Database Theory Column
Probabilistic Databases*

Dan Suciu
University of Washington

April 30, 2008

Abstract

Many applications today need to manage large data sets with uncertainties. In this paper we describe the foundations of managing data where the uncertainties are quantified as probabilities. We review the basic definitions of the probabilistic data model and present some fundamental theoretical results for query evaluation on probabilistic databases.

1 The Quest for Probabilistic Databases

Commercial databases today are deterministic. Relational databases are rooted in First Order Logic and Finite Model Theory, as initially envisioned by Codd [10], and were originally motivated by applications like banking, payroll, accounting, inventory, all of which require a precise semantics of the data. Subsequently, databases and data management techniques have been extended to handle richer data models, such as Nested Relations [44], Object-relational data [21], temporal data [45], spatial data [36], and semistructured data and XML [46]. All these extensions rely on a deterministic semantics for both data and queries.

Today, the database community needs to manage large volumes of data that is imprecise, or uncertain, and that contains an explicit representation of the uncertainty. Uncertain data occurs in large scale data integration [29], integration of life-science databases [39], in information extraction systems [24, 22, 33], in sensor data [8, 23, 19, 20], in activity recognition data [37, 9, 35]. Modeling and managing data where uncertainties are explicitly represented and numerically quantified requires a new paradigm whose foundation are based on probability theory, probabilistic inference [12], probability logic [2], and degrees of belief [6], in addition to Finite Model Theory. This paper presents the basic definitions of the probabilistic data model and some fundamental theoretical results for query evaluation on probabilistic databases. A precursor to probabilistic databases,

*This research was supported in part by Suciu's NSF CAREER grant IIS-0092955, NSF grants IIS-0415193, IIS-0627585, IIS-0513877, IIS-0428168, and a gift from Microsoft.

incomplete databases [32, 27], allowed the explicit representation of uncertainties, but no numerical quantification.

A Paradigm for Managing Uncertainties

The AI community has considered several approaches to representing uncertainties, *e.g.* rule-based systems, Dempster-Shafer, fuzzy sets, but over the past twenty years or so the probability model has been dominant [30]. Probabilistic databases adopt the same model. Thus, the data is probabilistic, where the probabilities are internal measures of the imprecision in the data. Users formulate queries using a standard query language, as if the data were precise. The system computes the answers, and for each answer computes a probability score representing its confidence in that answer.

The central problem in probabilistic databases is query evaluation. Given a Boolean query q and a probabilistic database, compute the answer of q on the database. On a deterministic database the answer is a Boolean value, `true` or `false`, but on a probabilistic database the answer to q is a probability, in notation $\mathbf{P}(q)$. We consider only Boolean queries, *i.e.* closed First Order Logic formulas, because a non-Boolean query $q(x)$ can be answered by evaluating repeatedly Boolean queries $q[a/x]$ obtained by substituting some constant a for x. Our interest is in studying the data complexity [49]: fix q and examine the complexity of computing $\mathbf{P}(q)$ as a function of the size of the probabilistic database.

The query evaluation problem can be reduced to a special case of inference in a probabilistic network, which is a problem extensively studied in the Knowledge Representation community, and which is known to be hard: both exact inference [11] and approximate inference [13] are NP-hard. The key distinction is that in probabilistic databases we refine the complexity analysis by separating the query from the data, since the size of the data is by far dominant. In light of this distinction, approximate inference is tractable [25], and, as we explain here, even precise inference is tractable for an important class of queries.

This paper, based on [14, 16], reviews the definition of probabilistic databases, defines the query evaluation problem and shows special cases when it is tractable.

2 The Possible Worlds Data Model

A probabilistic database is defined in terms of possible worlds, which extends incomplete databases [27]. Throughout this paper we restrict our discussion to relational data over a finite domain: extensions to continuous domains [19] and to XML [31, 1, 48] have also been considered.

We fix a relational schema $\mathcal{R} = (R_1, \ldots, R_k)$, where each R_i is a relation name and has a set of attributes $Attr(R_i)$. We also fix a set of key attributes, $Key(R_i) \subseteq Attr(R_i)$; we explain below why we include keys in our data model. Fixing a finite domain of atomic values, D, denote, for $i = 1, \ldots, k$, by Tup_i the set of typed tuples of the form $R_i(a_1, \ldots, a_k)$ where $a_1, \ldots, a_k \in D$ and

I_1

A	B	C	D
a_1	b_1	c_1	d_1
a_2	b_1	c_3	d_1
a_2	b_2	c_4	d_2

$\mathbf{P}(I_1) = 0.06$
$(= p_1 p_3 p_6)$

I_2

A	B	C	D
a_1	b_1	c_2	c_2
a_2	b_1	c_2	c_1
a_2	b_2	c_4	c_2

$\mathbf{P}(I_2) = 0.12$
$(= p_2 p_5 p_6)$

I_3

A	B	C	D
a_1	b_1	c_1	d_1
a_2	b_2	c_4	d_2

$\mathbf{P}(I_3) = 0.04$
$(= p_1(1\text{-}p_3\text{-}p_4\text{-}p_5)p_6)$

Figure 1: A probabilistic database $PDB = (\{I_1, I_2, I_3, \ldots\}, \mathbf{P})$ with schema $R(\underline{A, B}, C, D)$; we show only three possible worlds.

A	B	C	D	P
a_1	b_1	c_1	d_1	$p_1 = 0.25$
		c_2	d_2	$p_2 = 0.75$
a_2	b_1	c_3	d_1	$p_3 = 0.3$
		c_1	d_3	$p_4 = 0.3$
		c_2	d_1	$p_5 = 0.2$
a_2	b_2	c_4	d_2	$p_6 = 0.8$
		c_5	d_2	$p_7 = 0.2$

Figure 2: Representation of a disjoint-independent probabilistic database. There are seven possible tuples, which are here grouped by their keys, for readability. There are 16 possible worlds; three are shown in Fig. 1.

$k = |Attr(R_i)|$ is the arity of R_i. Also, denote $Tup = \bigcup_i Tup_i$ the disjoint union of all typed tuples. A database instance is any subset $I \subseteq Tup$ that satisfies all key constraints. We write $Key(t)$ for a typed tuple of arity $|Key(R_i)|$ consisting of the key attributes of a tuple $t \in Tup_i$.

The main idea in a probabilistic database is that the state of the database, *i.e.* the instance I is not known. Instead the database can be in any one of a finite number of possible states $I_1, I_2, \ldots$, called *possible worlds*, each with some probability.

Definition 2.1 *A* probabilistic database *is a probability space* $PDB = (W, \mathbf{P})$ *where the set of outcomes is a set of* possible worlds $W = \{I_1, \ldots, I_n\}$. *In other words, it is a function* $\mathbf{P} : W \to [0, 1]$ *s.t.* $\sum_{I \in W} \mathbf{P}(I) = 1$.

Fig. 1 illustrates three possible worlds of a probabilistic database. There are more than three worlds in this probabilistic database, and their probabilities must sum up to 1. The intuition is that we have a database with schema $R(\underline{A, B}, C, D)$ (underlined attributes are key attributes), but we are not sure about the content of the database: there are several possible contents, each with a probability.

A Boolean query q (*i.e.* a closed First Order Logic sentence) defines the event $\{I \mid I \models q\}$ over a probabilistic database, and its marginal probability is $\mathbf{P}(q) = \sum_{I \in W : I \models q} \mathbf{P}(I)$. To any tuple t we associate the Boolean query $t \in I$ and denote its marginal probability $\mathbf{P}(t) = \sum_{I \in W : t \in I} \mathbf{P}(I)$. Note that $t \neq t'$ and $Key(t) = Key(t')$ implies $\mathbf{P}(t, t') = 0$, *i.e.* t, t' are disjoint events.

Disjoint-Independent Databases In practice we cannot enumerate all possible worlds, since there are usually too many. Instead, we need to find some concise way to *represent* the probabilistic database. The *representation problem* for probabilistic database is an active research topic [17, 7, 27, 5, 4, 27]. In our discussion we restrict to a simple, yet popular representation formalism that can represent all disjoint-independent databases.

Any tuple that occurs in some possible world is called a *possible tuple* for the probabilistic database, and we denote T the set of possible tuples. The representation formalism is simply this: for each relation name R_i we store the set of all possible tuples of type R_i together with their marginal probabilities. If the probabilistic database is *disjoint-independent*, then one can recover the entire probability space from these marginal tuple probabilities. More precisely, call a probabilistic database *disjoint-independent* if any set of possible tuples with distinct keys is independent: $\forall t_1, \ldots, t_n \in T$, $Key(t_i) \neq Key(t_j)$ for $i \neq j$ implies $\mathbf{P}(t_1, \ldots, t_n) = \mathbf{P}(t_1) \cdots \mathbf{P}(t_n)$. It suffices to consider only sets of tuples with distinct keys because if any two tuples have the same key then they are disjoint events, and then $\mathbf{P}(t_1, \ldots, t_n) = 0$. Thus, in a disjoint-independent database all tuples are independent, except for the tuples that have conflicting keys (which are disjoint). If, in addition, for every relation R_i in the schema, $Key(R_i) = Attr(R_i)$, then there are no disjoint tuples, and we say that the probabilistic database is *independent*.

Let $K = \{Key(t) \mid t \in T\}$ and let $I \subseteq T$ be an instance. For every key value $k \in K$ denote $p_k^I = \mathbf{P}(t)$ if there exists a (necessarily unique) tuple $t \in I$ s.t. $Key(t) = k$, and $p_k^I = 1 - \sum_{t \in T : Key(t) = k} \mathbf{P}(t)$ if there is no tuple $t \in I$ s.t. $Key(t) = k$. Then one can check that $\mathbf{P}(I) = \prod_{k \in K} p_k^I$. In other words, for a disjoint-independent database the probability distribution $\mathbf{P}(-)$ can be recovered completely from the marginal probabilities $\mathbf{P}(t)$ for all possible tuples $t \in T$.

3 Query Evaluation

We study the query evaluation problem: given a query q and a disjoint-independent database PDB, compute the probability $\mathbf{P}(q)$. We first describe the connection between this problem and the probability of Boolean formulas, which has been studied in the literature; in particular it follows that for any fixed q (*i.e.* any First Order sentence), computing $\mathbf{P}(q)$ is in #P in the size of PDB. We then analyze the data complexity of $\mathbf{P}(q)$ for a fixed query, and establish a dichotomy for conjunctive queries without self-joins: every query is either #P-hard or in PTIME. A conjunctive query is an FO sentence consisting of positive literals, $\wedge$ and $\exists$, while a conjunctive query without selfjoins is one in which each relation

symbol occurs at most once.

From Queries to Boolean Formulas

Consider n Boolean variables $\bar{X} = \{X_1, \ldots, X_n\}$. A disjoint-independent probability space with outcomes $2^{\bar{X}}$ (the set of truth assignments) is given by a partition $\bar{X} = \bar{X}_1 \cup \ldots \cup \bar{X}_m$ and a probability function $\mathbf{P} : \bar{X} \rightarrow [0,1]$ s.t. $\forall j = 1, m$, $\sum_{X \in \bar{X}_j} \mathbf{P}(X) \leq 1$. A truth assignment is chosen at random by independently choosing in each set $\bar{X}_j$ at most one variable X_i that will be set to *true*, while all others are set to false: namely X_i is chosen with probability $\mathbf{P}(X_i)$. We will assume that all probabilities are given as rational numbers, $\mathbf{P}(X_i) = p_i/q_i$, $i = 1, n$, where p_i, q_i are integers.

We call this probability space *independent* if $\forall j$, $|\bar{X}_j| = 1$ (*i.e.* there are no disjoint variables), and *uniform* if it is independent and $\forall i, \mathbf{P}(X_i) = 1/2$.

Let Φ be a Boolean formula over $X_1, \ldots, X_n$; the goal is to compute its probability, $\mathbf{P}(\Phi) = \sum_{\theta \in 2^{\bar{X}} : \theta(\Phi)} \mathbf{P}(\theta)$.

The complexity class #P consists of problems of the following form: given an NP machine, compute the number of accepting computations [38]. Let $\#\Phi$ denote the number of satisfying assignments for Φ. Valiant [47] has shown that the problem: given Φ, compute $\#\Phi$, is #P-complete.

A statement like "computing $\mathbf{P}(\Phi)$ is in #P" is technically non-sense, because computing $\mathbf{P}(\Phi)$ is not a counting problem. However, one can show that for any partition of $\bar{X}$ into m sets $\bar{X}_1, \ldots, \bar{X}_m$, there exists a polynomial-time computable, numerical function $F(p_1, q_1, \ldots, p_n, q_n)$ s.t. for every formula Φ, the number $F \cdot \mathbf{P}(\Phi)$ is an integer, and computing $F \cdot \mathbf{P}(\Phi)$ is in #P. For example, in the case of a uniform distribution, the function F is $F = 2^n$, because in this case $2^n \cdot \mathbf{P}(\Phi) = \#\Phi$, and computing $\#\Phi$ is in #P. In this sense:

Theorem 3.1 *[25, 16] Computing* $\mathbf{P}(\Phi)$ *is in #P.*

While computing $\mathbf{P}(\Phi)$ for a DNF Φ is still #P-hard, Luby and Karp [34] have shown that it has a FPTRAS (fully poly-time randomized approximation scheme). More precisely: there exists a randomized algorithm A with inputs Φ, ε, δ, which runs in polynomial time in $|\Phi|$, $1/\varepsilon$, and $1/\delta$, and returns a value $\tilde{p}$ s.t. $\mathbf{P}_A(|\tilde{p}/p - 1| > \varepsilon) < \delta$. Here $\mathbf{P}_A$ denotes the probability over the random choices of the algorithm. Grädel et al. [25] show how to extend this to independent probabilities, and we have extended it to disjoint-independent probabilities [16]:

Theorem 3.2 *Computing* $\mathbf{P}(\Phi)$ *for a disjoint-independent probability space* $\mathbf{P}$ *and a DNF formula* Φ *has a FPTRAS.*

We now establish the connection between the probability of Boolean expressions and the probability of a query on a disjoint-independent database. Let $PDB = (T, \mathbf{P})$ be a database with possible tuples $T = \{t_1, \ldots, t_n\}$. Associate a Boolean variable X_i to each tuple t_i and define a disjoint-independent probability space on their truth assignments by partitioning the variables X_i according

to their keys (X_i, X_j are in the same partition iff $Key(t_i) = Key(t_j)$), and by defining $\mathbf{P}(X_i) = \mathbf{P}(t_i)$. This creates a one-to-one correspondence between the possible worlds of PDB and the truth assignments 2^X, which preserves the probabilities.

Consider a Boolean query, q, expressed in First Order Logic (FO) over the vocabulary $\mathcal{R}$. The *intensional formula* associated to q and database PDB is a Boolean formula Φ_q^{PDB}, or simply Φ_q when PDB is understood from the context, defined inductively as follows:

$$\begin{aligned}
\Phi_{R(\bar{a})} &= \begin{cases} X_i & \text{if } R(\bar{a}) = t_i \in T \\ false & \text{if } R(\bar{a}) \notin T \end{cases} \\
\Phi_{true} &= true \quad \Phi_{false} = false \quad \Phi_{\neg q} = \neg(\Phi_q) \\
\Phi_{q_1 \wedge q_2} &= \Phi_{q_1} \wedge \Phi_{q_2} \quad \Phi_{q_1 \vee q_2} = \Phi_{q_1} \vee \Phi_{q_2} \\
\Phi_{\exists x.q(x)} &= \bigvee_{a \in D} \Phi_{q[a/x]} \quad \Phi_{\forall x.q(x)} = \bigwedge_{a \in D} \Phi_{q[a/x]}
\end{aligned}$$

Here D represents the active domain of PDB (*i.e.* all constants occurring in any possible tuple in T), $q[a/x]$ denotes the formula q where the variable x is substituted with a, and interpreted predicates over constants (*e.g.* $a < b$ or $a = b$) are replaced by $true$ or $false$ respectively. If q has v variables, then the size of Φ_q is $O(|q| \cdot |D|^v)$. The connection between Boolean formulas and Boolean queries is:

Proposition 3.3 *For every query q and any database $PDB = (T, \mathbf{P})$, $\mathbf{P}(q) = \mathbf{P}(\Phi_q)$.*

A *Boolean conjunctive query* is a formula of the form $q = \exists \bar{x}.g_1 \wedge \ldots \wedge g_k$, where each g_i is a positive atomic literal $R_j(\ldots)$, called a *subgoal*. We write q as:

$$q = g_1, g_2, \ldots, g_k \tag{1}$$

Obviously, in this case Φ_q is a positive DNF expression. For a simple illustration, suppose $q = R(x), S(x, y)$ and that we have five possible tuples: $t_1 = R(a)$, $t_2 = R(b)$, $t_3 = S(a, c)$, $t_4 = S(a, d)$, $t_5 = S(b, d)$ to which we associate the Boolean variables X_1, X_2, Y_1, Y_2, Y_3, then $\Phi_q = X_1Y_1 \vee X_1Y_2 \vee X_2Y_3$. Our discussion implies:

Theorem 3.4 *For a fixed a Boolean query q, computing $\mathbf{P}(q)$ on a disjoint-independent database (1) is in #P, (2) admits a FPTRAS when q if conjunctive [25].*

A Dichotomy for Queries without Self-joins

We now establish the following dichotomy for queries without self-joins: computing $\mathbf{P}(q)$ is either #P-hard or is in PTIME in the size of the database

$PDB = (T, \mathbf{P})$. A query conjunctive q is said to be without self-joins if each relational symbol occurs at most once in the query body [15, 14]. For example $R(x, y), R(y, z)$ has self-joins, $R(x, y), S(y, z)$ does not.

Theorem 3.5 *For each of the queries below (where $k, m \geq 1$), computing $\mathbf{P}(q)$ is #P-hard in the size of the database:*

$$\begin{aligned} h_1 &= R(\underline{x}), S(\underline{x, y}), T(\underline{y}) \\ h_2^+ &= R_1(\underline{x}, y), \ldots, R_k(\underline{x}, y), S(\underline{y}) \\ h_3^+ &= R_1(\underline{x}, y), \ldots, R_k(\underline{x}, y), S_1(x, \underline{y}), \ldots, S_m(x, \underline{y}) \end{aligned}$$

The underlined positions represent the key attributes, thus, in h_1 the database is tuple independent, while in h_2^+, h_3^+ it is disjoint-independent. When $k = m = 1$ then we omit the + superscript and write:

$$\begin{aligned} h_2 &= R(\underline{x}, y), S(\underline{y}) \\ h_3 &= R(\underline{x}, y), S(x, \underline{y}) \end{aligned}$$

The significance of these three (classes of) queries is that the hardness of any other conjunctive query without self-joins follows from a simple reduction from one of these three (Lemma 3.7). By contrast, the hardness of these three queries is shown directly [16] (by reducing Positive Partitioned 2DNF [40] to h_1, and PERMANENT [47] to h_2^+, h_3^+) and these proofs are more involved.

Previously, the complexity has been studied only for independent probabilistic databases. De Rougemont [18] claimed that it is is in PTIME. Grädel at al. [18, 25] corrected this and proved that the query $R(\underline{x}), R(\underline{y}), S_1(\underline{x, z}), S_2(\underline{y, z})$ is #P-hard, by reduction from regular (non-partitioned) 2DNF: note that this query has a self-join (R occurs twice); h_1 does not have a self-join, and was first shown to be #P-hard in [15]. The hardness of h_2^+ and h_3^+ was discussed in [14, 16].

A PTIME Algorithm We describe here an algorithm that evaluates $\mathbf{P}(q)$ in polynomial time in the size of the database, which works for some queries, and fails for others. We need some notations. $Vars(q)$ and $Sg(q)$ are the set of variables, and the set of subgoals respectively. If $g \in Sg(q)$ then $Vars(g)$ and $KVars(g)$ denote all variables in g, and all variables in the key positions in g: *e.g.* for $g = R(\underline{x, a, y}, x, z)$, $Vars(g) = \{x, y, z\}$, $KVars(g) = \{x, y\}$. For $x \in Vars(q)$, let $sg(x) = \{g \mid g \in Sg(q), x \in KVars(g)\}$. Given a database $PDB = (T, \mathbf{P})$, D is its active domain (*i.e.* the set of all constants that occur in any of the tuples in T).

Algorithm 3.1 computes $\mathbf{P}(q)$ by recursion on the structure of q. If q consists of two connected components q_1, q_2, then it returns $\mathbf{P}(q_1)\mathbf{P}(q_2)$, *e.g.* $\mathbf{P}(R(\underline{x}), S(\underline{y}, z), T(\underline{y})) = \mathbf{P}(R(\underline{x}))\mathbf{P}(S(\underline{y}, z), T(\underline{y}))$. If some variable x occurs in a key position in all subgoals, then it applies the independent-project rule: *e.g.* $\mathbf{P}(R(\underline{x})) = 1 - \prod_{a \in D}(1 - \mathbf{P}(R(\underline{a})))$ (this is the probability that R is nonempty). For another example, we apply an independent project on x in

Algorithm 3.1 `Safe-Eval`
Input: query q and database $PDB = (T, \mathbf{P})$
Output: $\mathbf{P}(q)$

1: **Base Case: if** $q = R(\bar{a})$
return if $R(\bar{a}) \in T$ **then** $\mathbf{P}(R(\bar{a}))$ **else 0**
2: **Join: if** $q = q_1, q_2$ and $Vars(q_1) \cap Vars(q_2) = \emptyset$
return $\mathbf{P}(q_1)\mathbf{P}(q_2)$
3: **Independent project: if** $sg(x) = Sg(q)$
return $1 - \prod_{a \in D}(1 - \mathbf{P}(q[a/x]))$
4: **Disjoint project: if** $\exists g(x \in Vars(g), KVars(g) = \emptyset)$
return $\sum_{a \in D} \mathbf{P}(q[a/x])$
5: **Otherwise: FAIL**

$q = R(\underline{x}, y), S(\underline{x, y})$: this is correct because $q[a/x]$ and $q[b/x]$ are independent events whenever $a \neq b$. If there exists a subgoal g whose key positions are constants, then it applies a disjoint project on any variable in g: *e.g.* x is such a variable in $q = R(\underline{x}, y), S(\underline{c, d}, x)$, and any two events $q[a/x]$, $q[b/x]$ are disjoint because of the S subgoal.

We call a query *safe* if algorithm `Safe-Eval` terminates successfully; otherwise we call it *unsafe*. Safety is a property that depends only on the query q, not on the database PDB, and it can be checked in PTIME in the size of q by simply running the algorithm over a domain of size 1, $D = \{a\}$.

Proposition 3.6 *[16] For any safe query q, the algorithm computes correctly $\mathbf{P}(q)$ and runs in time $O(|q| \cdot |D|^{|Vars(q)|})$.*

We first described `Safe-Eval` in [14], in a format more suitable for an implementation, by translating q into an algebra plan using joins, independent projects, and disjoint projects. Andritsos et al. [3] describe a query evaluation algorithm similar in spirit but for a more restricted class of queries.

The Dichotomy Property We define below a rewrite rule $q \Rightarrow q'$ between two queries, where g, g' denote subgoals. The intuition is that if $q \Rightarrow q'$ then evaluating $\mathbf{P}(q')$ can be reduced in polynomial time to evaluating $\mathbf{P}(q)$.

$(R1)$	q	$\Rightarrow$	$q[a/x]$	if $x \in Vars(q), a \in D$
$(R2)$	q	$\Rightarrow$	q_1	if $q = q_1, q_2, Vars(q_1) \cap Vars(q_2) = \emptyset$
$(R3)$	q	$\Rightarrow$	$q[y/x]$	if $\exists g \in Sg(q), x, y \in Vars(g)$
$(R4)$	q, g	$\Rightarrow$	q	if $KVars(g) = Vars(g)$
$(R5)$	q, g	$\Rightarrow$	q, g'	if $KVars(g') = KVars(g)$, $Vars(g') = Vars(g), arity(g') < arity(g)$

The rules are designed such that whenever $q \Rightarrow q'$ then the evaluation of q' on a disjoint-independent database PDB' can be easily reduced to the evaluation of q on some other disjoint-independent database PDB. For example, rule $(R4)$ allows us to drop a subgoal g: to evaluate q' on PDB' simply add a relation for

g and make it deterministic by setting all tuple probabilities $\mathbf{P}(t) = 1$. This is possible because $KVar(g) = Var(g)$, *i.e.* all variables occur some key position, like in $R(\underline{x, y, z}, x, a)$: for every key value we need only one possible tuple with that key value, hence we can set its probability = 1. Similarly, rule $(R5)$ allows us to replace a subgoal g with another subgoal g' by dropping some variables and/or constants, provided that g' has the same set of variables in key positions, and the same set of variables (in key and non-key positions).

Lemma 3.7 *If $q \Rightarrow^* q'$ and q' is #P-hard, then q is #P-hard.*

Thus, $\Rightarrow$ gives us a convenient tool for checking if a query is hard, by trying to rewrite it to one of the known hard queries. For example, consider the queries q and q' below: `Safe-Eval` fails immediately on both queries, *i.e.* none of its cases apply. We show that both are hard by rewriting them to h_1 and h_3^+ respectively. By abuse of notations we reuse the same relation name for g and g' when applying rule $(R5)$ (*e.g.* S in the second line and S in the third line should strictly speaking be different relation symbols):

$$\begin{array}{lcll}
 & q & = & R(\underline{x}), R'(\underline{x}), S(\underline{x, y}, y), T(\underline{y, z, b}) & \\
(R4) & & \Rightarrow & R(\underline{x}), S(\underline{x, y}, y), T(\underline{y, z, b}) & \\
(R5) & & \Rightarrow^* & R(\underline{x}), S(\underline{x, y}), T(\underline{y}) & = h_1 \\
 & q' & = & R(\underline{x}, y), S(\underline{y}, z), T(\underline{z}, x), U(\underline{y}, x) & \\
(R3) & & \Rightarrow & R(\underline{x}, y), S(\underline{y}, x), T(\underline{x}, x), U(\underline{y}, x) & \\
(R4) & & \Rightarrow & R(\underline{x}, y), S(\underline{y}, x), U(\underline{y}, x) & = h_3^+
\end{array}$$

Call a query q *final* if it is unsafe, and $\forall q'$, if $q \Rightarrow q'$ then q' is safe. Clearly every unsafe query rewrites to a final query: simply apply $\Rightarrow$ repeatedly until all rewritings are to safe queries. One can prove:

Lemma 3.8 *h_1, h_2^+, h_3^+ are the only final queries.*

This implies immediately the dichotomy property:

Theorem 3.9 *Let q be a query without self-joins. Then one of the following holds:*

- *q is unsafe and q rewrites to one of h_1, h_2^+, h_3^+. In particular, q is #P-hard.*
- *q is safe. In particular, it is in PTIME.*

Proof: We need to show that q is unsafe iff it rewrites to one of h_1, h_2^+, or h_3^+. The important direction results from Lemma 3.8 and our discussion above: if q is unsafe, then it rewrites to one of these three queries. For other direction we need to show that if q is safe, then q does not rewrite to any of the queries h_1, h_2^+, or h_3^+. Note that this follows trivially if one assumes PTIME $\neq$ #P. A direct proof proceeds by induction on the number of recursion steps taken by Algorithm `Safe-Eval` on the query q. The Base Case in the algorithm ($q = R(\bar{a})$) is obvious: q does not rewrite to anything. For the Join case, $q = q_1, q_2$, we

note that if q rewrites to one of the three h-queries then so must either q_1 or q_2. For the Independent project case, there is some variable x occurring in key positions in all subgoals of q. If $q \Rightarrow^* h_i$, for some $i = 1, 2, 3$ then that variable must be removed somehow, and the only rule that can eliminate it is $(R1)$, which substitutes x with a constant. But then one can prove that $q[a/x] \Rightarrow^* h_i$, contradicting the induction hypothesis. For the Disjoint Project case, we note that the subgoal g has no variables in key positions, hence it must be eliminated by rule $(R2)$ (If rule $(R4)$ ever applies to g, then Rule $(R2)$ also applies, since $KVar(g) = \emptyset$ implies $Var(g) = \emptyset$, hence g is disconnected from the rest of the query.) In other words, x belongs to a connected component that is eliminated during the rewriting $q \Rightarrow^* h_i$, hence we also have $q[a/x] \Rightarrow^* h_i$, contradiction. □

The Complexity of the Complexity We complete our analysis by studying the following problem: given a relational schema $\mathcal{R}$ and conjunctive query q without self-joins over $\mathcal{R}$, decide whether q is safe[1]. We have seen that this problem is in PTIME; here we establish tighter bounds:

Proposition 3.10 *[16] (1) The problem: "given an independent schema $\mathcal{R}$ (i.e. $Key(R_i) = Attr(R_i)$ for all $R_i \in \mathcal{R}$) and a query q, decide whether q is safe" is in AC^0. (2) The problem: "given a disjoint-independent schema $\mathcal{R}$ (i.e. $Key(R_i) \subseteq Attr(R_i)$ for all $R_i \in \mathcal{R}$) and a query q, decide whether q is safe" is PTIME complete.*

4 Conclusions and Future Challenges

In many applications today it is becoming prohibitively expensive, and even impossible, to enforce a precise semantics, by completely cleaning the data and removing all types of uncertainties. Probabilistic databases manage data with an explicit representation and quantification of uncertainties, and query evaluation becomes a special form of probabilistic inference.

Probabilistic databases need to scale both in the size of the data, and in the complexity of the probabilistic model. Scaling in any of these two dimensions is a challenging problem. Consider the size of the data. Most often the data does not fit in main memory, but instead needs to be read from disk, sequentially or using some index. One approach to scale query processing is to restrict the query evaluation on only the top k answers: that is, the answers to a (non-Boolean) query are ranked in decreasing order of their probability score, and only the top k are returned to the user, where k is a small number, typically 10 or 20. While there may be many (*e.g.* thousands) of possible answers, the systems computes only the top k, and thus needs to do far fewer probabilistic inferences. But this creates a chicken and egg problem: we do not know which tuples are the top k until we have evaluated all probabilities [41].

In addition, probabilistic databases need to handle more complex probabilistic models than the disjoint-independent databases discussed in this paper.

[1]For a fixed $\mathcal{R}$ there are only finitely many queries without self-joins: this is the reason why $\mathcal{R}$ is part of the input.

Correlations between tuples or attribute values occur naturally in practice: for example in information extraction, the extraction of one data item may depend on the extraction of a previous item, hence the two items are correlated probabilistic events [28]. The explicit representation of such correlations in probabilistic databases is called *lineage*, or *provenance* [7, 26], and it complicates significantly the query evaluation problem. Some simple database optimization techniques, such as predicate pushdown or semijoin reduction, still help, but they are limited in scope. More powerful optimization techniques currently used in database systems, such as using materialized views during query processing, become difficult to use because of the extra cost of the lineage information. A new line of research seeks to simplify the lineage information by either finding an equivalent but simpler lineage [42], or by using Fourier approximations of Boolean expressions to approximate the lineage [43].

Acknowledgments The author was partially supported by NSF Grants IIS-0713576, IIS-0513877, IIS-0627585 IIS-0428168, IIS-0454425 and a Gift from Microsoft.

References

[1] S. Abiteboul and P. Senellart. Querying and updating probabilistic information in XML. In *EDBT*, pages 1059–1068, 2006.

[2] E. Adams. *A Primer of Probability Logic*. CSLI Publications, Stanford, California, 1998.

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

[4] L. Antova, C. Koch, and D. Olteanu. 10^(10^6) worlds and beyond: Efficient representation and processing of incomplete information. In *ICDE*, 2007.

[5] L. Antova, C. Koch, and D. Olteanu. World-set decompositions: Expressiveness and efficient algorithms. In *ICDT*, pages 194–208, 2007.

[6] F. Bacchus, A. Grove, J. Halpern, and D. Koller. From statistical knowledge bases to degrees of belief. *Artificial Intelligence*, 87(1-2):75–143, 1996.

[7] O. Benjelloun, A. Das Sarma, A. Halevy, and J. Widom. ULDBs: Databases with uncertainty and lineage. In *VLDB*, pages 953–964, 2006.

[8] G. Borriello and F. Zhao. World-Wide Sensor Web: 2006 UW-MSR Summer Institute Semiahmoo Resort, Blaine, WA, 2006. `www.cs.washington.edu/mssi/2006/schedule.html`.

[9] T. Choudhury, M. Philipose, D. Wyatt, and J. Lester. Towards activity databases: Using sensors and statistical models to summarize people's lives. *IEEE Data Eng. Bull*, 29(1):49–58, March 2006.

[10] E. F. Codd. A relational model for large shared databank. *Communications of the ACM*, 13(6):377–387, June 1970.

[11] G. Cooper. Computational complexity of probabilistic inference using bayesian belief networks (research note). *Artificial Intelligence*, 42:393–405, 1990.

[12] R. Cowell, P. Dawid, S. Lauritzen, and D. Spiegelhalter, editors. *Probabilistic Networks and Expert Systems*. Springer, 1999.

[13] P. Dagum and M. Luby. Approximating probabilistic inference in bayesian belief networks is NP-hard. *Artificial Intelligence*, 60:141–153, 1993.

[14] N. Dalvi, C. Re, and D. Suciu. Query evaluation on probabilistic databases. *IEEE Data Engineering Bulletin*, 29(1):25–31, 2006.

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

[16] N. Dalvi and D. Suciu. Management of probabilistic data: Foundations and challenges. In *PODS*, pages 1–12, Beijing, China, 2007. (invited talk).

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

[18] M. de Rougemont. The reliability of queries. In *PODS*, pages 286–291, 1995.

[19] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, pages 588–599, 2004.

[20] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Using probabilistic models for data management in acquisitional environments. In *CIDR*, pages 317–328, 2005.

[21] O. Deux. The story of O_2. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):91–108, March 1990.

[22] A. Doan, R. Ramakrishnan, F. Chen, P. DeRose, Y. Lee, R. McCann, M. Sayyadian, and W. Shen. Community information management. *IEEE Data Engineering Bulletin, Special Issue on Probabilistic Data Management*, 29(1):64–72, March 2006.

[23] M. B. et al. Data management in the world-wide sensor web. *IEEE Pervasive Computing*, 2007. To appear.

[24] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A. Popescu, T. Shaked, S. Soderland, D. Weld, and A. Yates. Web-scale information extraction in KnowItAll: (preliminary results). In *WWW*, pages 100–110, 2004.

[25] E. Grädel, Y. Gurevich, and C. Hirsch. The complexity of query reliability. In *PODS*, pages 227–234, 1998.

[26] T. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.

[27] T. Green and V. Tannen. Models for incomplete and probabilistic information. *IEEE Data Engineering Bulletin*, 29(1):17–24, March 2006.

[28] R. Gupta and S. Sarawagi. Creating probabilistic databases from information extraction models. In *VLDB*, pages 965–976, 2006.

[29] A. Halevy, A. Rajaraman, and J. Ordille. Data integration: The teenage years. In *VLDB*, pages 9–16, 2006.

[30] D. Heckerman. Tutorial on graphical models, June 2002.

[31] E. Hung, L. Getoor, and V. Subrahmanian. PXML: A probabilistic semistructured data model and algebra. In *ICDE*, 2003.

[32] T. Imielinski and W. Lipski. Incomplete information in relational databases. *Journal of the ACM*, 31:761–791, October 1984.

[33] T. Jayram, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Zhu. Avatar information extraction system. *IEEE Data Engineering Bulletin*, 29(1):40–48, 2006.

[34] R. Karp and M. Luby. Monte-Carlo algorithms for enumeration and reliability problems. In *Proceedings of the annual ACM symposium on Theory of computing*, 1983.

[35] N. Khoussainova, M. Balazinska, and D. Suciu. Towards correcting input data errors probabilistically using integrity constraints. In *MobiDB*, pages 43–50, 2006.

[36] G. Kuper, L. Libkin, and J. P. (Eds.). *Constraint Databases*. Springer, 2000.

[37] J. Lester, T. Choudhury, N. Kern, G. Borriello, and B. Hannaford. A hybrid discriminative/generative approach for modeling human activities. In *IJCAI*, pages 766–772, 2005.

[38] C. Papadimitriou. *Computational Complexity*. Addison Wesley Publishing Company, 1994.

[39] S. Philippi and J. Kohler. Addressing the problems with life-science databases for traditional uses and systems biology. *Nature Reviews Genetics*, 7:481–488, June 2006.

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

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

[42] C. Re and D.Suciu. Materialized views in probabilistic databases for information exchange and query optimization. In *Proceedings of VLDB*, 2007.

[43] C. Re and D. Suciu. Approximate lineage for probabilistic databases. Technical Report 2008-03-02, University of Washington, Seattle, WA, 2008.

[44] H.-J. Schek and M. H. Scholl. The relational model with relation-valued attributes. *Information Systems*, 11(2):137–147, 1986.

[45] R. Snodgrass. The temporal query language TQUEL. *ACM Transactions on Database Systems*, 12(2):247–298, June 1987.

[46] D. Suciu. An overview of semistructured data. *SIGACT News*, 29(4):28–38, December 1998.

[47] L. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8:410–421, 1979.

[48] M. van Keulen, A. de Keijzer, and W. Alink. A probabilistic XML approach to data integration. In *ICDE*, pages 459–470, 2005.

[49] M. Y. Vardi. The complexity of relational query languages. In *Proceedings of 14th ACM SIGACT Symposium on the Theory of Computing*, pages 137–146, San Francisco, California, 1982.