Representing Tuple and Attribute Uncertainty in Probabilistic Databases

Prithviraj Sen sen@cs.umd.edu
Amol Deshpande amol@cs.umd.edu
Lise Getoor getoor@cs.umd.edu

Computer Science Department, University of Maryland, College Park, MD, 20742.

Abstract

There has been a recent surge in work in probabilistic databases, propelled in large part by the huge increase in noisy data sources — sensor data, experimental data, data from uncurated sources, and many others. There is a growing need to be able to flexibly represent the uncertainties in the data, and to efficiently query the data. Building on existing probabilistic database work, we present a unifying framework which allows a flexible representation of correlated tuple and attribute level uncertainties. An important capability of our representation is the ability to represent shared correlation structures in the data. We provide motivating examples to illustrate when such shared correlation structures are likely to exist. Representing shared correlations structures allows the use of sophisticated inference techniques based on lifted probabilistic inference that, in turn, allows us to achieve significant speedups while computing probabilities for results of user-submitted queries.

1 Introduction

An increasing number of real-world applications are demanding support for managing, storing, and querying uncertain data in relational database systems. This has led to a recent resurgence of research in the area of probabilistic databases. This work has spanned a wide range of issues such as query optimization [11], indexing uncertain data [22], query languages [4, 2] and data models that can represent complex correlations [13, 20, 21] naturally produced from various applications [14, 13, 3]. The underlying data models for much of this work are based on probability theory coupled with *possible worlds semantics* where a probabilistic database is defined as a probability distribution over numerous possible databases. Although this definition results in intuitive querying semantics, probabilistic databases are still far from being efficient and scalable, with the query processing problems known to be #P-complete even for very simple queries [11].

As opposed to query evaluation for traditional databases containing exact data, query evaluation for probabilistic databases requires that we calculate probabilities associated with result tuples present in the answer to user-submitted queries [11], and this, in turn, requires probabilistic inference [21]. The main reason for most of the complexity results associated with probabilistic databases is a consequence of this close connection with probabilistic inference, which is known to be an NP-Hard problem, in general [9]. However, in many cases, it is possible to leverage special properties of the uncertain data at hand to reduce the complexity of query evaluation. One such property is the presence of *shared correlation structures*. Many applications produce uncertain data with probability distributions and correlations copied many times over; this is because the distributions and correlations typically come from general statistics that *do not* vary on a per-tuple basis. For instance, Andritsos et al. [1] report a customer relationship management application where the objective is to merge data from two or more source databases and each source database is assigned a probability value based on the quality of the information it contains. Even here, probabilities don't change from tuple to tuple, since tuples from the same source are assigned the same source probability. Also, Dalvi and Suciu [12] report uncertain data collected as part of the Cancer Genome Anatomy Project that assigns tags to genes and this information is uncertain because of the inherent uncertainties in the experiments conducted to produce the tag-gene association. Once again, the uncertainties in the experimental setup are likely to vary in systematic ways from one experiment to another, and will result in repetition in probability values. As we show in this paper, it is possible to exploit such shared correlation structures using sophisticated inference techniques to significantly reduce the time associated with probabilistic inference for query evaluation.

One prerequisite to being able to exploit shared correlation structures is to design a probabilistic database that can represent the shared correlation structures in the base data in the first place. Unfortunately, most of the recent work in probabilistic databases has been concentrated on the development of tuple-level uncertainty probabilistic databases [15, 11, 13, 21]. Many applications produce data that is more naturally represented using attribute-level uncertainty as opposed to tuple-level uncertainty, e.g., sensor network

AdID	SellerID	Date	Type	Model	MPG	Price
101	201	1/1	Sedan	Civic (EX)	?	$6,000
102	201	1/10	Sedan	Civic (DX)	?	$4,000
103	?	1/15	?	Civic	?	$12,000
104	202	1/1	Hybrid	Civic	?	$20,000
105	202	1/1	Hybrid	Civic	?	$20,000

SellerID	Reputation
201	Shady
202	Good

(a)

AdID	f(t.E)
101	0.3
102	0.6
103	0.8
104	0.2
105	0.2

t_{103}.SellerId	f(t_{103}.SellerId)
201	0.6
202	0.4

t_{103}.Type	f(t_{103}.Type)
Sedan	0.3
Hybrid	0.7

Type	Model	MPG	f(Type,Model,MPG)
Sedan	Civic (EX)	26	0.2
		28	0.6
		30	0.2
Sedan	Civic (DX)	32	0.1
		35	0.7
		37	0.2
Sedan	Civic	28	0.4
		35	0.6
Hybrid	Civic	45	0.4
		50	0.6

t_{101}.E	t_{102}.E	f(t_{101}.E, t_{102}.E)
True	True	0.8
True	False	0.1
False	True	0.1
False	False	0.9

(b)

Figure 1: (a) A simple car advertisement database (b) Factors describing the probabilities and correlations among tuples and attributes.

datasets [14], mobile object databases [8] etc. (see Barbara et al [3] for more examples). Although converting a database with attribute-level uncertainty to a database with tuple-level uncertainty is a fairly simple operation, this transformation usually leads to a loss of shared correlation structures (besides resulting in a database that requires storing a significantly larger number of tuples).

In general, we may require both tuple and attribute level uncertainty to faithfully represent the shared correlation structures present in real-world data and, for this purpose, in this paper, we propose a probabilistic database model that can represent correlated attribute and tuple level uncertainty, thus fully exploiting the representational power of the probability theory. Our proposed model uses small functions called *parameterized factors* to compactly represent the uncertainty in the data, and fundamentally supports sharing of both the uncertainty representation and the correlation structures. We also present a preliminary experimental study to demonstrate how beneficial such sharing can be during query processing.

2 Motivating Example

Consider the small database shown in Figure 1 containing information regarding pre-owned cars for sale. The database consists of two tables, $Ad(AdID, SellerID, Date, Type, Model, MPG, Price)$ and $Seller(SellerID, Reputation)$. Note that this is a very simple model of a pre-owned car sales database and, for simplicity, we have removed a number of attributes that would otherwise be present in such a database such as odometer readings and year of manufacture.

Probabilities factor into this domain in a variety of ways. First off, we may have some uncertainty about the existence of a tuple. In this domain, perhaps older ads are likely to correspond to cars that have already been sold. We may want to represent this explicitly by associating a probability with each tuple. Figure 1(b) shows a probability factor $f(t.E)$ which describes the probability of each ad existing.

AdID	SellerID	Date	Type	Model	MPG	Price	P(t)
101	201	1/1	Sedan	Civic (EX)	26	$6,000	0.0693
101	201	1/1	Sedan	Civic (EX)	28	$6,000	0.208
101	201	1/1	Sedan	Civic (EX)	30	$6,000	0.0693
102	201	1/10	Sedan	Civic (DX)	32	$4,000	0.0413
102	201	1/10	Sedan	Civic (DX)	35	$4,000	0.2893
102	201	1/10	Sedan	Civic (DX)	37	$4,000	0.0826
103	201	1/15	Sedan	Civic	28	$12,000	0.0576
103	201	1/15	Sedan	Civic	35	$12,000	0.0864
103	201	1/15	Hybrid	Civic	45	$12,000	0.1344
103	201	1/15	Hybrid	Civic	50	$12,000	0.2016
103	202	1/15	Sedan	Civic	28	$12,000	0.0384
103	202	1/15	Sedan	Civic	35	$12,000	0.0576
103	202	1/15	Hybrid	Civic	45	$12,000	0.0896
103	202	1/15	Hybrid	Civic	50	$12,000	0.1344
104	202	1/1	Hybrid	Civic	45	$20,000	0.08
104	202	1/1	Hybrid	Civic	50	$20,000	0.12
105	202	1/1	Hybrid	Civic	45	$20,000	0.08
105	202	1/1	Hybrid	Civic	50	$20,000	0.12

Figure 2: A probabilistic database for the car advertisement database represented using tuple probabilities.

In this example, there are also several unknown attribute values, shown as question marks. Consider the third tuple in the *Ads* table, which we will denote t_{103}. Both the *SellerID* and the *Type* of this car is unknown. We can represent this uncertainty using tuple-specific attribute factors, which describe the likelihoods of the different possible sellers and models. These are the tables $f(t_{103}.SellerID)$ and $f(t_{103}.Type)$, shown in Figure 1(b). Finally, there is uncertainty over MPG. The final factor in Figure 1(b) shows the likelihoods for combinations of Type, Model, and MPGs.

Given these probabilities, one approach to representing the probabilistic database is to "flatten" it out into a probabilistic database which has only tuple uncertainty. Figure 2 shows an attempt to do this. This requires computing joint probability distributions by multiplying the appropriate numbers from the various factors. For instance, the 7^{th} tuple in Figure 2 shows t_{103} in Figure 1(a) instantiated

with *SellerID*=201, *Type*=Sedan, *MPG*=28 and the probability value of 0.0576 is obtained by multiplying 0.8 (from $f(t.E)$), 0.6 (from $f(t_{103.SellerID})$), 0.3 (from $f(t_{103.Type})$) and 0.4 (from $f(Type, Model, MPG)$). The main advantage of this approach, as indicated in Section 1, is that there have been several recent proposals for probabilistic database systems which handle such tuple-level uncertainty databases involving mutually exclusive groups of tuples. However, there are several disadvantages. First, notice that there is a considerably larger number of tuples in Figure 2 than there is in Figure 1. In fact, this increase in size is a known problem with computing joint probability distributions and has been pointed out in other contexts such as selectivity estimation in databases [16]. The other, more subtle but more serious issue is the loss of shared correlation structures. Note that the factor $f(Type, Model, MPG)$ (Figure 1(b)) is actually a shared factor and is associated with any tuple that has any of its *Type*, *Model* or *MPG* values missing. Unfortunately, in Figure 2, this shared correlation is not easily visible, especially since most of the tuples in Figure 2 are associated with distinct tuple probabilities.

Our aim is to perform efficient query processing by utilizing shared correlation structures. To this end, we propose a probabilistic database model that not only supports correlated tuple and attribute level uncertainty but also *sharing* of probabilistic factors.

3 PRDB: A Generic Probabilistic DB Model

We begin with some notation. Following [11], we use the superscript p to emphasize the fact that we are dealing with a probabilistic database throughout the paper. Let D^p denote a probabilistic database with tuple and attribute level uncertainty. Let D^p contain a set of relations $\mathscr{R} = \{R_1^p, \ldots, R_m^p\}$. Let U denote the set of all possible constants; U contains a special *unassigned* constant $\emptyset$. Let $Attr(R_i^p)$ denote the set of attributes for relation R_i^p. Let t denote a tuple in R_i^p. For every $t \in R_i^p$ and $A \in Attr(R_i^p)$, we use $t.A$ to denote a random variable with domain $Dom(t.A)$ such that $Dom(t.A) \subseteq U$. Intuitively, $t.A$ captures the uncertainty associated with the value of attribute A of tuple t. Further, for every $t \in R_i^p$ we also have a special random variable $t.E$ that can take assignments from the binary domain {`true`, `false`} to capture the uncertainty associated with t's existence. We use $\mathscr{X}$ to denote the set of all random variables in probabilistic database D^p.

To express shared correlation structures we will need to familiarize ourselves with some concepts borrowed from the machine learning literature [19, 6, 7]. Essentially, for PRDB, we need to have constructs that allow us to express encapsulations of random variables associated with the same kind of uncertainty and random variables involved in the same kind of correlations. For this purpose we next define the notions of *tuple identifier collections*, *parameterized random variables* and *parameterized factors*.

A *tuple identifier collection* (TIC) I is a collection of tuple identifiers such that corresponding to each $i \in I$ there is a tuple in D^p. Let $\mathbf{I}$ denote a set of TICs $\{I_1, I_2 \ldots I_m\}$. A *parameterized random variable* (ParRV) V encapsulates a collection of random variables having the same domain and is represented by a triple $< \eta, \mathbf{I}, M >$ where:

- η is the type of V and is of the form $R^p.A$ or $R^p.E$ that indicates that each random variable mapped to by M belongs to the attribute A of relation R^p (in which case $A \in Attr(R^p)$) or tuple existence random variables of relation R^p, as the case might be.
- M is a one-to-one mapping that maps each list of tuple identifiers $\mathbf{i} = \{i_1, i_2, \ldots i_k\}$ such that $i_1 \in I_1, i_2 \in I_2, \ldots i_k \in I_k$ to an RV r that is of type η.

A useful operation with any ParRV $V =< \eta, \mathbf{I}, M >$ is to extract a *ground random variable* given $\mathbf{i}$, an element from the cartesian product of $\mathbf{I}$, and this we denote by $V(\mathbf{i})$ which simply returns the random variable that M maps to. We refer to the random variables encapsulated by ParRV V as the set of *ground random variables* of V. Further, we will refer to the domain of V (which is identical for each ground random variable) by $Dom(V)$. We will also refer to the various components of the ParRV V by appropriate subscripts such that V_η, $V_\mathbf{I}$ and V_M refer to V's type, collection of TICs and mapping respectively.

A *factor* is the basic unit of representing uncertainty and correlations in most of the work on probabilistic graphical models [18, 10]. Factors are simply functions over small sets of random variables that map each joint instantiation to real numbers between 0 and 1. A *parameterized factor* (ParFactor) F encapsulates a collection of factors and is represented by a triple $< \mathbf{I}, \mathbf{V}, \phi >$ where:

- $\mathbf{I}$ is a set of TICs
- $\mathbf{V}$ is a set of ParRVs such that $V_\mathbf{I} \subseteq \mathbf{I}\ \forall V \in \mathbf{V}$
- ϕ is a function such that $0 \leq \phi(\mathbf{V} = \mathbf{v}) \leq 1$ for each joint instantiation $\mathbf{v} \in \times_{V \in \mathbf{V}} Dom(V)$.

A ParFactor $F =< \mathbf{I}, \mathbf{V}, \phi >$ simply encapsulates all the factors obtained by substituting the groundings of $\mathbf{V}$ in ϕ. In other words F represents $\{\phi(\mathbf{V}(\mathbf{i}))\}_{\mathbf{i} \in \mathbf{I}}$. This is better explained using an example. Consider the 4^{th} tuple (t_{104}) and 5^{th} tuple (t_{105}) in the *Ad* relation in Figure 1. Each of these tuples has the same attribute value missing, *MPG*. Thus, the uncertainty associated with these two tuples can be expressed using the same function $f(Type, Model, MPG)$ shown in Figure 1(b). The naive way to do this is to represent the uncertainty using two ground factors over ground random variables $f(t_{104}.Type, t_{104}.Model, t_{104}.MPG)$ and $f(t_{105}.Type, t_{105}.Model, t_{105}.MPG)$ and copy the probability values once for each factor. An alternative approach is to represent the uncertainty using ParRVs and ParFactors which represents the sharing of factors. For this we

first define a TIC $I = \{t_{104}, t_{105}\}$. We then define three ParRVs:

- $V_{Type} =< Ad.Type, \{I\}, M_{Type} >$, where $Ad.Type$ is the type of V_{Type} and $M_{Type}(t_{104}) = t_{104}.Type$, $M_{Type}(t_{105}) = t_{105}.Type$.
- $V_{Model} =< Ad.Model, \{I\}, M_{Model} >$, where $Ad.Model$ is the type of V_{Model} and $M_{Model}(t_{104}) = t_{104}.Model$, $M_{Model}(t_{105}) = t_{105}.Model$.
- $V_{MPG} =< Ad.MPG, \{I\}, M_{MPG} >$, where $Ad.MPG$ is the type of V_{MPG} and $M_{MPG}(t_{104}) = t_{104}.MPG$, $M_{MPG}(t_{105}) = t_{105}.MPG$.

Now we can represent the two ground factors using a single ParFactor $F =< \{I\}, \{V_{Type}, V_{Model}, V_{MPG}\}, \phi >$ where ϕ is essentially the function shown in Figure 1(b), for instance, $\phi(V_{Type} = \text{Hybrid}, V_{Model} = \text{Civic}, V_{MPG} = 45) = 0.4$. Grounding out F we get the same two factors over the ground random variables. For instance, grounding out the ParRVs for t_{104} we get, $V_{Type}(t_{104}) = t_{104}.Type$, $V_{Model}(t_{104}) = t_{104}.Model$ and $V_{MPG}(t_{104}) = t_{104}.MPG$, substituting the ParFactor F with these ground random variables gives us the ground factor $\phi(t_{104}.Type, t_{104}.Model, t_{104}.MPG)$ which is exactly what the factor $f(t_{104}.Type, t_{104}.Model, t_{104}.MPG)$ represents. Similarly, grounding F with t_{105} gives us the other ground factor. Just like ParRVs, for any ParFactor $F =< \mathbf{I}, \mathbf{V}, \phi >$ we represent its components with suitable subscripts, in other words, $F_{\mathbf{I}}$ represents the TICs, $F_{\mathbf{V}}$ represents the ParRVs and F_ϕ represents the factor over ParRVs. We can also extract a ground factor from F by the operation $F(\mathbf{i})$ (where $\mathbf{i} \in \times_{I \in F_{\mathbf{I}}} I$) which simply returns the factor $F_\phi(F_{\mathbf{V}}(\mathbf{i}))$.

A D^p, besides containing a set of relations $\mathscr{R}$, also contains a set of ParFactors $\mathscr{F}$. Like many previous approaches, we define the semantics of our probabilistic database using possible worlds semantics [17, 11]. Possible world semantics interprets a probabilistic database as a probability distribution over a number of possible databases. We refer to each possible database as an instance of the probabilistic database and each instance is obtained by choosing a (sub)set of the uncertain tuples to be present in the instance and assigning exactly one value to each uncertain attribute from its domain. The probability assigned to each instance is simply the product of the relevant probabilities returned by the set of ground factors represented by $\mathscr{F}$ and divided by a constant such that the sum of probabilities over all instances is 1.

More formally, an instantiation of D^p is an assignment $\mathbf{x}$ to all the ground random variables in $\mathscr{X}$ from their respective domains. Not all assignments are legal; if $t.E = false$ then we require $t.A = \emptyset$. In other words, if a tuple does not exist, then its attributes should be unassigned. Any legal assignment $\mathbf{x}$ of $\mathscr{X}$ gives us a corresponding *certain* instance of D^p. These are the possible worlds, and D^p defines a distribution over them.

Definition 3.1 *A probabilistic database D^p consists of a set of relations $\mathscr{R}$ and a set of ParFactors $\mathscr{F}$. Let $\mathscr{X}$ denote all the random variables present in D^p and let $\mathbf{x}$, denoting a legal assignment to $\mathscr{X}$, represent a possible world of D^p. The probability of the possible world $\mathbf{x}$ is:*

$$\Pr(\mathscr{X} = \mathbf{x}) = \frac{1}{Z} \prod_{F \in \mathscr{F}} \prod_{\theta \sim F_{\mathbf{I}}} F_\phi(F_{\mathbf{V}}(\theta) = \mathbf{x}_{F_{\mathbf{V}}(\theta)}) \qquad (1)$$

where $\theta \sim F_{\mathbf{I}}$ denotes an element from the cartesian product of all the TICs in $F_{\mathbf{I}}$, $\mathbf{x}_{F_{\mathbf{V}}(\theta)}$ denotes the assignments restricted to the ground random variables $F_{\mathbf{V}}(\theta)$ and Z denotes the normalization constant $\sum_{\mathbf{x}'} \prod_{F \in \mathscr{F}} \prod_{\theta \sim F_{\mathbf{I}}} F_\phi(F_{\mathbf{V}}(\theta) = \mathbf{x}'_{F_{\mathbf{V}}(\theta)})$.

Note that we do not impose any restrictions on the types of ParRVs involved in any ParFactors in D^p. This means that if the user wants then s/he can define a ParFactor over ParRVs of types $R^p.A$, $R^p.E$, $R'^p.A$ or $R'^p.E$, where R^p and R'^p are relation names and A is an attribute name from the relation, or any combination of these types. Thus, it should be easy to express correlations involving just attributes or attributes and tuple existence random variables or tuple existence correlations.

4 Experiments

In this section we report some preliminary experiments performed on synthetic data that illustrate the reduction in times required to perform inference. We compare two approaches. Earlier work [21] has shown that standard probabilistic inference algorithms, such as variable elimination [23] (VarElim), can be used to perform inference for query evaluation in probabilistic databases. However, these approaches were mainly geared towards probabilistic databases with tuple-level uncertainty only. It is relatively easy to extend the same approach to handle both tuple and attribute level uncertainty. One approach we compare against is this extension of the ideas presented in our earlier work with VarElim for the underlying inference method. As indicated earlier, representing shared correlation structures using ParFactors allows us the use of more sophisticated inference algorithms. These inference algorithms generally go by the name of lifted inference algorithms and one such inference algorithm is called *parameterized variable elimination* (ParVE) [19, 6]. We use ParVE to illustrate the reduction in runtimes achieved by utlizing the shared correlation structures in the probabilistic database by comparing against the runtimes achieved by VarElim.

As described in our earlier work [21], one approach to query evaluation in probabilistic databases is by introducing correlations among intermediate tuples and the tuples they are produced from. For instance, consider a join issued

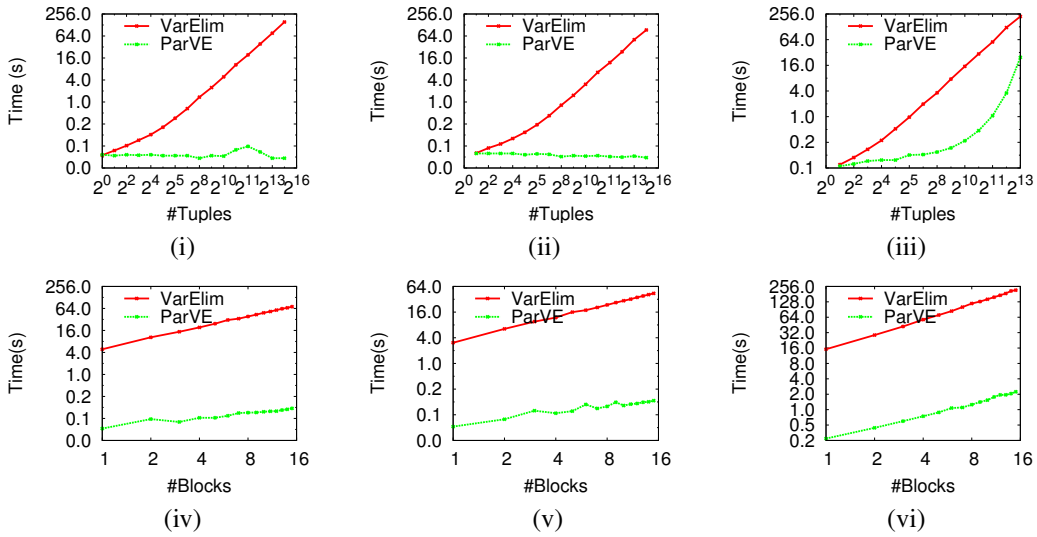

Figure 3: Experiments with selection and join queries on synthetic data. All plots are in log-log scale.

on two relations R_1^p and R_2^p. Let r denote the result tuple produced by the join between tuples $t_1 \in R_1^p$ and $t_2 \in R_2^p$ such that t_1 and t_2 satisfy the join condition. Note that r is only produced by a possible world if both t_1 and t_2 are present in it, which means $r.E$, $t_1.E$ and $t_2.E$ are correlated. In prior work [21], we showed how such correlations can be produced and represented using factors automatically at runtime while evaluating queries. Unfortunately, those techniques are inadequate now that we would like to use ParRVs and ParFactors to represent the same correlations in a shared manner. To see that shared correlations are introduced while evaluating queries notice that every time a tuple $t_1 \in R_1^p$ joins with another tuple $t_2 \in R_2^p$ to produce a tuple r, we need to introduce a logical-$\wedge$ factor to enforce the correlation connecting the three tuples' existence. More importantly, notice that the same logical-$\wedge$ needs to be introduced for any such triplet of tuples thus leading to copies of factors and motivating the use of ParFactors while query evaluation. Due to space constraints it is not possible to include the details regarding the introduction of ParRVs and ParFactors while query evaluation here but we refer the interested reader to the extended version of the paper (citation omitted for blind review purposes).

All our experiments were performed on a dual-proc Xeon 2.8GHz machine with 3GB of memory. Our code is written in JAVA. In all these experiments we only report the times required to run inference and do not include the times required to perform various other operations such as times required to read the data from disk etc.

We studied the variation in run times by varying two parameters. In every synthetic relation we generate, we include many tuples that share the same correlation structure, we refer to a collection of such tuples with identical correlations and factors as a *block* of tuples. Tuples from different blocks have a different correlation structure. One of the parameters we varied in all our experiments is the number of blocks per relation (#Blocks). The other parameter we varied is the number of tuples per block (#Tuples).

In our first experiment, we considered a simple selection query on a three attribute relation $R_1(A,B,C)$. For each tuple $t \in R_1$, each of $t.A$, $t.B$ and $t.C$ are uncertain attributes with domains of size 10. Moreover, $\forall t \in R_1$, $t.C$ is correlated with $t.B$ which in turn is correlated with $t.A$. The query we considered is $\sigma^p_{C=c}R_1$, where c is a constant present in the domain of every $t.C$ random variable in R_1. Thus in this case, VarElim needs to compute the probabilities for each result tuple, separately. On the other, by exploiting the shared factors, ParVE can reduce the computation required for inference by computing once per block of tuples. In Figure 3 (i), we generated R_1 so that all tuples belonged to the same block and varied the number of tuples from 1 to 2^{15}. The plot shows that VarElim's execution time increases linearly with #Tuples, whereas, the times for ParVE remain, roughly, constant. In Figure 3 (iv) we kept the number of tuples per block constant (1024) but varied the number of blocks in R_1. In this case, since ParVE needs to compute probabilities once per block, it's time increases linearly with the number of blocks but there is still a considerable improvement in execution times compared to VarElim.

In the above experiment, we considered a relation R_1 with a maximum of 2^{15} (32,768) tuples, which may not seem like a very big relation, but we would like to point out that this number, in fact, compares favourably with experiments performed on most probabilistic databases, especially the ones that can only represent tuple-level uncertainty [15, 11, 13]. Recall that R_1 contains tuples with three attributes and each attribute has a domain of size 10. Thus "flattening" R_1 into a relation with only tuple-level uncer-

tainty would mean taking each tuple from R_1 and representing it with $10 \times 10 \times 10$ uncertain tuples (recall the example in Section 2) which would result in a relation with $2^{15} \times 10 \times 10 \times 10 = 32,768,000$ tuples.

In our second experiment we tried a similar selection query on a relation $R_2(A,B)$ that contains tuple uncertainty, tuple correlations and attribute correlations. More specifically, every odd-numbered tuple $t_o \in R_2$ has some uncertainty of existence besides a correlation between $t_o.A$ and $t_o.B$ both of which have domains of size 10. Every even-numbered tuple $t_e \in R_2$, however, has tuple uncertainty that depends on the previous odd-numbered tuple t'_o's $t'_o.E$ tuple existence random variable besides the correlation between $t_e.A$ and $t_e.B$. Figure 3 (ii) and Figure 3 (v) show the runtimes obtained by varying #Tuples and #Blocks, respectively, for the query $\sigma^p_{B=b}R_2$ where b is a constant.

For the third query we tried a two-relation join followed by a projection $\prod^p_{\mathbf{D}}(R_3(A,B) \bowtie^p R_4(C,D))$ where both relations consist of attribute correlations such that each tuple from R_3 joined with exactly one tuple from R_4 and all the tuples produced by the join project to the same result tuple. The reader needs to be familiar with the details of the ParVE algorithm to fully comprehend the results of this query (Figure 3 (iii) and Figure 3 (vi)), please see the extended version of the paper for more details. Essentially, ParVE involves splitting and merging of ParFactors and unlike the previous two queries, this query requires a fair number of splits and merges. The main result is that, even though the execution times of ParVE does not remain linear with the number of blocks in the relations, it still provides significant speedups compared to VarElim. For instance, in Figure 3 (iii), even when #Tuples is 2^{13} ParVE requires 24.6 sec. to run whereas VarElim requires about 224 sec.

5 Conclusion

There is a growing need to be able to flexibly represent the uncertainties in the data, and to efficiently query the data. Building on existing work within the probabilistic database and probabilistic knowledge representation communities, we have presented PRDB, a unifying framework which leverages on shared correlation structures to achieve significant speedups during query processing.

PRDB allows one to faithfully represent the uncertainties present in data. Such faithful representations open up new possibilities for richer data mining tasks. Consider, for example, the task of determining faulty sensors from a database of sensor readings. Data collected from sensor networks is usually associated with errors and cannot be represented using traditional relational databases. Moreover, sensor network datasets also harbour strong correlations [14] that can be represented using the factor-based representation of PRDB. Note that such correlations cannot be faithfully represented by many recently proposed probabilistic database models [5, 20] which only allow us to represent mutual exclusivity dependencies. Determining the readings corresponding to faulty sensors may be formulated as an outlier detection problem and a data mining algorithm that attempts to detect such outliers will need to examine the dependency network formed by the tuple and attribute based random variables in the database which is only possible in the case of PRDB since it stores these dependencies in a compact and coherent fashion.

References

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

[2] L. Antova, C. Koch, and D. Olteanu. From complete to incomplete information and back. In *SIGMOD*, 2007.

[3] D. Barbara, H. Garcia-Molina, and D. Porter. The management of probabilistic data. *IEEE TKDE*, 1992.

[4] O. Benjelloun, A. Das Sarma, C. Hayworth, and J. Widom. An introduction to ULDBs and the Trio system. In *IEEE Data Engineering Bulletin*, 2006.

[5] O. Benjelloun, A. Sarma, A. Halevy, and J. Widom. ULDBs: Databases with uncertainty and lineage. In *VLDB'06*.

[6] R. d. Braz, E. Amir, and D. Roth. Lifted first-order probabilistic inference. In *IJCAI*, 2005.

[7] R. d. Braz, E. Amir, and D. Roth. MPE and partial inversion in lifted probabilistic variable elimination. In *AAAI*, 2006.

[8] R. Cheng, D. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *SIGMOD'03*.

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

[10] R. G. Cowell, S. L. Lauritzen, and D. J. Spiegelhater. *Probabilistic Networks and Expert Systems*. Springer, 1999.

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

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

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

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

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

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

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

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

[19] D. Poole. First-order probabilistic inference. In *IJCAI*, 2003.

[20] C. Re, N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, 2007.

[21] P. Sen and A. Deshpande. Representing and querying correlated tuples in probabilistic databases. In *ICDE*, 2007.

[22] S. Singh, C. Mayfield, S. Prabhakar, R. Shah, and S. Hambrusch. Indexing uncertain categorical data. In *ICDE*, 2007.

[23] N. L. Zhang and D. Poole. A simple approach to bayesian network computations. In *Canadian Conf. on Artificial Intelligence*, 1994.