in such cases the resulting PGM will either be tree-structured or have low tree-width, thus allowing us to execute the query efficiently. One of the important open problems in this area is developing algorithms for identifying query plans that result in PGMs with low tree-widths for a given probabilistic database and a given query.

Representing Uncertainty using Lineage

Several works [26, 58, 6, 48, 49] have proposed using explicit *boolean* formulas to capture the relationship between the base tuples and the intermediate tuples. In the Trio system [58, 6], such formulas are called *lineage*, and are computed during the query evaluation. The result tuple probabilities are then computed on demand by evaluating the lineage formulas. In recent work, Re et al. [49] presented techniques for approximate compression of such lineage formulas for more efficient storage and query evaluation.

The PGM constructed in our approach can be thought of as a generalization of such boolean formulas, since the PGM can represent more complex relationships than can be captured using a boolean formula. Further, the PGM naturally captures common subexpressions between the lineage formulas corresponding to different result tuples, and avoids re-computation during the inference process. Evaluation of boolean formulas can be seen as a special case of probabilistic inference, and thus techniques from exact or approximate inference literature can be directly applied to evaluating the lineage formulas as well. However lineage formula evaluation admits efficient approximation schemes (e.g. based on polynomial approximation [49]) that may not translate to general probabilistic graphical models.

Probabilistic Relational Models

Probabilistic relational models (PRMs) [25, 27] extend Bayesian networks with the concepts of objects, their properties and relations between them. In a way, they are to Bayesian networks as relational logic is to propositional logic. PRMs can also be thought of as a generalization of the probabilistic database framework that we presented in this chapter, and extending our approach to transparently and efficiently handle a PRM-based model is one of the important research directions that we plan to pursue in future. We begin with illustrating PRMs using a simple example, and then discuss the challenges in integrating them with our approach.

A PRM contains a relational component that describes the relational schema of the domain, and a probabilistic component that captures the probabilistic dependencies that hold in the domain. Figure 1.12 shows a simple example PRM over a relational schema containing three relations, *Author*, *Paper*, and *Review*. For simplicity the relationship *AuthorOf* is modeled as many-to-one (with a single author per paper), whereas the relationship *Reviewed* is many-to-many. Along with the relational schema, a PRM specifies a probabilistic model over the attributes of the relations. A key difference between Bayesian networks and PRMs is that an attribute in one relation may depend on an attribute in another relation. For example, the *quality* of a *paper* may depend on the properties of the *author* (as shown in the figure).

When defining a dependence across a many-to-one relationship, a mechanism to aggregate the attribute values must be specified as well. For instance, the *accepted* attribute for a paper is modeled as dependent on the *mood* attribute from the review relation. However a single paper may have multiple reviews, and we must somehow combine the values of *mood* attribute from those reviews; the example PRM uses the *MODE* of the attribute values for this purpose.

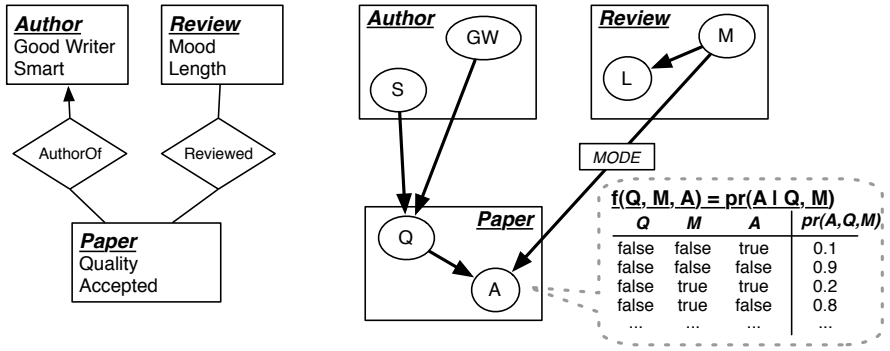


Figure 1.12. A probabilistic relational model defined over an example relational schema. Similar to Bayesian networks, the model parameters consist of conditional probability distributions for each node given its parents.

Now, given a relational skeleton that specifies the primary keys and foreign keys for the tuples, the PRM defines a probability distribution over the attributes of the tuples. Figure 1.13 shows an example of this, with two papers with keys *P1* and *P2*, both by the author *A1*. The PRM then specifies a joint probability distribution over the random variables as shown in the figure. If the skeleton also specifies the values of some of the attributes, those can be treated as evidence in a straightforward way.

PRMs can also capture uncertainty in the link structure (i.e., the key-foreign key dependencies). We refer the reader to [27] for more details.

Conceptually it is straightforward to extend our probabilistic model to allow the dependences to be defined using a PRM (shared factors is one step in that direction); the real challenge is doing inference over such models (see below). We are planning to explore closer integration between these two areas in the future.

Lifted Inference

Many first-order machine learning models such as PRMs allow defining rich, compact probability distributions over large collections of random variables. Inference over such models can be tricky, and the initial approaches to inference involved *grounding* out the graphical model by explicitly creating random variables (as shown in Figure 1.13) and then using standard inference algorithms. This can however result in very large graphical models, and can involve much redundant inference (since most of the factors are shared). Lifted inference techniques aim to address this situation by avoiding propositionalization (grounding) as much as possible [45, 46, 17, 55, 41, 53]. Most of

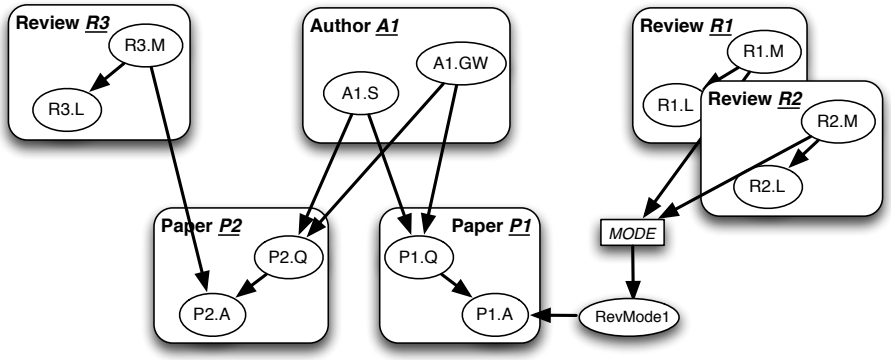


Figure 1.13. An instance of the example PRM with two papers: $P1$, $P2$, with the same author $A1$. For $P1$, we use an explicit random variable for representing the mode of $R1.M$ and $R2.M$. No such variable is needed for $P2$ since it only has one review.

this work assumes that the input is a first-order probabilistic model (such as a PRM). Poole [46] presents a modified version of the variable elimination algorithm [59] for this purpose. Braz et al. [17] and Milch et al. [41] present algorithms that look for specific types of structures in the first-order model, and exploit these for efficient inference. Singla et al. [55] develop a modified loopy belief propagation algorithm (for approximate inference) for lifted inference.

As discussed above, in our recent work [53], we developed a general-purpose lifted inference algorithm for probabilistic query evaluation. Our algorithm however does not operate on the first-order representation, and we are currently working on combining our approach with the techniques developed in the lifted inference literature.

Scalable Inference using a Relational Database

Finally a very related but at the same time fundamentally different problem is that of expressing inference tasks as database queries. Consider the Bayesian network shown in Figure 1.2, and consider the (inference) task of finding the marginal probability distribution over $income\ (I)$. As seen before, this can be written as:

$$\Pr(I) = \Sigma_{L,A,D,E}\ \ f_1(L)f_2(A)f_3(D)f_4(E,A)f_5(I,E,D)$$

If the factors (CPDs) become very large, we might choose to store them as relations in a database (called *functional relations* by Bravo et al. [9]). For example, the relations corresponding to f_1 and f_5 may have schemas *F1(L, prob)*, and *F5(I, E, D, prob)* respectively. Then this inference task can be written as an SQL query as follows:

```
select I, sum(F1.prob * F2.prob * F3.prob * F4.prob * F5.prob)
from F1 join F2 join F3 join F4 join F5
group by I
```

This approach not only enables easy and persistent maintenance of Bayesian networks, but can also enable significant performance optimizations (we refer the reader to Bravo et al. [9] for a more detailed discussion).

However note that this approach is only suitable when the number of random variables is small (i.e. the size of the network is small), since each factor must be stored as a separate relation. The number of uncertain facts in a probabilistic database is likely to be very large and continuously changing, and storing each factor as a different relation would be infeasible in those cases. Second, the main "query/inference" tasks that need to be supported in the two scenarios are quite different. In probabilistic databases, the SQL queries operate on the values of the random variables, concatenating or aggregating them, whereas inference in Bayesian networks is typically concerned with marginalization and conditioning. Supporting both types of tasks in a unified manner remains one of the most important open problems in this area.

6. Conclusions

Graphical models are a versatile tool that have been applied to many database problems such as selectivity estimation [28, 21, 43, 31], sensor network data management [23], information extraction [12, 51], data integration [54, 29] to name a few. In this chapter, we presented a simple and intuitive framework for managing large-scale uncertain data using graphical models, that allows us to capture complex uncertainties and correlations in the data in a uniform manner. We showed how the problem of query evaluation in uncertain databases can be seen to be equivalent to probabilistic inference in an appropriately constructed graphical model. This equivalence enables us to employ the formidable machinery developed in the probabilistic reasoning literature over the years for answering queries over probabilistic databases. We believe it will also lead to a deeper understanding of how to devise more efficient inference algorithms for large-scale, structured probabilistic models.

Acknowledgments

This work was supported in part by the National Science Foundation under Grants No. 0438866 and 0546136. We thank Sunita Sarawagi who co-presented a tutorial on graphical models at VLDB 2007 with one of the authors, and Brian Milch for stimulating discussions regarding lifted inference.

References

[1] Periklis Andritsos, Ariel Fuxman, and Renee J. Miller. Clean answers over dirty databases. In *International Conference on Data Engineering (ICDE)*, 2006.

[2] Stefan Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability - a survey. *BIT Numerical Mathematics*, 1985.

[3] Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. A tutorial on particle filters for on-line non-linear/non-gaussian Bayesian tracking. *IEEE Transactions of Signal Processing*, 50(2), 2002.

[4] Daniel Barbara, Hector Garcia-Molina, and Daryl Porter. The management of probabilistic data. *IEEE Transactions on Knowledge and Data Engineering*, 1992.

[5] Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30(3):479–513, 1983.

[6] Omar Benjelloun, Anish Das Sarma, Alon Halevy, and Jennifer Widom. ULDBs: Databases with uncertainty and lineage. In *International Conference on Very Large Data Bases (VLDB)*, 2006.

[7] Umberto Bertele and Francesco Brioschi. *Nonserial Dynamic Programming*. Academic Press, New York, 1972.

[8] Jihad Boulos, Nilesh Dalvi, Bhushan Mandhani, Chris Re, Shobhit Mathur, and Dan Suciu. Mystiq: A system for finding more answers by using probabilities. In *ACM SIGMOD International conference on Management of Data*, 2005.

[9] Héctor Corrada Bravo and Raghu Ramakrishnan. Optimizing MPF queries: decision support and probabilistic inference. In *ACM SIGMOD International conference on Management of Data*, pages 701–712, 2007.

[10] Roger Cavallo and Michael Pittarelli. The theory of probabilistic databases. In *International Conference on Very Large Data Bases (VLDB)*, 1987.

[11] Reynold Cheng, Dmitri Kalashnikov, and Sunil Prabhakar. Evaluating probabilistic queries over imprecise data. In *ACM SIGMOD International conference on Management of Data*, 2003.

[12] William W. Cohen and Sunita Sarawagi. Exploiting dictionaries in named entity extraction: Combining semi-markov extraction processes and data integration methods. In *SIGKDD*, 2004.

[13] Robert G. Cowell, A. Philip Dawid, Steffen L. Lauritzen, and David J. Spiegelhater. *Probabilistic Networks and Expert Systems*. Springer, 1999.

[14] Nilesh Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. In *International Conference on Very Large Data Bases (VLDB)*, 2004.

[15] Nilesh Dalvi and Dan Suciu. Management of probabilistic data: Foundations and challenges. In *PODS*, 2007.

[16] Anish Das Sarma, Omar Benjelloun, Alon Halevy, and Jennifer Widom. Working models for uncertain data. In *International Conference on Data Engineering (ICDE)*, 2006.

[17] Rodrigo de Salvo Braz, Eyal Amir, and Dan Roth. Lifted first-order probabilistic inference. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, 2005.

[18] Rina Dechter. Constraint networks. *Encyclopedia of Artificial Intelligence*, 1992.

[19] Rina Dechter. Bucket elimination: A unifying framework for probabilistic inference. In *Uncertainty in Artificial Intelligence (UAI)*, 1996.

[20] Amol Deshpande, Minos Garofalakis, and Michael Jordan. Efficient stepwise selection in decomposable models. In *Proceedings of the 17th Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 128–135, 2001.

[21] Amol Deshpande, Minos Garofalakis, and Rajeev Rastogi. Independence is Good: Dependency-Based Histogram Synopses for High-Dimensional Data. In *ACM SIGMOD International conference on Management of Data*, 2001.

[22] Amol Deshpande, Carlos Guestrin, Sam Madden, Joseph M. Hellerstein, and Wei Hong. Model-driven data acquisition in sensor networks. In *International Conference on Very Large Data Bases (VLDB)*, 2004.

[23] Amol Deshpande, Carlos Guestrin, and Samuel Madden. Using probabilistic models for data management in acquisitional environments. In *Conference on Innovative Data Systems Research (CIDR)*, 2005.

[24] Debabrata Dey and Sumit Sarkar. A probabilistic relational model and algebra. *ACM Transactions on Database Systems (TODS)*, 1996.

[25] Nir Friedman, Lise Getoor, Daphne Koller, and Avi Pfeffer. Learning probabilistic relational models. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, 1999.

[26] Norbert Fuhr and Thomas Rolleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Transactions on Information Systems (TODS)*, 1997.

[27] Lise Getoor and Ben Taskar, editors. *Introduction to Statistical Relational Learning*. MIT Press, Cambridge, MA, USA, 2007.

[28] Lise Getoor, Ben Taskar, and Daphne Koller. Selectivity estimation using probabilistic models. In *ACM SIGMOD International conference on Management of Data*, 2001.

[29] Rahul Gupta and Sunita Sarawagi. Creating probabilistic databases from information extraction models. In *International Conference on Very Large Data Bases (VLDB)*, 2006.

[30] Cecil Huang and Adnan Darwiche. Inference in belief networks: A procedural guide. *International Journal of Approximate Reasoning*, 1994.

[31] Ihab F. Ilyas, Volker Markl, Peter Haas, Paul Brown, and Ashraf Aboulnaga. Cords: automatic discovery of correlations and soft functional dependencies. In *SIGMOD*, 2004.

[32] Tomasz Imielinski and Witold Lipski, Jr. Incomplete information in relational databases. *Journal of the ACM*, 1984.

[33] Tommi Jaakkola and Michael I. Jordan. Variational probabilistic inference and the QMR-DT network. *Journal of Artificial Intelligence Research*, 10:291–322, 1999.

[34] T. S. Jayram, Rajasekar Krishnamurthy, Sriram Raghavan, Shivakumar Vaithyanathan, and Huaiyu Zhu. Avatar information extraction system. In *IEEE Data Engineering Bulletin*, 2006.

[35] Michael I. Jordan, editor. *Learning in graphical models*. MIT Press, Cambridge, MA, USA, 1999.

[36] Michael I. Jordan, Zoubin Ghahramani, Tommi S. Jaakkola, and Lawrence K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 1999.

[37] Rudolph E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[38] Bhargav Kanagal and Amol Deshpande. Online filtering, smoothing and probabilistic modeling of streaming data. In *ICDE*, 2008.

[39] Laks V. S. Lakshmanan, Nicola Leone, Robert Ross, and V. S. Subrahmanian. Probview: a flexible probabilistic database system. *ACM Transactions on Database Systems (TODS)*, 1997.

[40] Blackford Middleton, Michael Shwe, David Heckerman, Max Henrion, Eric Horvitz, Harold Lehmann, and Gregory Cooper. Probabilistic diagnosis using a reformulation of the internist-1/qmr knowledge base. *Methods of Information in Medicine*, 30:241–255, 1991.

[41] Brian Milch, Luke Zettlemoyer, Kristian Kersting, Michael Haimes, and Leslie Kaelbling. Lifted probabilistic inference with counting formulas. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 2008.

[42] Kevin P. Murphy, Yair Weiss, and Michael I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Uncertainty in Artificial Intelligence (UAI)*, pages 467–475, 1999.

[43] Dmitry Pavlov, Heikki Mannila, and Padhraic Smyth. Beyond independence: Probabilistic models for query approximation on binary transaction data. *IEEE TKDE*, 2003.

[44] Judaea Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

[45] Avi Pfeffer, Daphne Koller, Brian Milch, and Ken Takusagawa. SPOOK: A system for probabilistic object-oriented knowledge representation. In *Uncertainty in Artificial Intelligence (UAI)*, 1999.

[46] David Poole. First-order probabilistic inference. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, 2003.

[47] Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, 77(2):257–286, 1989.

[48] Chris Re, Nilesh Dalvi, and Dan Suciu. Efficient top-k query evaluation on probabilistic data. In *International Conference on Data Engineering (ICDE)*, 2007.

[49] Chris Re and Dan Suciu. Approximate lineage for probabilistic databases. In *International Conference on Very Large Data Bases (VLDB)*, 2008.

[50] Irina Rish. *Efficient Reasoning in Graphical Models*. PhD thesis, University of California, Irvine, 1999.

[51] Sunita Sarawagi. Efficient inference on sequence segmentation models. In *ICML*, 2006.

[52] Prithviraj Sen and Amol Deshpande. Representing and querying correlated tuples in probabilistic databases. In *International Conference on Data Engineering (ICDE)*, 2007.

[53] Prithviraj Sen, Amol Deshpande, and Lise Getoor. Exploiting shared correlations in probabilistic databases. In *International Conference on Very Large Data Bases (VLDB)*, 2008.

[54] Parag Singla and Pedro Domingos. Multi-relational record linkage. In *Proceedings of 3rd Workshop on Multi-Relational Data Mining at ACM SIGKDD*, Seattle, WA, 2004.

[55] Parag Singla and Pedro Domingos. Lifted first-order belief propagation. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 2008.

[56] Padhraic Smyth. Belief networks, hidden Markov models, and Markov random fields: a unifying view. *Pattern Recognition Letters*, 18(11-13), 1997.

[57] Greg Welch and Gary Bishop. An introduction to Kalman filter. `http://www.cs.unc.edu/~welch/kalman/kalmanIntro.html`, 2002.

[58] Jennifer Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *Conference on Innovative Data Systems Research (CIDR)*, 2005.

[59] Nevin Lianwen Zhang and David Poole. A simple approach to Bayesian network computations. In *Canadian Conference on Artificial Intelligence*, 1994.

[60] Nevin Lianwen Zhang and David Poole. Exploiting causal independence in Bayesian network inference. *Journal of Artificial Intelligence Research*, 1996.