MayBMS: A System for Managing Large Uncertain and Probabilistic Databases*

Christoph Koch

Department of Computer Science
Cornell University, Ithaca, NY
koch@cs.cornell.edu

Abstract

MayBMS is a state-of-the-art probabilistic database management system that has been built as an extension of Postgres, an open-source relational database management system. MayBMS follows a principled approach to leveraging the strengths of previous database research for achieving scalability. This article describes the main goals of this project, the design of query and update language, efficient exact and approximate query processing, and algorithmic and systems aspects.

Acknowledgments. My collaborators on the MayBMS project are Dan Olteanu (Oxford University), Lyublena Antova (Cornell), Jiewen Huang (Oxford), and Michaela Goetz (Cornell). Thomas Jansen and Ali Baran Sari are alumni of the MayBMS team. I thank Dan Suciu for the inspirational talk he gave at a Dagstuhl seminar in February of 2005, which triggered my interest in probabilistic databases and the start of the project. I am also indebted to Joseph Halpern for insightful discussions. The project was previously supported by German Science Foundation (DFG) grant KO 3491/1-1 and by funding provided by the Center for Bioinformatics (ZBI) at Saarland University. It is currently supported by NSF grant IIS-0812272, a KDD grant, and a gift from Intel.

1 Introduction

Database systems for uncertain and probabilistic data promise to have many applications. Query processing on uncertain data occurs in the contexts of data warehousing, data integration, and of processing data extracted from the Web. Data cleaning can be fruitfully approached as a problem of reducing uncertainty in data and requires the management and processing of large amounts of uncertain data. Decision support and diagnosis systems employ hypothetical (what-if) queries. Scientific databases, which store outcomes of scientific experiments, frequently contain uncertain data such as incomplete observations or imprecise measurements. Sensor and RFID data is inherently uncertain. Applications in the contexts of fighting crime or terrorism, tracking moving objects, surveillance, and plagiarism detection essentially rely on techniques for processing and managing large uncertain datasets. Beyond that, many further potential applications of probabilistic databases exist and will manifest themselves once such systems become available.

*This article will appear as Chapter 6 of Charu Aggarwal, ed., *Managing and Mining Uncertain Data*, Springer-Verlag, 2008/9.

Inference in uncertain data is a field in which the Artificial Intelligence research community has made much progress in the past years. Some of the most exciting AI applications, such as using graphical models in biology, belong to this area. While a number of papers on uncertain data and probabilistic databases have been written within the data management research community over the past decades, this area has moved into the focus of research interest only very recently, and work on scalable systems has only just started.

The *MayBMS* project[1] aims at creating a probabilistic database management system by leveraging techniques developed by the data management research community. The MayBMS project is founded on the thesis that a principled effort to use previous insights from databases will allow for substantial progress towards more robust and scalable systems for managing and querying large uncertain datasets. This will have a positive impact on current applications such as in computational science and will allow for entirely new data management applications.

Central themes in our research include the creation of foundations of query languages for probabilistic databases by developing analogs of relational algebra [22, 21] and SQL [6, 8] and the development of efficient query processing techniques [5, 25, 3, 23, 24, 17]. In practice, the efficient evaluation of queries on probabilistic data requires approximation techniques, and another important goal was to understand which approximation guarantees can be made for complex, realistic query languages [22, 15].

We have worked on developing a complete database management system for uncertain and probabilistic data. Apart from data representation and storage mechanisms, a query language, and query processing techniques, our work covers query optimization, an update language, concurrency control and recovery, and APIs for uncertain data.

MayBMS stands alone as a complete probabilistic database management system that supports a very powerful, compositional query language for which nevertheless worst-case efficiency and result quality guarantees can be made. Central to this is our choice of essentially using probabilistic versions of conditional tables [18] as the representation system, but in a form engineered for admitting the efficient evaluation and automatic optimization of most operations of our language using robust and mature relational database technology [3].

The structure of this article is as follows. Section 2 sketches our model of probabilistic databases. Section 3 outlines desiderata that have guided the design of our query languages. Section 4 introduces our query algebra and illustrates it by examples. The section also gives an overview over theoretical results, in particular on expressiveness, that have been achieved for this algebra. Section 5 introduces U-relations, the representation system of MayBMS. Section 6 shows how most of the operations of our algebra can be evaluated efficiently using mature relational database techniques. Moreover, the problem of efficiently processing the remaining operations is discussed and an overview of the known results on the (worst-case) complexity of the query algebra is given. Section 7 presents the query and update language of MayBMS, which is based on our algebra but uses an extension of SQL as syntax. Section 8 discusses further systems issues. Section 9 concludes.

This article is meant to provide an overview over the MayBMS project and some topics are covered in a sketchy fashion. For details on the various techniques, experiments, and the theoretical contributions, the reader is referred to the original technical papers on MayBMS that can be found in the references.

[1] MayBMS is read as "maybe-MS", like DBMS.

2 Probabilistic Databases

Informally, our model of probabilistic databases is the following. The schema of a probabilistic database is simply a relational database schema. Given such a schema, a probabilistic database is a finite set of database instances of that schema (called possible worlds), where each world has a weight (called probability) between 0 and 1 and the weights of all worlds sum up to 1. In a subjectivist Bayesian interpretation, one of the possible worlds is "true", but we do not know which one, and the probabilities represent degrees of belief in the various possible worlds. Note that this is only the conceptual model. The physical representation of the set of possible worlds in the MayBMS system is quite different (see Section 5).

Given a schema with relation names $R_1, \ldots, R_k$. We use $sch(R_l)$ to denote the attributes of relation schema R_l. Formally, a *probabilistic database* is a *finite* set of structures

$$\mathbf{W} = \{\langle R_1^1, \ldots, R_k^1, p^{[1]}\rangle, \ldots, \langle R_1^n, \ldots, R_k^n, p^{[n]}\rangle\}$$

of relations $R_1^i, \ldots, R_k^i$ and numbers $0 < p^{[i]} \leq 1$ such that

$$\sum_{1 \leq i \leq n} p^{[i]} = 1.$$

We call an element $\langle R_1^i, \ldots, R_k^i, p^{[i]}\rangle \in \mathbf{W}$ a *possible world*, and $p^{[i]}$ its probability. We use superscripts for indexing possible worlds. To avoid confusion with exponentiation, we sometimes use bracketed superscripts $\cdot^{[i]}$. We call a relation R *complete* or *certain* if its instantiations are the same in all possible worlds of $\mathbf{W}$, i.e., if $R^1 = \cdots = R^n$.

Tuple *confidence* refers to the probability of the event $\vec{t} \in R$, where R is one of the relation names of the schema, with

$$\Pr[\vec{t} \in R] = \sum_{1 \leq i \leq n:\, \vec{t} \in R^i} p^{[i]}.$$

3 Query Language Desiderata

At the time of writing this, there is no accepted standard query language for probabilistic databases. In fact, we do not even agree today what use cases and functionality such systems should support. It seems to be proper to start the query language discussion with the definition of design *desiderata*. The following are those used in the design of MayBMS.

1. Efficient query evaluation.

2. The right degree of expressive power. The language should be powerful enough to support important queries. On the other hand, it should not be too strong, because expressiveness generally comes at a price: high evaluation complexity and infeasibility of query optimization. Can a case be made that some language is in a natural way a probabilistic databases analog of the relationally complete languages (such as relational algebra) – an expressiveness yardstick?

3. Genericity. The semantics of a query language should be independent from details of how the data is represented. Queries should behave in the same way no matter how the probabilistic data is stored. This is a basic requirement that is even part of the traditional definition of what constitutes a query (cf. e.g. [1]), but it is nontrivial to achieve for probabilistic databases [6, 4].

4. The ability to transform data. Queries on probabilistic databases are often interpreted quite narrowly in the literature. It is the author's view that queries in general should be compositional mappings between databases, in this case probabilistic databases. This is a property taken for granted in relational databases. It allows for the definition of clean database update languages.

5. The ability to introduce additional uncertainty. This may appear to be a controversial goal, since uncertainty is commonly considered undesirable, and probabilistic databases are there to deal with it by providing useful functionality *despite* uncertainty. However, it can be argued that an uncertainty-introduction operation is important for at least three reasons: (1) for compositionality, and to allow construction of an uncertain database from scratch (as part of the update language); (2) to support what-if queries; and (3) to extend the hypothesis space modeled by the probabilistic database. The latter is needed to accommodate the results of experiments or new evidence, and to define queries that map from prior to posterior probabilistic databases. This is a nontrivial issue, and will be discussed in more detail later.

The next section introduces a query algebra and argues that it satisfies each of these desiderata.

4 The Algebra

This section covers the core query algebra of MayBMS: *probabilistic world-set algebra* (probabilistic WSA) [6, 22, 21]. Informally, probabilistic world-set algebra consists of the operations of relational algebra, an operation for computing tuple confidence conf, and the repair-key operation for *introducing* uncertainty. The operations of relational algebra are evaluated individually, in "parallel", in each possible world. The operation conf(R) computes, for each tuple that occurs in relation R in at least one world, the sum of the probabilities of the worlds in which the tuple occurs. The result is a certain relation, or viewed differently, a relation that is the same in all possible worlds. Finally, repair-key$_{\vec{A}@P}(R)$, where $\vec{A}, P$ are attributes of R, conceptually nondeterministically chooses a maximal repair of key $\vec{A}$. This operation turns a possible world R^i into the set of worlds consisting of all possible *maximal repairs* of key $\vec{A}$. A repair of key $\vec{A}$ in relation R^i is a subset of R^i for which $\vec{A}$ is a key. It uses the numerically-valued column P for weighting the newly created alternative repairs.

Formally, probabilistic world-set algebra consists of the following operations:

- The operations of relational algebra (selection σ, projection π, product $\times$, union $\cup$, difference $-$, and attribute renaming ρ), which are applied in each possible world independently.

  The semantics of operations Θ on probabilistic database $\mathbf{W}$ is $[\![\Theta(R_l)]\!](\mathbf{W}) := \{\langle R_1, \ldots, R_k, \Theta(R_l), p\rangle \mid \langle R_1, \ldots, R_k, p\rangle \in \mathbf{W}\}$ for unary operations ($1 \leq l \leq k$). For binary operations, the semantics is $[\![\Theta(R_l, R_m)]\!](\mathbf{W}) := \{\langle R_1, \ldots, R_k, \Theta(R_l, R_m), p\rangle \mid \langle R_1, \ldots, R_k, p\rangle \in \mathbf{W}\}$.

  Selection conditions are Boolean combinations of atomic conditions (i.e., negation is permitted even in the positive fragment of the algebra). Arithmetic expressions may occur in atomic conditions and in the arguments of π and ρ. For example, $\rho_{A+B\to C}(R)$

in each world adds up the A and B values of each tuple of R and keeps them in a new C attribute.

- An operation for computing tuple confidence,

$$[\![\text{conf}(R_l)]\!](\mathbf{W}) := \{\langle R_1, \ldots, R_k, S, p\rangle \mid \langle R_1, \ldots, R_k, p\rangle \in \mathbf{W}\}$$

where, w.l.o.g., $P \notin sch(R_l)$, and

$$S = \{\langle \vec{t}, P : \Pr[\vec{t} \in R_l]\rangle \mid \vec{t} \in \bigcup_i R_l^i\},$$

with schema $sch(S) = sch(R_l) \cup \{P\}$. The result of $\text{conf}(R_l)$, the relation S, is the same in all possible worlds, i.e., it is a certain relation.

By our definition of probabilistic databases, each possible world has nonzero probability. As a consequence, conf does not return tuples with probability 0.

For example, on probabilistic database

R^1	A	B
	a	b
	b	c

$p^{[1]} = .3$

R^2	A	B
	a	b
	c	d

$p^{[2]} = .2$

R^3	A	B
	a	c
	c	d

$p^{[3]} = .5$

$\text{conf}(R)$ computes, for each possible tuple, the sum of the weights of the possible worlds in which it occurs, here

conf(R)	A	B	P
	a	b	.5
	a	c	.5
	b	c	.3
	c	d	.7

- An uncertainty-introducing operation, *repair-key*, which can be thought of as sampling a maximum repair of a key for a relation. Repairing a key of a complete relation R means to compute, as possible worlds, all subset-maximal relations obtainable from R by removing tuples such that a key constraint is satisfied. We will use this as a method for constructing probabilistic databases, with probabilities derived from relative weights attached to the tuples of R.

  We say that relation R' is a *maximal repair* of a functional dependency (fd, cf. [1]) for relation R if R' is a maximal subset of R which satisfies that functional dependency, i.e., a subset $R' \subseteq R$ that satisfies the fd such that there is no relation R'' with $R' \subset R'' \subseteq R$ that satisfies the fd.

  Let $\vec{A}, B \in sch(R_l)$. For each possible world $\langle R_1, \ldots, R_k, p\rangle \in \mathbf{W}$, let column B of R contain only numerical values greater than 0 and let R_l satisfy the fd $(sch(R_l) - B) \to sch(R_l)$. Then,

$$[\![\text{repair-key}_{\vec{A}@B}(R_l)]\!](\mathbf{W}) :=$$
$$\Big\{\langle R_1, \ldots, R_k, \pi_{sch(R_l)-B}(\hat{R}_l), \hat{p}\rangle \mid \langle R_1, \ldots, R_k, p\rangle \in \mathbf{W},$$
$$\hat{R}_l \text{ is a maximal repair of fd } \vec{A} \to sch(R_l),$$
$$\hat{p} = p \cdot \prod_{\vec{t} \in \hat{R}_l} \frac{\vec{t}.B}{\sum_{\vec{s} \in R_l : \vec{s}.\vec{A} = \vec{t}.\vec{A}} \vec{s}.B}\Big\}$$

Such a repair operation, apart from its usefulness for the purpose implicit in its name, is a powerful way of constructing probabilistic databases from complete relations.

Example 4.1 Consider the example of tossing a biased coin twice. We start with a certain database

R	Toss	Face	FProb	
	1	H	.4	
	1	T	.6	$p = 1$
	2	H	.4	
	2	T	.6	

that represents the possible outcomes of tossing the coin twice. We turn this into a probabilistic database that represents this information using alternative possible worlds for the four outcomes using the query $S := \text{repair-key}_{\text{Toss@FProb}}(R)$. The resulting possible worlds are

S^1	Toss	Face
	1	H
	2	H

S^2	Toss	Face
	1	H
	2	T

S^3	Toss	Face
	1	T
	2	H

S^4	Toss	Face
	1	T
	2	T

with probabilities $p^{[1]} = p \cdot \frac{.4}{.4+.6} \cdot \frac{.4}{.4+.6} = .16$, $p^{[2]} = p^{[3]} = .24$, and $p^{[4]} = .36$. □

The fragment of probabilistic WSA which excludes the difference operation is called *positive* probabilistic WSA.

Computing possible and certain tuples is redundant with conf:

$$\begin{aligned} \text{poss}(R) &:= \pi_{sch(R)}(\text{conf}(R)) \\ \text{cert}(R) &:= \pi_{sch(R)}(\sigma_{P=1}(\text{conf}(R))) \end{aligned}$$

Example 4.2 A bag of coins contains two fair coins and one double-headed coin. We take one coin out of the bag but do not look at its two faces to determine its type (fair or double-headed) for certain. Instead we toss the coin twice to collect evidence about its type.

We start out with a complete database (i.e., a relational database, or a probabilistic database with one possible world of probability 1) consisting of three relations, Coins, Faces, and Tosses (see Figure 1 for all tables used in this example). We first pick a coin from the bag and model that the coin be either fair or double-headed. In probabilistic WSA this is expressed as

$$R := \text{repair-key}_{\emptyset@\text{Count}}(\text{Coins}).$$

This results in a probabilistic database of two possible worlds,

$$\{\langle \text{Coins}, \text{Faces}, R^f, p^f = 2/3\rangle, \langle \text{Coins}, \text{Faces}, R^{dh}, p^{dh} = 1/3\rangle\}.$$

Coins	Type	Count
	fair	2
	2headed	1

Faces	Type	Face	FProb
	fair	H	.5
	fair	T	.5
	2headed	H	1

Tosses	Toss
	1
	2

R^f	Type
	fair

R^{dh}	Type
	2headed

$S^{f.HH}$	Type	Toss	Face
	fair	1	H
	fair	2	H

$p^{f.HH} = 1/6$

$S^{f.HT}$	Type	Toss	Face
	fair	1	H
	fair	2	T

$p^{f.HT} = 1/6$

S^{dh}	Type	Toss	Face
	2headed	1	H
	2headed	2	H

$p^{dh} = 1/3$

$S^{f.TH}$	Type	Toss	Face
	fair	1	T
	fair	2	H

$p^{f.TH} = 1/6$

$S^{f.TT}$	Type	Toss	Face
	fair	1	T
	fair	2	T

$p^{f.TT} = 1/6$

Ev	Toss	Face
	1	H
	2	H

Q	Type	P
	fair	$(1/6)/(1/2) = 1/3$
	2headed	$(1/3)/(1/2) = 2/3$

Figure 1: Tables of Example 4.2.

The possible outcomes of tossing the coin twice can be modeled as

$$S := \text{repair-key}_{\text{Toss@FProb}}(R \bowtie \text{Faces} \times \text{Tosses}).$$

This turns the two possible worlds into five, since there are four possible outcomes of tossing the fair coin twice, and only one for the double-headed coin.

Let $T := \pi_{\text{Toss,Face}}(S)$. The posterior probability that a coin of type x was picked, given the *evidence* Ev (see Figure 1) that both tosses result in H, is

$$\Pr[x \in R \mid T = Ev] = \frac{\Pr[x \in R \wedge T = Ev]}{\Pr[T = Ev]}.$$

Let A be a relational algebra expression for the Boolean query $T = Ev$. Then we can compute a table of pairs $\langle x, \Pr[x \in R \mid T = Ev]\rangle$ as

$$Q := \pi_{\text{Type},P_1/P_2 \to P}(\rho_{P \to P_1}(\text{conf}(R \times A)) \times \rho_{P \to P_2}(\text{conf}(A))).$$

The prior probability that the chosen coin was fair was 2/3; after taking the evidence from two coin tosses into account, the posterior probability Pr[the coin is fair | both tosses result in H] is only 1/3. Given the evidence from the coin tosses, the coin is now more likely to be double-headed. □

Example 4.3 We redefine the query of Example 4.2 such that repair-key is only applied to certain relations. Starting from the database obtained by computing R, with its two possible worlds, we perform the query $S_0 := \text{repair-key}_{\text{Type,Toss@FProb}}(\text{Faces} \times \text{Tosses})$ to model the possible outcomes of tossing the chosen coin twice. The probabilistic database representing these repairs consists of eight possible worlds, with the two possible R relations

of Example 4.2 and, independently, four possible S_0 relations. Let $S := R \bowtie S_0$. While we now have eight possible worlds rather than five, the four worlds in which the double-headed coin was picked all agree on S with the one world in which the double-headed coin was picked in Example 4.2, and the sum of their probabilities is the same as the probability of that world. It follows that the new definition of S is equivalent to the one of Example 4.2 and the rest of the query is the same. □

Discussion The repair-key operation admits an interesting class of queries: Like in Example 4.2, we can start with a probabilistic database of prior probabilities, add further evidence (in Example 4.2, the result of the coin tosses) and then compute interesting posterior probabilities. The adding of further evidence may require extending the hypothesis space first. For this, the repair-key operation is essential. Even though our goal is not to update the database, we have to be able to introduce uncertainty just to be able to model new evidence – say, experimental data. Many natural and important probabilistic database queries cannot be expressed without the repair-key operation. The coin tossing example was admittedly a toy example (though hopefully easy to understand). Real applications such as diagnosis or processing scientific data involve technically similar questions.

Regarding our desiderata, it is quite straightforward to see that probabilistic WSA is generic (3): see also the proof for the non-probabilistic language in [6]. It is clearly a data transformation query language (4) that supports powerful queries for defining databases. The repair-key operation is our construct for uncertainty introduction (5). The evaluation efficiency (1) of probabilistic WSA is studied in Section 6. The expressiveness desideratum (2) is discussed next.

An expressiveness yardstick In [6] a non-probabilistic version of world-set algebra is introduced. It replaces the confidence operation with an operation poss for computing possible tuples. Using poss, repair-key, and the operations of relational algebra, powerful queries are expressible. For instance, the certain answers of a query on an uncertain database can be computed using poss and difference. Compared to the poss operation described above, the operation of [6] is more powerful. The syntax is $\text{poss}_{\vec{A}}(Q)$, where $\vec{A}$ is a set of column names of Q. The operation partitions the set of possible worlds into the groups of those worlds that agree on $\pi_{\vec{A}}(Q)$. The result in each world is the set of tuples possible in Q within the world's group. Thus, this operation supports the grouping of possible worlds just like the group-by construct in SQL supports the grouping of tuples.

The main focus of [6] is to study the fragment of (non-probabilistic) WSA in which repair-key is replaced by the choice-of operation, definable as $\text{choice-of}_{\vec{A}@P}(R) := R \bowtie \text{repair-key}_{\emptyset@P}(\pi_{\vec{A},P}(R))$. The choice-of operation introduces uncertainty like the repair-key operation, but can only cause a polynomial, rather than exponential, increase of the number of possible worlds. This language has the property that query evaluation on enumerative representations of possible worlds is in PTIME (see Section 6 for more on this). Moreover, it is *conservative* over relational algebra in the sense that any query that starts with a certain database (a classical relational database) and produces a certain database is equivalent to a relational algebra query and can be efficiently rewritten into relational algebra. This is a nontrivial result, because in this language we can produce uncertain intermediate results consisting of many possible worlds using the choice-of operator. This allows us to express and efficiently answer hypothetical (what-if) queries.

(Full non-probabilistic) WSA consists of the relational algebra operations, repair-key,

and $\text{poss}_{\vec{A}}$. In [21], it is shown that WSA precisely captures second-order logic. Leaving aside inessential details about interpreting second-order logic over uncertain databases – it can be done in a clean way – this result shows that a query is expressible in WSA if and only if it is expressible in second-order logic. WSA seems to be the first algebraic (i.e., variable and quantifier-free) language known to have exactly the same expressive power as second-order logic.

More importantly for us, it can be argued that this establishes WSA as the natural analog of relational algebra for uncertain databases. Indeed, while it is well known that useful queries (such as transitive closure or counting queries, cf. [1]) cannot be expressed in it, relational algebra is a very popular expressiveness yardstick for relational query languages (and query languages that are as expressive as relational algebra are called *relationally complete*). Relational algebra is also exactly as expressive as the *domain-independent* first-order queries [1], also known as the *relational calculus*. Second-order logic is just first-order logic extended by (existential) quantification over relations ("Does there exist a relation R such that ϕ holds?", where ϕ is a formula). This is the essence of (what-if) reasoning over uncertain data. For example, the query of Example 4.2 employed what-if reasoning over relations twice via the repair-key operation, first considering alternative choices of coin and then alternative outcomes to coin tossing experiments.

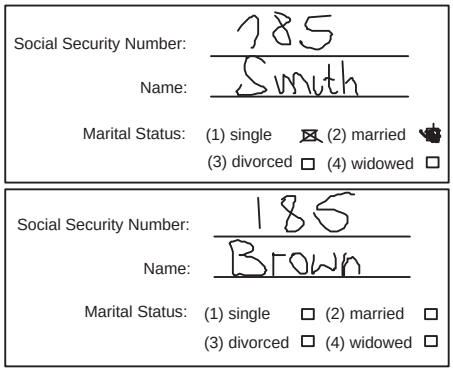
It is unknown whether probabilistic WSA as defined in this article can express all the queries of WSA (with $\text{poss}_{\vec{A}}$). Given the known data complexity bounds for the two languages (see Section 6) alone, there is no reason to assume that this is not the case. On the other hand, it seems unlikely, and a mapping from WSA to probabilistic WSA, if it exists, must be nontrivial.

It would be easy to define a sufficiently strong extension of probabilistic WSA by just generalizing conf to a world-grouping $\text{conf}_{\vec{A}}$ operation. In this article, this is not done because we do not know how to obtain any even just moderately efficient implementation of this operation (or of $\text{poss}_{\vec{A}}$) on succinct data representations.

5 Representing Probabilistic Data

This section discusses the method used for representing and storing probabilistic data and correlations in MayBMS. We start by motivating the problem of finding a practical representation system.

Example 5.1 Consider a census scenario, in which a large number of individuals manually fill in forms. The data in these forms subsequently has to be put into a database, but no matter whether this is done automatically using OCR or by hand, some uncertainty may remain about the correct values for some of the answers. Below are two simple filled in forms. Each one contains the social security number, name, and marital status of one person.



The first person, Smith, seems to have checked marital status "single" after first mistakenly checking "married", but it could also be the opposite. The second person, Brown, did not answer the marital status question. The social security numbers also have several possible readings. Smith's could be 185 or 785 (depending on whether Smith originally is from the US or from Europe) and Brown's may either be 185 or 186.

In an SQL database, uncertainty can be managed using null values, using a table

(TID)	SSN	N	M
t_1	null	Smith	null
t_2	null	Brown	null

Using nulls, information is lost about the values considered possible for the various fields. Moreover, it is not possible to express correlations such as that, while social security numbers may be uncertain, no two distinct individuals can have the same. In this example, we can exclude the case that both Smith and Brown have social security number 185. Finally, we cannot store probabilities for the various alternative possible worlds. □

This leads to three natural desiderata for a representation system: (*) Expressiveness, that is, the power to represent all (relevant) probabilistic databases, (*) succinctness, that is, space-efficient storage of the uncertain data, and (*) efficient real-world query processing.

Often there are many rather (but not quite) independent local alternatives in probabilistic data, which multiply up to a very large number of possible worlds. For example, the US census consists of many dozens of questions for about 300 million individuals. Suppose forms are digitized using OCR and the resulting data contains just two possible readings for 0.1% of the answers before cleaning. Then, there are on the order of $2^{10,000,000}$ possible worlds, and each one will take close to one Terabyte of data to store. Clearly, we need a way of representing this data that is much better than a naive enumeration of possible worlds.

Also, the repair-key operator of probabilistic world-set algebra in general causes an exponential increase in the number of possible worlds.

There is a trade-off between succinctness on one hand and efficient processing on the other. Computing confidence $\mathrm{conf}(Q)$ of conjunctive queries Q on tuple-independent databases is #P-hard – one such hard query [13] (in datalog notation [1]) is

$$Q \leftarrow R(x), S(x, y), T(y).$$

At the same time, much more expressive queries can be evaluated efficiently on nonsuccinct representations (enumerations of possible worlds) [6]. Query evaluation in probabilistic databases is not hard because of the presence of probabilities, but because of the succinct storage of alternative possible worlds! We can still have the goal of doing well in practice.

Conditional tables MayBMS uses a purely relational representation system for probabilistic databases called *U-relational databases*, which is based on probabilistic versions of the classical *conditional tables* (c-tables) of the database literature [18]. Conditional tables are a relational representation system based on the notion of *labeled null values* or *variables*, that is, null values that have a name. The name makes it possible to use the same variable x in several fields of a database, indicating that the value of x is unknown but must be the same in all those fields in which x occurs. Tables with variables are also known as *v-tables*.

Formally, c-tables are v-tables extended by a column for holding a local condition. That is, each tuple of a c-table has a Boolean condition constructed using "and", "or", and "not" from atomic conditions of the form $x = c$ or $x = y$, where c are constants and x and y are *variables*. Possible worlds are determined by functions θ that map each variable that occurs in at least one of the local conditions in the c-tables of the database to a constant. The database in that possible world is obtained by (1) selecting those tuples whose local condition ϕ satisfies the variable assignment θ, i.e., that becomes true if each variable x in ϕ is replaced by $\theta(x)$, (2) replacing all variables y in the value fields of these tuples by $\theta(y)$, and (3) projecting away the local condition column.

Conditional tables are sometimes defined to include a notion of *global condition*, which we do not use: We want each probabilistic database to have at least one possible world.

Conditional tables are a so-called *strong representation system*: They are closed under the application of relational algebra queries. The set of worlds obtained by evaluating a relational algebra query in each possible world represented by a conditional table can again be straightforwardly represented by a conditional table. Moreover, the local conditions are in a sense the most natural and simple formalism possible to represent the result of queries on data with labeled nulls. The local conditions just represent the information necessary to preserve correctness and can also be understood to be just data provenance information [10].

U-Relational Databases In our model, probabilistic databases are finite sets of possible worlds with probability weights. It follows that each variable naturally has a finite domain, the set of values it can take across all possible worlds. This has several consequences. First, variables can be considered *finite random variables*. Second, only allowing for variables to occur in local conditions, but not in attribute fields of the tuples, means no restriction of expressiveness. Moreover, we may assume without loss of generality that each atomic condition is of the form $x = c$ (i.e., we never have to compare variables).

If we start with a c-table in which each local condition is a conjunction of no more than k atomic conditions, then a positive relational algebra query on this uncertain database will result in a c-table in which each local condition is a conjunction of no more than k' atoms, where k' only depends on k and the query, but not on the data. If k is small, it is reasonable to actually hard-wire it in the schema, and represent local conditions by k pairs of columns to store atoms of the form $x = c$.

These are the main ideas of our representation system, U-relations. Random variables are assumed independent in the *current* MayBMS system, but as we will see, this means no restriction of generality. Nevertheless, it is one goal of future work to support graphical models for representing more correlated joint probability distributions below our U-relations. This would allow us to represent *learned* distributions in the form of e.g. Bayesian networks directly in the system (without the need to map them to local conditions) and run queries on top, representing the inferred correlations using local conditions. The latter seem to be

better suited for representing the incremental correlations constructed by queries.

One further idea employed in U-relational databases is to use vertical partitioning [9, 26] for representing *attribute-level uncertainty*, i.e., to allow to decompose tuples in case several fields of a tuple are independently uncertain.

Example 5.2 The following set of tables is a U-relational database representation for the census data scenario of Example 5.1, extended by suitable probabilities for the various alternative values the fields can take (represented by table W).

$U_{R[SSN]}$	V	D	TID	SSN
	x	1	t_1	185
	x	2	t_1	785
	y	1	t_2	185
	y	2	t_2	186

$U_{R[M]}$	V	D	TID	M
	v	1	t_1	1
	v	2	t_1	2
	w	1	t_2	1
	w	2	t_2	2
	w	3	t_2	3
	w	4	t_2	4

$U_{R[N]}$	TID	N
	t_1	Smith
	t_2	Brown

W	V	D	P
	x	1	.4
	x	2	.6
	y	1	.7
	y	2	.3
	v	1	.8
	v	2	.2
	w	1	.25
	w	2	.25
	w	3	.25
	w	4	.25

Formally, a U-relational database consists of a set of independent random variables with finite domains (here, x, y, v, w), a set of U-relations, and a ternary table W (the *world-table*) for representing distributions. The W table stores, for each variable, which values it can take and with what probability. The schema of each U-relation consists of a *set* of pairs (V_i, D_i) of *condition columns* representing variable assignments and a set of *value columns* for representing the data values of tuples.

The semantics of U-relational databases is as follows. Each possible world is identified by a valuation θ that assigns one of the possible values to each variable. The probability of the possible world is the product of weights of the values of the variables. A tuple of a U-relation, stripped of its condition columns, is in a given possible world if its variable assignments are consistent with θ. Attribute-level uncertainty is achieved through vertical decompositioning, so one of the value columns is used for storing tuple ids and undoing the vertical decomposition on demand.

Example 5.3 Consider the U-relational database of Example 5.2 and the possible world

$$\theta = \{x \mapsto 1, y \mapsto 2, v \mapsto 1, w \mapsto 1\}.$$

The probability weight of this world is $.4 \cdot .3 \cdot .8 \cdot .25 = .024$. By removing all the tuples whose condition columns are inconsistent with θ and projecting away the condition columns, we obtain the relations

$R[SSN]$	TID	SSN
	t_1	185
	t_2	186

$R[M]$	TID	M
	t_1	1
	t_2	1

$R[N]$	TID	N
	t_1	Smith
	t_2	Brown

which are just a vertically decomposed version of R in the chosen possible world. That is, R is $R[SSN] \bowtie R[M] \bowtie R[N]$ in that possible world. □

Properties of U-relations U-relational databases are a *complete* representation system for (finite) probabilistic databases [3]. This means that any probabilistic database can be represented in this formalism. In particular, it follows that U-relations are closed under query evaluation using any generic query language, i.e., starting from a represented database, the query result can again be represented as a U-relational database. Completeness also implies that any (finite) correlation structure among tuples can be represented, despite the fact that we currently assume that the random variables that our correlations are constructed from (using tuple conditions) are independent: The intuition that some form of graphical model for finite distributions may be more powerful (i.e., able to represent distributions that cannot be represented by U-relations) is *false*.

Historical Note The first prototype of MayBMS [5, 7, 25] did not use U-relations for representations, but a different representation system called *world-set decompositions* [5]. These representations are based on factorizations of the space of possible worlds. They can also be thought of as shallow Bayesian networks. The problem with this approach is that some selection operations can cause an exponential blowup of the representations. This problem is not shared by U-relations, even though they are strictly more succinct than world-set decompositions. This was the reason for introducing U-relations in [3] and developing a new prototype of MayBMS based on U-relations.

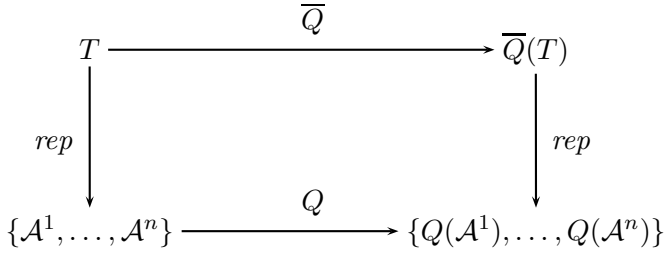
6 Conceptual Query Evaluation, Rewritings, and Asymptotic Efficiency

This section gives a complete solution for efficiently evaluating a large fragment of probabilistic world-set algebra using relational database technology. Then we discuss the evaluation of the remaining operations of probabilistic WSA, namely difference and tuple confidence. Finally, an overview of known worst-case computational complexity results is given.

Translating queries down to the representation relations Let *rep* be the *representation function*, which maps a U-relational database to the set of possible worlds it represents. Our goal is to give a reduction that maps any positive relational algebra query Q over probabilistic databases represented as U-relational databases T to an equivalent positive relational algebra query $\overline{Q}$ of polynomial size such that

$$rep(\overline{Q}(T)) = \{Q(\mathcal{A}^i) \mid \mathcal{A}^i \in rep(T)\}$$

where the $\mathcal{A}^i$ are relational database instances (possible worlds) or, as a commutative diagram,



The following is such a reduction, which maps the operations of positive relational algebra, poss, and repair-key to relational algebra over U-relational representations:

$$\begin{aligned}
[\![R \times S]\!] &:= \pi_{(U_R.\overline{VD} \cup U_S.\overline{VD}) \to \overline{VD}, sch(R), sch(S)}(\\
&\qquad U_R \bowtie_{U_R.\overline{VD} \text{ consistent with } U_S.\overline{VD}} U_S) \\
[\![\sigma_\phi R]\!] &:= \sigma_\phi(U_R) \\
[\![\pi_{\vec{B}} R]\!] &:= \pi_{\overline{VD}, \vec{B}}(R) \\
[\![R \cup S]\!] &:= U_R \cup U_S \\
[\![\text{poss}(R)]\!] &:= \pi_{sch(R)}(U_R).
\end{aligned}$$

The consistency test for conditions can be expressed simply using Boolean conditions (see Example 6.2, and [3]). Note that the product operation, applied to two U-relations of k and l (V_i, D_i) column pairs, respectively, returns a U-relation with $k + l$ (V_i, D_i) column pairs.

For simplicity, let us assume that the elements of $\pi_{\langle \vec{A} \rangle}(U_R)$ are not yet used as variable names. Moreover, let us assume that the B value column of U_R, which is to provide weights for the alternative values of the columns $sch(R) - (\vec{A} \cup B)$ for each tuple $\vec{a}$ in $\pi_{\langle \vec{A} \rangle}(U_R)$, are probabilities, i.e., sum up to one for each $\vec{a}$ and do not first have to be normalized as described in the definition of the semantics of repair-key in Section 4. The operation $S := \text{repair-key}_{\vec{A}@B}(R)$ for complete relation R is translated as

$$U_S := \pi_{\langle \vec{A} \rangle \to V, \langle (sch(R) - \vec{A}) - \{B\} \rangle \to D, sch(R)} U_R$$

with

$$W := W \cup \pi_{\langle \vec{A} \rangle \to V, \langle (sch(R) - \vec{A}) - \{B\} \rangle \to D, B \to P} U_R.$$

Here, $\langle \cdot \rangle$ turns tuples of values into atomic values that can be stored in single fields.

That is, repair-key starting from a complete relation is just a projection/copying of columns, even though we may create an exponential number of possible worlds.

Example 6.1 Consider again the relation R of Example 4.1, which represents information about tossing a biased coin twice, and the query $S := \text{repair-key}_{\text{Toss@FProb}}(R)$. The result is

U_S	V	D	Toss	Face	FProb
	1	H	1	H	.4
	1	T	1	T	.6
	2	H	2	H	.4
	2	T	2	T	.6

W	V	D	P
	1	H	.4
	1	T	.6
	2	H	.4
	2	T	.6

as a U-relational database. □

The projection technique only works if the relation that repair-key is applied to is certain. However, this means no loss of generality (cf. [21], and see also Example 4.3).

The next example demonstrates the application of the rewrite rules to compile a query down to relational algebra on the U-relations.

Example 6.2 We revisit our census example with U-relations $U_{R[SSN]}$ and $U_{R[N]}$. We ask for possible names of persons who have SSN 185, $\text{poss}(\pi_N(\sigma_{SSN=185}(R)))$. To undo the

vertical partitioning, the query is evaluated as $\text{poss}(\pi_N(\sigma_{SSN=185}(R[SSN] \bowtie R[N])))$. We rewrite the query using our rewrite rules into $\pi_N(\sigma_{SSN=185}(U_{R[SSN]} \bowtie_{\psi \wedge \phi} U_{R[N]}))$, where ψ ensures that we only generate tuples that occur in some worlds,

$$\psi := (U_{R[SSN]}.V = U_{R[N]}.V \Rightarrow U_{R[SSN]}.D = U_{R[N]}.D),$$

and ϕ ensures that the vertical partitioning is correctly undone,

$$\phi := (U_{R[SSN]}.TID = U_{R[N]}.TID).$$

□

Properties of the relational-algebra reduction The relational algebra rewriting down to positive relational algebra on U-relations has a number of nice properties. First, since relational algebra has PTIME (even AC_0) data complexity, the query language of positive relational algebra, repair-key, and poss on probabilistic databases represented by U-relations has the same. The rewriting is in fact a *parsimonious translation*: The number of algebra operations does not increase and each of the operations selection, projection, join, and union remains of the same kind. Query plans are hardly more complicated than the input queries. As a consequence, we were able to observe that off-the-shelf relational database query optimizers do well in practice [3].

Thus, for all but two operations of probabilistic world-set algebra, it seems that there is a very efficient solution that builds on relational database technology. These remaining operations are confidence computation and relational algebra difference.

Approximate confidence computation To compute the confidence in a tuple of data values occurring possibly in several tuples of a U-relation, we have to compute the probability of the disjunction of the local conditions of all these tuples. We have to eliminate duplicate tuples because we are interested in the probability of the data tuples rather than some abstract notion of tuple identity that is really an artifact of our representation. That is, we have to compute the probability of a DNF, i.e., the sum of the weights of the worlds identified with valuations θ of the random variables such that the DNF becomes true under θ. This problem is #P-complete [16, 13]. The result is not the sum of the probabilities of the individual conjunctive local conditions, because they may, intuitively, "overlap".

Example 6.3 Consider a U-relation with schema $\{V, D\}$ (representing a nullary relation) and two tuples $\langle x, 1\rangle$, and $\langle y, 1\rangle$, with the W relation from Example 5.2. Then the confidence in the nullary tuple $\langle\rangle$ is $\Pr[x \mapsto 1 \vee y \mapsto 1] = \Pr[x \mapsto 1] + \Pr[y \mapsto 1] - \Pr[x \mapsto 1 \wedge y \mapsto 1] = .82$. □

Confidence computation can be efficiently approximated by Monte Carlo simulation [16, 13, 22]. The technique is based on the Karp-Luby fully polynomial-time randomized approximation scheme (FPRAS) for counting the number of solutions to a DNF formula [19, 20, 12]. There is an efficiently computable unbiased estimator that in expectation returns the probability p of a DNF of n clauses (i.e., the local condition tuples of a Boolean U-relation) such that computing the average of a polynomial number of such Monte Carlo steps (= calls to the Karp-Luby unbiased estimator) is an (ϵ, δ)-approximation for the probability: If the average $\hat{p}$ is taken over at least $\lceil 3 \cdot n \cdot \log(2/\delta)/\epsilon^2 \rceil$ Monte Carlo steps, then $\Pr\big[|p - \hat{p}| \geq \epsilon \cdot p\big] \leq \delta$. The paper [12] improves upon this by determining smaller numbers (within a constant factor from optimal) of necessary iterations to achieve an (ϵ, δ)-approximation.

Avoiding the difference operation Difference $R-S$ is conceptually simple on c-tables. Without loss of generality, assume that S does not contain tuples $\langle \vec{a}, \psi_1 \rangle, \ldots, \langle \vec{a}, \psi_n \rangle$ that are duplicates if the local conditions are disregarded. (Otherwise, we replace them by $\langle \vec{a}, \psi_1 \vee \cdots \vee \psi_n \rangle$.) For each tuple $\langle \vec{a}, \phi \rangle$ of R, if $\langle \vec{a}, \psi \rangle$ is in S then output $\langle \vec{a}, \phi \wedge \neg\psi \rangle$; otherwise, output $\langle \vec{a}, \phi \rangle$. Testing whether a tuple is possible in the result of a query involving difference is already NP-hard [2]. For U-relations, we in addition have to turn $\phi \wedge \neg\psi$ into a DNF to represent the result as a U-relation. This may lead to an exponentially large output and a very large number of $\vec{V}\vec{D}$ columns may be required to represent the conditions. For these reasons, MayBMS currently does not implement the difference operation.

In many practical applications, the difference operation can be avoided. Difference is only hard on uncertain relations. On such relations, it can only lead to displayable query results in queries that close the possible worlds semantics using conf, computing a single certain relation. Probably the most important application of the difference operation is for encoding universal constraints, for example in data cleaning. But if the confidence operation is applied on top of a universal query, there is a trick that will often allow to rewrite the query into an existential one (which can be expressed in positive relational algebra plus conf, without difference) [22].

Example 6.4 The example uses the census scenario and the uncertain relation $R[SSN]$ with columns TID and SSS discussed earlier; below we will call this relation just simply R. Consider the query of finding, for each TID t_i and SSN s, the confidence in the statement that s is the correct SSN for the individual associated with the tuple identified by t_i, assuming that social security numbers uniquely identify individuals, that is, assuming that the functional dependency $SSN \rightarrow TID$ (subsequently called ψ) holds. In other words, the query asks, for each TID t_i and SSN s, to find the probability $\Pr[\phi \mid \psi]$, where $\phi(t_i, s) = \exists t \in R\ t.TID = t_i \wedge t.SSN = s$. Constraint ψ can be thought of as a data cleaning constraint that ensures that the SSN fields in no two distinct census forms (belonging to two different individuals) are interpreted as the same number.

We compute the desired conditional probabilities, for each possible pair of a TID and an SSN, as $\Pr[\phi \mid \psi] = \Pr[\phi \wedge \psi]/\Pr[\psi]$. Here ϕ is existential (expressible in positive relational algebra) and ψ is an equality-generating dependency (i.e., a special universal query) [1]. The trick is to turn relational difference into the subtraction of probabilities, $\Pr[\phi \wedge \psi] = \Pr[\phi] - \Pr[\phi \wedge \neg\psi]$ and $\Pr[\psi] = 1 - \Pr[\neg\psi]$, where $\neg\psi = \exists t, t' \in R\ t.SSN = t'.SSN \wedge t.TID \neq t'.TID$ is existential (with inequalities). Thus $\neg\psi$ and $\phi \wedge \neg\psi$ are expressible in positive relational algebra. This works for a considerable superset of the equality-generating dependencies [22], which in turn subsume useful data cleaning constraints, such as *conditional functional dependencies* [11].

Let $R_{\neg\psi}$ be the relational algebra expression for $\neg\psi$,

$$\pi_{\emptyset}(R \bowtie_{TID=TID' \wedge SSN \neq SSN'} \rho_{TID \rightarrow TID'; SSN \rightarrow SSN'}(R)),$$

and let S be

$$\rho_{P \rightarrow P_\phi}(\mathrm{conf}(R)) \bowtie \rho_{P \rightarrow P_{\phi \wedge \neg\psi}}\big(\mathrm{conf}(R \times R_{\neg\psi}) \cup \pi_{TID,SSN,0 \rightarrow P}(\mathrm{conf}(R) - \mathrm{conf}(R \times R_{\neg\psi}))\big) \times \rho_{P \rightarrow P_{\neg\psi}}(\mathrm{conf}(R_{\neg\psi})).$$

The overall example query can be expressed as

$$T := \pi_{TID,SSN,(P_\phi - P_{\phi \wedge \neg\psi})/(1-P_{\neg\psi}) \rightarrow P}(S).$$

For the example table R given above, S and T are

Language Fragment	Complexity	Reference
On non-succinct representations:		
RA + conf + possible + choice-of	**in PTIME** (SQL)	[22]
RA + possible + repair-key	**NP-&coNP-hard**, **in $\mathbf{P}^{NP}$**	[6] [21]
RA + possible$_Q$ + repair-key	**PHIER-compl.**	[21]
On U-relations:		
Pos.RA + repair-key + possible	**in AC0**	[3]
RA + possible	**co-NP-hard**	Abiteboul et al. [2]
Conjunctive queries + conf	**#P-hard**	Dalvi, Suciu [13]
Probabilistic WSA	**in $\mathbf{P}^{\#P}$**	[22]
Pos.RA + repair-key + possible + approx.conf + egds	**in PTIME**	[22]

Figure 2: Complexity results for (probabilistic) world-set algebra. RA denotes relational algebra.

S	TID	SSN	P_ϕ	$P_{\phi\wedge\neg\psi}$	$P_{\neg\psi}$
	t_1	185	.4	.28	.28
	t_1	785	.6	0	.28
	t_2	185	.7	.28	.28
	t_2	186	.3	0	.28

T	TID	SSN	P
	t_1	185	1/6
	t_1	785	5/6
	t_2	185	7/12
	t_2	186	5/12

Complexity Overview Figure 2 gives an overview over the known complexity results for the various fragments of probabilistic WSA. Two different representations are considered, non-succinct representations that basically consist of enumerations of the possible worlds [6] and succinct representations: U-relational databases. In the non-succinct case, only the repair-key operation, which may cause an exponential explosion in the number of possible worlds, makes queries hard. All other operations, including confidence computation, are easy. In fact, we may add much of SQL – for instance, aggregations – to the language and it still can be processed efficiently, even by a reduction of the query to an SQL query on a suitable non-succinct relational representation.

When U-relations are used as representation system, the succinctness causes both difference [2] and confidence computation [13] independently to make queries NP-hard. Full probabilistic world-set algebra is essentially not harder than the language of [13], even though it is substantially more expressive.

It is worth noting that repair-key by itself, despite the blowup of possible worlds, does not make queries hard. For the language consisting of positive relational algebra, repair-key, and poss, we have shown by construction that it has PTIME complexity: We have given a positive relational algebra rewriting to queries on the representations earlier in this section. Thus queries are even in the highly parallelizable complexity class AC_0.

The final result in Figure 2 concerns the language consisting of the positive relational algebra operations, repair-key, (ϵ, δ)-approximation of confidence computation, and the generalized equality generating dependencies of [22] for which we can rewrite difference of uncertain relations to difference of confidence values (see Example 6.4). The result is that queries of that language that close the possible worlds semantics – i.e., that use conf to

compute a certain relation – are in PTIME overall. In [22], a stronger result than just the claim that each of the operations of such a query is individually in PTIME is proven. It is shown that, leaving aside a few pitfalls, global approximation guarantees can be achieved in polynomial time, i.e., results of entire queries in this language can be approximated arbitrarily closely in polynomial time.

This is a non-obvious result because the query language is compositional and selections can be made based on approximated confidence values. In a query $\sigma_{P=0.5}(\text{approx.conf}(R))$, an approximated P value will almost always be slightly off, even if the exact P value is indeed 0.5, and the selection of tuples made based on whether P is 0.5 is nearly completely arbitrary. In [22, 15], it is shown that this is essentially an unsurmountable problem. All we can tell is that if P is very different from 0.5, then the probability that the tuple should be in the answer is very small. If atomic selection conditions on (approximated) probabilities usually admit ranges such as $P < 0.5$ or $0.4 < P < 0.6$, then query approximation will nevertheless be meaningful: we are able to approximate query results unless probability values are very close or equal to the constants used as interval bounds. (These special points are called *singularities* in [22].)

The results of [22] have been obtained for powerful conditions that may use arithmetics over several approximated attributes, which is important if conditional probabilities have to be checked in selection conditions or if several probabilities have to be compared. The algorithm that gives overall (ϵ, δ)-approximation guarantees in polynomial time is not strikingly practical. Further progress on this has been made in [15], but more work is needed.

7 The MayBMS Query and Update Language

This section describes the query and update language of MayBMS, which is based on SQL. In fact, our language is a generalization of SQL on classical relational databases. To simplify the presentation, a fragment of the full language supported in MayBMS is presented here.

The representation system used in MayBMS, U-relations, has classical relational tables as a special case, which we will call *typed-certain (t-certain) tables* in this section. Tables that are not t-certain are called uncertain. Note that this notion of certainty is purely syntactic, and $\text{cert}(R) = \pi_{sch(R)}(\sigma_{P=1}(\text{conf}(R)))$ may well be equal to the projection of a U-relation U_R to its attribute (non-condition) columns despite R not being t-certain according to this definition.

Aggregates In MayBMS, full SQL is supported on t-certain tables. Beyond t-certain tables, some restrictions are in place to assure that query evaluation is feasible. In particular, we do not support the standard SQL aggregates such as `sum` or `count` on uncertain relations. This can be easily justified: In general, these aggregates will produce exponentially many different numerical results in the various possible worlds, and there is no way of representing these results efficiently. However, MayBMS supports a different set of aggregate operations on uncertain relations. These include the computations of *expected* sums and counts (using aggregates `esum` and `ecount`).

Moreover, the confidence computation operation is an aggregate in the MayBMS query language. This is a deviation from the language flavor of our algebra, but there is a justification for this. The algebra presented earlier assumed a set-based semantics for relations, where operations such as projections automatically remove duplicates. In the MayBMS query language, just like in SQL, duplicates have to be eliminated explicitly, and

confidence is naturally an aggregate that computes a single confidence value for each group of tuples that agree on (a subset of) the non-condition columns. By using aggregation syntax for `conf` and not supporting `select distinct` on uncertain relations, we avoid a need for conditions beyond the special conjunctions that can be stored with each tuple in U-relations.

All the aggregates on uncertain tables produce t-certain tables.

Duplicate tuples SQL databases in general support multiset tables, i.e., tables in which there may be duplicate tuples. There is no conceptual difficulty at all in supporting multiset U-relations. In fact, since U-relations are just relations in which some columns are interpreted to have a special meaning (conditions), just storing them in a standard relational database management system (which supports duplicates in tables) yields support for multiset U-relations.

Syntax The MayBMS query language is compositional and built from uncertain and t-certain queries. The uncertain queries are those that produce a possibly uncertain relation (represented by a U-relation with more than zero V and D columns). Uncertain queries can be constructed, inductively, from t-certain queries, `select-from-where` queries over uncertain tables, the multiset union of uncertain queries (using the SQL `union` construct), and statements of the form

```
repair key <attributes> in <t-certain-query>
weight by <attribute>
```

Note that repair-key is a query, rather than an update statement. The `select-from-where` queries may use any t-certain subqueries in the conditions, plus uncertain subqueries in atomic conditions of the form

```
<tuple> in <uncertain-query>
```

that occur positively in the condition. (That is, if the condition is turned into DNF, these literals are not negated.)

The t-certain queries (i.e., queries that produce a t-certain table) are given by

- all constructs of SQL on t-certain tables and t-certain subqueries, extended by a new aggregate

  ```
  argmax(<argument-attribute>, <value-attribute>)
  ```

  which outputs all the `argument-attribute` values in the current group (determined by the group-by clause) whose tuples have a maximum `value-attribute` value within the group. Thus, this is the typical argmax construct from mathematics added as an SQL extension.

- `select-from-where-group-by` on uncertain queries using aggregates `conf`, `esum`, and `ecount`, but none of the standard SQL aggregates. There is an exact and an approximate version of the `conf` aggregate. The latter takes two parameters ϵ and δ (see the earlier discussion of the Karp-Luby FPRAS).

The aggregates `esum` and `ecount` compute expected sums and counts across groups of tuples. While it may seem that these aggregates are at least as hard as confidence computation (which is #P-hard), this is in fact not so. These aggregates can be efficiently computed exploiting linearity of expectation. A query

```
select A, esum(B) from R group by A;
```

is equivalent to a query

```
select A, sum(B * P) from R' group by A;
```

where `R'` is obtained from the U-relation of `R` by replacing each local condition $V_1, D_1, \ldots, V_k, D_k$ by the probability $\Pr[V_1 = D_1 \wedge \cdots \wedge V_k = D_k]$, not eliminating duplicates. That is, expected sums can be computed efficiently tuple by tuple, and only require to determine the probability of a conjunction, which is easy, rather than a DNF of variable assignments as in the case of the `conf` aggregate. The `ecount` aggregate is a special case of `esum` applied to a column of ones.

Example 7.1 The query of Example 4.2 can be expressed in the query language of MayBMS as follows. Let `R` be `repair key in Coins weight by Count` and let `S` be

```
select R.Type, Toss, Face
from (repair key Type, Toss in (select * from Faces, Tosses)
      weight by FProb) S0, R
where R.Type = S0.Type;
```

It is not hard to verify that $\pi_{\text{Toss,Face}}(S) \neq Ev$ exactly if there exist tuples $\vec{s} \in S, \vec{t} \in Ev$ such that $\vec{s}.\text{Toss} = \vec{t}.\text{Toss}$ and $\vec{s}.\text{Face} \neq \vec{t}.\text{Face}$. Let `C` be

```
select S.Type from S, Ev
where S.Toss = Ev.Toss and S.Face <> Ev.Face;
```

Then we can compute `Q` using the trick of Example 6.4 as

```
select Type, (P1-P2)/(1-P3) as P
from (select Type, conf() as P1 from S group by Type) Q1,
     ((select Type, conf() as P2 from C group by Type)
      union
      (
         (select Type, 0 as P2 from Coins)
         except
         (select Type, 0 as P2 from
             (select Type, conf() from C group by Type) Dummy)
      )) Q2,
     (select conf() as P3 from C) Q3
where Q1.Type = Q2.Type;
```

The argmax aggregate can be used to compute maximum-a-posteriori (MAP) and maximum-likelihood estimates. For example, The MAP coin type

$$\text{argmax}_{\text{Type}} \Pr[\text{evidence is twice heads} \wedge \text{coin type is Type}]$$

can be computed as `select argmax(Type, P) from Q` because the normalizing factor `(1-P3)` has no impact on argmax. Thus, the answer in this example is the double-headed coin. (See table Q of Figure 1: The fair coin has $P = 1/3$, while the double-headed coin has $P = 2/3$.)

The maximum likelihood estimate

$$\text{argmax}_{\text{Type}} \Pr[\text{evidence is twice heads} \mid \text{coin type is Type}]$$

can be computed as

```
select argmax(Q.Type, Q.P/R'.P)
from Q, (select Type, conf() as P from R) R'
where Q.Type = R'.Type;
```

Here, again, the result is 2headed, but this time with likelihood 1. (The fair coin has likelihood 1/4). □

Updates MayBMS supports the usual schema modification and update statements of SQL. In fact, our use of U-relations makes this quite easy. An insertion of the form

```
insert into <uncertain-table> (<uncertain-query>);
```

is just the standard SQL insertion for tables we interpret as U-relations. Thus, the table inserted into must have the right number (that is, a sufficient number) of condition columns. Schema-modifying operations such as

```
create table <uncertain-table> as (<uncertain-query>);
```

are similarly straightforward. A deletion

```
delete from <uncertain-table>
where <condition>;
```

admits conditions that refer to the attributes of the current tuple and may use t-certain subqueries. Updates can be thought of as combinations of deletions and insertions, but in practice there are of course ways of implementing updates much more efficiently.

Conditioning Apart from the basic update operations of SQL, MayBMS also supports an update operation `assert` for conditioning, or knowledge compilation. The assert operation takes a Boolean positive relational algebra query ϕ in SQL syntax as an argument, i.e., a select-from-where-union query without aggregation. It conditions the database using this *constraint* ϕ, i.e., conceptually it removes all the possible worlds in which ϕ evaluates to false and renormalizes the probabilities so that they sum up to one again.

Formally, the semantics is thus

$$[\![\text{assert}(\phi)]\!](\mathbf{W}) := \{(R_1, \ldots, R_k, p/p_0) \mid (R_1, \ldots, R_k, p) \in \mathbf{W}, \\ (R_1, \ldots, R_k) \vDash \phi,\ p_0 = \sum_{(R'_1,\ldots,R'_k,p)\in\mathbf{W},(R'_1,\ldots,R'_k)\vDash\phi} p\}.$$

If the condition is inconsistent with the database, i.e., would delete all possible worlds when executed, the assert operation fails with an error (and does not modify the database).

Example 7.2 Consider the four possible worlds for the $R[SSN]$ relation of the census example.

$R[SSN]^1$	TID	SSN
	t_1	185
	t_2	185

$R[SSN]^2$	TID	SSN
	t_1	185
	t_2	186

$R[SSN]^3$	TID	SSN
	t_1	785
	t_2	185

$R[SSN]^4$	TID	SSN
	t_1	785
	t_2	186

To assert the functional dependency $R : SSN \rightarrow TID$, which states that no two individuals can have the same SSN, we can express the functional dependency as a Boolean query Q and execute assert(Q). This deletes the first of the four worlds and renormalizes the probabilities to sum up to one. □

Knowledge compilation using assert has obvious applications in areas such as data cleaning, where we may start with an uncertain database and then chase [1] a set of integrity constraints to reduce uncertainty. The assert operation can apply a set of constraints to a probabilistic database and materialize the cleaned, less uncertain database.

The assert operation is at least as hard as exact confidence operation (it is also practically no harder [23], and essentially the same algorithms can be used for both problems), but differently from confidence computation, the result has to be computed exactly and currently there is no clear notion of useful approximation to a cleaned database.

8 The MayBMS System

The MayBMS system has been under development since 2005 and has undergone several transformations. From the beginning, our choice was to develop MayBMS as an extension of the Postgres server backend. Two prototypes have been demonstrated at ICDE 2007 [7] and VLDB 2007 [8]. Currently, MayBMS is approaching its first release. MayBMS is open source and the source code is available through

http://maybms.sourceforge.net

The academic homepage of the MayBMS project is at

http://www.cs.cornell.edu/database/maybms/

Test data generators and further resources such as main-memory implementations of some of our algorithms have been made available through these Web pages as well.

We are aware of several research prototype probabilistic database management systems that are built as front-end applications of Postgres, but of no other system that aims to develop a fully integrated system. Our backend is accessible through several APIs, with efficient internal operators for computing and managing probabilistic data.

Representations, relational encoding, and query optimization Our representation system, U-relations, is basically implemented as described earlier, with one small exception. With each pair of columns V_i, D_i in the condition, we also store a column P_i for the probability weight of alternative D_i for variable V_i, straight from the W relation. While the operations of relational algebra, as observed earlier, do not use probability values, confidence computation does. This denormalization (the extension by P_i columns) removes the need to look up any probabilities in the W table in our exact confidence computation algorithms.

Our experiments show that the relational encoding of positive relational algebra which is possible for U-relations is so simple – it is a parsimonious transformation, i.e., the number of relational algebra operations is not increased – that the standard Postgres query optimizer actually does well at finding good query plans (see [3]).

Approximate confidence computation MayBMS implements both an approximation algorithm and several exact algorithms for confidence computation. The approximation algorithm is a combination of the Karp-Luby unbiased estimator for DNF counting [19, 20] in a modified version adapted for confidence computation in probabilistic databases (cf. e.g. [22]) and the Dagum-Karp-Luby-Ross optimal algorithm for Monte Carlo estimation [12]. The latter is based on sequential analysis and determines the number of invocations of the Karp-Luby estimator needed to achieve the required bound by running the estimator a small number of times to estimate its mean and variance. We actually use the probabilistic variant of a version of the Karp-Luby estimator described in the book [27] which computes fractional estimates that have smaller variance than the zero-one estimates of the classical Karp-Luby estimator.

Exact confidence computation Our exact algorithm for confidence computation is described in [23]. It is based on an extended version of the Davis-Putnam procedure [14] that is the basis of the best exact Satisfiability solvers in AI. Given a DNF (of which each clause is a conjunctive local condition), the algorithm employs a combination of variable elimination (as in Davis-Putnam) and decomposition of the DNF into independent subsets of clauses (i.e., subsets that do not share variables), with cost-estimation heuristics for choosing whether to use the former (and for which variable) or the latter.

Example 8.1 Consider the U-relation U representing a nullary table and the W table of Figure 3. The local conditions of U are $\Phi = \{\{x \mapsto 1\}, \{x \mapsto 2, y \mapsto 1\}, \{x \mapsto 2, z \mapsto 1\}, \{u \mapsto 1, v \mapsto 1\}, \{u \mapsto 2\}\}$.

The algorithm proceeds recursively. We first choose to exploit the fact that the Φ can be split into two independent sets, the first using only the variables $\{x, y, z\}$ and the second only using $\{u, v\}$. We recurse into the first set and eliminate the variable x. This requires us to consider two cases, the alternative values 1 and 2 for x (alternative 3 does not have to be considered because in each of the clauses to be considered, x is mapped to either 1 or 2. In the case that x maps to 2, we eliminate x from the set of clauses that are compatible with the variable assignment $x \mapsto 2$, i.e., the set $\{\{y \mapsto 1\}, \{z \mapsto 1\}\}$, and can decompose exploiting the independence of the two clauses. Once y and z are eliminated, respectively, the conditions have been reduced to "true". The alternative paths of the computation tree, shown in Figure 3, are processed analogously.

On returning from the recursion, we compute the probabilities of the subtrees in the obvious way. For two independent sets S_1, S_2 of clauses with probabilities p_1 and p_2, the

U	V_1	D_1	V_2	D_2
	x	1	x	1
	x	2	y	1
	x	2	z	1
	u	1	v	1
	u	2	u	2

W	V	D	P
	x	1	.1
	x	2	.4
	x	3	.5
	y	1	.2
	y	2	.8
	z	1	.4
	z	2	.6
	u	1	.7
	u	2	.3
	v	1	.5
	v	2	.5

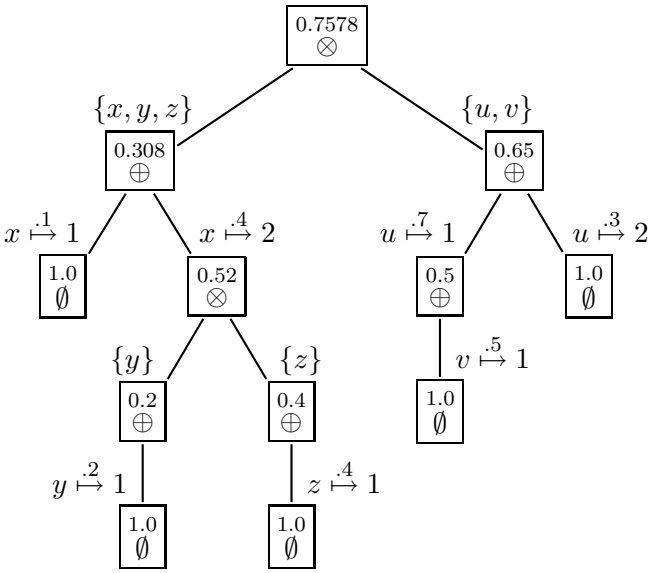


Figure 3: Exact confidence computation.

probability of $S_1 \cup S_2$ is $1-(1-p_1)\cdot(1-p_2)$. For variable elimination branches, the probability is the sum of the products of the probabilities of the subtrees and the probabilities of the variable assignments used for elimination.

It is not hard to verify that the probability of Φ, i.e., the confidence in tuple $\langle\rangle$, is 0.7578. □

Our exact algorithm solves a #P-hard problem and exhibits exponential running time in the worst case. However, like some other algorithms for combinatorial problems, this algorithm shows a clear easy-hard-easy pattern. Outside a narrow range of variable-to-clause count ratios, it very pronouncedly outperforms the (polynomial-time) approximation techniques [23]. It is straightforward to extend this algorithm to condition a probabilistic database (i.e., to compute "assert") [23].

Hierarchical queries The tuple-independent databases are those probabilistic databases in which, for each tuple, a probability can be given such that the tuple occurs in the database with that probability and the tuples are uncorrelated. It is known since the work of Dalvi and Suciu [13] that there is a class of conjunctive queries, the hierarchical queries Q, for which computing conf(Q) exactly on tuple-independent probabilistic databases is feasible in polynomial time.

In fact, these queries can essentially be computed using SQL queries that involve several nested aggregate-group-by queries. On the other hand, it was also shown in [13] that for any conjunctive query Q that is not hierarchical, computing conf(Q) is #P-hard with respect to data complexity. Dalvi and Suciu introduce the notion of *safe plans* that are at once certificates that a query is hierarchical and query plans with aggregation operators that

can be used for evaluating the queries.

Dan Olteanu's group at Oxford has recently extended this work in three ways, and implemented it in MayBMS [17]. First, the observation is used that in the case that a query has a safe plan, it is not necessary to use that safe plan for query evaluation. Instead we can choose our plan from a large set of possible plans, some of which will be much better and use fewer levels of aggregation than the canonical safe plans of [13]. Second, a special low-level operator for processing these aggregations has been implemented, which reduces the number of data scans needed [17]. Finally, the fact is exploited that the #P-hardness result for any single nonhierarchical query of [13] only applies as long as the problem is that of evaluating the query on an arbitrary probabilistic database of suitable schema. If further information about permissible databases is available in the form of functional dependencies that the databases must satisfy, then a larger class of queries can be processed by our approach.

Olteanu and Huang [24] have also obtained results on polynomial-time confidence computation on fragments of conjunctive queries with inequalities, using a powerful framework based on Ordered Binary Decision Diagrams.

Updates, concurrency control and recovery As a consequence of our choice of a purely relational representation system, these issues cause surprisingly little difficulty. U-relations are just relational tables and updates are just modifications of these tables that can be expressed using the standard SQL update operations. However, finding a suitable programming model and API for efficiently supporting programming access without exposing the user applications to internals of the representation system (which will differ among the various probabilistic DBMS) is a difficult problem. A full statement of this problem and some first results can be found in [4].

9 Conclusions

The aim of the MayBMS system is to be the first robust and scalable probabilistic database system that can be used in real applications. By our choice of running the entire project as an open-source project with the goal of creating mature code and serious documentation for developers, we hope to be able to accelerate progress in the field by making a testbed for new algorithms available to the research community.

Our possibly most important goal is to extend MayBMS to support continuous distributions. The path towards this goal is clearly sketched by our use of, essentially, a class of conditional tables for data representation. Our representations will not be hard to generalize, but some of the advantages of U-relations will be lost. There will be a need for a special column type "condition" for storing the more general local conditions needed, which has implications on operator implementations and will require us to study query optimization closely: We will not be able to rely as much on standard query optimizers to produce good plans as we currently do.

Another major goal is an extensive and careful experimental comparison of ours versus the graphical models approach, and to understand where the sweet spots of the two directions lie. More generally, it will be important to start working on a fair benchmark for probabilistic databases and, ideally, AI systems, even though it may still be too early to see the full set of dimensions that the space of systems will have, which is necessary to be able to define a benchmark that will remain fair and useful for some time.

A final grand goal is a query and update language specification that is a widely acceptable candidate for a future standard. This will be essential for wide acceptance of probabilistic databases. We expect our past work on the foundations of query algebras [6, 22, 21] to be useful in such an effort.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] S. Abiteboul, P. Kanellakis, and G. Grahne. "On the Representation and Querying of Sets of Possible Worlds". *Theor. Comput. Sci.*, **78**(1):158–187, 1991.

[3] L. Antova, T. Jansen, C. Koch, and D. Olteanu. "Fast and Simple Relational Processing of Uncertain Data". In *Proc. ICDE*, 2008.

[4] L. Antova and C. Koch. "On APIs for Probabilistic Databases". In *Proc. 2nd International Workshop on Management of Uncertain Data*, Auckland, New Zealand, 2008.

[5] L. Antova, C. Koch, and D. Olteanu. "10^{10^6} Worlds and Beyond: Efficient Representation and Processing of Incomplete Information". In *Proc. ICDE*, 2007.

[6] L. Antova, C. Koch, and D. Olteanu. "From Complete to Incomplete Information and Back". In *Proc. SIGMOD*, 2007.

[7] L. Antova, C. Koch, and D. Olteanu. "MayBMS: Managing Incomplete Information with Probabilistic World-Set Decompositions". In *Proc. ICDE*, 2007.

[8] L. Antova, C. Koch, and D. Olteanu. "Query Language Support for Incomplete Information in the MayBMS System". In *Proc. VLDB*, 2007.

[9] D. S. Batory. "On Searching Transposed Files". *ACM Trans. Database Syst.*, **4**(4):531–544, 1979.

[10] O. Benjelloun, A. D. Sarma, C. Hayworth, and J. Widom. "An Introduction to ULDBs and the Trio System". *IEEE Data Engineering Bulletin*, 2006.

[11] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. "Conditional Functional Dependencies for Data Cleaning". In *Proc. ICDE*, 2007.

[12] P. Dagum, R. M. Karp, M. Luby, and S. M. Ross. "An Optimal Algorithm for Monte Carlo Estimation". *SIAM J. Comput.*, **29**(5):1484–1496, 2000.

[13] N. Dalvi and D. Suciu. "Efficient query evaluation on probabilistic databases". *VLDB Journal*, **16**(4):523–544, 2007.

[14] M. Davis and H. Putnam. "A Computing Procedure for Quantification Theory". *Journal of ACM*, **7**(3):201–215, 1960.

[15] M. Goetz and C. Koch. "A Compositional Framework for Complex Queries over Uncertain Data", 2008. Under submission.

[16] E. Grädel, Y. Gurevich, and C. Hirsch. "The Complexity of Query Reliability". In *Proc. PODS*, pages 227–234, 1998.

[17] J. Huang, D. Olteanu, and C. Koch. "Lazy versus Eager Query Plans for Tuple-Independent Probabilistic Databases". In *Proc. ICDE*, 2009. To appear.

[18] T. Imielinski and W. Lipski. "Incomplete information in relational databases". *Journal of ACM*, **31**(4):761–791, 1984.

[19] R. M. Karp and M. Luby. "Monte-Carlo Algorithms for Enumeration and Reliability Problems". In *Proc. FOCS*, pages 56–64, 1983.

[20] R. M. Karp, M. Luby, and N. Madras. "Monte-Carlo Approximation Algorithms for Enumeration Problems". *J. Algorithms*, **10**(3):429–448, 1989.

[21] C. Koch. "A Compositional Query Algebra for Second-Order Logic and Uncertain Databases". Technical Report arXiv:0807.4620, 2008.

[22] C. Koch. "Approximating Predicates and Expressive Queries on Probabilistic Databases". In *Proc. PODS*, 2008.

[23] C. Koch and D. Olteanu. "Conditioning Probabilistic Databases". In *Proc. VLDB*, 2008.

[24] D. Olteanu and J. Huang. Conjunctive queries with inequalities on probabilistic databases. In *Proc. SUM*, 2008.

[25] D. Olteanu, C. Koch, and L. Antova. "World-set Decompositions: Expressiveness and Efficient Algorithms". *Theoretical Computer Science*, **403**(23):265–284, 2008.

[26] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. J. O'Neil, P. E. O'Neil, A. Rasin, N. Tran, and S. B. Zdonik. "C-Store: A Column-oriented DBMS". In *Proc. VLDB*, pages 553–564, 2005.

[27] V. V. Vazirani. *Approximation Algorithms*. Springer, 2001.