Query Evaluation on Probabilistic Databases

Christopher Ré, Nilesh Dalvi and Dan Suciu
University of Washington

1 The Probabilistic Data

In this paper we consider the query evaluation problem: how can we evaluate SQL queries on probabilistic databases? Our discussion is restricted to single-block SQL queries using standard syntax, with a modified semantics: each tuple in the answer is associated with a probability representing our confidence in that tuple belonging to the answer. We present here a short summary of the research done at the University of Washington into this problem.

Consider the probabilistic database in Fig. 1. `Product`^{p} contains three products; their names and their prices are known, but we are unsure about their color and shape. Gizmo may be red and oval, or it may be blue and square, with probabilities $p_1 = 0.25$ and $p_2 = 0.75$ respectively. Camera has three possible combinations of color and shape, and IPod has two. Thus, the table defines for each product a probability distribution on its colors and shapes. Since each color-shape combination excludes the others, we must have $p_1+p_2 \leq 1$, $p_3+p_4+p_5 \leq 1$ and $p_6 + p_7 \leq 1$, which indeed holds for our table. When the sum is strictly less than one then that product may not occur in the table at all: for example Camera may be missing from the table with probability 1-p_3-p_4-p_5. Each probabilistic table is stored in a standard relational database: for example `Product`^{p} becomes the table in Fig. 2 (a). For any two tuples in `Product`^{p}, if they have the same values of the key attributes `prod` and `price` then they are exclusive (i.e. disjoint) probabilistic events, otherwise they are independent events.

The meaning of a probabilistic database is a probability distribution on possible worlds. `Product`^{p} has 16 possible worlds, since there are two choices for the color and shape for Gizmo, four for Camera (including removing Camera altogether) and two for IPod. Fig. 2 (b) illustrate two possible worlds and their probabilities.

`Product`^{p}

prod	price	color	shape	p
Gizmo	20	red	oval	$p_1 = 0.25$
		blue	square	$p_2 = 0.75$
Camera	80	green	oval	$p_3 = 0.3$
		red	round	$p_4 = 0.3$
		blue	oval	$p_5 = 0.2$
IPod	300	white	square	$p_6 = 0.8$
		black	square	$p_7 = 0.2$

`Order`

prod	price	cust
Gizmo	20	Sue
Gizmo	80	Fred
IPod	300	Fred

`Customer`^{p}

cust	city	p
Sue	New York	$q_1 = 0.5$
	Boston	$q_2 = 0.2$
	Seattle	$q_3 = 0.3$
Fred	Boston	$q_4 = 0.4$
	Seattle	$q_5 = 0.3$

Figure 1: Probabilistic database

Keys In this paper we impose the restriction that every deterministic attribute is part of a key. Formally, each probabilistic table `R` has a key, `R.Key`, and by definition this set of attributes must form a key in each

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Partially supported by Suciu's NSF Career Award IIS-00992955 and NSF Grants IIS-0428168, 61-2252, 61-2103, 0513877.

$\texttt{Product}^p$

prod	price	color	shape	p
Gizmo	20	red	oval	p_1
Gizmo	20	blue	square	p_2
Camera	80	green	oval	p_3
Camera	80	red	round	p_4
Camera	80	blue	oval	p_5
IPod	300	white	square	p_6
IPod	300	black	square	p_7

(a)

Product

prod	price	color	shape
Gizmo	20	blue	square
Camera	80	blue	oval
IPod	300	white	square

$p_2 p_5 p_6$

prod	price	color	shape
Gizmo	20	red	oval
IPod	300	white	square

$p_1(1\text{-}p_3\text{-}p_4\text{-}p_5)p_6$

(b)

Figure 2: Representation of a probabilistic table (a) and two possible worlds (b)

possible world. Intuitively, the attributes in `R.Key` are deterministic while the others are probabilistic. For example, in `Product(prod, price, shape, color)` the key `Product.Key` is {`prod, price`}, and one can see that it is a key in each of the two possible worlds in Fig. 2 (b). When a probabilistic table has only deterministic attributes, like in `R(A,B)`, the meaning is that each tuple occurs in the database with some probability ≤ 1, and any two tuples are independent events.

2 Easy Queries

Q_1:
```
SELECT DISTINCT prod, price
FROM Product
WHERE shape='oval'
```
$$Q_1(x,y) \; :- \; \texttt{Product}(\underline{x,y},\texttt{'oval'},z)$$

Q_2:
```
SELECT DISTINCT city
FROM Customer
```
$$Q_2(z) \; :- \; \texttt{Customer}(\underline{x,y},z)$$

Q_3:
```
SELECT DISTINCT *
FROM Product, Order, Customer
WHERE Product.prod = Order.prod
and Product.price = Order.price
and Order.cust = Customer.cust
```
$$Q_3(*) \; :- \; \texttt{Product}(\underline{x,y},z), \texttt{Order}(x,y,u) \; \texttt{Customer}(\underline{u},v)$$

Figure 3: Three simple queries, expressed in SQL and in datalog

We start by illustrating with three simple queries in Fig. 3. The left columns shows the queries in SQL syntax, the right column shows the same queries in datalog notation. In datalog we will underline the variables that occur in the key positions. The queries are standard, i.e. they are written assuming that the database is deterministic, and ignore any probabilistic information. However, their semantics is modified: each tuple returned has an associated probability representing our confidence in that answer. For example the first query, Q_1, asks for all the oval products in the database, and it returns:

prod	price	p
Gizmo	20	p_1
Camera	80	$p_3 + p_5$

In general, given a query Q and a tuple t, the probability that t is an answer to Q is the sum of the probabilities of all possible worlds where Q returns t. For Q_1, the probability of Gizmo is thus the sum of the probabilities of the 8 possible worlds for `Product` (out of 16) where Gizmo appears as oval, and this turns out (after simplifications) to be p_1. In the case of Q_1 these probabilities can be computed without enumerating all possible worlds, directly from the table in Fig. 2 (a) by the following process: (1) Select all rows with `shape='oval'`, (2) project on `prod`, `price`, and `p` (the probability), (3) eliminate duplicates, by replacing their probabilities with the sum, because they are disjoint events. We call the operation consisting of the last two steps a disjoint project:

Disjoint Project, $\pi^{pD}_{\bar{A}}$ If k tuples with probabilities $p_1, \cdots, p_k$ have the same value, $\bar{a}$, for their $\bar{A}$ attributes, then the disjoint project will associated the tuple $\bar{a}$ with the probability $p_1 + \cdots + p_k$. The disjoint project is correctly applied if any two tuples that share the same values of the $\bar{A}$ attributes are disjoint events.

Q_1 can therefore be computed by the following plan: $Q_1 = \pi^{pD}_{\texttt{prod,price}}(\sigma_{\texttt{shape='oval'}}(\texttt{Product}^p))$. $\pi^{pD}_{\texttt{prod,price}}$ is correct, because any two tuples in $\texttt{Product}^p$ that have the same `prod` and `price` are disjoint events.

The second query asks for all cities in the `Customer` table, and its answer is:

city	p
New York	q_1
Boston	1-(1-q_2)(1-q_4)
Seattle	1-(1-q_3)(1-q_5)

This answer can also be obtained by a projection with a duplicate elimination, but now the probabilities $p_1, p_2, p_3, \ldots$ of duplicate values are replaced with 1-(1-p_1)(1-p_2)(1-p_3) $\ldots$, since in this case all duplicate occurrences of the same city are independent. We call this an independent project:

Independent Project, $\pi^{pI}_{\bar{A}}$ If k tuples with probabilities $p_1, \cdots, p_k$ have the same value, $\bar{a}$, for their $\bar{A}$ attributes, then the independent project will associated the tuple $\bar{a}$ with the probability 1-(1-p_1)(1-p_2)$\cdots$(1-p_k). The independent project is correctly applied if any two tuples that share the same values of the $\bar{A}$ attributes are independent events.

Thus, the disjoint project and the independent project compute the same set of tuples, but with different probabilities: the former assumes disjoint probabilistic events, where $\mathbf{P}(t \vee t') = \mathbf{P}(t) + \mathbf{P}(t')$, while the second assumes independent probabilistic events, where $\mathbf{P}(t \vee t') = 1 - (1 - \mathbf{P}(t))(1 - \mathbf{P}(t')$. Continuing our example, the following plan computes Q_2: $Q_2 = \pi^{pI}_{\texttt{city}}(\texttt{Customer}^p)$. Here $\pi^{pI}_{\texttt{city}}$ is correct because any two tuples in $\texttt{Customer}^p$ that have the same `city` are independent events.

Finally, the third query illustrates the use of a join, and its answer is:

prod	price	color	shape	cust	city	p
Gizmo	20	red	oval	Sue	New York	p_1q_1
Gizmo	20	red	oval	Sue	Boston	p_1q_2
Gizmo	20	red	oval	Sue	Seattle	p_1q_3
Gizmo	20	blue	square	Sue	New York	p_2q_1
...	...					

It can be computed by modifying the join operator to multiply the probabilities of the input tables:

Join, $\bowtie^p$ Whenever it joins two tuples with probabilities p_1 and p_2, it sets the probability of the resulting tuple to be p_1p_2.

A plan for Q_3 is: $Q_3 = \texttt{Product} \bowtie^p \texttt{Order} \bowtie^p \texttt{Customer}$.

```
Schema:  R(A),S(A,B),T(B)
H1:  SELECT DISTINCT 'true' AS A
     FROM R, S, T
     WHERE R.A=S.A and S.B=T.B
```

$$H_1 \quad :- \quad R(\underline{x}), S(\underline{x,y}), T(\underline{y})$$

```
Schema:  R(A,B),S(B)
H2:  SELECT DISTINCT 'true' AS A
     FROM R, S
     WHERE R.B=S.B
```

$$H_2 \quad :- \quad R(\underline{x}, y), S(\underline{y})$$

```
Schema:  R(A,B),S(C,B)
H3:  SELECT DISTINCT 'true' AS A
     FROM R, S
     WHERE R.B=S.B
```

$$H_3 \quad :- \quad R(\underline{x}, y), S(\underline{z}, y)$$

Figure 4: Three queries that are #P-complete

3 Hard Queries

Unfortunately, not all queries can be computed as easily as the ones before. Consider the three queries in Fig. 4. All are boolean queries, i.e. they return either 'true' or nothing, but they still have a probabilistic semantics, and we have to compute the probability of the answer 'true'. Their schemas are kept as simple as possible: e.g. in H_1 table R has a single attribute A which forms a key (hence any two tuples are independent events). None of these queries can be computed in the style described in the previous section: for example, $\pi^{pI}_{\emptyset}(\mathsf{R} \bowtie \mathsf{S} \bowtie \mathsf{T})$ is an incorrect plan because two distinct rows in $\mathsf{R} \bowtie \mathsf{S} \bowtie \mathsf{T}$ may share the same tuple in R, hence they are not necessarily independent events. In fact, we have:

Theorem 1: Each of the queries H_1, H_2, H_3 in Fig. 4 is #P-complete.

The complexity class #P is the counting version of NP, i.e. it denotes the class of problems that count the number of solutions to an NP problem. If a problem is #P-hard, then there is no polynomial time algorithm for it unless P = NP; in this case none of H_1, H_2, H_3 has a simple plan using the operators in Sec. 1. Both here and in the following section we assume that all relations are probabilistic, but some results extend to a mix of probabilistic and deterministic tables. For example H_1 is #P-complete even if the table S is deterministic.

4 The Boundary Between Hard and Easy Queries

We show now which queries are in PTIME and which are #P-complete. We consider a conjunctive query q in which no relation name occurs more than once (i.e. without self-joins). We use the following notations: *Head*(q) is the set of head variables in q, *FreeVar(q)* is the set of free variables (i.e. non-head variables) in q, `R.Key` is the set of free variables in the key position of the relation R, `R.NonKey` is the set of free variables in the non-key positions of the relation R, `R.Pred` is the predicate that q applies to R. For $x \in$ *FreeVar*(q), denote q_x a new query whose body is identical with q and where *Head*$(q_x) =$ *Head*$(q) \cup \{x\}$.

Algorithm 3.1 takes a conjunctive query q and produces a relational plan for q using the operators described in Sec. 2. If it succeeds, then the query is in PTIME; if it fails then the query is #P-complete.

Theorem 2:

1. Algorithm 3.1 is sound, i.e. if it produces a relational plan for a query q, then the plan correctly computes the output tuple probabilities for q.

Algorithm 3.1 FIND-PLAN(q)

If q has a single relation R and no free variables, then return $\sigma_{\texttt{R.Pred}}(\mathsf{R})$.
Otherwise:

1. If there exists $x \in \mathit{FreeVar}(q)$ s.t. $x \in$ `R.Key` for every relation R in q, then return:

$$\pi^{pI}_{\mathit{Head}(q)}(\text{FIND-PLAN}(q_x))$$

2. If there exists $x \in \mathit{FreeVar}(q)$ and there exists a relation R s.t. $x \in$ `R.NonKey` and `R.Key`$\cap \mathit{FreeVar}(q) = \emptyset$, then return:

$$\pi^{pD}_{\mathit{Head}(q)}(\text{FIND-PLAN}(q_x))$$

3. If the relations in q can be partitioned into q_1 and q_2 such that they do not share any free variables, then return:

$$\text{FIND-PLAN}(q_1) \bowtie^p \text{FIND-PLAN}(q_2)$$

If none of the three conditions above holds, then q is #P-complete.

2. Algorithm 3.1 is complete, i.e. it does not produce a relational plan for a query only if the query is #P-hard.

As a consequence, every query that has a PTIME data complexity can in fact be evaluated using a relational plan. Any relational database engine can be used to support these queries, since the probabilistic projections and joins can be expressed in SQL using aggregate operations and multiplications.

Example 1: In the remainder of this section we illustrate with the following schema, obtained as an extension of our running example in Sec. 1.

```
Product(prod, price, color, shape)
Order(prod, price, cust)
CustomerFemale(cust, city, profession)
CustomerMale(cust, city, profession)
CitySalesRep(city, salesRep, phone)
```

All tables are now probabilistic: for example each entry in `Order` has some probability ≤ 1. The customers are partitioned into female and male customers, and we have a new table with sales representatives in each city. The following query returns all cities of male customers who have ordered a product with price 300:

$$Q(c) \quad :- \quad \texttt{Order}(\underline{x, 300, y}), \texttt{CustomerMale}(\underline{y}, c, z)$$

Here $\mathit{Head}(Q) = \{c\}$, $\mathit{FreeVar}(Q) = \{x, y, z\}$. Condition (1) of the algorithm is satisfied by the variable y, since $y \in$ `Order.Key` and $y \in$ `CustomerMale.Key`, hence we generate the plan: $Q = \pi^{pI}_c(Q_y)$ where the new query Q_y is:

$$Q_y(c, y) \quad :- \quad \texttt{Order}(\underline{x, 300, y}), \texttt{CustomerMale}(\underline{y}, c, z)$$

The independence assumption needed for $\pi^{pI}_c(Q_y)$ to be correct indeed holds, since any two distinct rows in $Q_y(c, y)$ that have the same value of c must have distinct values of y, hence they consists of two independent tuples in `Order` and two independent tuples in `CustomerMale`. Now $\mathit{Head}(Q_y) = \{c, y\}$, $\mathit{FreeVar}(Q_y) = \{x, z\}$ and Q_y satisfies condition (2) of the algorithm (with $z \in$ `CustomerMale.NonKey` and $\texttt{CustomerMale.Key} = \{y\} \subseteq \mathit{Head}(Q_y)$), hence we generate the plan: $Q = \pi_c^{pI}(\pi_{c,y}^{pD}(Q_{y,z}))$ where the new query $Q_{y,z}$ is:

$$Q_{y,z}(c, y, z) \quad :- \quad \texttt{Order}(\underline{x, 300, y}), \texttt{CustomerMale}(\underline{y}, c, z)$$

The disjointness assumption needed for $\pi_{c,y}^{pD}(Q_{y,z})$ to be correct also holds, since any two distinct rows in $Q_{y,z}(c, y, z)$ that have the same values for c and y must have distinct values for z, hence they represent disjoint events in `CustomerMale`. $Q_{y,z}$ satisfies condition (3) and we compute it as a join between `Order` and `CustomerMale`. The predicate `Order.Pred` is $\texttt{price} =' 300'$, hence we obtain the following complete plan for Q:

$$Q = \pi_c^{pI}(\pi_{c,y}^{pD}(\sigma_{\texttt{price}='300'}(\texttt{Order}) \bowtie^p \texttt{CustomerMale}))$$

Recall the three #P-complete queries H_1, H_2, H_3 in Fig. 4. It turns out that, in some sense, these are the only #P-complete queries: every other query that is #P-complete has one of these three as a subpattern. Formally:

Theorem 3: Let q be any conjunctive query on which none of the three cases in Algorithm 3.1 applies (hence Q is #P-complete). Then one of the following holds:

1. There are three relations `R`, `S`, `T` and two free variables $x, y \in \mathit{FreeVar}(q)$ such that `R.Key` contains x but not y, `S.Key` contains both x, y, and `T.Key` contains y but not x. In notation:

$$\texttt{R}(\underline{x}, \ldots), \texttt{S}(\underline{x, y}, \ldots), \texttt{T}(\underline{y}, \ldots)$$

2. There are two relations `R` and `S` and two free variables $x, y \in \mathit{FreeVar}(q)$ s.t. such that x occurs in `R.Key` but not in `S`, and y occurs in `R` and in `S.Key` but not in `R.Key`. In notation:

$$\texttt{R}(\underline{x}, y, \ldots), \texttt{S}(\underline{y}, \ldots)$$

3. There are two relations `R` and `S` and three free variables $x, y, z \in \mathit{FreeVar}(q)$ s.t. x occurs in `R.Key` but not in `S`, x occurs in `S.Key` but not in `R`, and y occurs in both `R` and `S` but neither in `R.Key` nor in `S.Key`. In notation:

$$\texttt{R}(\underline{x}, y, \ldots), \texttt{S}(\underline{z}, y, \ldots)$$

Obviously, H_1 satisfies condition (1), H_2 satisfies condition (2), and H_3 satisfies condition (3). The theorem says that if a query is hard, then it must have one of H_1, H_2, H_3 as a subpattern.

Example 2: Continuing Example 1, consider the following three queries:

$$\begin{aligned}
HQ_1(c) \quad &:- \quad \texttt{Product}(\underline{x, v}, -, \texttt{'red'}), \texttt{Orders}(\underline{x, v, y}), \texttt{CustomerFemale}(\underline{y}, c, -) \\
HQ_2(sr) \quad &:- \quad \texttt{CustomerMale}(\underline{x}, y, \texttt{'lawyer'}), \texttt{CitySalesReps}(\underline{y}, sr, z) \\
HQ_3(c) \quad &:- \quad \texttt{CustomerMale}(\underline{x}, c, y), \texttt{CustomerFemale}(\underline{z}, c, y)
\end{aligned}$$

None of the three cases of the algorithm applies to these queries, hence all three are #P-complete. The first query asks for all cities where some female customer purchased some red product; it matches pattern (1). The second query asks for all sale representatives in cities that have lawyer customers: it matches pattern (2). The third query looks for all cities that have a male and a female customer with the same profession; it matches pattern (3).

Finally, note that the three patterns are a necessary condition for the query to be #P-complete, but they are sufficient conditions only after one has applied Algorithm 3.1 until it got stuck. In other words, there are queries that have one or more of the three patterns, but are still in PTIME since the algorithm eliminates free variables in a way in which it makes the patterns disappear. For example:

$$Q(v) \quad :- \quad R(\underline{x}), S(\underline{x,y}), T(\underline{y}), U(\underline{u}, y), V(\underline{v}, u)$$

The query contains the subpattern (1) (the first three relations are identical to H_1), yet it is in PTIME. This is because it is possible to remove variables in order u, y, x and obtain the following plan:

$$Q = \pi_v^{pD}(V \bowtie^p \pi_{v,u}^{pD}(U \bowtie^p T \bowtie^p \pi_y^{pI}(R \bowtie^p S)))$$

Theorem 3 has interesting connections to several existing probabilistic systems. In Cavallo and Pittarelli's system [2], all the tuples in a table `R` represent disjoint events, which corresponds in our model to `R.Key` $= \emptyset$. None of the three patterns of Theorem 3 can occur, because each pattern asks for at least one variable to occur in a key position, and therefore all the queries in Cavallo and Pittarelli's model have PTIME data complexity. Barbara et al. [1] and then Dey et al. [4] consider a system that allows arbitrary tables, i.e. `R.Key` can be any subset of the attributes of `R`, but they consider restricted SQL queries: all key attributes must be included in the `SELECT` clause. In datalog terminology, `R.Key` $\subseteq$ `Head`(q) for every table `R`, hence none of the three patterns in Theorem 3 can occur since each looks for at least one variable in a key position that does *not* occur in the query head. Thus, all queries discussed by Barbara et al. are in PTIME. Theorem 3 indicates that a much larger class of queries can be efficiently supported by their system. Finally, in our previous work [3], we consider a system where `R.Key` is the set of all attributes. In this case only case (1) of Theorem 3 applies, and one can check that now the pattern is a sufficient condition for #P-completeness: this is precisely Theorem 5.2 of [3].

5 Future Work

We identify three future research problems. (1) Self joins: we currently do not know the boundary between PTIME and #P-complete queries when some relation name occurs two ore more times in the query (i.e. queries with self-joins). (2) Query optimization: the relational operators π^{pD}, π^{pI}, $\bowtie^p$ and σ^p do not follow the same rules as the standard relational algebra. A combination of cost-based optimization and safe-plan generation is needed. (3) Queries that are #P-hard require simulation based techniques [5], which are expensive. However, often there are subqueries that admit safe-plans: this calls for investigations of mixed techniques, combining safe plans with simulations.

References

[1] Daniel Barbará, Hector Garcia-Molina, and Daryl Porter. The management of probabilistic data. *IEEE Trans. Knowl. Data Eng.*, 4(5):487–502, 1992.

[2] Roger Cavallo and Michael Pittarelli. The theory of probabilistic databases. In *VLDB*, pages 71–81, 1987.

[3] Nilesh Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, 2004.

[4] Debabrata Dey and Sumit Sarkar. A probabilistic relational model and algebra. *ACM Trans. Database Syst.*, 21(3):339–369, 1996.

[5] Richard Karp and Michael Luby. Monte-carlo algorithms for enumeration and reliability problems. In *STOC*, 1983.