Uncertain Data Models

Christoph Koch
EPFL

Dan Olteanu
University of Oxford

SYNOMYMS

data models for incomplete information, probabilistic data models, representation systems

DEFINITION

An uncertain data model is a system for representing incomplete or uncertain data. An uncertain database asserts that a database is in one of multiple alternative states (possible worlds), each being a standard database. A probability distribution can be assigned to the set of possible worlds.

A detailed account of relational uncertain data models and their evolution, as well as related computational aspects is given in a recent research monograph [1].

SCIENTIFIC FUNDAMENTALS

Basic terminology and possible worlds semantics

An *uncertain data model* or *representation system* is an abstraction and method for representing uncertain or incomplete data in a database system.

This article focuses on uncertain *relational* data models, that is, data models that aim to represent relational databases consisting of *finite* relations that hold data tuples, with the additional challenge that the data to be represented is only incompletely determined. This incompleteness or uncertainty may manifest itself by unknown or alternative field values or tuples, for instance.

An uncertain database can be captured by *possible worlds semantics*: An uncertain relational database can be thought of as a (possibly uncountably infinite) set of possible worlds, each possible world a classical relational database, which all adhere to the same schema.

Possible worlds semantics is an intuitive thought model, but an impractical representation system; infinite sets of worlds cannot be explicitly stored, and even finite sets of worlds can often be stored much more compactly than by naive enumeration.

Expressive power of representation systems

A *representation system* is a method and data structure for physically storing an uncertain or probabilistic database as an actual finite data object called a *representation*, a bit sequence (or another "classical" data object that we are comfortable handling, such as a relational table) that can be stored on a physical storage device, with finite resource needs. The representation must completely and unambiguously specify the uncertain database.

By a *family* of uncertain databases, we understand the set of all uncertain databases satisfying a given common set of (schema) constraints. A representation system is called *complete* for a family of uncertain databases if it assigns a representation to each element of the family.

This is more subtle than it may seem. Even if unlimited storage is available, there are only countably many possible distinct bit sequences, so in general we can only hope to find representation systems for countably infinite families; but since there are uncountably many countable families, these representable families are extremely rare. For instance, consider the family of uncertain relational databases with real-typed fields. There are uncountably many real numbers, so there are uncountably many uncertain relational databases with real-typed fields, and there cannot be a complete representation system for them. There are countably many integers, but uncountably many *sets* of integers, so there cannot be a complete representation system for relational databases with integer-typed fields either. On the other hand, consider the family of uncertain databases

$$\underbrace{\{\underbrace{\{\underbrace{\{\langle x, y\rangle\}}_{\text{world}} \mid y \in \mathbb{R}\}}_{\text{uncertain database}} \mid x \in \mathbb{R}\}}_{\text{family}},$$

i.e., for each $x \in \mathbb{R}$, the set of possible worlds consisting of a singleton binary relation containing a pair of x and one other real number y. This is an uncountable family of uncertain databases; each uncertain database in this family is an uncountable set of possible worlds; yet there is a complete representation system – the previous sentence.

On the other hand, there are complete representation systems for all families of uncertain databases captured by finite sets of possible worlds – naive enumeration of the possible worlds is one of them.

Given the difficulty of creating complete representation systems for uncertain databases, studying the *expressive power* of incomplete representation systems is worthwhile. One key notion in this context is that of a *strong representation system*. A representation system ρ is called strong for a given query language $\mathcal{L}$ if, for each uncertain database (set of possible worlds) $\mathbf{I}$ representable by ρ, and each query $Q \in \mathcal{L}$, the set of possible worlds $\{Q(I) \mid I \in \mathbf{I}\}$ is representable by ρ. Strength is a nontrivial property of representation systems that in general needs to be proven; when ρ is not strong for $\mathcal{L}$, it may still be strong for a sublanguage of $\mathcal{L}$.

Conditional tables

Conditional tables (c-tables) are a representation system for uncertain relational databases which is notable for its versatility and for the seminal role it has played in the development of uncertain data models.

Syntactically, a c-table is a relational table extended as follows. Given a set V of typed variables (i.e., it is known for each variable which domain it ranges over, be it integers, strings, Booleans, or other), a c-table is a (multi-)set of items

$$\langle t_1, \ldots, t_k \mid \phi \rangle$$

where the t_i is either a constant or a variable from V (in each case its type is consistent with the schema of the table) and ϕ is a Boolean combination (using $\wedge$, $\vee$, and $\neg$) of atomic formulae $s \leq t$, where s, t are either variables from V or constants. (Obviously, comparison operations $=$, $\neq$, and $<$ are just syntactic sugar.) A *c-table database* is a structure $\langle R_1, \ldots, R_l; \Psi \rangle$ consisting of a number of c-tables $R_1, \ldots, R_l$ (according to schema) and a global condition Ψ (of the same format as the per-tuple conditions).

The semantics of a c-table database is best formalised by possible worlds semantics. Let θ be a function that maps each variable of V to an element of its domain and each constant to itself. For Boolean conditions, let $\theta(s \leq t) := \theta(s) \leq \theta(t)$ and let θ commute with $\wedge$, $\vee$, and $\neg$. For c-table R_i, let

$$\theta(R_i) := \{\langle \theta(t_1), \ldots, \theta(t_{\text{arity}(R_i)}) \rangle \mid \langle t_1, \ldots, t_{\text{arity}(R_i)} \mid \phi \rangle \in R_i, \theta(\phi) \text{ is true}\}$$

For c-table database $\langle R_1, \ldots, R_l; \Psi \rangle$, its set of possible worlds is

$$\{\langle \theta(R_1), \ldots, \theta(R_l) \rangle \mid \theta, \theta(\Psi) \text{ is true}\}.$$

Example: Consider the c-table database

R_1	A	
	x	$x \leq y$
	2	$x \leq 0$

R_2	B	
	y	$y \leq x$

$$\Psi = 0 \leq x \wedge x \leq 1 \wedge 0 \leq y \wedge y \leq 1$$

over schema $R_1(A : int), R_2(B : int)$ where $V = \{x, y\}$. Due to the global condition only admitting values $0, 1$ for x and y, there are four possible worlds:

- $\theta = \{x \mapsto 0, y \mapsto 0\} : R_1 = \{0, 2\}, R_2 = \{0\}$
- $\theta = \{x \mapsto 0, y \mapsto 1\} : R_1 = \{0, 2\}, R_2 = \emptyset$
- $\theta = \{x \mapsto 1, y \mapsto 0\} : R_1 = \emptyset, R_2 = \{0\}$
- $\theta = \{x \mapsto 1, y \mapsto 1\} : R_1 = \{1\}, R_2 = \{1\}$

In the literature, c-table databases are just called c-tables, even if they consist of multiple tables. We will do the same from now on.

C-tables have a number of interesting properties. Most importantly, they are a strong representation system for positive relational algebra. That is, although they are not complete representation systems, they are closed under evaluation of relational algebra queries even though they can represent (some) uncountable families of possible worlds.

The proof is not difficult. Relational projection and union on c-tables are like on relational tables but also copy the input conditions to the result. (Under set semantics, multiple tuples agreeing on the data columns but differing on the conditions are collapsed into a single tuple with a disjunction of conditions.) Selections add the selection condition to the condition of each tuple:

$$\sigma_\psi(R_i) = \{\langle \vec{t} \mid \phi \wedge \psi \rangle \mid \langle \vec{t} \mid \phi \rangle \in R_i\}.$$

The relational product operation pairs two c-table tuples just like in the classical product operation; the condition of the new tuples is the conjunction ($\wedge$) of the conditions of the input tuples:

$$R_i \times R_j = \{\langle \vec{t_i}, \vec{t_j} \mid \phi_i \wedge \phi_j \rangle \mid \langle \vec{t_i} \mid \phi_i \rangle \in R_i, \langle \vec{t_j} \mid \phi_j \rangle \in R_j\}$$

Naive tables are c-tables without conditions (or viewed differently, in which all conditions are set to true). Syntactically, naive tables differ from relational databases in that variables may occur in fields. Superficially, naive tables resemble relational databases with SQL NULL values, but the semantics is substantially different and query evaluation on naive tables is more complex. In naive tables, if a variable is re-used in multiple places, all possible worlds will have the same concrete value in all these places; such a constraint cannot be expressed using SQL NULL values. Note that for naive tables, query evaluation is co-NP-hard for data complexity (fixed queries, variable data) already for very limited query languages, such as select-project-join queries.

Of course, SQL tables with NULL values are a representation system for uncertain databases of their own, and given their wide adoption, they deserve mention here. However, there are many subtleties in how the SQL standard treats NULL values and how it weakens SQL query semantics to avoid NP-hardness, which are beyond the scope of this article. Because of these limitations, SQL tables with NULL values are *not* a strong representation system for positive relational algebra under its classical semantics. To understand this, note that we could build any naive table using a positive relational algebra view from a naive table in which no variable occurs twice. So the limitation to single occurrences of variables is not sufficient to avoid NP-hard query evaluation. Instead, SQL semantics is forgetful of valuations of NULL values and makes decisions locally to break dependencies that would otherwise cause complexity. For example, consider the query

```
select * from (select A as A1, A as A2 from R) as V where A1 = A2
```

Assume that this query is run on a singleton relation R with a NULL value. Then SQL will produce an empty result, locally deciding that it knows neither A1 nor A2 and conservatively evaluating the query condition to false, even though a strong representation system would evaluate the condition to true and produce a result tuple. (By definition, a strong representation system produces a representation of the query evaluated in each possible world individually. Here, A1 and A2 have the same value in all possible worlds.)

Probabilistic c-tables and graphical models

A *probabilistic database* is an uncertain database with a probability distribution over the set of possible worlds. To avoid mathematical subtleties, we subsequently assume discrete (countable) sets $\mathbf{I}$ of possible worlds. Then, a probability distribution is a function $p : \mathbf{I} \to \mathbb{R}$ such that, for each possible world $I \in \mathbf{I}$, $p(I) > 0$ and $\sum_{I \in \mathbf{I}} p(I) = 1$.

Given that probabilistic databases are uncertain databases with additional structure (the probability distribution), what was said about uncertain databases above applies to probabilistic databases in analogy.

We can represent a probabilistic database by a pair of a representation system for the uncertain database that uses a set of variables V and obtains each possible world by a valuation of these variables (as was the case for c-tables) *plus* a suitable representation of the joint probability distribution of these variables, interpreted as random variables.

Such a joint distribution of the random variables $V = \langle v_1, \ldots, v_n \rangle$ can be represented by a mapping $p : \mathrm{Dom}(v_1) \times \ldots \times \mathrm{Dom}(v_n) \to \mathbb{R}$, and if the domains $\mathrm{Dom}(v_i)$ are finite, it can be stored as a (large) relational table. Of course, such a naive representation consumes much space, and often more succinct decompositions are possible, in the sense of relational decomposition (factorisation with respect to the product operation $\times$ of relational algebra) or, as a strict generalisation of this idea, Bayesian Networks and probabilistic graphical models.

We have thus described a representation system for probabilistic databases as a pair of

1. a method for succinctly capturing the data inside a possible world, by a function that maps a valuation of the random variables to a classical relational database, and

2. a representation of the joint probability distribution of the random variables.

These two constituent parts are essentially orthogonal, and only linked through the names of the random variables.

This is a convenient separation of concerns analogous to the separation of data from queries in databases. It is possible to amalgamate these two parts more closely, but in general this will be at the cost of both clarity *and* succinctness of representation. Two forms of amalgamation are possible. The first, evolving from the database tradition, is to annotate tuples with probabilities, integrating the second part into the first. This direction is studied in more detail in the next section. The other form of amalgamation is by integrating the first part into the second, typically implemented as probabilistic graphical models in which the names of random variables carry inline data values or even structure.

A particular amalgamation may appear convenient for a particular application; moreover, in the context of graphical models, before recent work on lifted inference, query, data, and distribution were never separated.

One special case of (probabilistic) c-tables, called *U-relations*, is worth noting, since the formalism allows to evaluate positive relational algebra using pure relational algebra only, without condition parsers or advanced plug-ins. U-relations are (probabilistic) c-tables in which (random) variables are Boolean, the global condition is true, variables occur only in the conditions, not in the data fields, and per-tuple conditions are conjunctions over atomic conditions testing whether a variable is true. Thus, a conjunction of k atomic conditions can be represented by k additional columns to a table, holding variable names. On such a representation, positive relational algebra can be answered by a transform to another, a little more complex positive relational algebra query that is aware of the relational encoding of conditions, and an uncertain or probabilistic database system can be built on top of a classical relational database system simply by adding a query rewriting front-end. Despite the restrictions on the model, U-relations are not just a strong representation system for positive relational algebra but a *complete representation system for finite sets of possible worlds*.

From weaker representation systems to lineage

Depending on how uncertain data is initially obtained, certain independence properties may hold that lend themselves to particularly simple and readable representations. Two such models shall be mentioned here, both for probabilistic databases that represent finite sets of possible worlds.

Tuple-independent databases are relational databases in which each tuple, independently, either is or is not in the database, with a certain probability. Given the simple scenario, it is natural to associate a tuple's probability directly with the tuple, and store it in an additional probability column of the table. This representation system thus does not need explicit variables. It is also a succinct representation system – a tuple-independent database with n tuples represents 2^n possible worlds. The probability of a possible world is the product

$$\Big(\prod_{\text{tuple } \vec{t} \text{ chosen}} p(\vec{t})\Big) * \prod_{\text{tuple } \vec{t} \text{ not chosen}} (1 - p(\vec{t})).$$

A block-independent-disjoint table (BID table) is a table in which there are multiple groups (blocks) of tuples. Within each block, tuples are mutually exclusive and are associated with probabilities that sum up to one within the block. Tuple choices within blocks are independent across blocks. A possible world consists of one tuple pick from each block. The probability of the possible world is the product of the in-block probabilities of the picked tuples.

It is not hard to see how the independence properties of these two models can arise in practice; for instance, independent tuples may arise in information retrieval systems; BID tables can be the result of OCR algorithms that can do better than make a best guess, but are able to suggest local alternatives in case a text snippet cannot be recognized with certainty.

Both models are succinct, but neither is a strong representation system for a significant query language (with joins). This leaves two options, either to run monolithic query evaluation algorithms which may not make use of the compositionally of the query language on such input representations, or to abandon the representation in favor of a stronger one for query evaluation.

Assume we load a tuple-independent database into a probabilistic c-table, assigning each tuple its own new independent random variable. Then a tuple-independent table R becomes a c-table R as follows (the representation of the joint distribution of the independent variables x_i, $x_i \mapsto p_i$, is not shown below).

R	sch(R)	p
	$\vec{t}_1$	p_1
	$\vdots$	
	$\vec{t}_n$	p_n

R	sch(R)	ϕ
	$\vec{t}_1$	$x_1 = true$
	$\vdots$	
	$\vec{t}_n$	$x_n = true$

Now we can evaluate positive relational algebra as discussed earlier. What is significant to observe is that as we perform query operations, tuples and their conditions change, but the random variables *remain the same and independent* and the representation of the distribution does not need to be modified. As more dependencies are introduced, they get reflected in the per-tuple conditions. (Even though we seem to create new statistical dependencies, they only get reflected in the data, where they do not cause a substantial loss of succinctness.) For instance, a join of two relations will result in pairs whose conditions are the conjunctions of the conditions of the constituent tuples. Each original input tuple was associated with its unique variable, and this information does not get lost along the way, allowing us to interpret a tuple's condition, fairly, as its *lineage.*

KEY APPLICATIONS

Applications of uncertain data models include information retrieval, sensor data management, natural language processing, statistical data management, and data analytics.

CROSS REFERENCES

Query Processing over Uncertain Data; Graphical Models; Probabilistic Relational Models

Recommended Reading

[1] D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases.* Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.