Uncertainty in Data: A Survey on Probabilistic Databases

DESH RAJ, Indian Institute of Technology Guwahati
KANHAIYA RATHI, Indian Institute of Technology Guwahati

A probabilistic database management system (DBMS) is one that stores large volumes of probabilistic (uncertain or imperfect) data and supports complex queries. To efficiently describe any DBMS model, such as relational, graph, or object-oriented model, we need: (i) the algebra followed by the data objects, (ii) method of information retrieval, and (iii) performance of the model with real-world-sized data. In this paper, we provide an extensive summary and analysis of probabilistic databases. The available literature on the subject has been categorized into five major sections, namely algebra, query evaluation, conditioning, scaling and implementation, for the sake of convenience and distinction, and the progress made in each of these categories primarily over the last two decades has been reviewed. An overview of the design and performance of algorithms associated with each of these sections has been given to facilitate intuition for current and future research prospects. While the classes in themselves span a fairly large volume of research, an attempt has been made to avoid unnecessary detail wherever possible in favor of theoretical understanding and summarization. Through a reevaluation of performance parameters, we show that query evaluation is indeed the bottleneck in efficient development and implementation of the probabilistic DBMS model.

Categories and Subject Descriptors: H.2.4 [**Database Management**]: Systems

General Terms: Design, Algorithms, Performance, Implementation

Additional Key Words and Phrases: Algebra, query evaluation, scaling, conditioning, maybms

ACM Reference Format:
Desh Raj and Kanhaiya Rathi, 2016. Uncertainty in Data: A Survey on Probabilistic Databases. *ACM Comput. Surv.* 16, 4, Article 1 (April 2016), 15 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Relational databases have been extensively studied and used to organize data into forms that are convenient to store and efficient to retrieve. This model was effectively described in a presentation of ideas by Codd in 1970, and has since been widely accepted as the most suitable form for data organization, with SQL (Structured Query Language) famously utilizing this model.

The relational DBMS model is structured around the assumption that any data is known in its entirety and is certain. It does not allow any provision for uncertain or imperfect data. In the recent past, a number of applications have been recognized that require large, imprecise data sets. These include, and are not limited to, sensor and RFID data (uncertainty due to error in measurement, information retrieval (ambiguity in natural-language text), privacy and security.

This survey paper was conceptualized and written in partial fulfillment of the deliverables required for the course *CS 344 Databases*, undertaken by the authors at Indian Institute of Technology Guwahati, during Spring 2016.

Author's addresses: D. Raj and K. Rathi, Dept. of CSE, IIT Guwahati.

DOI: http://dx.doi.org/10.1145/0000000.0000000

Probabilistic databases model imprecision/uncertainty as probabilistic data. They were first introduced as a concept by Wong and Shoshani [1982]. Cavallo and Pittarelli studied them alongside traditional relational models. They argued that most of the theory related to the relational model can also be applied to probabilistic DBMS, and probabilistic dependencies may only be studied as constraints for the relational model. The following is an extract from Cavallo's *The Theory of Probabilistic Databases* [Cavallo and Pittarelli 1987]:

> "It is also the case that probabilistic databases generalize relational databases in the sense that any RDB can be represented by a PDB in such a way that important properties are preserved. A result of this is that useful concepts derived in the context of probability distributions may be applied in relational database theory. For example, by using probabilistic information theory with relational databases that have been transformed to probabilistic databases, cumbersome proofs of a number of significant results in relational database theory are simplified."

Barbara [Barbará et al. 1992] performed extensive studies on the management of probabilistic models, independent from relational models. Probabilistic analogues to relational algebra operators were proposed, and a set of new operators was also presented that had no counterpart in relational algebra. The work was thus instrumental in developing a richer algebraic language for the probabilistic DBMS model. Lakshmanan [Lakshmanan et al. 1997; Lakshmanan and Sadri 2001] further refined the model into a *single unified framework*, performed studies on complexity of queries involved in the model, and proposed algorithms for maintaining materialized probabilistic views.

Fuhr and Roellke [Fuhr and Rölleke 1997] integrated information retrieval into the model by designing probabilistic document indexing and search term weighting. They introduced the concept of vague predicates as opposed to Boolean values returned by a query, which allowed for imprecise data in the relational model. Zimanyi [Zimányi 1997; Zimányi and Pirotte 1997] studied a similar problem for assimilating vagueness into relational model of DBMS.

More recently, researchers have been involved in the study of conditioning, scaling and implementation issues of probabilistic databases. Koch and Olteanu [Koch and Olteanu 2008] proposed approximation algorithms to solve the problem of exponential complexity in conditioning, a process which refers to the frequent updation of databases to integrate new information. As recently as 2016, however, experimental data has shown that approximation can be avoided by using constraint-based methods for conditioning [Zhu et al. 2016]. Scaling, as a separate area of interest, has not been addressed in much detail though, and very few algorithms, including one which uses machine learning to scale probabilistic DBMS to a distributed setting [Blanco], are known. Due to the complexity involved in uncertainty-based querying, probabilistic models have not been efficiently employed inspite of their demand in numerous applications. The MayBMS project by Cornell and Oxford [Koch 2009; Huang et al. 2009], and the Trio project by Stanford [Widom 2004] are the most popular implementations of the model at present.

The remainder of the paper is organized as follows. In Section 2, we briefly describe the algebra of probabilistic databases, which will provide a base for understanding query evaluation which is discussed in Section 3. In the subsequent sections 4 and 5, conditioning and scaling issues have been discussed, and section 6 explores popular implementations of the model. We conclude by revisiting the issues ingrained in the probabilistic model, and also explain why query evaluation is the bottleneck for probabilistic databases.

2. ALGEBRA OF PROBABILISTIC DATABASES

In this section, we discuss the probabilistic set operators proposed by Pittarelli [Pittarelli 1994]. Some of these operators, such as *projection*, *selection*, and (maximum entropy) *join*, are analogous to operators in relational algebra, and may be shown to be homomorphic to the relational algebra structures (In abstract algebra, a *homomorphism* is a structure-preserving map between two algebraic structures such as groups, rings, or vector spaces). Others like *pooling*, *extension*, and *threshold*, are proposed as new operators. Most of the results obtained for these set operators are strictly probabilistic and may be shown to have practical applications because of the same reason.

As an example of probabilistic database with tuple-level uncertainty, we consider the relation R shown in table I. An entry $p_i(t) = x$ can be interpreted as the relative frequency with which some part from the sample possesses a tuple of attributes.

Table I: Relation R

Type	Plant	Defective	$p(t)$
Chain	Lubbock	No	0.15
Chain	Lubbock	Yes	0.01
Chain	Waco	No	0.22
Chain	Waco	Yes	0.20
Sprocket	Lubbock	No	0.12
Sprocket	Lubbock	Yes	0.01
Sprocket	Waco	No	0.26
Sprocket	Waco	Yes	0.03

2.1. Projection

The *projection* of p with scheme V onto $A \subseteq V$ is the distribution $\pi_A(p)$, where

$$\pi_p(a) = \sum_{t \in \mathrm{dom}(V), t[A]=a} p(t).$$

Marginal probabilities $p(S)$, $S \in \mathrm{dom}(V)$, are computed as

$$p(S) = \sum_{t \in S} p(t).$$

With tuples $t \in \mathrm{dom}(V)$ viewed as disjoint events, these definitions follow from the additivity of probability.

Relational projection may be defined in terms of the characteristic functions as

$$\pi_A(r)(a) = \mathbf{max}_{t \in \mathrm{dom}(V), t[A]=a} r(t).$$

It follows from these definitions that

$$t_{pr}(\pi_A(p)) = \pi_A(t_{pr}(p)),$$

i.e., t_{pr} is a homomorphism from (P_V, P_A, π_A) to (R_V, R_A, π_A), and that

$$\pi_A(r) = t_{pr}(\pi_A(t_{rp}(r))).$$

We state the following without proof:

(1) $\pi_V(p) = p$, if V is the scheme for p.
(2) $A \subseteq B$ implies $\pi_A(\pi_B(p)) = \pi_A(p)$.
(3) $\pi_A(\pi_B(r)) = \pi_A(r)$, if $A \subseteq B$.

The relations shown in Tables II and III are the projections $\pi_{X_1}(p)$ and $\pi_{X_2}(p)$ on R, with X_1 = {Type,Plant}, and X_2 = {Plant,Defective}.

Table II: Relation $R_{2,1} = \pi_{X_1}(p)$ (R), where X_1 = {Type,Plant}

Type	Plant	$p_1(t)$
Chain	Lubbock	0.16
Chain	Waco	0.42
Sprocket	Lubbock	0.13
Sprocket	Waco	0.29

Table III: Relation $R_{2,2} = \pi_{X_2}(p)$ (R), where X_2 = {Plant,Defective}

Plant	Defective	$p_2(t)$
Lubbock	No	0.27
Lubbock	Yes	0.02
Waco	No	0.48
Waco	Yes	0.23

2.2. Selection

For $S \subseteq \text{dom}(V)$,

$$\sigma_S(p)(t) = \begin{cases} 0, & \text{if } t \notin S \\ \frac{p(t)}{\sum_{t \in S} p(t)}, & \text{otherwise.} \end{cases}$$

$\sigma_S(p)$ is undefined when $\sum_{t \in S} p(t) = 0$. The mapping t_{pr} is homomorphic with respect to select also:

$$\sigma_A(t_{pr}(p)) = t_{pr}(\sigma_A(p)).$$

The relation shown in Table IV is the selection $\sigma_{\text{Plant='Waco'}}(p)$ on the relation R.

Table IV: Relation $R_3 = \sigma_{\text{Plant='Waco'}}(p)(R)$

Type	Plant	Defective	$p_2(t)$
Chain	Waco	No	0.31
Chain	Waco	Yes	0.28
Sprocket	Waco	No	0.37
Sprocket	Waco	Yes	0.04

2.3. Join

Let $P = p_1, p_2$, with structure V_1, V_2. The (pairwise) join of P is the probability distribution $J(P) \in E(P)$, whose components are calculated as

$$J(P)(t) = \frac{p_1(a) x p_2(b)}{\sum_{c|c[V_1 \cap V_2]=b[V_1 \cap V_2]} p_2 c}$$

where $a = t[V_1]$ and $b = t[V_2]$. The denominator of the above expression equals 1 if $V_1 \cap V_2 = \emptyset$

Table V: Relation $R_4 = J(p_1, p_2)(t)$

Type	Plant	Defective	$J(\{p_1, p_2\})(t)$
Chain	Lubbock	No	0.149
Chain	Lubbock	Yes	0.011
Chain	Waco	No	0.284
Chain	Waco	Yes	0.136
Sprocket	Lubbock	No	0.121
Sprocket	Lubbock	Yes	0.009
Sprocket	Waco	No	0.196
Sprocket	Waco	Yes	0.094

For example, on applying join on the relations $R_{2,1}$ and $R_{2,2}$, we get the relation shown in Table V.

We may further derive equations for cases when either of the relations is partially or completely dependent on the other, using probability theory. The advantage offered by embedding relational algebra in the probabilistic via the mapping t_{pr} is that nontrivial lossless decompositions of relations are allowed.

2.4. Pooling

Pooling is analogous to the Union set operator in relational algebra. It is used when it is required to combine multiple subjective estimates of a probability distribution into a single distribution. The most common method, known as linear pooling, computes a weighted average of the estimates:

$$p = w_1 p_1 + \ldots + w_n p_n$$

where $w_i > 0$, $\sum_i w_i = 1$, and $w_i > w_j$ iff estimate p_i is judged more trustworthy than estimate p_j.

Constructing an example for the pooling operator is a trivial exercise and is left out for sake of brevity.

2.5. Extension

For a distribution p with scheme A, its *extension* to the scheme V, $A \subseteq V$, is the set of all preimages of p under the mapping π_A:

$$E_V(p) = \{p' \in P_V | \pi_A(p') = p\}.$$

The extension of a database P is the intersection of the extensions of its elements:

$$E_V(P) = \bigcap_{p \in P} E_V(p).$$

For example, the database formed by relations $R_5, 1$ and $R_5, 2$ shown in tables VI and VII respectively represents partial information about manufactured parts.

Table VI: Relation $R_{5,1}$

Type	$p_1(t)$
Chain	0.58
Sprocket	0.42

Its extension to {Type,Plant} is the set of solutions p to the system:
$p(\text{Chain}, \text{Lubbock}) + p(\text{Chain}, \text{Waco}) = 0.58$

Table VII: Relation $R_{5,2}$

Plant	$p_2(t)$
Lubbock	0.29
Waco	0.71

$p(\text{Sprocket}, \text{Lubbock}) + p(\text{Sprocket}, \text{Waco}) = 0.42$
$p(\text{Chain}, \text{Lubbock}) + p(\text{Sprocket}, \text{Lubbock}) = 0.29$
$p(\text{Chain}, \text{Waco}) + p(\text{Sprocket}, \text{Waco}) = 0.71$
$p(t) \geq 0$.

2.6. Threshold

The *threshold* operator renormalizes a probability distribution after eliminating components failing to exceed a specified value:

$$T_x(p)(t) = \begin{cases} 0, & \text{if } p(t) \leq x, \\ \frac{p(t)}{\sum_{p(t)>x} p(t)}, & \text{otherwise.} \end{cases}$$

($T_x(p)$ is undefined when $\sum_{p(t)>x} p(t) = 0$.)

3. QUERY EVALUATION

A *query evaluation plan* consists of an extended relational algebra tree, with additional annotations at each node indicating the access methods to use for each table and implementation method to use for each operator [Ramakrishnan and Gehrke 2000]. Queries may be written intuitively and it is the task of the DBMS to execute them in the most efficient manner possible.

Query evaluation consists of three major processes as shown in Fig. 1:

(1) *Parsing*: It involves taking a query written in a simple descriptive language (such as SQL) and converting it into DBMS-level instructions.
(2) *Optimization*: It involves determining a plan for answering a query.
(3) *Execution*: It invloves obtaining the answer to query using the DBMS engine.

In their popular review of probabilistic databases, Dalvi and Suciu identified query evaluation as the hardest technical challenge in an efficient implementation of the model [Dalvi et al. 2009]. This is mainly because in probabilistic databases, unlike in their relational counterpart, the answer to a query must also include probability inference in addition to the lineage evaluation. We first present some definitions to lay groundwork for the discussions in this section.

Definition 3.1. Lineage: The *lineage* of a tuple is defined as an annotation that describes its derivation. It is used to represent probabilistic data as well as to represent query results.

Definition 3.2. Safe plan: It refers to a sequence of instructions for evaluating a query, such that the probabilistic inference on the answer gives the correct or desired value.

Definition 3.3. Disjunctive Normal Form: A disjunctive normal form (DNF) in boolean logic is defined as a standardization (or normalization) of a logical formula which is a disjunction of conjunctive clauses. It can also be described as an OR of ANDs or a sum of products.

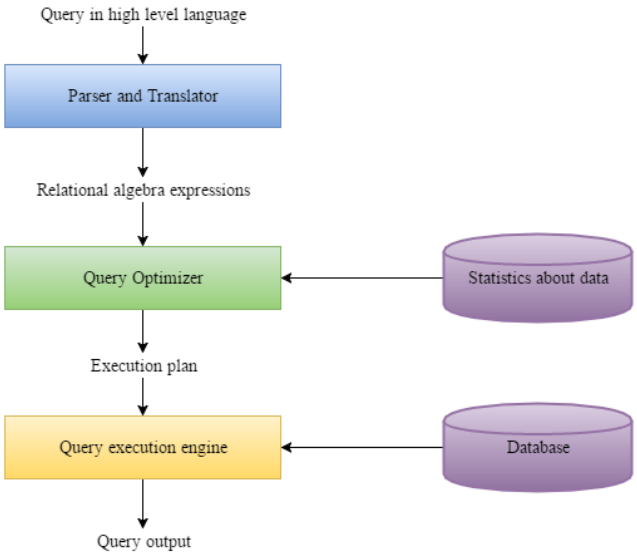

Fig. 1: Different stages of query evaluation.

Definition 3.4. Data complexity: The data complexity of a query q is the complexity of evaluating $q^{rank}(D^p)$ as a function of the size of D^p, where D^p is any probabilistic database.

In the remainder of this section, we discuss various strategies that have been applied for efficient query evaluation in the probabilistic DBMS model.

3.1. Separate lineage evaluation and probability inference

In this approach, the tasks of evaluating an answer to the query is performed independent of the associated probabilities. The probabilities are computed in parallel, using approaches such as the Monte Carlo approximation algorithm for DNF formulas [Karp et al. 1989; Dalvi and Suciu 2007], variable elimination algorithm [Poole and Zhang 2003], and the Davis-Putnam procedure [Davis and Putnam 1960]. In this section, we limit our discussion to the first mentioned algorithm, while the Davis-Putnam procedure has been explored as a method of conditioning probabilistic databases, in Section 4.

Consider the database consisting of relations S^p and T^p described in tables VIII and X respectively.

Table VIII: Relation S^p

A	**B**
'm'	1
'n'	1

Table IX: Relation T^p

C	**D**
1	'p'

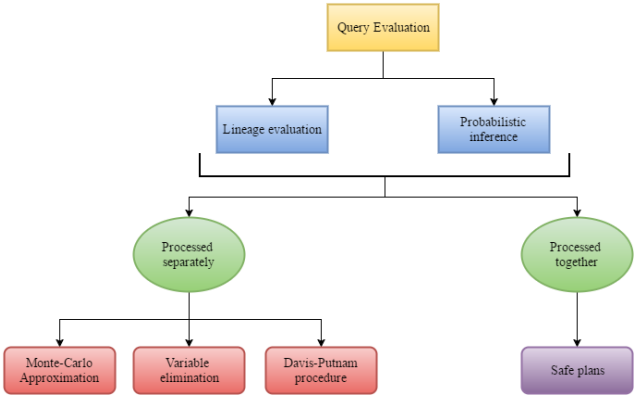

Fig. 2: Query evaluation strategies.

Suppose the tuple-level uncertainty for s_1 and s_2 are 0.8 and 0.5, and that for t_1 is 0.6. Consider the following query:

$$q(u) : S^p(x, y), T^p(z, u), y = z$$

A new query q' is created for returning all the variables in its body, which is as follows:

$$q'(u, x, y, z) : S^p(x, y), T^p(z, u), y = z$$

When this query is evaluated over the database, the following result is obtained:

Table X: Relation T^p

x	**y**	**z**	**u**	**E**
'm'	1	1	'p'	$s_1 \cap t_1$
'n'	1	1	'p'	$s_2 \cap t_1$

where E denotes the set of all complex events (a *complex event* is an expression constructed from atomic events using the operators $\wedge$, $\vee$, $\neg$).

Every event in the new query is a conjunction of atomic events, without any projects. This is because it returns all the variables that are present in its body. A final projection is performed at the end of this query, which gives us a single tuple {'p'} whose event is the DNF expression $(s_1 \wedge t_1) \vee (s_2 \wedge t_1)$.

Even though we have reduced our query to a DNF formula, the problem of determining its probability is still #P-complete. Using Monte-Carlo approximation, however, we can get the probability in a time that is linear in the number of tuples N that were merged in the final projection to determine the output tuples. This is because given a DNF formula with N clauses, and any ϵ and δ, the algorithm runs in time $O(N/\epsilon^2 \ln 1/\delta)$, and guarantees that the probability of error being greater than ϵ is less than δ.

This method has also been employed successfully for evaluating top-ranked queries, which is a common requirement in applications like search engines, in probabilistic databases [Re et al. 2007].

ALGORITHM 1: SafePlan (q)

Result: Safe plan for query q, if it exists.
if $Head(q) = Attr(q)$ **then**
  return any plan P for q;
  (P is projection-free, hence safe)
end
for $A \in Attr(q) - Head(q)$ **do**
  let q_A be the query obtained from q by adding A to the head variables;
  if $\pi_{Head(q)}(q_A)$ *is a safe operator* **then**
    return $\pi_{Head(q)}$(SafePlan(q_A));
  end
end
Split q into $q_1 \bowtie_c q_2$ s.t. $\forall R_1 \in Rels(q_1) R_2 \in Rels(q_2)$ R_1, R_2 are separated.;
if *no such splits exist* **then**
  return error("No safe plans exist");
end
return SafePlan(q_1)$\bowtie_c$ SafePlan(q_2)

3.2. Integrating probabilistic inference into lineage evaluation

Researchers have found that computing output probabilities inside the database engine rather than a separate probabilistic inference step may improve the performance of query evaluation over Monte Carlo simulation method by up to two orders of magnitude [Re et al. 2007]. The catch, however, is that such a method can only be applied to particular kinds of queries, which are known as *safe queries*. The relational plan which computes the output probability correctly for such queries is called a *safe plan*.

Dalvi and Suciu proposed an algorithm (1) which determines a safe plan for any query, if it exists.

In Algorithm 1, the terminology is as defined below:

— *Rels(q)* = $\{R_1, \ldots, R_k\}$, all relation names occuring in q.
— *Attr(q)*= all attributes in all relations in q.
— *Head(q)* = the set of attributes that are in the output of the query q.

The algorithm for determining safe plans has been proven to be sound and complete, i.e., any plan returned by the algorithm is safe, and if a query is safe, the algorithm determines a safe plan for it. It can also be shown that the algorithm fails to determine safe plans for a query if and only if its data complexity is #P-complete. In all other cases, the data complexity of q is PTIME. Further, Dalvi and Sucio proposed a definite set of rules for accurately identifying an *unsafe* query in linear time.

From the above discussion, we may conclude that although it is possible and algorithmically efficient to evaluate safe queries, the same does not hold true when the queries are unsafe. Since in any database system, consistency is key for evaluating queries, the probabilistic model fails in this regard.

4. CONDITIONING PROBABILISTIC DATABASES

The problem of *conditioning* involves transforming a probabilistic database of priors into a posterior probabilistic database which is materialized for subsequent query processing or further refinement. This is done by removing those branches (worlds) of the DBMS which do not satisfy a given condition.

In Section 3, we discussed the Monte Carlo approximation method of determining probabilistic inferences. At every stage of query evaluation, if the probability of tuples obtained do not satisfy a particular condition, they can be pruned out using the *thresh-*

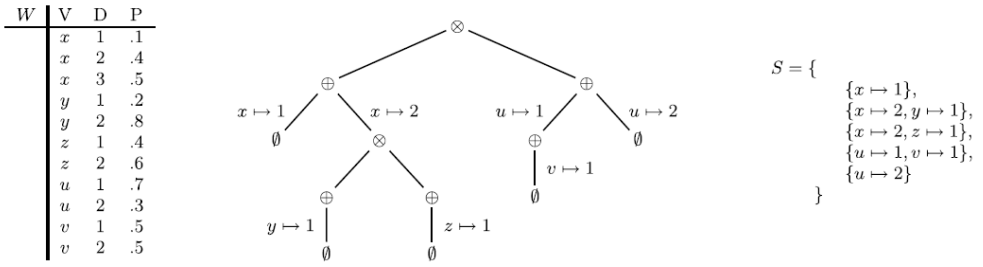

Fig. 3: World-set table W, a ws-tree R, and an equivalent ws-set S.

old operator. In this section, we analyze other algorithms which have been applied for the same purposes by various researchers.

4.1. Davis-Putnam method

Koch and Olteanu [Koch and Olteanu 2008] proposed an efficient conditioning algorithm based on the Davis-Putnam (DP) method and the theory of *world sets*. The basic structure of the method is described as follows:

(1) A probabilistic model based on world-set descriptor sets (*ws-sets*) is first introduced.
(2) An algorithm is proposed for converting the *ws-sets* into *ws-trees*, which essentially describe the structural decomposition of ws-sets.
(3) Once a ws-tree is known, the exact confidence computation and conditioning can be performed in linear time. For this reason, the problem of conditioning reduces to the problem of efficiently determining small ws-tree decompositions.
(4) A decomposition procedure is generated motivated by the DP method for checking propositional satisfiability.
(5) A database-conditioning algorithm is developed based on the ws-tree decomposition.

The DP algorithm was chosen for the reason that it remains the most exact approximation of the NP-complete satsifiability problem till date, and most SAT solvers implement either the original DP algorithm or the refined DPLL algorithm [Davis et al. 1962].

Since the decomposition step of the algorithm is the hardest in terms of time complexity, we concentrate our discussion around it. The exact conditioning algorithm implements this method recursively, renormalizing the database at every instance of the recursion.

4.1.1. Constructing world-set trees. Algorithm 2 shows the proposed divide-and-conquer method of constructing ws-tree from a ws-set. A brief outline of the algorithm is as follows:

(1) The input ws-set S is partitioned into independent disjoint sets when possible, or into overlapping sets that are consistent with different assignments of a variable.
(2) In the case of independent partitions, a $\otimes$-node is added to the tree and the algorithm is recursively called on the partitions.
(3) In the case of overlap, we eliminate a variable by iterating over all its possible assignments, and each of these assignments is added to the tree through a $\oplus$-node.
(4) If at any step a null ws-descriptor is found, a $\emptyset$ leaf is added to the ws-tree.

ALGORITHM 2: ComputeTree (WS-set S)

Result: WS-tree for a WS-set S
if $S = \emptyset$ **then**
  return $\bot$;
else if $\emptyset \in S$ **then**
  return $\emptyset$;
else
  choose one of the following;
  (independent partitioning):;
  if *there are non-empty and independent ws-sets* $S_1, \ldots, S_I$ *such that* $S = S_1 \cup \ldots \cup S_I$ **then**
    return $\otimes_{i \in I}(\text{ComputeTree}(S_i))$;
  end
  (variable elimination):;
  choose a variable x in S;
  $T \leftarrow \{d | d \in S, i \in \text{dom}_x : \{x \mapsto i\} \subseteq d\}$;
  $\forall i \in \text{dom}_x : S_{x \mapsto i} \leftarrow \{\{y_1 \mapsto j_1, \ldots, y_m \mapsto j_m\} | \{x \mapsto i, y_1 \mapsto j_1, \ldots, y_m \mapsto j_m\} \in S\}$;
  return $\oplus_{i \in \text{dom}_x}(x \mapsto i : \text{ComputeTree}(S_{x \mapsto i} \cup T))$;

In the case of overlapping partitions, different choices of the eliminating variable leads to different ws-trees. This *variable ordering* problem itself can affect the overall time complexity of the algorithm due to different heuristics for different orderings. Koch and Olteanu performed heuristic evaluations for two different orderings, namely *minlog* and *minmax*, and found that both of these orderings required time linear in the size of the ws-sets.

4.2. Constraint-based conditioning

In Koch and Olteanu's DP-based method, if the volume of constraints integrated into the probabilistic database is big, then the corresponding ws-trees can span multiple levels, hence causing the ws-tree used for conditioning to be inefficient. For instance, some selection operations may cause an exponential blow-up of the representations. This problem is not present in the intensional representation that was introduced by Zhu and Zhang [Zhu et al. 2016].

While the DP-method involved all the variables in the probabilistic database at every step, this algorithm focuses only on variables in the conditioning constraint while avoiding the involvement of other variables. This modification ensures that the size of the ws-tree generated is not too deep so as to be computationally expensive, while maintaining the efficiency of the DP-method. To further optimize functional-dependency based constraints, two different strategies were proposed:

(1) A pruning strategy that reduces the time for obtaining the set of satisfactory assignments; and
(2) A variable elimination strategy that minimizes the set of generated variables.

The details of the algorithm proposed in this framework are cumbersome and have been omitted for sake of brevity.

5. SCALING ISSUES IN THE PROBABILISTIC MODEL

From our discussions of query evaluation and conditioning of probabilistic databases, we have come to understand that probabilistic inference is the primary bottleneck in handling large sets of uncertain data using this model. *Scaling* is a generic term used to refer to a process of modifying a small-scale model such that it is suitable for implementation on real-world heuristics. With probabilistic DBMS models, researchers have

recently turned their focus on using machine learning strategies to make the models scalable [Blanco].

Distributed computing is being increasingly integrated with graph database architectures, and organizations have already started implementing parallelism on top of their existing DBMS architecture. Apache Hadoop, Trinity, FlockDB, Titan and Google's MapReduce are some of the popular implementations of distributed graph DBMS models. However, there is no distributed implementation of probabilistic databases at present, which may be a point of concern given the numerous applications of uncertainty in data sets.

6. IMPLEMENTATIONS

In this section, we analyze two popular implementations of the probabilistic database model, namely MayBMS [Koch 2009; Huang et al. 2009] and Trio [Widom 2004].

6.1. MayBMS

MayBMS was developed as a joint project by Cornell and Oxford in 2009. It is an extension of the open-source PostgreSQL server backend, which sets it apart from the existing research prototypes such as MystiQ and Trio. Koch and Olteanu, who have studied query evaluation extensively, were involved in the MayBMS project, and they have argued that simple PHP applications such as data cleaning, human resource management, and analysis of social networks, can be easily built on top of the MayBMS architecture.

The salient features of MayBMS are listed below:

— They store data in U-relational databases [Antova et al. 2008], which are a set of relations extended with condition and probability columns. This ensures a succinct and complete representation of a large set of possible worlds.
— The MayBMS query language [Olteanu and Huang 2008] has constructs that map (i) uncertain tables to tcertain tables, such as confidence computation constructs, (ii) uncertain to uncertain and t-certain to t-certain tables, such as full SQL, and (iii) t-certain to uncertain tables, such as constructs that extend the hypothesis space and create new possible worlds. Here, *t-certain* or *typed-certain* relations refer to U-relations without condition and probability columns.
— For confidence interval computation, MayBMS uses a combination of the Karp-Luby unbiased estimator for DNF queries (adapted to probailistic databases), and the Monte Carlo approximation algorithm [Karp et al. 1989].
— The exact method of query evaluation in MayBMS is the same as that described in Section 4.1, using the Davis-Putnam algorithm.
— Updates and concurrency is handled easily using SQL operations since the architecture relies on U-relations [Götz and Koch 2009].
— MayBMS implements U-relations on top of existing PostgreSQL relations such that the system catalog can distinguish between the two. Further, confidence computation is implemented as an operator using PostgreSQL executor, and other constructs are rewritten to SQL [Antova et al. 2007].

6.2. Trio

Trio, a Stanford initiative, is based on an extended relational model called ULDBs (Uncertainty Lineage Databases) [Benjelloun et al. 2006], and it supports a SQL-based query language called TriQL. It aims to aid varied applications including but not limited to scientific and sensor data management, data cleaning and integration, information extraction systems, and approximate and hypothetical query processing.

The salient features of Trio are listed below:

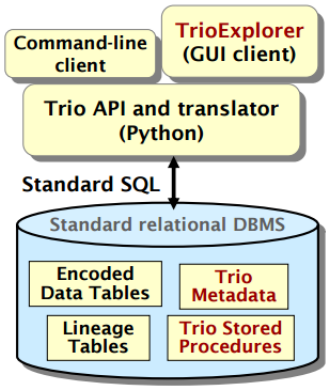

Fig. 4: Trio basic system architecture.

— Trio consists of a three-layer architecture [Mutsuzaki et al. 2007]: (i) Trio GUI Client that accepts TriQL queries; (ii) a Python parser that translates the TriQL to SQL queries; and (iii) a standard relational DBMS that executes SQL queries.
— The lineage information for each ULDB table T is stored in a separate relation [Sarma et al. 2010]. Each tuple is assigned a unique identifier to determine whether it has an associated confidence value. With this scheme, it is possible apply various optimizations on the model using methods like indexing, which is implemented in Trio.
— To process a query, a u-tuple is expected from the TriSQL. Once this is obtained, Trio issues instructions to create tables for lineage, and indexes. It then executes the queries, while simultaneously inserting the new alternatives and results into the lineage tables, so that round trips between the Python module and underlying DBMS may be avoided.
— Further, Trio implements operators like join [Ikeda and Widom 2009; Agrawal and Widom 2009] and aggregations. [Murthy et al. 2011].

7. CONCLUSION

Probabilistic databases are an intuitive approach for many applications such as data cleaning, information retrieval and human resource management, to name a few. However, due to some ground-level issues that have plagued the development of this model, they are yet to be effectively implemented on a large scale.

After the need for uncertainty in data was concretely established, it was not long before an exhaustive algebra of probabilistic databases was proposed and accepted. While this algebra had innovative probabilistic analogues for its relational counterpart and also a new set of operators, query evaluation became the bottleneck in its application.

The issue of query evaluation in this model originates from the necessity of probabilistic inference while processing queries. As discussed in Sections 3 and 4, the problem of confidence interval computation for most of the common queries may be #P-complete, which makes it computationally impossible to implement efficiently. Even with the help of approximation algorithms like Monte-Carlo or Davis-Putnam, there remains the difficulty of high complexity for some types of queries.

While researchers have been looking at machine learning algorithms for confidence value computation, such that the values could be learned from an analogous graph

model, there is not much literature to suggest optimistic outcomes. Moreover, while distributed computing has been used to great effect with relational and graph DBMS models, it has no counterpart in probabilistic databases.

Until query evaluation is performed efficiently and scaling is addressed prominently, probabilistic databases may remain confined to research prototypes regardless of their wide applicability.

REFERENCES

Parag Agrawal and Jennifer Widom. 2009. Confidence-aware join algorithms. In *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*. IEEE, 628–639.

Lyublena Antova, Thomas Jansen, Christoph Koch, and Dan Olteanu. 2008. Fast and simple relational processing of uncertain data. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*. IEEE, 983–992.

Lyublena Antova, Christoph Koch, and Dan Olteanu. 2007. Query language support for incomplete information in the MayBMS system. In *Proceedings of the 33rd international conference on Very large data bases*. VLDB Endowment, 1422–1425.

Daniel Barbará, Hector Garcia-Molina, and Daryl Porter. 1992. The management of probabilistic data. *Knowledge and Data Engineering, IEEE Transactions on* 4, 5 (1992), 487–502.

Omar Benjelloun, Anish Das Sarma, Chris Hayworth, and Jennifer Widom. 2006. An introduction to ULDBs and the Trio system. *IEEE Data Engineering Bulletin, March 2006* (2006).

Hernán Blanco. Scaling Probabilistic Databases. (????).

Roger Cavallo and Michael Pittarelli. 1987. The theory of probabilistic databases.. In *VLDB*, Vol. 87. 1–4.

Nilesh Dalvi, Christopher Ré, and Dan Suciu. 2009. Probabilistic databases: diamonds in the dirt. *Commun. ACM* 52, 7 (2009), 86–94.

Nilesh Dalvi and Dan Suciu. 2007. Efficient query evaluation on probabilistic databases. *The VLDB Journal* 16, 4 (2007), 523–544.

Martin Davis, George Logemann, and Donald Loveland. 1962. A machine program for theorem-proving. *Commun. ACM* 5, 7 (1962), 394–397.

Martin Davis and Hilary Putnam. 1960. A computing procedure for quantification theory. *Journal of the ACM (JACM)* 7, 3 (1960), 201–215.

Norbert Fuhr and Thomas Rölleke. 1997. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Transactions on Information Systems (TOIS)* 15, 1 (1997), 32–66.

Michaela Götz and Christoph Koch. 2009. A compositional framework for complex queries over uncertain data. In *Proceedings of the 12th international conference on database theory*. ACM, 149–161.

Jiewen Huang, Lyublena Antova, Christoph Koch, and Dan Olteanu. 2009. MayBMS: a probabilistic database management system. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. ACM, 1071–1074.

Robert Ikeda and Jennifer Widom. 2009. Outerjoins in uncertain databases. (2009).

Richard M Karp, Michael Luby, and Neal Madras. 1989. Monte-Carlo approximation algorithms for enumeration problems. *Journal of algorithms* 10, 3 (1989), 429–448.

Christoph Koch. 2009. MayBMS: A system for managing large uncertain and probabilistic databases. *Managing and Mining Uncertain Data* (2009), 149.

Christoph Koch and Dan Olteanu. 2008. Conditioning probabilistic databases. *Proceedings of the VLDB Endowment* 1, 1 (2008), 313–325.

Laks VS Lakshmanan, Nicola Leone, Robert Ross, and Venkatramanan Siva Subrahmanian. 1997. Probview: A flexible probabilistic database system. *ACM Transactions on Database Systems (TODS)* 22, 3 (1997), 419–469.

Laks V. S. Lakshmanan and Fereidoon Sadri. 2001. On a theory of probabilistic deductive databases. *TPLP* 1, 1 (2001), 5–42.

Raghotham Murthy, Robert Ikeda, and Jennifer Widom. 2011. Making aggregation work in uncertain and probabilistic databases. *Knowledge and Data Engineering, IEEE Transactions on* 23, 8 (2011), 1261–1273.

Michi Mutsuzaki, Martin Theobald, Ander De Keijzer, Jennifer Widom, Parag Agrawal, Omar Benjelloun, Anish Das Sarma, Raghotham Murthy, and Tomoe Sugihara. 2007. Trio-One: Layering uncertainty and lineage on a conventional DBMS. (2007).

Dan Olteanu and Jiewen Huang. 2008. Using OBDDs for efficient query evaluation on probabilistic databases. In *Scalable Uncertainty Management*. Springer, 326–340.

Michael Pittarelli. 1994. An algebra for probabilistic databases. *Knowledge and Data Engineering, IEEE Transactions on* 6, 2 (1994), 293–303.

David Poole and Nevin Lianwen Zhang. 2003. Exploiting contextual independence in probabilistic inference. *J. Artif. Intell. Res.(JAIR)* 18 (2003), 263–313.

Raghu Ramakrishnan and Johannes Gehrke. 2000. *Database Management Systems* (2nd ed.). Osborne/McGraw-Hill, Berkeley, CA, USA.

Cristina Re, Nilesh Dalvi, and Daniel Suciu. 2007. Efficient top-k query evaluation on probabilistic data. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*. IEEE, 886–895.

Anish Das Sarma, Martin Theobald, and Jennifer Widom. 2010. LIVE: a lineage-supported versioned DBMS. In *Scientific and Statistical Database Management*. Springer, 416–433.

Jennifer Widom. 2004. Trio: A system for integrated management of data, accuracy, and lineage. *Technical Report* (2004).

Hong Zhu, Caicai Zhang, Zhongsheng Cao, and Ruiming Tang. 2016. On efficient conditioning of probabilistic relational databases. *Knowledge-Based Systems* 92 (2016), 112–126.

Esteban Zimányi. 1997. Query evaluation in probabilistic relational databases. *Theoretical Computer Science* 171, 1 (1997), 179–219.

Esteban Zimányi and Alain Pirotte. 1997. Imperfect information in relational databases. In *Uncertainty Management in Information Systems*. Springer, 35–87.