Representing and Querying Correlated Tuples in Probabilistic Databases

Prithviraj Sen Amol Deshpande
Department of Computer Science, University of Maryland, College Park, MD 20742.
E-mail: {sen, amol}@cs.umd.edu

Abstract

Probabilistic databases have received considerable attention recently due to the need for storing uncertain data produced by many real world applications. The widespread use of probabilistic databases is hampered by two limitations: (1) current probabilistic databases make simplistic assumptions about the data (e.g., complete independence among tuples) that make it difficult to use them in applications that naturally produce correlated data, and (2) most probabilistic databases can only answer a restricted subset of the queries that can be expressed using traditional query languages. We address both these limitations by proposing a framework that can represent not only probabilistic tuples, but also correlations that may be present among them. Our proposed framework naturally lends itself to the possible world semantics thus preserving the precise query semantics extant in current probabilistic databases. 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* probabilistic graphical model. *We present several optimizations specific to probabilistic databases that enable efficient query evaluation. We validate our approach by presenting an experimental evaluation that illustrates the effectiveness of our techniques at answering various queries using real and synthetic datasets.*

1 Introduction

Database research has primarily concentrated on how to store and query *exact* data. This has led to the development of techniques that allow the user to express and efficiently process complex queries in a declarative fashion over large collections of data. Unfortunately, many real-world applications produce large amounts of *uncertain* data. In such cases, databases need to do more than simply store and retrieve; they have to help the user sift through the uncertainty and find the results *most likely* to *be* the answer.

Numerous approaches have been proposed to handle uncertainty in databases [2, 7, 16, 6, 17, 19, 12, 29]. Among these, *tuple-level uncertainty models* [6, 17, 19, 12, 29], that associate existence probabilities with tuples, are considered more attractive for various reasons: (a) they typically result in relations that are in 1NF, (b) they provide simple and intuitive querying semantics, and (c) they are easier to store and operate on. However, these models often make simplistic and highly restrictive assumptions about the data (e.g., complete independence among base tuples [19, 12]). In particular, they cannot easily model or handle *dependencies/correlations*[1] among tuples. The ability to do so is critical for two reasons:

Natural dependencies in the data: Many application domains naturally produce correlated data. 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* [6, 1]. Real-world datasets such as the Christmas Bird Count [14] naturally contain complex correlations among tuples. Data generated by sensor networks is typically highly correlated, both in time and space [16]. Data produced through use of machine learning techniques (e.g. classification labels) typically exhibits complex correlation patterns.

Dependencies during query evaluation: The problem of handling dependencies among tuples arises naturally during query evaluation *even when one assumes that the base data tuples are independent* (Section 2.1). In other words, the independent tuples assumption is not closed under the relational operators, specifically *join* [19, 12].

Past work on tuple-level uncertainty models has addressed this problem by either restricting the set of queries that can be evaluated against such a database (e.g. *safe plans* [12]), or by restricting the dependencies that can be modeled (e.g. *ProbView* [29]). Neither of these approaches, however, is satisfactory for a large class of real-world applications.

In this paper, we propose a *tuple-level uncertainty model* built on the foundations of statistical modeling techniques that allows us to uniformly handle dependencies in the data, while keeping the basic probabilistic framework simple and

[1] From here onwards, we use the terms "dependencies" and "correlations" interchangeably.

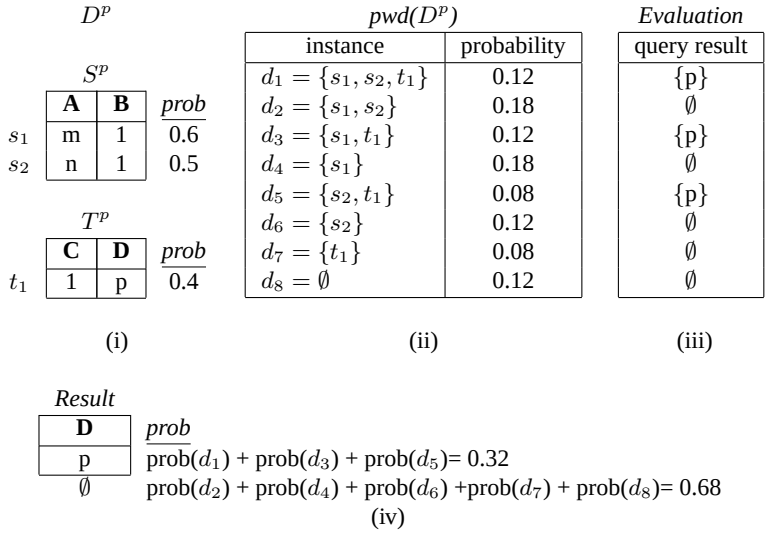


Figure 1. Example reproduced with minor changes from [12]: (i) A probabilistic database with independent tuples; (ii) corresponding possible worlds; (iii) evaluating $\prod_{\mathbf{D}}(S^p \bowtie_{\mathbf{B}=\mathbf{C}} T^p)$ over $pwd(D^p)$; (iv) computing result probabilities.

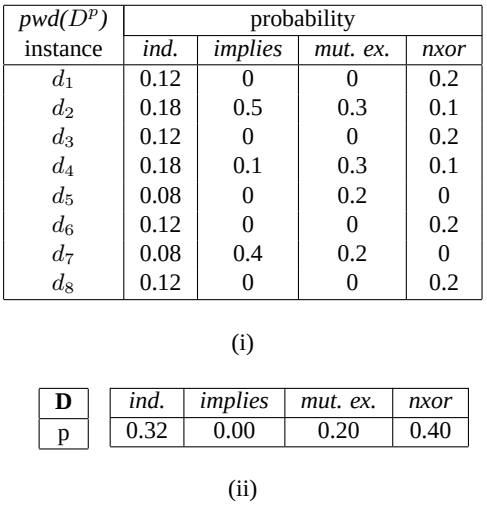


Figure 2. (i) $pwd(D^p)$ for various dependencies and the (ii) corresponding query results.

intuitive. The salient features of our proposed approach, and our main contributions are as follows:

- We propose a uniform framework for expressing uncertainties and dependencies through use of random variables and joint probability distributions. Unlike prior approaches, our proposed model is closed under relational algebra operations.
- We 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 allows us to choose from various inference algorithms (exact or approximate) for query evaluation, depending on our requirements of accuracy and speed.
- We develop several optimizations specific to probabilistic databases resulting in efficient query execution in spite of the rich dependency modeling that we allow.
- We present experimental results from a prototype implementation over several real and synthetic datasets that demonstrate the need for modeling and reasoning about dependencies and the efficacy of our techniques at evaluating various queries including aggregate operators.

We begin with some background on tuple-level uncertainty models and probabilistic graphical models (Section 2). We then present our proposed model for representing correlated data (Section 3), discuss implementation issues 4) and present an experimental evaluation over a prototype implementation (Section 5). Finally, we discuss related work in Section 6, and conclude in Section 7.

2 Background

2.1 Independent Tuples Model [19, 12]

One of the most commonly used tuple-level uncertainty models, the *independent tuples model* [19, 12], associates existence probabilities with individual tuples and assumes that the tuples are independent of each other. Figure 1 (i) shows an example of such a database, D^p, with relations S^p (containing tuples s_1 and s_2 with probabilities 0.6 and 0.5 resp) and T^p (containing tuple t_1 with probability 0.4).

Such a probabilistic database can be interpreted as a probability distribution over the set of all possible deterministic database instances, called *possible worlds* (denoted by $pwd(D)$) [25, 19, 12]. Each deterministic instance (world) contains a subset of the tuples present in the probabilistic database, and the probability associated with it can be calculated directly using the independence assumption (by multiplying together the existence probabilities of tuples present in it and non-existence probabilities of tuples not present in it). Figure 1 (ii) shows all the possible worlds for D^p and their associated probabilities. For example, the probability of $d_2 = \{s_1, s_2\}$ is computed as $0.6 \times 0.5 \times (1-0.4) = 0.18$.

This possible worlds interpretation lends highly intuitive and precise semantics for query evaluation over probabilistic databases. Let q be a query issued on a probabilistic database D^p. We evaluate such a query against each possible world in $pwd(D^p)$ separately, thus resulting in another set of (*result*) possible worlds (with the same associated probabilities). The final result is obtained by taking a union of all the *result* possible worlds, and by associating a probability with each tuple in them to be the sum of

the probabilities of the *result* possible worlds that contain it. For instance, Figure 1 (iii) shows the results of executing $\prod_{\mathbf{D}}(S^p \bowtie_{\mathbf{B}=\mathbf{C}} T^p)$ on each possible world of D^p and Figure 1 (iv) shows the final probability computation.

Evaluating a query via the set of possible worlds is clearly intractable as the number of possible worlds is exponential in the number of tuples contained in the database. Previous literature [19, 12] has suggested two query evaluation strategies instead, called *extensional* and *intensional* semantics. Intensional semantics guarantee results in accordance with possible worlds semantics but are computationally expensive. Extensional semantics, on the other hand, are computationally cheaper but do not guarantee results in accordance with the possible worlds semantics. This is because, even if base tuples are independent of each other, the intermediate tuples that are generated during query evaluation are typically correlated. For instance, in Figure 1, the join operation if performed before the projection results in two intermediate tuples, s_1t_1 and s_2t_1, that are not independent of each other as they share t_1.

2.2 Tuple Correlations

As we discussed in Section 1, tuple correlations also occur naturally in many application domains, and ignoring such correlations can result in highly inaccurate and unintuitive query results.

Consider the four sets of possible worlds shown in Figure 2 (i) derived from the same database shown in Figure 1 (i), but containing different sets of dependencies that we might want to represent:

1. *ind.*: where s_1, s_2, and t_1 are independent of each other.
2. *implies*: presence of t_1 implies absence of s_1 and s_2 ($t_1 \Rightarrow \neg s_1 \wedge \neg s_2$).
3. *mutual exclusivity* (*mut. ex.*): $t_1 \Rightarrow \neg s_1$ and $s_1 \Rightarrow \neg t_1$.
4. *nxor*: high positive correlation between t_1 and s_1, presence (absence) of one almost certainly implies the presence (absence) of the other.

Figure 2 (ii) shows the result of applying the example query from Figure 1 to these four possible worlds. As we can see, although the tuple probabilities associated with s_1, s_2 and t_1 are identical, the query results are drastically different across these four databases. Note that, since both the approaches (intensional and extensional semantics) discussed in the previous section assume base tuple independence, neither can be directly used to do query evaluation in such cases.

2.3 Probabilistic Graphical Models and Factored Representations

Probabilistic graphical models form a powerful class of approaches that can compactly represent and reason about complex dependency patterns involving large numbers of correlated random variables [31, 10]. The key idea underlying these approaches is the use of *factored representations* for modeling the correlations.

$$Pr(X_1 = x_1, X_2 = x_2, X_3 = x_3) = f_1(X_1 = x_1) f_{12}(X_1 = x_1, X_2 = x_2) f_{23}(X_2 = x_2, X_3 = x_3)$$

x_1	f_1
0	0.6
1	0.4

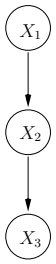
x_1	x_2	f_{12}
0	0	0.9
0	1	0.1
1	0	0.1
1	1	0.9

x_2	x_3	f_{23}
0	0	0.7
0	1	0.3
1	0	0.3
1	1	0.7

(i)

x_1	x_2	x_3	Pr
0	0	0	0.378
0	0	1	0.162
0	1	0	0.018
0	1	1	0.042
1	0	0	0.028
1	0	1	0.012
1	1	0	0.108
1	1	1	0.252

(ii)



(iii)

Figure 3. Example involving three dependent random variables each with a binary domain: (i) factored representation (ii) resulting joint probability distribution (iii) graphical model representation.

Let X denote a random variable with a domain $dom(X)$ and let $Pr(X)$ denote a probability distribution over it.

Similarly, let $\mathbf{X} = \{X_1, X_2, X_3 \ldots, X_n\}$ denote a set of n random variables each with its own associated domain $dom(X_i)$, and $Pr(\mathbf{X})$ denote the join probability distribution over them.

Definition 2.1. *A factor*[2] $f(\mathbf{X})$ *is a function of a (small) set of random variables* $\mathbf{X} = \{X_1, \ldots, X_n\}$ *such that* $0 \leq f(\mathbf{X} = \mathbf{x}) \leq 1 \; \forall \mathbf{x} \in dom(X_1) \times \ldots \times dom(X_n)$.

A factored representation of $Pr(\mathbf{X})$ allows the distribution to be represented compactly as a product of factors:

$$Pr(\mathbf{X} = \mathbf{x}) = \prod_{i=1}^{m} f_i(\mathbf{X}_i = \mathbf{x}_i) \qquad (1)$$

where $\mathbf{X}_i \subseteq \mathbf{X}$ is the set of random variables restricted to factor f_i and $\mathbf{x}_i$ is the corresponding assignment. Figure 3 shows a small example of a factored representation of a joint probability distribution over three random variables.

Computing *marginal probabilities* is a common operation when dealing with such joint probability distributions.

[2]Factors can be seen as a generalization of *conditional probability tables* in Bayesian networks [31].

$Pr(\mathbf{X}_{D^p}) = f^{ind}_{t_1}(X_{t_1}) f^{implies}_{t_1,s_2}(X_{t_1}, X_{s_2})$
$f^{implies}_{t_1,s_1}(X_{t_1}, X_{s_1})$

X_{t_1}	X_{s_1}	$f^{implies}_{t_1,s_1}$
0	0	0
0	1	1
1	0	1
1	1	0

X_{t_1}	X_{s_2}	$f^{implies}_{t_1,s_2}$
0	0	1/6
0	1	5/6
1	0	1
1	1	0

(i)

$Pr(\mathbf{X}_{D^p}) = f^{ind}_{s_2}(X_{s_2})$
$f^{mutex}_{t_1,s_1}(X_{t_1}, X_{s_1})$

X_{t_1}	X_{s_1}	$f^{mutex}_{t_1,s_1}$
0	0	0
0	1	0.6
1	0	0.4
1	1	0

(ii)

$Pr(\mathbf{X}_{D^p}) = f^{ind}_{s_2}(X_{s_2})$
$f^{nxor}_{t_1,s_1}(X_{t_1}, X_{s_1})$

X_{t_1}	X_{s_1}	$f^{nxor}_{t_1,s_1}$
0	0	0.4
0	1	0.2
1	0	0
1	1	0.4

(iii)

Figure 4. Representing probabilistic databases with dependencies (examples from Figure 2): (i) "implies" dependency; (ii) "mut. ex." dependency; (iii) "nxor" dependency. $f^{ind}_{t_1}(X_{t_1})$ and $f^{ind}_{s_2}(X_{s_2})$ refer to the independent tuple factors for t_1 and s_2.

It falls under the general class of operations known as *inference*. Given a random variable $X \in \mathbf{X}$ and an assignment $x \in dom(X)$, the marginal probability computation problem from the joint distribution $Pr(\mathbf{X})$ is:

$$Pr(X = x) = \sum_{\mathbf{x} \sim x} Pr(\mathbf{X} = \mathbf{x}) \quad (2)$$

where $\mathbf{x} \sim x$ denotes an assignment to $\mathbf{X}$ that agrees with $X = x$ and $\mathbf{x}$ is a valid assignment to $\mathbf{X}$. We will discuss inference techniques in Section 4.1.

3 Proposed Approach

We now describe how to represent dependencies in probabilistic databases through the use of factors and random variables. We then consider the query evaluation problem over such databases, and begin by presenting an example that grounds the basic ideas underlying our approach. We then follow up with a more detailed description of the overall query evaluation procedure.

3.1 Representing Dependencies

Let t denote a tuple belonging to relation R such that t is a mapping from attributes in R to constants. In our framework, every tuple t is associated with a unique boolean valued random variable X_t where 0 represents `false` and 1 represents `true`.

A *probabilistic relation* R consists of a set of tuples with their corresponding random variables and a *probabilistic database* D consists of a collection of *probabilistic relations*. We refer to the collection of all the random variables associated with tuples in the probabilistic database D by the symbol $\mathbf{X}_D$.

Each instance in $pwd(D)$ can now be expressed as a complete assignment to the set of random variables $\mathbf{X}_D$ denoted by $\mathbf{x}_D \in \{0,1\}^{|\mathbf{X}_D|}$. For example, d_2 in Figure 2 (i) corresponds to the assignment $X_{s_1} = 1, X_{s_2} = 1, X_{t_1} = 0$.

We can now represent dependencies by defining factors on the tuple-based random variables in the database. The probability of an instance can be computed by computing the joint probability of the assignment to $\mathbf{X}_D$ which can in turn be obtained by multiplying all factors defined on the tuple-based random variables in the database (Eq. (1)).

Example: Representing Independent Tuples.
We illustrate our approach by expressing the probabilistic database in Figure 1 (i) (with three independent tuples) in our formulation. This can be achieved by defining one factor per independent tuple:

X_{s_1}	$f^{ind}_{s_1}$
0	0.4
1	0.6

X_{s_2}	$f^{ind}_{s_2}$
0	0.5
1	0.5

X_{t_1}	$f^{ind}_{t_1}$
0	0.6
1	0.4

To compute the probability for an instance we multiply these factors. For instance:

$$\begin{aligned} Pr(d_2 = \{s_1, s_2\}) &= Pr(X_{s_1} = 1, X_{s_2} = 1, X_{t_1} = 0) \\ &= f^{ind}_{s_1}(X_{s_1} = 1) f^{ind}_{s_2}(X_{s_2} = 1) f^{ind}_{t_1}(X_{t_1} = 0) \\ &= 0.6 \times 0.5 \times 0.6 = 0.18 \end{aligned}$$

Figure 4 shows the factored representations for the other three probabilistic databases in Figure 2 (i).

3.2 Query Evaluation: Example

We begin our discussion on query evaluation by presenting a small example. Consider the database shown in Figure 2 (i) with the "nxor" dependency (Figure 4 (iii)). Figure 5 describes the execution of $\prod_{\mathbf{D}}(S^p \bowtie_{\mathbf{B}=\mathbf{C}} T^p)$ on this database.

Consider the intermediate tuples introduced during the execution of this query. The tuples i_1 and i_2, produced by the join (Figure 5), are clearly uncertain tuples since they are not produced in every instance of the database. Similarly, the result tuple r_1 is also a probabilistic tuple. Let us take a closer look at the inter-tuple dependencies:

- i_1 is produced by possible world d iff d contains both s_1 and t_1 (i.e., $s_1 \wedge t_1 \Leftrightarrow i_1$).
- Similarly, $s_2 \wedge t_1 \Leftrightarrow i_2$.
- Finally, r_1 is produced iff either i_1 or i_2 is produced.

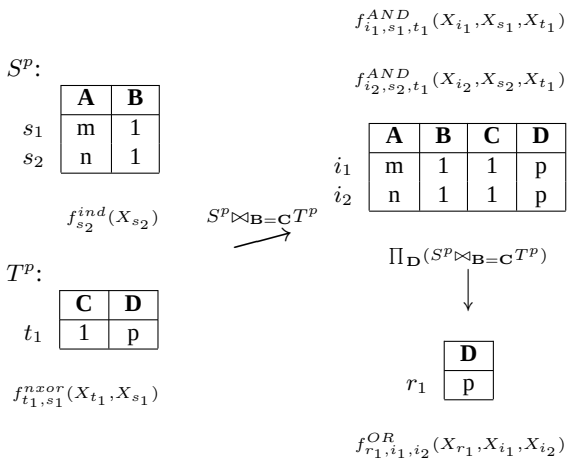


Figure 5. Solving $\prod_{\mathbf{D}}(S^p \bowtie_{\mathbf{B}=\mathbf{C}} T^p)$ on D^p with "nxor" dependency (Figure 2 (i)).

Figure 5 shows the factors for these dependencies:

- $f^{and}_{i_1,s_1,t_1}$ and $f^{and}_{i_2,s_2,t_1}$ return 1 when the first argument is the logical-`and` of the last two arguments (see Figure 6 (i) for the full definition of $f^{and}_{i_1,s_1,t_1}$).
- $f^{OR}_{r_1,i_1,i_2}$ returns 1 when the first argument is the logical-`or` of the last two arguments (Figure 6 (ii)).

Consider the factored probability distribution induced by this query, i.e. the product of all the factors introduced including the factors among the base tuples. It turns out that the marginal probability of $Pr(X_{r_1} = 1)$ returns the correct answer 0.40 matching the number in Figure 2 (ii).

3.3 Query Evaluation: Details

3.3.1 Generating factors during query evaluation

The query evaluation procedure presented through the example above requires that we encode dependencies among (intermediate) tuples by introducing factors. We now redefine the three operators $\sigma, \prod, \times$ to produce factors expressing such dependencies. Let us denote the new operators by $\sigma^p, \prod^p, \times^p$ where the superscript emphasizes that they are operators for probabilistic databases. We will assume that our query q does not contain two copies of the same relation and consists of only the above three operators. In the longer version of the paper [34] we describe redefinitions for all relational algebra operators that do not make these assumptions and can handle all relational algebra queries.

- select ($\sigma^{\mathbf{p}}_{\mathbf{c}}$): Suppose σ^p_c operator with predicate c acts on tuple t and produces new tuple r (both tuples containing the same mapping attribute-value mappings). There are two cases to consider, if t does not satisfy the predicate c, then r cannot be produced and this is enforced by a `false` factor on X_r that returns 1 if $X_r = 0$ and 0 when $X_r = 1$. The other case is when t satisfies c and in this case, X_r holds true in possible world d if and only if X_t holds true. This is enforced by an `equals` factor that takes arguments X_r and X_t and returns 1 whenever $X_r = X_t$ and 0 o.w.
- join ($\times^{\mathbf{p}}$): Let tuples t and t' join to return new tuple r. In this case, as we saw in the example, X_r will hold true in possible world d if and only if both X_t and $X_{t'}$ hold true. This can be enforced with the `and` factor that we introduced in the example (Figure 6 (i)).
- project ($\prod^{\mathbf{p}}_{\mathbf{A}}$): Let tuples $t_1, \ldots, t_n$ project to form new tuple r. In this case, X_r holds true in possible world d if at least one of $X_{t_1}, \ldots, X_{t_n}$ hold true in d. This can be enforced with a straightforward generalization of the `or` factor, introduced in the example (Figure 6 (ii)), $f^{or}_{r,t_1,\ldots t_n}(X_r, X_{t_1}, \ldots X_{t_n})$ that returns 1 if the first argument is the logical-`or` of the last n arguments.

Figure 7 defines the above operators in functional representation where $\mathcal{F}_q(t)$ denotes the set of factors required to generate tuple t inductively on query plan q and Figure 7 describes how we generate factors for a particular tuple t and add them to $\mathcal{F}_q(t)$.

X_{i_1}	X_{s_1}	X_{t_1}	$f^{and}_{i_1,s_1,t_1}$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

(i)

X_{r_1}	X_{i_1}	X_{i_2}	$f^{or}_{r_1,i_1,i_2}$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

(ii)

Figure 6. Definitions of (i) $f^{AND}_{i_1,s_1,t_1}(X_{i_1}, X_{s_1}, X_{t_1})$ and (ii) $f^{OR}_{r_1,i_1,i_2}(X_{r_1}, X_{i_1}, X_{i_2})$ from Figure 5.

$$\mathcal{F}_{\sigma^p_{c(q)}}(t) = \begin{cases} f^{false}(X_t) & \text{if } c(t) \text{is false} \\ f^{equals}(X_t, X_{t'}) \cup \mathcal{F}_q(t') & \text{if } c(t) \text{ is true} \\ & \text{where } t = t' \end{cases}$$

$$\mathcal{F}_{\prod^p_A q}(t) = f^{or}(X_t, \{X_{t'} | t = \textstyle\prod_A t'\}) \cup \{\mathcal{F}_q(t') | t = \textstyle\prod_A t'\}$$

$$\mathcal{F}_{q \times^p q'}(t,t') = f^{and}(X_{t \times t'}, X_t, X_{t'}) \cup \mathcal{F}_q(t) \cup \mathcal{F}_{q'}(t')$$

Figure 7. Definitions for relational algebra operators.

3.3.2 Formulating the Query Evaluation Problem

We now justify our approach of generating factors for query evaluation. Let $Pr_q(t)$ denote the probability associated with result tuple t produced by query q issued on the database D. Thus from possible world semantics:

$$Pr_q(t) = \sum_{\mathbf{x} \in \{0,1\}^{|\mathbf{X}_D|},\ t \sim q(\mathbf{x})} P(\mathbf{X}_D = \mathbf{x})$$

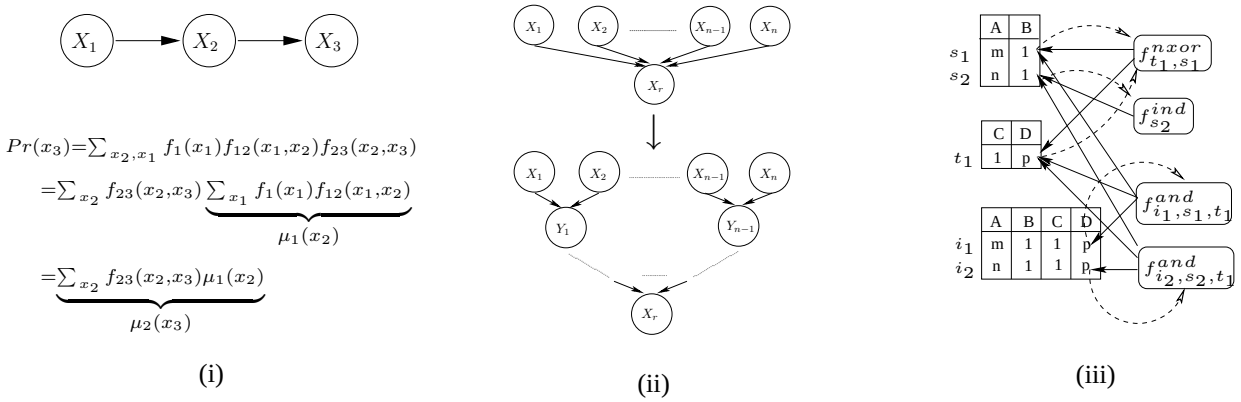


Figure 8. (i) Marginal probability computation for X_3 using variable elimination on the distribution shown in Figure 3. (ii) Transforming a graphical model using decomposability (iii) Partitions for the "nxor" dependency shown in Figure 2 (i) with the partitions introduced by the join shown in Figure 5.

where $t \sim q(\mathbf{x})$ is true if t is included in the result of q applied to possible world $\mathbf{x}$.

Let X_t denote the random variable associated with t. We would like to express the above computation as a marginal probability computation problem on an appropriate distribution $Pr(\mathbf{X}_D, X_t)$ such that:

$$Pr_q(t) = \sum_{\mathbf{x}\in\{0,1\}^{|\mathbf{X}_D|}} Pr(\mathbf{X}_D = \mathbf{x}, X_t = 1)$$

where $Pr(\mathbf{X}_D, X_t)$ satisfies:

$$Pr(\mathbf{X}_D = \mathbf{x}, X_t = 1) = \begin{cases} Pr(\mathbf{X}_D = \mathbf{x}) & \text{if } t \sim q(\mathbf{x}) \\ 0 & \text{o.w.} \end{cases} \quad (3)$$

It is easy to show that by introducing factors that maintain dependencies for the tuples generated by the query we will, in fact, construct a joint distribution that satisfies Eq. (3) (see [34] for full derivation). Computing the result from this distribution is thus a matter of computing marginal probabilities that requires inference in probabilistic graphical models.

4 Query Execution

In this section, we provide further details as to how we execute queries in probabilistic databases with dependencies. We begin with a description of variable elimination, the inference algorithm used in our current implementation, and discuss various optimizations to perform variable elimination efficiently. After that we discuss ways to store probabilistic databases with correlated tuples.

4.1 Inference in Graphical Models

Exact probabilistic inference is known to be NP-hard in general [9]. However, many applications provide graphical models with a graph structure that allow efficient probabilistic computation [37]. Variable elimination (VE), also known as *bucket elimination*, [37, 15] is an exact inference algorithm that has the ability to exploit this structure. VE can be used to compute the marginal probabilities of a single random variable from a joint distribution. The main advantages of VE are simplicity and generality.

Computing the marginals of a random variable X requires that we sum out all the other random variables present in the joint distribution (Eq. (2)). Figure 8 (i) shows how VE computes the marginal probability corresponding to $X_3 = x_3$ for the joint distribution described in Figure 3. In Figure 8 (i) we first sum over X_1 producing a new factor $\mu_1(X_2)$ and then sum over X_2 producing a factor $\mu_2(X_3)$ from which we can retrieve the required result.

The complexity of VE depends on some natural parameters relating to the connectivity of the graph underlying the graphical model corresponding to the joint probability distribution [33]. The inference problem is easy if the graphical model is or closely resembles a tree and the problem becomes progressively harder as the graphical model deviates more from being a tree. Interestingly, in the context of probabilistic databases with independent base tuples, Dalvi and Suciu [12] identified a class of queries that allow efficient evaluation (queries with *safe plans*). In the longer version of the paper [34], we show that safe plans give rise to inference problems with tree-structured graphical models where running inference is easy.

Another possible reason for inference being difficult is due to the presence of factors involving a large number of random variables. Projection and aggregate operations can produce large factors but we can easily reduce the size of these factors by exploiting *decomposability* [38, 33]. This allows us to break any large projection factor into numerous (linear in the number of tuples involved in the projection) constant-sized 3-argument factors. Figure 8 (ii) shows the pictorial representation of this optimization. The top graphical model represents n tuples projecting into one result tuple producing a large factor. The bottom graphical

model shows the transformed graphical model that contains $n-1$ new random variables and consists of only 3-argument factors. All aggregate operators (e.g., `sum`, `max` etc.) are also decomposable with the exception of `avg`. To compute `avg`, we first compute `sum` and `count`, both of which are decomposable, and then compute the average.

4.2 Representing Probabilistic Relations

Earlier approaches represented probabilistic relations by storing uncertainty with each tuple in isolation. This approach is inadequate for our purposes since the same tuple can be involved in multiple dependencies. In our implementation, we store the data and uncertainty parts separately. The tuples are stored as part of relations. To store uncertainty, we introduce the concept of a *partition*. **A partition consists of a factor and a set of** *references* **to the tuples whose random variables form the arguments to that factor**. Besides, each tuple t in a relation also contains a list of pointers to the partitions that contain references to the tuple. The relations and partitions together form a doubly-linked data structure so that we can move from one to the other. Figure 8 (iii) shows the arrangement of relations and partitions for the database with the "nxor" dependency (Figure 4 (iii)) where dotted lines represent the pointers from tuples to partitions and solid lines show the references.

4.3 Query Execution Steps

Here is a brief sketch of the steps we follow for SPJ queries:

1. *Perform early selections*: We push down and execute as many selections as possible.
2. *Perform early projections*: We also push down as many projections as possible.
3. *Join phase*: In this phase, we perform all the join operations and perform them as part of one multi-join operation, creating partitions and intermediate relations (Figure 8 (iii) shows the partitions introduced due to a join operation on the database shown in Figure 4 (iii)).
4. *Perform final projection*: Here we project onto the result attributes specified by the user in the query.
5. *Probability computation*: For each result tuple, we recursively collect all partitions required to perform inference and compute the required marginal probability.

We refer the interested reader to [34] for full details.

4.4 Further Optimizations

The memory required to hold all partitions to compute probability for result tuple r may exceed available memory. To alleviate this problem we further divide the joint distribution for r into many independent parts such that no tuple reference from part i is involved in part j, where $i \neq j$, and compute the marginal probabilities for each part separately. The probabilities from the various parts can be combined easily since we know they are independent. A simple search for connected components can be used to determine these independent parts where we start with a graph of tuple references collected from the various partitions and each reference is connected to all other references within the same partition via edges. As part of our future work we aim to use external memory graph algorithms [36] for these tasks.

When exact probabilistic inference turns out to be too expensive we have the flexibility of switching to approximate inference techniques depending on the user's requirements. Just like exact inference, approximate inference is also known to be NP-hard [11]. However there exist a fairly large variety of approximate inference algorithms that perform well in practice in a variety of cases each varying in speed and accuracy e.g., Markov Chain Monte Carlo techniques [23], Variational Methods [28] etc.

5 Experimental Study

In this section, we present an experimental evaluation demonstrating the need for modeling tuple correlations, and the effectiveness of our techniques at modeling such correlations and evaluating queries over them. This evaluation was done using a prototype system that we are currently building on top of the Apache Derby Java DBMS [26]. Our system supports the query execution strategies discussed in the previous section.

Following Fuhr et al [19] and Dalvi et al [12], we generate probabilistic relations by issuing *similarity predicates* over deterministic relations. Briefly, given a predicate of the form $R.a \approx k$, where a is a string attribute, and k is a string constant, the system assigns a probability to each tuple t, based on how *similar* $t.a$ is to k. Following the prior work [12], we compute the *3-gram distance* [35] between $t.a$ and k, and convert it to a posterior probability by assuming that the distance is normally distributed with mean 0, and variance σ (σ is a parameter to the system). The details can be found in [12].

5.1 Need for Modeling Dependencies

Consider a *publications* database containing two relations: (1) `PUBS(PID, Title)`, and (2) `AUTHS(PID, Name)`, where PID is the unique publication id, and consider the task of retrieving all publications with title y written by an author with name x. Assuming that the user is not sure of the spellings x and y, we might use the following query to perform the above task:

$$\prod_{Title}(\sigma_{Name \approx x}(\texttt{AUTHS}) \bowtie \sigma_{Title \approx y}(\texttt{PUBS}))$$

As discussed above, the similarity predicates will cause both the relations to be converted into probabilistic relations, $\texttt{AUTHS}^p$ and $\texttt{PUBS}^p$. However, note that $\texttt{AUTHS}^p$

Title
Reinforcement learning with hidden states (by L. Lin, T. Mitchell)
Feudal Reinforcement Learning (by C. Atkeson, P. Dayan, . . .)
Reasoning (by C. Bereiter, M. Scardamalia)
. . .

(i) *MUTEX_DB* results at $\sigma = 10, 50, 100$

Title
Reinforcement learning with hidden states (by L. Lin, T. Mitchell)
Feudal Reinforcement Learning (by C. Atkeson, P. Dayan, . . .)
Reasoning (by C. Bereiter, M. Scardamalia)
. . .

(ii) *IND_DB* results at $\sigma = 10$

Title
Feudal Reinforcement Learning (by C. Atkeson, P. Dayan, . . .)
Decision making and problem solving (G. Dantzig, R. Hogarth, . . .)
Multimodal Learning Interfaces (by U. Bub, R. Houghton, . . .)
. . .

(iii) *IND_DB* results at $\sigma = 50$

Title
Decision making and problem solving (G. Dantzig, R. Hogarth, . . .)
HERMES: A heterogeneous reasoning and mediator system (by S. Adali, A. Brink, . . .)
Induction and reasoning from cases (by K. Althoff, E. Auriol, . . .)
. . .

(iv) *IND_DB* results at $\sigma = 100$

Figure 9. Top three results for a similarity query: (i) shows results from *MUTEX_DB*; (ii), (iii) and (iv) show results from *IND_DB*.

contains natural mutual exclusion dependencies with respect to this query. Since the user is looking for publications by a single author with name x, it is *not possible for x to match two* AUTHS^p *tuples corresponding to the same publication in the same possible world*. Thus, any two AUTHS^p tuples with the same PID exhibit a mutual exclusion dependency, and a possible world containing both of them should be assigned zero probability.

To illustrate the drawbacks of ignoring these mutual exclusion dependencies, we ran the above query with x = "T. Michel" and y = "Reinforment Leaning hiden stat" on two probabilistic databases, one assuming complete independence among tuples (*IND_DB*) and another that models the dependencies (*MUTEX_DB*). We ran the query on an extraction of 860 publications from the real-world CiteSeer dataset [22]. We report results across various settings of σ.

Figure 9 shows the top three results obtained from the two databases at three different settings of σ (we also list the author names to aid the reader's understanding). *MUTEX_DB* returns intuitive and similar results at all three values of σ. *IND_DB* returns reasonable results only at $\sigma = 10$, whereas at $\sigma = 50, 100$ it returns very odd results ("Decision making and problem solving" does not match the string "Reinforment Leaning hiden stat" very closely and yet it is assigned the highest rank at $\sigma = 100$). Figure 10 (i) shows the cumulative recall graph for *IND_DB* for various values of σ, where we plot the fraction of the top N results returned by *MUTEX_DB* that were present in the top N results returned by *IND_DB*. As we can see, at $\sigma = 50$ and 100, *IND_DB* exhibits poor recall.

Figure 9 shows that *IND_DB* favors publications with long author lists. This does not affect the results at low values of σ (=10) because, in that case, we use a "peaked" gaussian which assigns negligible probabilities to possible worlds with multiple AUTHS^p from the same publication. At larger settings of σ, however, these possible worlds are assigned larger probabilities and *IND_DB* returns poor results. *MUTEX_DB* assigns these possible worlds zero probabilities by modeling dependencies on the base tuples.

We would like to note that, although setting the value of σ carefully may have resulted in a good answer for *IND_DB* in this case, choosing σ is not easy in general and depends on various factors such as user preferences, distributions of the attributes in the database etc [12]. Modeling mutual exclusion dependencies explicitly using our approach naturally alleviates this problem.

5.2 Scalability

Next we study the scalability of our proposed query execution strategies using a randomly generated TPC-H dataset of size 10MB. For simplicity, we assume complete independence among the base tuples (though the intermediate tuples may still be correlated).

Figure 10 (ii) shows the execution times on TPC-H queries Q2 to Q8 (modified to remove the top-level aggregations). The first bar on each query indicates the time it took for our implementation to run the full query including all the database operations and the probabilistic computations. The second bar on each query indicates the time it took to run only the database operations using our Java implementation. Here are the summary of the results:

- As we can see in Figure 10 (ii), for most queries the additional cost of probability computations is comparable to the cost of normal query processing.
- The two exceptions are Q3 and Q4 which severely tested our probabilistic inference engine. By removing the aggregate operations, Q3 resulted in a relation of size in excess of 60,000 result tuples. Although Q4 resulted in a very small relation, each result tuple was associated with a probabilistic graphical model of size exceeding 15,000 random variables. Each of these graphical models are fairly sparse but book-keeping for such large data structures took a significant amount of time.
- Q7 and Q8 *are queries without safe plans [12] yet their*

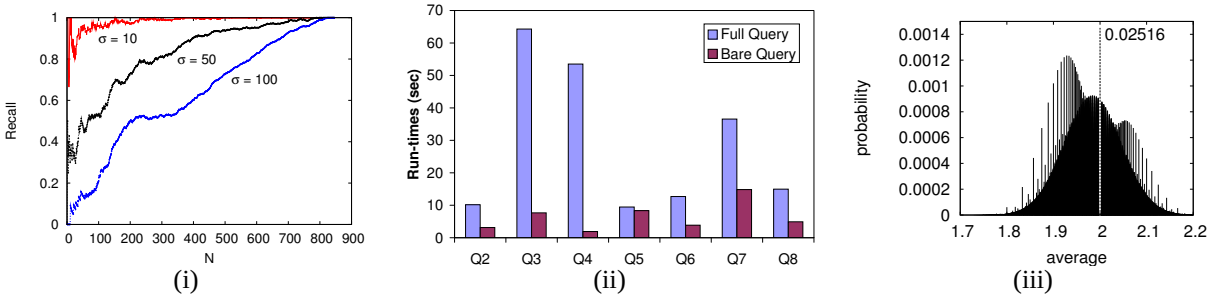


Figure 10. (i) Cumulative recall graph comparing results of *IND_DB* and *MUTEX_DB* for $\sigma = 10, 50, 100$. (ii) Run-times on TPC-H data. (iii) AVG aggregate computed over 500 randomly generated tuples with attribute values ranging from 1 to 5.

run-times are surprisingly fast. By taking a closer look we notice that both these queries gave rise to tree structured graphical models justifying our belief that there are queries that lend to efficient probabilistic inference even though they might not have safe plans.

5.3 Aggregate Operations

Our system also naturally supports efficient computation of a variety of aggregate operators over probabilistic relations; details can be found the longer version of the paper [34]. Figure 10 (iii) shows the result of running an *average* query over a synthetically generated dataset containing 500 tuples. As we can see, the final result can be a fairly complex probability distribution, which is quite common for aggregate operations. Effective computation of aggregates over large probabilistic databases is an open research problem that we plan to study in future.

6 Related Work

The database community has seen a lot of work on managing probabilistic, uncertain, incomplete, and/or fuzzy data in database systems (see e.g. [32, 27, 2, 29, 30, 24, 19, 5, 7, 12, 4]). With a rapid increase in the number of application domains such as data integration, pervasive computing etc., where uncertain data arises naturally, this area has seen renewed interest in recent years (see [20] for a survey of the ongoing research). From the uncertainty in artificial intelligence community, Friedman et. al [18] (PRM) address the problem of learning a probabilistic model from a given database. While PRMs can represent uncertainty in databases, Getoor et. al. [21] explore the application of PRMs to answering selectivity estimation queries but not queries expressed in standard database languages.

We will briefly discuss some of the closely related work in the area of *probabilistic* data management. Previous work in this area can be differentiated based on (a) whether probabilities are associated with *attributes* or with *tuples*, (b) whether the resulting relations are in the first normal form (which is highly desirable given the complexity of managing and querying data which is not in 1NF), and (c) whether the correlations typically present in real-world data can be modeled. Barbara et al [2] propose an approach that associates probabilities with attributes and can model arbitrary correlations between attributes *within a single tuple*. However the resulting relations are not in 1NF and further the semantics of some of the query operators are messy, both of which seriously limit the applicability of their approach. More recently, Choenni et al [8] discuss conceptually how to make the query semantics more consistent through use of Dempster-Schafer theory.

Cavallo et al [6] and Dey et al [17] propose and study tuple-level uncertainty models that explicitly capture *mutual exclusivity*. More recently, Andritsos et al [1] use a similar basic model to capture *dirty* data, and develop query evaluation algorithms for the same. In a series of papers, Fuhr and Rolleke [19] propose using tuple-level probabilities to model uncertain data in *information retrieval* context, and present the intensional and extensional query evaluation techniques discussed in Section 2. Extending this work, Dalvi and Suciu [12, 13] define *safe query plans* to be those for which extensional and intensional query evaluation produces identical results, and show how to generate a safe query plan for a query if one exists. Tuple independence is assumed for most of the work by both these groups. Lakshmanan et al [29] attempt to combine these different approaches by associating *probability intervals* with tuples in their *ProbView* system. Their model also supports various conjunction and disjunction *strategies* that allow a limited encoding of tuple interdependences.

Cheng et al [7] associate (continuous) probability distributions with attributes, and propose several query evaluation and indexing techniques over such data. Trio [14, 3] aims to provide a unified treatment of *data uncertainty* by studying the issues of completeness and closure under various alternative models for representing uncertainty. Their recent work [3] combines *data lineage* and *data accuracy* in a single system by tracking lineage of tuples derived from other tuples. However, they do not consider duplicate elimination and aggregation, among other operations.

7 Conclusions

There is an increasing need for database solutions for efficiently managing and querying uncertain data exhibiting complex correlation patterns. In this paper, we presented a simple and intuitive framework, based on probabilistic graphical models, for explicitly modeling correlations among tuples in a probabilistic database . Our experimental evaluation illustrates both the necessity of modeling tuple correlations, and the effectiveness of our techniques at representing and querying correlated datasets. Our research so far has raised several interesting challenges that we plan to pursue in future. Although conceptually our approach allows for capturing arbitrary tuple correlations, exact query evaluation over large datasets exhibiting complex correlations may not always be feasible. We plan to develop approximate query evaluation techniques that can be used in such cases. We are also planning to develop disk-based query evaluation algorithms so that our techniques can scale to very large datasets.

Acknowledgements: This work was supported by NSF Grants CNS-0509220, IIS-0546136 and IIS-0308030. We thank Lise Getoor for pointing us to relevant material on probability theory and inference methods. We also thank the anonymous reviewers for their helpful comments.

References

[1] P. Andritsos, A. Fuxman, and R. J. Miller. Clean answers over dirty databases. In *ICDE*, 2006.

[2] D. Barbara, H. Garcia-Molina, and D. Porter. The management of probabilistic data. In *KDE*, 1992.

[3] O. Benjelloun, A. Das Sarma, A. Halevy, and J. Widom. ULDBs: Databases with uncertainty and lineage. In *VLDB*, 2006.

[4] P. Bosc and O. Pivert. About projection-selection-join queries addressed to possibilistic relational databases. In *IEEE Tran. on Fuzzy Systems*, 2005.

[5] B. P. Buckles and F. E. Petry. A fuzzy model for relational databases. *Intl. Journal of Fuzzy Sets and Systems*, 1982.

[6] R. Cavallo and M. Pittarelli. The theory of probabilistic databases. In *VLDB*, 1987.

[7] R. Cheng, D. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *SIGMOD*, 2003.

[8] S. Choenni, H. E. Blok, and E. Leertouwer. Handling uncertainty and ignorance in databases: A rule to combine dependent data. In *Database Systems for Adv. Apps.*, 2006.

[9] G. F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 1990.

[10] R. Cowell, S. Lauritzen, and D. Spiegelhater. *Probabilistic Networks and Expert Systems*. Springer, New York, 1999.

[11] P. Dagum and M. Luby. Approximate probabilistic reasoning in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 1993.

[12] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, 2004.

[13] N. Dalvi and D. Suciu. Query answering using statistics and probabilistic views. In *VLDB*, 2005.

[14] A. Das Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working models for uncertain data. In *ICDE*, 2006.

[15] R. Dechter. Bucket elimination: A unifying framework for probabilistic inference. In *UAI*, 1996.

[16] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, 2004.

[17] D. Dey and S. Sarkar. A probabilistic relational model and algebra. *ACM Trans. on Database Systems*, 1996.

[18] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *IJCAI*, 1999.

[19] N. Fuhr and T. Rolleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. on Information Systems*, 1997.

[20] M. Garofalakis and D. Suciu, editors. *IEEE Data Engineering Bulletin Special Issue on Probabilistic Data Management*. March 2006.

[21] L. Getoor, B. Taskar, and D. Koller. Selectivity estimation using probabilistic models. In *SIGMOD*, 2001.

[22] C. Giles, K. Bollacker, and S. Lawrence. Citeseer: An automatic citation indexing system. In *Conf. on Digit. Libs.*, 1998.

[23] W. R. Gilks, S. Richardson, and D. J. Spiegelhalter. *Markov Chain Monte Carlo in Practice*. Chapman & Hall, 1996.

[24] G. Grahne. Horn tables - an efficient tool for handling incomplete information in databases. In *PODS*, 1989.

[25] J. Halpern. An analysis of first-order logics for reasoning about probability. *Artificial Intelligence*, 1990.

[26] http://db.apache.org/derby. The Apache Derby Project.

[27] T. Imielinski and W. Lipski, Jr. Incomplete information in relational databases. *Journal of the ACM*, 1984.

[28] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 1999.

[29] L. V. S. Lakshmanan, N. Leone, R. Ross, and V. S. Subrahmanian. Probview: a flexible probabilistic database system. *ACM Trans. on Database Systems*, 1997.

[30] S. K. Lee. An extended relational database model for uncertain and imprecise information. In *VLDB*, 1992.

[31] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

[32] H. Prade and C. Testemale. Generalizing database relational algebra for the treatment of incomplete or uncertain information and vague queries. *Information Sciences*, 1984.

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

[34] P. Sen and A. Deshpande. Representing and querying correlated tuples in probabilistic databases. Technical Report CS-TR-4820, University of Maryland, College Park, 2006.

[35] E. Ukkonen. Approximate string matching with q-grams and maximal matches. In *Theoretical Computer Science*, 1992.

[36] J. S. Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing Surveys*, 2001.

[37] N. L. Zhang and D. Poole. A simple approach to Bayesian network computations. In *Canadian Artif. Intel. Conf.*, 1994.

[38] N. L. Zhang and D. Poole. Exploiting causal independence in Bayesian network inference. *Journal of Artificial Intelligence Research*, 1996.