The VLDB Journal manuscript No.
(will be inserted by the editor)

Nilesh Dalvi · Dan Suciu

Efficient Query Evaluation on Probabilistic Databases

the date of receipt and acceptance should be inserted later

Abstract We describe a framework for supporting arbitrarily complex SQL queries with "uncertain" predicates. The query semantics is based on a probabilistic model and the results are ranked, much like in Information Retrieval. Our main focus is query evaluation. We describe an optimization algorithm that can compute efficiently most queries. We show, however, that the data complexity of some queries is $\#P$-complete, which implies that these queries do not admit any efficient evaluation methods. For these queries we describe both an approximation algorithm and a Monte-Carlo simulation algorithm.

1 Introduction

Databases and Information Retrieval [3] have taken two philosophically different approaches to queries. In databases SQL queries have a rich structure and a precise semantics. This makes it possible for users to formulate complex queries and for systems to apply complex optimizations, but users need to have a pretty detailed knowledge of the database in order to formulate queries. For example, a single misspelling of a constant in the WHERE clause leads to an empty set of answers, frustrating casual users. By contrast, a query in Information Retrieval (IR) is just a set of keywords and is easy for casual users to formulate. IR queries offer two important features that are missing in databases: the results are *ranked* and the matches may be *uncertain*, i.e. the answer may include documents that do not match all the keywords in the query[1]. While several proposals exist for extending SQL with uncertain matches and ranked results [1,20,15], they are either restricted to a single table, or, when they handle join queries, adopt an ad-hoc semantics.

University of Washington, Seattle.

[1] Some IR systems only return documents that contain all keywords, but this is a feature specific to those systems, and not of the underlying vector model used in IR.

```
SELECT   *
FROM     Actor A
WHERE    A.name ≈ 'Kevin'
  and    1995 =
         SELECT   MIN(F.year)
         FROM     Film F, Casts C
         WHERE    C.filmid = F.filmid
           and    C.actorid = A.actorid
           and    F.rating ≈ "high"
```

Fig. 1 An Approximate Query

To illustrate the point consider the query in Fig 1. It is a structurally rich query, asking for an actor whose name is like 'Kevin' and whose first movie with 'High' rating appeared in the year 1995.

The two $\approx$ operators indicate which predicates we intend as uncertain matches. Techniques like edit distances, ontology-based distances [14], IDF-similarity and QF-similarity [1] can be applied to a *single* table: to rank all `Actor` tuples (according to how well they match the first uncertain predicate), and to rank all `Film` tuples. But it is unclear how to rank the *entire* query. To date, no system combines structurally rich SQL queries with uncertain predicates and ranked results.

In this paper we propose such a system. We show that, using the concept of *possible worlds semantics*, database queries with approximate matches can be given meaning in a principled way and a ranked list of answers can be computed. Given a SQL query with uncertain predicates, we start by assigning a probability to each tuple in the input database according to how well it matches the uncertain predicates. Then we derive a probability for each tuple in the answer, and rank the answers accordingly.

An important characteristic of our approach is that *any* SQL query with approximate predicates has a meaning, including queries with joins, nested sub-queries, aggregates, group-by, and existential/universal quantifiers[2].

[2] In this paper we restrict our discussion to SQL queries whose normal semantics is a set, not a bag or an ordered list.

Queries have a probabilistic semantics, which is simple and easy to understand by both users and implementors.

While simple, the semantics gives no indication on how to evaluate the query. The main problem that we discuss in this paper is query evaluation. Our approach is to represent SQL queries in an algebra, and modify the operators to compute the probabilities of each output tuple. This is called *extensional semantics* [26], and is quite efficient. While this sounds simple, the problem is that it doesn't work: the probabilities computed this way are wrong in most cases, and lead to incorrect ranking. The reason is that, even if all tuples in the base relations are independent probabilistic events, the tuples in the intermediate results of a query plan have often correlated probabilities, making it impossible to compute the new probabilities precisely. The previous workaround is to use an *intensional semantics* [29,31,12], which represents tuple events symbolically and, hence, tracks tuple correlations precisely. However, as we show here, this approach is too inefficient to be practical. Our approach is different: we rewrite the query plan, searching for one where the extensional evaluation is correct. We show however that certain queries have a #P-complete data complexity under probabilistic semantics, and hence do not admit a correct extensional plan. While they are not frequent in practice (only 2 out of the 10 TPC/H queries fall in this category, and only when all their predicates are uncertain), we describe two techniques to address them: using heuristics to chose a plan that avoids large errors, and using a Monte-Carlo simulation algorithm, which is more expensive but can guarantee arbitrarily small errors.

Outline We give motivating examples in Sec. 2, define the problem in Sec. 3, and describe our techniques in Sec. 4-9. Sec. 11 reports experiments and Sec. 12 describes related work. We conclude in Sec. 13.

2 Examples

We illustrate the main concepts and techniques of this paper with two simple examples.

Probabilistic Database In a probabilistic database each tuple has a certain probability of belonging to the database. Figure 2 shows a probabilistic database D^p with two tables, S^p and T^p: the tuples in S^p have probabilities 0.8 and 0.5, and the unique tuple in T^p has probability 0.6. We use the superscript p to emphasize that a table or a database is probabilistic. We assume in this example that all the tuples represent independent probabilistic events.

The meaning of a probabilistic database is a probability distribution on all database instances, which we call *possible worlds*, and denote $pwd(D^p)$. Fig. 3 (a) shows the eight possible instances with non-zero probabilities, which are computed by simply multiplying the tuple probabilities, as we have assumed them to be independent. For example, the probability of D_2 is $0.8*(1-0.5)*0.6 = 0.24$, since the instance contains the tuples s_1 and t_1 and does not contain s_2.

S^p

	A	B	
s_1	'm'	1	0.8
s_2	'n'	1	0.5

T^p

	C	D	
t_1	1	'p'	0.6

Fig. 2 A probabilistic database D^p

$pwd(D^p) =$

world	prob.
$D_1 = \{s_1, s_2, t_1\}$	0.24
$D_2 = \{s_1, t_1\}$	0.24
$D_3 = \{s_2, t_1\}$	0.06
$D_4 = \{t_1\}$	0.06
$D_5 = \{s_1, s_2\}$	0.16
$D_6 = \{s_1\}$	0.16
$D_7 = \{s_2\}$	0.04
$D_8 = \phi$	0.04

(a)

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

(b)

$q^{pwd}(D^p) =$

answer	prob.
$\{'p'\}$	0.54
$\emptyset$	0.46

(c)

Fig. 3 (a) The possible worlds for D^p in Figure 2, (b) a query q, and (c) its possible answers.

We now illustrate query evaluation on probabilistic databases. Consider the conjunctive query q in Fig. 3 (b). Its meaning on D^p is a set of possible answers, shown in Fig. 3 (c). It is obtained by applying q to each deterministic database in $pwd(D^p)$, and adding the probabilities of all instances that return the same answer. In our example we have $q(D_1) = q(D_2) = q(D_3) = \{'p'\}$, and $q(D_4) = \ldots = q(D_8) = \emptyset$. Thus, the probability of the answer being $\{'p'\}$ is $0.24+0.24+0.06 = 0.54$, while that of the answer $\emptyset$ is 0.46. This defines the set of possible answers, denoted $q^{pwd}(D^p)$. Notice that we have never used the structure of the query explicitly, but only applied it to deterministic databases taken from $pwd(D^p)$. Thus, one can give a similar semantics to *any* query q, no matter how complex, because we only need to know its meaning on deterministic databases.

The set of possible answers $q^{pwd}(D^p)$ may be very large, and it is impractical to return it to the user. In our simple example, there are only two possible answers, since T^p has only one tuple. In general, if T^p has n tuples, there can be as many as 2^n possible answers, and it is impractical to return them to the user.

Our approach is to compute for each possible tuple t a *probability rank* that t belongs to any answer, and sort tuples sorted by this rank. We denote this $q^{rank}(D^p)$. In our example this is:

$$q^{rank}(D^p) = \begin{array}{|c|c|}\hline \textbf{D} & \textbf{Rank} \\ \hline \text{'p'} & 0.54 \\ \hline \end{array}$$

In this simple example $q^{rank}(D^p)$ contains a single tuple and the distinction between q^{pwd} and q^{rank} is blurred. To see this distinction clearer, consider another query, $q_1(x) :- S^p(x,y), T^p(z,u), y = z$. Here q_1^{pwd} and q_1^{rank} are given by:

$$q_1^{pwd}(D^p) = \begin{array}{|l|l|}\hline \text{answer} & \text{probability} \\ \hline \{'m','n'\} & 0.24 \\ \{'m'\} & 0.24 \\ \{'n'\} & 0.06 \\ \emptyset & 0.46 \\ \hline \end{array}$$

$$q_1^{rank}(D^p) = \begin{array}{|l|l|}\hline \textbf{D} & \textbf{Rank} \\ \hline \text{'m'} & 0.48 \\ \text{'n'} & 0.30 \\ \hline \end{array}$$

For example, the rank probability of $'m'$ is obtained as $Pr(\{'m','n'\}) + Pr(\{'m'\})$. While in general $q^{pwd}(D^p)$ may be exponentially large, $q^{rank}(D^p)$ is simply a set of tuples, which are sorted by **Rank**. The problem in this paper is now to compute $q^{rank}(D^p)$ efficiently.

Extensional Query Semantics A natural attempt to compute $q^{rank}(D^p)$ is to represent q as a query plan then compute the probabilities of all tuples in all intermediate results. For the query q in Fig. 3 (b), such a plan is $\mathcal{P} = \Pi_{\mathbf{D}}(S^p \bowtie_{\mathbf{B=C}} T^p)$, and the corresponding probabilities are shown in Fig. 4. The formulas for the probabilities assume tuple independence, are taken from [12] and are rather straightforward (we review them in Sec. 4). For example the probability of a joined tuple $s \bowtie t$ is the product of the probabilities of s and t. Clearly, this approach is much more efficient than computing the possible worlds $q^{pwd}(D^p)$ and then computing $q^{rank}(D^p)$, but it is wrong ! It's answer is 0.636, while it should be 0.54. The reason is that the two tuples in $S^p \bowtie_{\mathbf{B=C}} T^p$ are not independent events, hence the formula used in $\Pi_{\mathbf{D}}$ is wrong.

However, let us consider an alternative plan, $\mathcal{P}' = \Pi_{\mathbf{D}}((\Pi_{\mathbf{B}}(S^p)) \bowtie_{\mathbf{B=D}} T^p)$. The extensional evaluation of this expression is shown in Figure 5, and this time we do get the correct answer. As we will show later, this plan will always compute the correct answer to q, on any probabilistic tables S^p, T^p. In this paper we show how to find automatically a plan whose extensional evaluation returns the correct answer to a query q. Finding such a plan requires pushing projections early (as shown in this example) and choosing join orders carefully.

A	**B**	**C**	**D**	*prob*
'm'	1	1	'p'	0.8*0.6 = 0.48
'n'	1	1	'p'	0.5*0.6 = 0.30

(a) $S^p \bowtie_{\mathbf{B=C}} T^p$

D	*prob*
'p'	(1 - (1 - 0.48)(1 - 0.3)) = 0.636

(b) $\Pi_{\mathbf{D}}(S^p \bowtie_{\mathbf{B=C}} T^p)$

Fig. 4 Evaluation of $\Pi_{\mathbf{D}}(S^p \bowtie_{\mathbf{B=C}} T^p)$

B	*prob*
1	(1 - (1 - 0.8)(1 - 0.5)) = 0.9

(a) $\Pi_{\mathbf{B}}(S^p)$

B	**C**	**D**	*prob*
1	1	'p'	0.9 * 0.6 = 0.54

(b) $\Pi_{\mathbf{B}}(S^p) \bowtie_{\mathbf{B=C}} T^p$

D	*prob*
'p'	0.54

(c) $\Pi_{\mathbf{D}}(\Pi_{\mathbf{B}}(S^p) \bowtie_{\mathbf{B=C}} T^p)$

Fig. 5 Evaluation of $\Pi_{\mathbf{D}}(\Pi_{\mathbf{B}}(S^p) \bowtie_{\mathbf{B=C}} T^p)$

Queries with uncertain matches While query evaluation on probabilistic databases is an important problem in itself, our motivation comes from answering SQL queries with uncertain matches, and ranking their results. We illustrate here with a simple example on the Stanford movie database[21].

```
SELECT   DISTINCT F.title, F.year
FROM     Director D, Films F
WHERE    D.did = F.did
  and    D.name ≈ 'Copolla'
  and    F.title ≈ 'rain man'
  and    F.year ≈ 1995
```

The predicates on the director name and the movie title and year are here *uncertain*.

Our approach is to translate the query into a regular query over a probabilistic databases. Each tuple in the table `Films` is assigned a probability based on how well it matches the predicates `title ≈ 'rain man'` and `year ≈ 1995`. Several techniques for doing this exist already, and in this paper we will adopt existing ones: see Sec. 9. The result is a probabilistic table, denoted $\texttt{Films}^p$. Similarly, the uncertain predicate on `Director` generates a probabilistic table $\texttt{Director}^p$. Then, we evaluate the following query, obtained by dropping the similarity predicates from the original SQL query:

```
SELECT   DISTINCT F.title, F.year
FROM     Director^p D, Films^p F
WHERE    D.did = F.did
```

This is similar to the query q considered earlier (Figure 3 (b)), and the same extensional plan can be used to evaluate it. Our system returns:

title	year	rank
The Rainmaker (by Coppola)	1997	0.110
The Rain People (by Coppola)	1969	0.089
Rain Man (by Levinson)	1988	0.077
Finian's Rainbow (by Coppola)	1968	0.069
Tucker, Man and Dream (Coppola)	1989	0.061
Rain or Shine (by Capra)	1931	0.059
...	...	...

3 Problem Definition

We review here the basic definitions in probabilistic databases, based on ideas taken from several papers [32, 12, 2], and state our problem.

Basic Notations We write R for a relation name, $Attr(R)$ for the set of its attributes, and $r \subseteq U^k$ for a relation instance, where k is $arity(R)$ and U is a fixed, finite universe of atomic values. We denote with $\bar{R} = R_1, \ldots, R_n$ a database schema, and write $D = r_1, \ldots, r_n$ for a database instance of that schema. We consider functional dependencies in this paper, and denote with Γ a set of functional dependencies. We write $D \models \Gamma$ when D satisfies the functional dependencies in Γ.

Probabilistic Events In order to represent probabilistic events, we use a set of symbols, AE: each $e \in AE$ represents an independent probabilistic event, called *atomic event*. We fix a probability function $Pr : AE \to [0, 1]$, associating a probability to each atomic event. A special symbol $\bot \in AE$ denotes the impossible event, and we define $Pr(\bot) = 0$.

A *complex event* is an expression constructed from atomic events using the operators $\wedge$, $\vee$, $\neg$. E denotes the set of all complex events. For each complex event e, let $Pr(e)$ be its probability.

Example 1 Consider $e = (s_1 \wedge t_1) \vee (s_2 \wedge t_1)$, and assume $Pr(s_1) = 0.8$, $Pr(s_2) = 0.5$, $Pr(t_1) = 0.6$. To compute $Pr(e)$ we construct the truth table for $e(s_1, s_2, t_1)$ and identify the entries where e is true, namely $(1, 0, 1)$, $(0, 1, 1)$, $(1, 1, 1)$. The three entries have probabilities given by $Pr(s_1)(1-Pr(s_2))Pr(t_1) = 0.8\times0.5\times0.6 = 0.24$, $(1-Pr(s_1))Pr(s_2)Pr(t_1) = 0.06$ and $Pr(s_1)Pr(s_2)Pr(t_1) = 0.24$ respectively. Then $Pr(e)$ is their sum, 0.54.

This method of computing probabilities generalizes to any complex event $e(s_1, \ldots, s_k)$ which is a function of atomic events $s_1, \ldots, s_k$. But it is important to notice that this algorithm is exponential in k, the number of atomic events, because the size of the truth table for $e(s_1, \ldots, s_k)$ is 2^k. This cannot be avoided: it is known that computing $Pr(e)$ is #P-complete [36] even for complex events without negation.

Probabilistic Databases A *probabilistic relation* is a relation with a distinguished *event attribute* E, whose value is a complex event. We add the superscript p to mean "probabilistic". Thus, R^p denotes a relation name having E among it's attributes; r^p denotes an instance of R^p, i.e. where for each tuple $t \in r^p$, $t.E$ is a complex event; $\bar{R}^p$ denotes a database schema where some relations names have an E attribute; and, finally, Γ^p denotes a set of functional dependencies over $\bar{R}^p$, where the functional dependencies may use both regular attributes and the event attribute. Note that we allow for a probabilistic database schema $\bar{R}^p$ to consist of both probabilistic relations and deterministic relations.

As a convenient notation, by dropping the p superscript we mean the deterministic part, obtained by removing the event attribute. Thus, R is such that $Attr(R) = Attr(R^p) - \{E\}$, while r represents the project of r^p on all attributes other than E. The intuition is that users "see" only R, but the system needs to access the event attribute $R^p.E$ in order to compute correctly the probability ranks. For a simple illustration, consider the probabilistic relation name $S^p(A, B, E)$: then S has attributes A, B. Users care to see a binary table $S(A, B)$, while the system maintains probabilistic events in $S^p(A, B, E)$.

We assume that the set of functional dependencies Γ^p always contains, or implies the following functional dependency:

$$Attr(R) \to R^p.E$$

for every relation R^p: this ensures that we don't associate two different events e_1 and e_2 to the same tuple t (instead, we may want to associate $e_1 \vee e_2$ to t).

In addition to this *tabular* representation of a probabilistic relation, we consider a *functional* representation, where a probabilistic instance r^p, of type R^p, is described by the following function $e_R : U^k \to E$, where $k = arity(R)$. When t occurs in r^p and its event is $t.E = e$, then $e_R(t) = e$, otherwise $e_R(t) = \bot$. Conversely, one can recover r^p from the function e_R by collecting all tuples for which $e_R(t) \neq \bot$.

The probabilistic databases we consider have only atomic events: complex events are introduced only during query evaluation. If a probabilistic relation has a distinct atomic event for each tuple, we call it a *tuple-independent* probabilistic relation. For example, both the relations in Fig. 2 are tuple-independent, where the atomic events are s_1, s_2, t_1 respectively.

Possible Worlds Database A *possible worlds relation* of type R is simply a probability distribution on the set of all deterministic relations of type R. Similarly, a *possible worlds database* of schema $\bar{R}$ is a probability distribution on all possible data instances with the given schema.

Semantics of a probabilistic database Now, we give a simple and intuitive meaning to a probabilistic relation. Consider a probabilistic relation r^p of type R^p and let r be its deterministic part. The meaning of r^p is a possible worlds relation, which we denote $pwd(r^p)$ and is defined as follows. Let $e_R : U^k \to E$ be the functional representation of r^p. Given $s \subseteq U^k$, $Pr(s)$ is defined to

be $Pr(\bigwedge_{t\in s} e_R(t) \wedge \bigwedge_{t\notin s} \neg e_R(t))$. Intuitively, this is the probability that exactly the tuples in s are "in" and all the others are "out". One can check that $\sum_{s\subseteq U^k} Pr(s) = 1$. Also, it follows that for $s \not\subseteq r$, $Pr(s) = 0$ because elements not in r have the event $\perp$. Similarly, the meaning of a probabilistic database D^p is a probability distribution on all deterministic databases D, denoted $pwd(D^p)$. For an instance, $D = s_1, \ldots, s_n$ and its probability is $Pr(s_1)Pr(s_2)\ldots Pr(s_n)$.

We discuss next how possible worlds relations interact with functional dependencies, and focus on a single relation name R^p. Given a set of functional dependencies Γ^p, denote Γ its projection to the deterministic attributes of R^p, i.e. let Γ consist of those dependencies that can be derived from Γ^p and that do not refer to the event attribute $R^p.E$. For example, if R^p is $R^p(A, B, E)$ and $\Gamma^p = \{A \to E, E \to B\}$, then $\Gamma = \{A \to B\}$. Consider a probabilistic instance r^p that satisfies Γ^p (in notation: $r^p \models \Gamma^p$), and let Pr be the probability distribution on instances induced by r_p (the possible worlds relation). The question is if a relation s that does not satisfy Γ can have a non-zero probability, i.e. $Pr(s) > 0$. The answer is no: if $Pr(s) > 0$, s must be a subset of r (which is r^p without the event attribute), and, hence, must also satisfy the functional dependencies in Γ.

Query semantics Let q be a query of arity k over a deterministic schema $\bar{R}$. We define a very simple and intuitive semantics for the query. Users think of q as normal query on a deterministic database, but the database is given by a probability distribution rather than being fixed. As a result, the query's answer is also a probability distribution. Formally, given a query q and a probabilistic database D^p: $q^{pwd}(D^p)$ is the following probability distribution on all possible answers, $Pr_q : \mathcal{P}(U^k) \to [0, 1]$:

$$\forall S \subseteq U^k, Pr_q(S) = \sum_{D|q(D)=S} Pr(D)$$

We call this the *possible worlds semantics*. This definition makes sense for every query q that has a well defined semantics on all deterministic databases.

It is impossible to return $q^{pwd}(D^p)$ to the user. Instead, we compute a *probabilistic ranking* on all tuples $t \in U^k$, defined by the function: $rank_q(t) = \sum_S \{Pr_q(S) \mid S \subseteq U^k, t \in S\}$, for every tuple $t \in U^k$. We denote with $q^{rank}(D^p)$ a tabular representation of the function $rank_q$: this is a table with $k + 1$ attributes, where the first k represent a tuple in the query's answer while the last attribute, called **Rank** is a real number in $[0, 1]$ representing its probability.

The Query Evaluation Problem This paper addresses the following problem: given schema $\bar{R}^p, \Gamma^p$, a probabilistic database D^p and a query q over schema $\bar{R}$, compute the probabilistic rankings $q^{rank}(D^p)$.

Application to queries with uncertain predicates Consider now a deterministic database D and a query $q^{\approx}$ that explicitly mentions some uncertain predicates. We convert this problem into evaluating a deterministic query q, obtained by removing all uncertain predicates from $q^{\approx}$, on a probabilistic database, obtained by associating a probability $Pr(t)$ to each tuple t based on how well t satisfies the uncertain predicates.

3.1 Representation power of probabilistic databases

We have shown that every probabilistic database has a representation as a possible worlds database. We show here the converse: that probabilistic databases are powerful enough to represent any possible distribution on databases. Recall that in a probabilistic database, we can only store with each tuple an event that is a boolean function of independent atomic events.

Theorem 1 *Let W be any possible worlds database, i.e. a probability distribution on the set of data instances. Then, there exists a probabilistic database D^p such that $W = pwd(D^p)$.*

Proof We have a fixed, finite universe U. Let $a_1, a_2, \cdots$ be the arities of the relations in the schema and let $k = |U|^{a_1} + |U|^{a_2} + \cdots$. Thus, k denotes the number of distinct possible tuples. The number of possible databases is $n = 2^k$. Let $D_1, \cdots D_n$ be all possible data instances. $W(D_i)$ denotes the probability of D_i.

We will create $n-1$ independent events $E = \{e_1, \cdots, e_{n-1}\}$ with probabilities $p(e_1), \cdots, p(e_{n-1})$ and a boolean expression $f_W(D_i)$ in terms of these events for each data instance so that the following holds: $W(D_i) = p(f_W(D_i))$ for all i, and the events $f_W(D_i)$ are all disjoint.

We do it recursively. Let $p_L = \sum_{i=0}^{2^{k-1}} W(D_i)$. Consider the distribution W_L that is given by $W_L(D_i) = W(D_i)/p_L$ on domain $\{D_1, \cdots, D_{2^{k-1}}\}$ and distribution W_R given by $W_R(D_i) = W(D_i)/(1 - p_L)$ on the domain $\{D_{2^{k-1}+1}, \cdots, D_{2^k}\}$.

Recursively, represent W_L and W_R using $2^{k-1} - 1$ independent variables each. Also create a new variable e with $p(e) = p_L$. Define

$$f_W(D_i) = \begin{cases} f_{W_L}(D_i) \wedge e & i \le 2^{k-1} \\ f_{W_R}(D_i) \wedge \neg(e) & i > 2^{k-1} \end{cases}$$

We see that f_W uses $2^k - 1$ independent variables. It is easy to verify that $W(D_i) = p(f_W(D_i))$ and $f_W(D_i)$ are all disjoint events.

Finally, we create a probabilistic database as possible. For a relation R in the schema, and for every tuple t, let $e_R(t) = \bigvee_{\{D_i|t\in D_i.R\}} f_W(D_i)$. Since the events $f_W(D_i)$ are all disjoint, one can verify that when this probabilistic database is converted to possible worlds database, we get back W.

$$
\begin{array}{lcl}
e_{\sigma^i_c(\mathcal{P})}(t) & = & \begin{cases} e_{\mathcal{P}}(t) & \text{if } c(t) \text{ is true} \\ \bot & \text{if } c(t) \text{ is false} \end{cases} \\
e_{\Pi^i_{\bar{A}}(\mathcal{P})}(t) & = & \bigvee_{t':\Pi_{\bar{A}}(t')=t} e_{\mathcal{P}}(t') \\
e_{\mathcal{P}\times^i\mathcal{P}'}(t,t') & = & e_{\mathcal{P}}(t) \wedge e_{\mathcal{P}'}(t')
\end{array}
$$

Fig. 6 Intensional Evaluation

4 Query Evaluation

We turn now to the central problem, evaluating $q^{rank}(D^p)$ for a query q, and a probabilistic database D^p. Applying the definition directly is infeasible, since it involves iterating over a large set of database instances. Instead, we will first review the intensional evaluation of [12], then describe our approach in Sec 4.3.

We restrict our discussion first to conjunctive queries, or, equivalently select (distinct)-project-join queries. This helps us better understand the query evaluation problem and its complexity, and will consider more complex query expressions in Sec. 8. We use either datalog notation for our queries q, or plans $\mathcal{P}$ in the select/project/product algebra[3]: $\sigma, \Pi, \times$.

4.1 Intensional Query Evaluation

One method for evaluating queries on probabilistic databases is to use complex events. We review it here and discuss its limitations. Start by expressing q as a query plan, using the operators $\sigma, \Pi, \times$. Then modify each operator to compute the event attribute E in each intermediate result: denote $\sigma^i, \Pi^i, \times^i$ the modified operators. It is more convenient to introduce them in the functional representation, by defining the complex event $e_{\mathcal{P}}(t)$ for each tuple t, inductively on the query plan $\mathcal{P}$, as shown in Fig 6.

The tabular definitions for $\sigma^i, \Pi^i, \times^i$ follow easily: σ^i acts like σ then copies the complex events from the input tuples to the output tuples; Π^i associates to a tuple t the complex event $e_1 \vee \ldots \vee e_n$ obtained from the complex events of all input tuples $t_1, \ldots, t_n$ that project into t; and $\times^i$ simply associates to a product tuple (t, t') the complex event $e \wedge e'$.

Example 2 Let us consider the database D^p described in Fig. 2. Consider the query plan, $\mathcal{P} = \Pi_D(S^p \bowtie_{B=C} T^p)$. Fig. 7 shows the intensional evaluation of the query (we used the tuple names as atomic events). $\mathcal{P}^i(D^p)$ contains a single tuple $'p'$ with the event $(s_1 \wedge t_1) \vee (s_2 \wedge t_1)$.

It can be shown that $\mathcal{P}^i(D^p)$ does not depend on the particular choice of plan $\mathcal{P}$, and we denote $q^i(D^p)$ the value $\mathcal{P}^i(D^p)$ for any plan $\mathcal{P}$ for q, and call it the

[3] Notice that Π also does duplicate elimination

A	B	C	D	E
'm'	1	1	'p'	$s_1 \wedge t_1$
'n'	1	1	'p'	$s_2 \wedge t_1$

(a) $S^p \bowtie^i_{B=C} T^p$

D	E
'p'	$(s_1 \wedge t_1) \vee (s_2 \wedge t_1)$

(b) $\Pi^i_D(S^p \bowtie^i_{B=C} T^p)$

D	Rank
$'p'$	$Pr((s_1 \wedge t_1) \vee (s_2 \wedge t_1)) = 0.54$

(c) $q^{rank}(D^p) = Pr(\Pi^i_D(S^p \bowtie^i_{B=C} T^p))$

Fig. 7 Intensional Evaluation of $\Pi_D(S^p \bowtie_{B=C} T^p)$

intensional semantics[4] of q on the probabilistic database D^p. We prove now that it is equivalent to the possible worlds semantics, $q^{pwd}(D^p)$.

Theorem 2 *The intensional semantics and the possible worlds semantics on probabilistic databases are equivalent for conjunctive queries. More precisely, $pwd(q^i(D^p)) = q^{pwd}(D^p)$ for every probabilistic database D^p and every conjunctive query q.*

Proof Every tuple t in $q^i(D^p)$ has a complex event $t.E$ associated with it. $pwd(q^i(D^p))$ consists of a set of worlds, each world assigning a truth value to the set of atomic events. A tuple t belongs to a world in $pwd(q^i(D^p))$ if $t.E$ is true in that world.

$q^{pwd}(D^p)$ also consists of a set of worlds, each assigning a truth value to the set of atomic events. The content of a world in $q^{pwd}(D^p)$ is the output of q on the database defined by that world.

Given a world W (i.e. a deterministic database W), which is defined by an assignment of truth values to the atomic variables, let $q^{pwd}(D^p)[W]$ and $pwd(q^i(D^p))[W]$ denote the set of tuples in the corresponding worlds.

We will prove, by induction on the size of q, that for all W, $q^{pwd}(D^p)[W] = pwd(q^i(D^p))[W]$. This will show that both the semantics result in exactly the same possible worlds with same probabilities.

If a query just selects a single relation, this holds trivially. If q is a larger query, there are three possibilities:

1. $q = \Pi_A(q_1)$. Consider any world W. By induction hypothesis, $pwd(q_1^i(D^p))[W] = q_1^{pwd}(D^p)[W]$. Thus,

[4] In [12] this is the only query semantics considered.

$$
\begin{aligned}
t \in q^{pwd}(D^p)[W] &\Leftrightarrow t.E = \text{true} \\
&\Leftrightarrow (\vee_{t_1:\Pi_A(t_1)=t} t_1.E) = true \\
&\Leftrightarrow \exists t_1, \Pi_A(t_1) = t, \\
&\quad t_1 \in pwd(q_1^i(D^p))[W] \\
&\Leftrightarrow \exists t_1, \Pi_A(t_1) = t, \\
&\quad t_1 \in q_1^{pwd}(D^p)[W] \\
&\Leftrightarrow t \in q^{pwd}(D^p)[W]
\end{aligned}
$$

2. $q = \sigma_c(q_1)$. Consider any world W. Again, we have

$$pwd(q_1^i(D^p))[W] = q_1^{pwd}(D^p)[W]$$

t belongs to $pwd(q^i(D^p))[W]$ iff $t.E$ is true, i.e. t satisfies σ_c and t belongs to $pwd(q_1^i(D^p))[W]$. Similarly, t belongs to $pwd(q^i(D^p))[W]$ iff t satisfies σ_c and t belongs to $q_1^{pwd}(D^p)[W]$. Therefore, we get

$$pwd(q^i(D^p))[W] = q^{pwd}(D^p)[W]$$

3. $q = q_1 \bowtie q_2$. We have

$$
\begin{aligned}
pwd(q_1^i(D^p))[W] &= q_1^{pwd}(D^p)[W], \\
pwd(q_2^i(D^p))[W] &= q_2^{pwd}(D^p)[W]
\end{aligned}
$$

Given a tuple $t = (t_1, t_2)$ belonging to q, $t.E = t_1.E \wedge t_2.E1$. Thus,

$$
\begin{aligned}
t \in q^{pwd}(D^p)[W] &\Leftrightarrow t.E = \text{true} \\
&\Leftrightarrow t_1.E = \text{true},\ t_2.E = \text{true} \\
&\Leftrightarrow t_1 \in pwd(q_1^i(D^p))[W], \\
&\quad t_2 \in pwd(q_2^i(D^p))[W] \\
&\Leftrightarrow t_1 \in q_1^{pwd}(D^p)[W], \\
&\quad t_2 \in q_2^{pwd}(D^p)[W] \\
&\Leftrightarrow t \in q^{pwd}(D^p)[W]
\end{aligned}
$$

Thus, by induction, $pwd(q^i(D^p))$ and $q^{pwd}(D^p)$ are equal.

Theorem 2 allows us to compute $q^{rank}(D^p)$ as follows. First compute $q^i(D^p)$, then compute the probability $Pr(e)$ for each complex event. Then $q^{rank}(D^p) = Pr(q^i(D^p))$.

Example 3 Fig. 7(c) shows $p^{rank}(D^p)$ for Ex. 2. $Pr((s_1 \wedge t_1) \vee (s_2 \wedge t_1))$ was shown in Ex. 1.

It is very impractical to use the intensional semantics to compute the rank probabilities for two reasons. First, the event expressions in $q^i(D^p)$ can become very large, due to the projections. In the worst case the size of such an expression can become of the same order of magnitude as the database. This increases the complexity of the query operators significantly, and makes the task of an optimizer much harder, because now the cost per tuple is not longer constant. Second, for each tuple t one has to compute $Pr(e)$ for its event e, which is a #P-complete problem.

$$
\begin{array}{lcl}
Pr_R(t) & = & Pr(e_R(t)) \\
Pr_{\sigma_c^e(\mathcal{P})}(t) & = & \begin{cases} Pr_{\mathcal{P}}(t) & \text{if } c(t) \text{ is true} \\ 0 & \text{if } c(t) \text{ is false} \end{cases} \\
Pr_{\Pi^e_{\bar{A}}(\mathcal{P})}(t) & = & 1 - \prod_{t':\Pi_{\bar{A}}(t')=t}(1 - Pr_{\mathcal{P}}(t')) \\
Pr_{\mathcal{P} \times^e \mathcal{P}'}(t, t') & = & Pr_{\mathcal{P}}(t) \times Pr_{\mathcal{P}'}(t')
\end{array}
$$

Fig. 8 Extensional Evaluation

4.2 Extensional Query Evaluation

We now modify the query operators to compute probabilities rather than complex events: we denote $\sigma^e, \Pi^e, \times^e$ the modified operators. This is much more efficient, since it involves manipulating real numbers rather than event expressions. We define a number $Pr_{\mathcal{P}}(t) \in [0, 1]$ for each tuple t, by induction on the structure of the query plan $\mathcal{P}$. The inductive definitions in Fig 4.2 should be compared with those in Fig 6. Unlike the formulas in Fig. 6, the extensional operators assume that the input tuples have independent events. Recall that $e_R(t)$ is the event associated with tuple t in relation R as defined in Sec 3.

Again, the tabular definitions of $\sigma^e, \Pi^e, \times^e$ follow easily: σ^e acts like σ then propagates the tuples' probabilities from the input to the output, Π^e computes the probability of a tuples t as $1-(1-p_1)(1-p_2)\dots(1-p_n)$ where $p_1, \dots, p_n$ are the probabilities of all input tuples that project to t, while $\times$ computes the probability of each tuple (t, t') as $p \times p'$.

We call the result, $\mathcal{P}^e(D^p)$, the *extensional semantics* of the plan $\mathcal{P}$. If we know $\mathcal{P}^e(D^p) = q^{rank}(D^p)$, then we simply execute the plan under the extensional semantics. But, unfortunately, this is not always the case, as we saw in Sec. 2. Moreover, $\mathcal{P}^e(D^p)$ depends on the particular plan $\mathcal{P}$ chosen for q. Our goal is to find a plan for which the extensional semantics is correct.

Definition 1 Given a schema $\bar{R}^p, \Gamma^p$, a plan $\mathcal{P}$ for a query q is *safe* if $\mathcal{P}^e(D^p) = q^{rank}(D^p)$ for all instances D^p of that schema.

We show next how to find a safe plan.

4.3 The Safe-Plan Optimization Algorithm

Recall that a probabilistic database schema $\bar{R}^p$ may consists both of probabilistic relation names, which have an event attribute E, and deterministic relation names. Consider a conjunctive query q; we use the following notations:

- $Rels(q) = \{R_1, \dots, R_k\}$ all relation names occurring in q. We assume that each relation name occurs at most once in the query (more on this in Sec. 8).
- $PRels(q)$ = the probabilistic relation names in q, $PRels(q) \subseteq Rels(q)$.

- $Attr(q)$ = all attributes in all relations in q. To disambiguate, we denote attributes as $R_i.A$.
- $Head(q)$ = the set of attributes that are in the output of the query q. $Head(q) \subseteq Attr(q)$.

For a simple illustration, consider the three relations $R^p(A, B, C, E)$, $S(D, F)$, $T(G, H, K)$ and the query:

$$q(A, F) \ :- \ R^p(A, B, C), S(D, F), T(G, H, K), \\ A = D, D = H, F = K$$

Then $Rels(q) = \{R^p, S, T\}$, $PRels(q) = \{R^p\}$, $Attr(q) = \{A, B, C, D, F, G, H, K\}$ and $Head(q) = \{A, F\}$.

Let q be a conjunctive query. We define the *induced* functional dependencies $\Gamma^p(q)$ on $Attr(q)$:

- Every FD in Γ^p is also in $\Gamma^p(q)$.
- For every join predicate $R_i.A = R_j.B$, both $R_i.A \rightarrow R_j.B$ and $R_j.B \rightarrow R_i.A$ are in $\Gamma^p(q)$.
- For every selection predicate $R_i.A = c$, $\emptyset \rightarrow R_i.A$ is in $\Gamma^p(q)$.

We seek a safe plan $\mathcal{P}$, i.e. one that computes the probabilities correctly. For that each operator in $\mathcal{P}$ must be safe, i.e. compute correct probabilities: we define this formally next.

Let q_1, q_2 be two queries, and let $op \in \{\sigma, \Pi, \times\}$ be a relational operator. Consider the new query $op(q_1, q_2)$ (or just $op(q_1)$ when op is unary). We say that op^e is *safe* if $op^e(Pr(q_1^i(D^p)), Pr(q_2^i(D^p))) = Pr(op^i(q_1^i(D^p)), q_2^i(D^p))$ (and similarly for unary operators), $\forall D^p$ s.t. $D^p \models \Gamma^p$. In other words, op is safe if, when given correct probabilities for its inputs op^e computes correct probabilities for the output tuples.

Theorem 3 *Consider a database schema where all the probabilistic relations are tuple-independent. Let q, q' be conjunctive queries that do not share any relation name. Then,*

1. *σ_c^e is always safe in $\sigma_c(q)$.*
2. *$\times^e$ is always safe in $q \times q'$.*
3. *$\Pi^e_{A_1,\ldots,A_k}$ is safe in $\Pi_{A_1,\ldots,A_k}(q)$ iff for every $R^p \in PRels(q)$ the following can be inferred from $\Gamma^p(q)$:*

$$A_1, \ldots, A_k, R^p.E \rightarrow Head(q) \quad (1)$$

Proof

1. Follows trivially from definition.
2. Since we assume all relations in the query to be distinct, the complex events in the output of q and q' comprise of distinct atomic events. Thus, given a tuple $t_{join} = (t, t') \in q \times q'$, $Pr(t_{join}.E) = Pr(t.E \wedge t'.E) = Pr(t.E)Pr(t'.E)$. So, the join operator is safe. The independence assumed by the operator indeed holds.
3. For each output tuple t of the project, consider the set S_t of input tuples that map to t. The operator is safe if for each such t, the complex events corresponding to tuples in S_t are independent. Thus, among all tuples having the same value for $A_1, \cdots A_k$, no atomic event (i.e. $R^p.E$ for some probabilistic relation R^p) occurs in two of them having different values of $Head(q)$. Thus, the following functional dependency must hold for each R^p.

$$A_1, \ldots, A_k, R^p.E \rightarrow Head(q)$$

Note that we can replace $R^p.E$ in Eq.(1) by $Attr(R^p)$, since for tuple-independent relations, $R^p.E \rightarrow Attr(R^p)$ and $Attr(R^p) \rightarrow R^p.E$ always hold.

Obviously, a plan $\mathcal{P}$ consisting of only safe operators is safe. As we prove below, the converse also holds.

Theorem 4 *Let $\mathcal{P}$ be a safe relational algebra plan for a query q consisting of selects, projects and joins. Then, all operators in p are safe.*

Proof We will prove the following statement: If $\mathcal{P}$ is a plan for a query q that has at least one unsafe operator and t is any tuple that can be produced by q, there is a database D^p such that $\mathcal{P}^e(D^p)$ consists of a single tuple t with incorrect probability.

First, it is easy to see that for any plan $\mathcal{P}$ and database D^p, $\mathcal{P}^e(D^p) \geq q^{rank}(D^p)$. This is because the only operators that are unsafe are projects, and it is easy to see that they can only overestimate the probabilities. Thus, there can only be one-sided errors in the probabilities.

We will prove the theorem by induction on the size of the plan.

If $\mathcal{P}$ returns a single relation, its safe and the assertion holds trivially.

Consider a larger plan $\mathcal{P}$ and assume it is safe. There are three cases:

1. $\mathcal{P} = q_1 \bowtie q_2$. Since $\mathcal{P}$ is unsafe, at least one of q_1 and q_2 is unsafe. W.L.O.G, assume q_1 is unsafe. Given any tuple $t = (t_1, t_2)$, a database can be constructed such that q_1 results in a single tuple t_1 with incorrect probability. Since $Pr_{\mathcal{P}}(t_1, t_2) = Pr_{q_1}(t_1) * Pr_{q_2}(t_2)$, for the final probability to be correct, both $Pr_{q_1}(t_1)$ and $Pr_{q_2}(t_2)$ have to be correct (since errors can only be one-sided). Thus, we have constructed a database that results in a single tuple t with incorrect probability.
2. $\mathcal{P} = \Pi_A(q_1)$. If q_1 is safe, the final project operator must be unsafe and hence, by definition, there exists a database where the final result is incorrect. If t is any tuple in the output whose probability is incorrect, we can restrict the database to produce only the tuple t (whose probability is still incorrect).
   If q_1 is not safe, there is a database on which q_1 produces a single tuple with incorrect probability. Thus, the final project also produces a single tuple whose probability is still incorrect.
3. $\mathcal{P} = \sigma_c(q_1)$. Consider any tuple t that satisfies the condition c and create a database on which q_1 produces t with incorrect probability. Thus, $\sigma_c(q_1)$ also produces a single tuple with incorrect probability.

We explain safe plans with an example below.

Example 4 Continuing the example in Sec. 2, assume that both S^p and T^p are tuple-independent probabilistic relations. Hence Γ^p is:

$$S^p.A, S^p.B \rightarrow S^p.E$$
$$T^p.C, T^p.D \rightarrow T^p.E$$
$$S^p.E \rightarrow S^p.A, S^p.B$$
$$T^p.E \rightarrow T^p.C, T^p.D$$

The last two functional dependencies follow from the fact that a tuple-independent relation has a distinct atomic event for each tuple. Consider the plan $\Pi_D(S^p \bowtie_{B=C} T^p)$. We have shown in Fig. 4 that, when evaluated extensionally, this plan is incorrect. We explain here the reason: the operator Π^e_D is not safe. An intuitive justification can be seen immediately by inspecting the relation $S^p \bowtie^i_{B=C} T^p$ in Fig. 7 (a). The two complex events share the common atomic event t_1, hence they are correlated probabilistic events, while the formula for Π^e_D only works when these events are independent. We show how to detect formally that Π^e_D is unsafe. We need to check:

$$T^p.D, S^p.E \rightarrow S^p.A, S^p.B, T^p.C, T^p.D$$
$$T^p.D, T^p.E \rightarrow S^p.A, S^p.B, T^p.C, T^p.D$$

The first follows from Γ^p and from the join condition $B = C$, which adds $S^p.B \rightarrow T^p.C$ and $T^p.C \rightarrow S^p.B$. But the second fails: $T^p.D, T^p.E \not\rightarrow S^p.A$.

Example 5 Continuing the example, consider now the plan $\Pi_D(\Pi_B(S^p) \bowtie_{B=C} T^p)$. We will prove that Π^e_D is safe. For that we have to check:

$$T^p.D, S^p.E \rightarrow S^p.B, T^p.C, T^p.D$$
$$T^p.D, T^p.E \rightarrow S^p.B, T^p.C, T^p.D$$

Both hold, hence Π^e_D is safe. Similarly, Π^e_B is safe in $\Pi_B(S^p)$, which means that the entire plan is safe.

Before we describe our algorithm for finding a safe plan, we need some terminology.

Definition 2 *(Separate relations) Let q be a conjunctive query. Two relations $R_i, R_j \in Rels(q)$ are called connected if the query contains a join condition $R_i.A = R_j.B$ and either $R_i.A$ or $R_i.B$ is not in $Head(q)$. The relations R_i, R_j are called separate if they are not connected.*

Definition 3 *(Separation) Two sets of relations $\mathscr{R}_i$ and $\mathscr{R}_j$ are said to form a separation for query q iff*

1. *They partition the set $Rels(q)$*
2. *For any pair of relations R_i and R_j such that $R_i \in \mathscr{R}_i$ and $R_j \in \mathscr{R}_j$, they are separate.*

Algorithm 1 is our optimization algorithm for finding a safe plan. It proceeds top-down, as follows. First, it tries to do all safe projections late in the query plan. When no more late safe projections are possible for a query q, it tries to perform a join $\bowtie_c$ instead, by splitting q into $q_1 \bowtie_c q_2$. Since $\bowtie_c$ is the last operation in the query plan, all attributes in c must be in $Head(q)$. Hence, $Rels(q_1)$ and $Rels(q_2)$ must form a separation.

To find a separation, we construct a graph $G(q)$ which we call the *constraint graph*. The nodes of $G(q)$ are $Rels(q)$ and the edges are all pairs (R_i, R_j) of connected relations, i.e. s.t. q contains some join condition $R_i.A = R_j.B$ with either $R_i.A$ or $R_j.B$ not in $Head(q)$. Find the connected components of $G(q)$, and choose q_1 and q_2 to be any partition of these connected components: this defines $Rels(q_i)$ and $Attr(q_i)$ for $i = 1, 2$. Define $Head(q_i) = Head(q) \cap Attr(q_i)$, for $i = 1, 2$. If $G(q)$ is a connected graph, then the query has no safe plans (more on this below). If $G(q)$ has multiple connected components, then we have several choices for splitting q. If q has a safe plan at all, then we will show that all these choices lead to a safe plan; hence we can deploy any standard cost based optimizations algorithm that works in top-down fashion to select the cheapest plan among several possibilities[5]. More on this in Sec 6.

Finally, the algorithm terminates when no more projections are needed. The remaining join and/or selection operators can be done in any order.

Algorithm 1 SAFE-PLAN(q)

```
if Head(q) = Attr(q) then
    return any plan P for q
      (P is projection-free, hence safe)
end if
for A ∈ (Attr(q) − Head(q)) do
    let q_A be the query obtained from q
      by adding A to the head variables
    if Π_{Head(q)}(q_A) is a safe operator then
        return Π_{Head(q)}(SAFE-PLAN(q_A))
    end if
end for
Split q into q_1 ⋈_c q_2
    s.t. ∀R_1 ∈ Rels(q_1) R_2 ∈ Rels(q_2)
    R_1, R_2 are separated.
if no such split exists then
    return error("No safe plans exist")
end if
return SAFE-PLAN(q_1) ⋈_c SAFE-PLAN(q_2)
```

Example 6 Continuing the example in Sec. 2, consider the original query in Fig. 3 (b), which we rewrite now

[5] It is also possible to adapt our algorithm to work with a bottom-up optimizer.

as:

$$q(D) :- S^p(A,B), T^p(C,D), B = C$$

Here $Attr(q) = \{A,B,C,D\}$ and $Head(q) = \{D\}$. The algorithm first considers the three attributes A, B, C in $Attr(q) - Head(q)$, trying to see if they can be projected out late in the plan. A cannot be projected out. Indeed, the corresponding q_A is:

$$q_A(A,D) :- S^p(A,B), T^p(C,D), B = C$$

and Π^e_D is unsafe in $\Pi_D(q_A)$ because $T^p.D, T^p.E \not\rightarrow S^p.A$, as we saw in Example 4. However, B and C can be projected out. By successively doing the projections, we get the plan for q as $\Pi_D(\Pi_{BD}(q_{BC}))$, where:

$$q_{BC}(B,C,D) :- S^p(A,B), T^p(C,D), B = C$$

Now we process q_{BC}, where $Attr(q_{BC}) = \{A,B,C,D\}$, $Head(q_{BC}) = \{B,C,D\}$. No projection is possible, but we can split the query into $q_1 \bowtie_{B=C} q_2$ where q_1, q_2 are:

$$\begin{aligned} q_1(B) &:- S^p(A,B) \\ q_2(C,D) &:- T^p(C,D) \end{aligned}$$

The split $q_{BC} = q_1 \bowtie_{B=C} q_2$ is indeed possible since both B and C belong to $Head(q_{BC})$. Continuing with q_1, q_2, we are done in q_2, while in q_1 we still need to project out A, $q_1 = \Pi_B(S^p)$, which is safe since $B, S^p.E \rightarrow A$. Putting everything together gives us the following safe plan:

$$\mathcal{P}' = \Pi_D(\Pi_{DB}((\Pi_B(S^p) \bowtie_{B=C} T^p))$$

The plan can be further optimized. For instance, it can be shown that the projection Π_{DB} is redundant in the above join. We discuss the optimization of safe plans in details in Sec 6.

We state now the soundness of our algorithm: the proof follows easily from the fact that all projection operators are safe. We prove in the next section that the algorithm is also complete.

Proposition 1 *The* SAFE-PLAN *optimization algorithm is sound, i.e. any plan it returns is safe.*

Proof Define the size of the query q to be $|Rels(q)| + |Attr(q)| - |Head(q)|$. We will prove the proposition by induction on the size of the query.

Algorithm 1 returns a plan in the following three cases:

1. It returns at line 2. In this case, the plan consists only of joins and selects, and hence, is safe.
2. It returns at line 9. Note that q_A is smaller in size that q. By induction, SAFE-PLAN(q_A) is safe. Also, by definition, the final project operator is safe. Thus, the algorithm returns a safe plan.
3. The algorithm returns at line 16. q_1 and q_2 are both smaller that q. By induction, both SAFE-PLAN(q_1) and SAFE-PLAN(q_1) are safe plans. These plans are then connected by a join operator, which is always safe. So the returned plan is safe.

4.4 Completeness of Safe-Plan algorithm

We have shown that SAFE-PLAN algorithm is sound. We next prove that it is complete, i.e., if the query is safe, the algorithm finds a safe plan for it.

We start with few basic results about extensional evaluation.

Lemma 1 *Under extensional evaluation, for any query q,*

$$\Pi^e_A(\Pi^e_{A\cup B}(q)) = \Pi^e_A(q)$$

Proof These two expressions have the same output tuples. We only have to show that the tuples have the same probability in both expressions.

Consider a tuple t belonging to $\Pi_A(\Pi_{A\cup B}(q))$. Then,

$$\begin{aligned} & 1 - Pr_{\Pi_A(\Pi_{A\cup B}(q))}(t) \\ = & \prod_{(t'|\Pi_A(t')=t)} (1 - Pr_{\Pi_{A\cup B}(q)}(t')) \\ = & \prod_{(t'|\Pi_A(t')=t)} \prod_{(t''|\Pi_{A\cup B}(t'')=t')} (1 - Pr_q(t'')) \\ = & \prod_{(t''|\Pi_A(t'')=t)} (1 - Pr_q(t'')) \\ = & 1 - Pr_{\Pi_A(q)} \end{aligned}$$

This proves the lemma.

Lemma 2 *Consider a query q and let A be any attribute in $Head(q)$. If $\Pi_{(Head(q)\setminus A)}(q)$ has a safe plan, q also has a safe plan.*

Proof Let $\mathcal{P}$ be a safe plan for query $\Pi_{(Head(q)-A)}(q)$. By Lemma 1, we can assume that each project operator in $\mathcal{P}$ removes exactly one attribute. Consider the project operator *op* in $\mathcal{P}$ that removes A. Create a new plan $\mathcal{P}'$ with *op* removed from $\mathcal{P}$. We will now show that this plan is safe for q. Since $\mathcal{P}$ is a safe plan, every project operator satisfies the condition in Theorem 3 given by Equation (1). For the operators in $\mathcal{P}'$ that are not ancestors of *op*, Equation (1) remains same. So they are still safe. For the operators in $\mathcal{P}'$ that are ancestors of *op*, A gets added to the left of equation (1). So the functional dependencies still hold and the operators are still safe. Thus, a safe plan exists for q.

Lemma 3 *Consider a plan $\mathcal{P}_1 = \Pi_A(q_1 \bowtie q_2)$, where q_1 and q_2 are queries and $Heads(q_1) \subseteq A$. Also, let A contain all the attributes used in the join between q_1 and q_2. Let $\mathcal{P}_2 = q_1 \bowtie (\Pi_{A\cap Heads(q_2)}(q_2))$. Then $\mathcal{P}_1$ is safe implies that $\mathcal{P}_2$ is safe.*

Proof First, $\mathcal{P}_2$ is a valid plan because A contains all the attributes that are used in the final join.

Since $\Pi_A(q_1 \bowtie q_2)$ is safe, the following can be inferred from $\Gamma^p(q_1 \bowtie q_2)$ for each $R^p \in PRels(q_1 \bowtie q_2)$:

$$A, R^p.E \rightarrow Heads(q_1 \bowtie q_2)$$

The above can be rewritten as

$$Heads(q_1), A \cap Heads(q_2), R^p.E \rightarrow Heads(q_1) \cup Heads(q_2) \quad (2)$$

We know that plans q_1 and q_2 are safe. Thus, to show that $\mathcal{P}_2$ is safe, we only need to show that the project operator $\Pi_{A \cap Heads(q_2)}(q_2)$ is safe. So, for each $R^p \in PRels(q_2)$, $\Gamma^p(q_2)$ must imply the following:

$$A \cap Heads(q_2), R^p.E \rightarrow Heads(q_2) \quad (3)$$

Let Γ_{join} be the set of FDs introduced by the final join. Then, $\Gamma^p(q_1 \bowtie q_2) = \Gamma^p(q_1) \cup \Gamma^p(q_2) \cup \Gamma_{join}$. Since A contains all the attributes that occur in Γ_{join}, Γ_{join} does not help in inferring Equation (2). Thus, $\Gamma^p(q_1) \cup \Gamma^p(q_2)$ imply Equation (2) and hence the following:

$$Heads(q_1), A \cap Heads(q_2), R^p.E \rightarrow Heads(q_2)$$

But now, since Γ_{join} is not present, $Heads(q_1)$ and $\Gamma^p(q_1)$ have no contribution in the above equation. Thus, $\Gamma^p(q_2)$ alone implies Equation (3).

This shows that $\mathcal{P}_2$ is a safe plan.

Theorem 5 *Let q be a query that has a separation (S, T). Then, if q has a safe plan, there must be another safe plan of the form $\mathcal{P}_S \bowtie \mathcal{P}_T$ such that $PRels(\mathcal{P}_S) = S$ and $PRels(\mathcal{P}_T) = T$ (where the join may contain a set of select conditions).*

Proof We will prove this by induction on the size of the query. The base step corresponding to queries over a single relation holds trivially. Let $\mathcal{P}$ be a safe plan for q. There are three cases:

1. The top operator on $\mathcal{P}$ is a join. Thus, $\mathcal{P}$ can be written as $\mathcal{P}_1 \bowtie \mathcal{P}_2$. Let q_1 and q_2 be the queries corresponding to the plans $\mathcal{P}_1$ and $\mathcal{P}_2$. It is easy to see that $(PRels(\mathcal{P}_1) \cap S, PRels(\mathcal{P}_1) \cap T)$ form a separation for q_1. By induction, there is a safe plan $\mathcal{P}_{S1} \bowtie \mathcal{P}_{T1}$ for q_1 where $S1 = PRels(\mathcal{P}_1) \cap S$ and $T1 = PRels(\mathcal{P}_1) \cap T$. Similarly, there is a safe plan $\mathcal{P}_{S2} \bowtie \mathcal{P}_{T2}$ for q_2 where $S2 = PRels(\mathcal{P}_2) \cap S$ and $T2 = PRels(\mathcal{P}_2) \cap T$. Thus, the following is a safe plan for q:

   $$(\mathcal{P}_{S1} \bowtie \mathcal{P}_{S2}) \bowtie (\mathcal{P}_{T1} \bowtie \mathcal{P}_{T2})$$

   The safety of the about plan follows from the safety of the individual subplans and the safety of the join operators.
2. The top operator on $\mathcal{P}$ is a select. Thus, $\mathcal{P}$ can be written as $\sigma_c(\mathcal{P}')$. Let q' be the query corresponding to the plan $\mathcal{P}'$. (S, T) is still a separation for q'. By induction, there is a safe plan $\mathcal{P}'_S \bowtie \mathcal{P}'_T$ for q'. By simply adding the select condition to the top join, we get a safe plan as required.
3. The top operator on $\mathcal{P}$ is a project. Thus, $\mathcal{P}$ can be written as $\Pi_A(\mathcal{P}')$. Let q' be the query corresponding to $\mathcal{P}'$. Note that (S, T) is still a separation for q'. By induction hypothesis, there is a safe plan for q' of the form $\mathcal{P}'_S \bowtie \mathcal{P}'_T$. Also, A can be written as the disjoint union of A_S and A_T, where A_S are attributes of relations from S and A_T are attributes of T. Thus, the following plan is safe for q:

   $$\Pi_{(A_S, A_T)}(\mathcal{P}'_S, \mathcal{P}'_T)$$

   Using Lemma 1, we get the following equivalent safe plan:

   $$\Pi_{A_S, A_T}(\Pi_{Heads(\mathcal{P}'_S) \cup A_T}(\mathcal{P}'_S \bowtie \mathcal{P}'_T))$$

   We observe that Lemma 3 is applicable to the plan $(\Pi_{Heads(\mathcal{P}'_S) \cup A_S}(\mathcal{P}'_S \bowtie \mathcal{P}'_T))$ since (S, T) is a separation. Thus, the following equivalent plan is also safe:

   $$\Pi_{A_S}(\mathcal{P}'_S \bowtie (\Pi_{A_T}(\mathcal{P}'_T)))$$

   Another application of Lemma 3 yields the following plan

   $$(\Pi_{A_S}(\mathcal{P}'_S)) \bowtie (\Pi_{A_T}(\mathcal{P}'_T))$$

   Setting $\mathcal{P}_S = \Pi_{A_S}(\mathcal{P}'_S)$ and $\mathcal{P}_T = \Pi_{A_T}(\mathcal{P}'_T)$, we get the required result.

The theorem then follows from the principle of induction.

Theorem 6 *Algorithm* Safe-Plan *is complete.*

Proof Let q be the given query. Suppose it has an attribute A belonging to $Attr(q) - Head(q)$ such that the operator $\Pi_{Head(q)}(q_A)$ is a safe operator, where q_A is as defined in the algorithm. Then, by Lemma 2, q has a safe plan if and only if q_A has a safe plan. So the algorithm recursively solves for q_A. On the other hand, suppose there is no such attribute A. This implies that in any safe plan, the final operator cannot be a project operator as it is not safe. So a safe plan must consist of a join of two subplans. These two subplans must form a separation because after they are joined, there is no projection. Hence, there must exist a separation for q. If not, the algorithm correctly returns false. Otherwise, by Theorem 5, it is sufficient to solve each of the subproblems corresponding to the separation separately. This completes the proof of completeness of the algorithm.

5 Complexity Analysis

We show here a fundamental result on the complexity of query evaluation on probabilistic databases. It forms a sharp separation of conjunctive queries into queries with low and high data complexity, and shows that our optimization algorithm is complete.

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. If q has a safe plan $\mathcal{P}$, then its data complexity is in PTIME,

because all extensional operators are in PTIME. We start by showing that, for certain queries, the data complexity is $\#P$-complete. $\#P$ is the complexity class of some hard counting problems. Given a boolean formula φ, counting the number of satisfying assignments, denote it $\#\varphi$, is $\#P$-complete [36]. (Checking satisfiability, $\#\varphi > 0$, is NP-complete.) The data complexity of any conjunctive query is $\#P$, since $q^{rank}(D^p) = Pr(q^i(D^p))$. The following is a variant of a result on query reliability by Gradel et al. [13].

Theorem 7 *Consider the following conjunctive query on three probabilistic tables:*

$$q() : -L^p(x), J(x, y), R^p(y)$$

Here L^p, R^p are extensional probabilistic tables and J is deterministic[6]*. The data complexity for q is $\#P$-hard.*

We used here a more standard datalog notation for conjunctive queries. In our notation the query becomes: $q : -L^p(A), J(B, C), R^p(D), A = B, C = D$.

Proof (Sketch) Provan and Ball [27] showed that computing $\#\varphi$ is $\#P$-complete even for *bipartite monotone 2-DNF* boolean formulas φ, i.e. when the propositional variables can be partitioned into $X = \{x_1, \ldots, x_m\}$ and $Y = \{y_1, \ldots, y_n\}$ s.t. $\varphi = C_1 \vee \ldots \vee C_l$ where each clause C_i has the form $x_j \wedge y_k$, $x_j \in X, y_k \in Y$. (The satisfiability problem, $\#\varphi > 0$, is trivially true.). Given φ, construct the instance D^p where L^p is X, R^p is Y and J is the set of pairs (x_j, y_k) that occur in some clause C_i. Assign independent probability events to tuples in L^p, R^p, with probabilities 1/2. Then $q^{rank}(D^p)$ returns a single tuple, with probability $\#\varphi/2^{m+n}$. Thus, computing $q^{rank}(D^p)$ is at least as hard as computing $\#\varphi$.

We state now the main theoretical result in this paper. We consider it to be a fundamental property of query evaluation on probabilistic databases. First, we need some notations. Given a set of attributes $S \subseteq Attr(q)$, we denote with $S^+ = \{A \in Attr(q) \mid \Gamma^p(q) \vdash S \rightarrow A\}$, i.e. the set of attributes A such that $S \rightarrow A$ can be inferred from $\Gamma^p(q)$. We say that S determines A if $A \in S^+$. For each relation R, denote $Attr^0(R) = (Attr(R))^+ - (Head(q))^+$. That is, $Attr^0(R)$ consists of all attributes that are determined by R but not by $Head(q)$ [7].

Theorem 8 (Fundamental Theorem of Queries on Probabilistic DBs) *Consider a schema $\bar{R}^p, \Gamma^p$ which consists of a set of probabilistic and deterministic relations. Let q be a conjunctive query s.t. each relation occurs at most once. Assuming $\#P \neq PTIME$ the following statements are equivalent:*

1. *The data complexity of q is $\#P$-complete.*
2. *The* Safe-Plan *optimization algorithm fails to return a plan.*
3. *There exists a sequence of relations $R_0, R_1, \cdots R_k$, R_{k+1}, and attributes A_1 and A_2, with the following properties:*
   (a) *R_0 and R_{k+1} are probabilistic*
   (b) *$Attr^0(R_0) \cap Attr^0(R_1)$ contains A_1 but $Attr^0(R_0)$ does not contain A_2*
   (c) *$Attr^0(R_k) \cap Attr^0(R_{k+1})$ contains A_2 but $Attr^0(R_{k+1})$ does not contain A_1*
   (d) *$Attr^0(R_i) \cap Attr^0(R_{i+1}) \not\subseteq Attr^0(R_0) \cup Attr^0(R_{k+1})$ for $1 \leq i \leq k-1$*

Before we give the proof, let is look at some examples.

Example 7 Consider the following four queries:

$$\begin{aligned}
q_1() &:- L^p(A), J(B, C), R^p(D), A = B, C = D \\
q_2(U) &:- L^p(A), J(B, C), R^p(D), A = B, C = D, \\
&\quad S(U, U_1), U_1 = A \\
q_3() &:- R_0^p(A), R_1(B, C), R_2(D, F), R_3^p(H), \\
&\quad A = B, C = D, F = H \\
q_4() &:- R_0^p(A, B), R_1(C, D), R_2(F, G), R_3^p(H, I) \\
&\quad B = D, D = G, G = I, A = C, F = H
\end{aligned}$$

q_1 is exactly the query in Theorem 7, which we know is unsafe. Let us show that it satisfies the condition (3) of the above theorem.

Since $Head(q_1) = \emptyset$ and Γ^p contains only trivial functional dependencies, we have $Attr^0(L) = \{A, B\}$, $Attr^0(J) = \{A, B, C, D\}$, $Attr^0(R) = \{C, D\}$, and condition (3) above follows by taking $A_1 = A$, $A_2 = D$, $R_0 = L^p$, $R_1 = J$, and $R_2 = R^p$. Indeed L^p and R^p are probabilistic relations, $Attr^0(L) \cap Attr^0(J)$ contains A but $Attr^0(L)$ does not contain D, and $Attr^0(J) \cap Attr^0(R)$ contains D but $Attr^0(R)$ does not contain A. Also, (d) holds vacuously because $k = 1$ here.

If we modify $q_1()$ to $q_1(A, B, C, D)$, the query becomes safe, since no projections are needed; this illustrates the need to remove the head variables in the definition of $Attr^0$.

Let us now look at q_2, which is a variation of the previous query. Assume that U is a key for the relation $S(U, U_1)$. Now, $Head(q_2) = \{U\}$ and $(Head(q_2))^+ = \{U, U_1, A, B\}$, since we have the functional dependency $U \rightarrow U_1$ and the query equates U_1, A and B. Now, $Attr^0(L) = \emptyset$, $Attr^0(J) = \{C, D\}$ and $Attr^0(L) = \{C, D\}$. The conditions of Thm. 8 are not satisfied and in fact, the query is safe. It can be verified that the following plan is safe:

$$\Pi^e_U(S \bowtie^e_{U_1=A} L^p \bowtie^e_{A=B} (\Pi^e_B(J \bowtie^e_{C=D} R^p)))$$

q_2 illustrates the need to take $Head^+$ rather than $Head$ in the definition of $Attr^0$.

[6] Allowing J to be deterministic strengthens the result. The theorem remains true if J is probabilistic.

[7] In general, $Attr(R)$ and $Attr^0(R)$ may overlap, without one of them having to contain the other.

Query q_3 shows that the length of the sequence of relations that satisfy condition (3) could be arbitrarily large[8]. The only probabilistic relations here are R_0^p and R_1^p, and the only sequence that satisfies Thm 8 is R_0^p, R_1, R_2, R_3^p. This query is unsafe, while if we drop any relation from the query, it becomes safe.

Finally, query q_4 shows the need for the property (d) in condition (3). With A_1 and A and A_2 as H, the query satisfies properties (a),(b) and (c) but fails (d). This query is safe, with the following safe plan

$$\Pi^e_{\{\}}(\ \Pi_B(R_0^p \bowtie^e_{A=C,B=D} R_1) \bowtie^e_{B=D} \Pi_D(R_2 \bowtie^e_{F=H,G=I} R_3^p))$$

Theorem 8 provides a sharp separation of feasible and infeasible queries on probabilistic databases. It also shows that all the queries that can be evaluated in polynomial time can in fact be evaluated using a safe plan.

Before we give the proof, we need a simple result.

Lemma 4 *Let q by any query and A be any attribute not in $Head(q)$. Let q_A denote the query obtained by adding A to the head variables. Then, $\Pi^e_A(q_A)$ is safe if $A \in Attr^0(R^p)$ for all $R^p \in PRels(q)$.*

Proof For $\Pi^e_A(q_A)$ to be safe, Eq 1 must hold. It holds because for any probabilistic relation R^p, if $A \in Attr^0(R^p)$, then $Attr(R^p) \rightarrow A$ and hence, $R^p.E \rightarrow A$.

Now we prove Thm 8.

Proof **(Theorem 8)**

(1) $\Rightarrow$ (2) is obvious, since any safe plan has data complexity in PTIME.

(2) $\Rightarrow$ (3) Let us assume that Safe-Plan algorithm fails on q. Then, it must have a sub-query q' such that (i) it has no separation and (ii) for every $A \in Attr(q') - Head(q')$, $\Pi^e_{Head(q')}(q'_A)$ is not safe where q'_A is the query q' with A added to head.

For each attribute in $Attr(q') - Head(q')$, by Lemma 4, there is at least one probabilistic relation that does not determine that attribute. Let A_1 be the attribute that is determined by the largest number of probabilistic relations and let $\bar{R}_{A_1}$ be the set of those probabilistic relations. There is at least one probabilistic relation not contained in $\bar{R}_{A_1}$. Let R' be one such relation. Since the query has no separation, the constraint graph $G(q')$ (defined in Sec 4.3) is connected. Its easy to see that an edge between two relations R_1 and R_2 in $G(q')$ implies $Attr^0(R_1)$ and $Attr^0(R_2)$ intersect. Thus, there is a sequence of relations R_1, $R_2 \cdots R_k, R_{k+1} = R'$ such that $R_1 \in \bar{R}_{A_1}$ and

$$Attr^0(R_i) \cap Attr^0(R_{i+1}) \neq \emptyset \quad (1 \leq i \leq k) \tag{4}$$

Consider the shortest such sequence. Let A_2 be any attribute in $Attr^0(R_k) \cap Attr^0(R_{k+1})$. There is at least one relation $R_0 \in \bar{R}_{A_1}$ that does not determine A_2 (otherwise $|\bar{R}_{A_2}|$ would be strictly greater that $|\bar{R}_{A_1}|$). We claim that the sequence $R_0, R_1, \ldots, R_k, R_{k+1}$ satisfies all the four properties. We have R_0 and R_{k+1} probabilistic, hence (a) holds. (b) and (c) follows from our construction. If $k = 1$, (d) holds vacuously. If $k > 1$, observe that $Attr^0(R_0)$ must be disjoint from $Attr^0(R_i)$ for $i > 1$, otherwise we can obtain a shorter sequence that connects $\bar{R}_{A_1}$ and R'. Similarly, $Attr^0(R_{k+1})$ must be disjoint from $Attr^0(R_i)$ for $1 \leq i \leq k-1$. (d) follows from these properties.

(3) $\Rightarrow$ (1) We prove this by extending the ideas in Th. 7. We show a reduction from the problem of counting the number of assignments of *bipartite monotone 2-DNF* to evaluating the query q.

Let $X = \{x_1, \ldots, x_m\}$, $Y = \{y_1, \ldots, y_n\}$ and $\varphi = C_1 \vee \ldots \vee C_l$ where each clause C_i has the form $x_{f(i)} \wedge y_{f(i)}$. We will construct a database as follows. Consider a single relation R_U, which we call the *universal table*, whose attributes are the union of the attributes of all the relations that occur in the query. We partition the attributes into four sets: S_X consists of attributes in $Attr^0(R_0) - Attr^0(R_{k+1})$, S_Y consists of $Attr^0(R_{k+1}) - Attr^0(R_0)$, S_0 is $(Attr^0(R_0) \cup Attr^0(R_{k+1})) \cup (Head(q))+$, and S_C consists of all the remaining attributes.

We populate R_U with l tuples, one corresponding to each clause, as follows: in row i, assign a value of $x_{f(i)}$ to all variables in S_X, a value of $y_{f(i)}$ to variables in S_Y, a constant value 0 to variables in S_0 and a value of C_i to variables in S_C. From this table, construct individual tables by taking the projections on corresponding columns. For tables R_0 and R_{k+1}, let all events have a probability of 0.5 (which we can assign because both of them are probabilistic) and let all other tables have events with probability 1. The resulting database satisfies all the functional dependencies. To see why, consider any FD, $\bar{A} \rightarrow A'$, where $\bar{A}$ is a set of attributes and A' is an attribute. For this to be violated, one of the following must hold: (i) $\bar{A} \subseteq S_0$ and $A' \notin S_0$, (ii) $\bar{A} \subseteq S_0 \cup S_X$ and $A' \in S_Y$, (iii) $\bar{A} \subseteq S_0 \cup S_Y$ and $A' \in S_X$. None of the three cases is possible: S_0 is equal to $(Attr(R_0))^+ \cap (Attr(R_{k+1}))^+$, $S_0 \cup S_X$ is $(Attr(R_0))^+$ and $S_0 \cup S_Y$ is $(Attr(R_{k+1}))^+$, and all of these sets are closed under implication.

Claim: The query q, when evaluated on the above database, returns a single tuple with all attributes 0, with probability equal to $\#\varphi / 2^{m+n}$.

First, since $Attr^0(R_0)$ does not intersect with S_C but contains S_X, the relation R_0 will have m tuples corresponding to m distinct values of S_X attributes. We use the variable names $x_1, \ldots, x_m$ to denote corresponding events. Similarly, the relation R_{k+1} will have n distinct tuples with events denoted by $y_1, \ldots, y_n$. Now, consider the intensional evaluation of q that first joins everything and then projects on $Head(q)$. When we join all the tables back using the join conditions in the query, we get back the R_U table. This is due to the S_C variables. Note

[8] Cor. 1 says that when the schema contains only probabilistic relations, we never need a sequence of relations more than three.

that for $1 \leq i < k$, $Attr^0(R_i) \cap Attr^0(R_{i+1})$ must contain at least one attribute from the set S_C because of the property (d). Thus, S_C attributes connect a tuple x_i in R_0 with y_j in R_{k+1} if and only if $(x_i \wedge y_j)$ is a clause in φ. The join results in precisely l tuples, with complex events corresponding to the clauses in φ. When we project this table on $Head(q)$, since $Head(q) \subseteq S_0$, we get just one tuple whose complex event is the formula φ. Since all truth assignments have the same probability, the probability of φ is $\#\varphi/2^{m+n}$.

Corollary 1 *Consider the special case of Thm. 8 where all relations are probabilistic, i.e. there are no deterministic relations. Then, condition (3) of the theorem can be replaced by the following simpler condition:*

3'. There exists three relations R_0, R_1 and R_2, and attributes A_1 and A_2, s.t. $Attr^0(R_0) \cap Attr^0(R_1)$ contains A_1 but not A_2, and $Attr^0(R_1) \cap Attr^0(R_2)$ contains A_2 but not A_1.

Proof It is easy to check that $(3')$ above implies (3) in Thm. 8. To prove in the other direction, consider the sequence $R_0, R_1, \ldots, R_k, R_{k+1}$ in (3). We will show that R_0, R_1 and R_2 satisfy the properties above. If $k = 1$, this holds trivially. If $k > 1$, let A_2' be any attribute in $Attr^0(R_1) \cap Attr^0(R_2) - Attr^0(R_0)$, which exists by property (d). Thus, $Attr^0(R_0) \cap Attr^0(R_1)$ contains A_1 but not A_2', and $Attr^0(R_1) \cap Attr^0(R_2)$ contains A_2' but not A_1.

When Γ^p is empty, i.e. there are no functional dependencies expect for the trivial dependencies, Cor. 1 gives us a very elegant characterization of the queries with $\#P$-complete complexity. A query is $\#P$-complete, if it contains the following pattern (shown in datalog notation), with $x, y \notin Head(q)$

$$q(\ldots) :- R_0(x, \ldots), R_1(x, y, \ldots), R_2(y, \ldots), \cdots$$

If such a pattern can not be found in the query, the data complexity of the query is $PTIME$.

6 Query Optimization

We have shown in Sec 4 that the relational algebra operators can be modified to compute the probabilities, and we introduced three new operators: σ^e, Π^e and $\bowtie^e$. Further, we saw that different relational algebra plans for the same query can give different answers under extensional semantics, and only a subset of plans are safe, i.e. give correct answers. In this section, we consider the problem of finding an efficient safe plan using a cost-based optimizer. Recall that the Safe-Plan algorithm in Sec 4.3 only gives us one safe plan. Traditional query optimizers start from one query plan and use relational algebra equivalences to search for alternate plans. However, σ^e, Π^e and $\bowtie^e$ do not satisfy the traditional equivalences [9]. For instance, Π^e and $\bowtie^e$ do not commute. Therefore, we give a set of transformation rules for these operators that can be used to search for alternate safe plans.

A *transformation rule* takes a relational algebra plan consisting of these extensional operators and produces a new plan. We say that a transformation rule is *sound* if, when applied to a safe plan, it results in a safe plan[10]. Below are some transformation rules:

Rule 1: [*Join Commutativity*] Extensional joins are commutative

$$R \bowtie^e S \Leftrightarrow S \bowtie^e R$$

Rule 2: [*Join Associativity*] Extensional joins are associative

$$R \bowtie^e (S \bowtie^e T) \Leftrightarrow (R \bowtie^e S) \bowtie^e T$$

Rule 3: [*Cascading Projections*] Successively eliminating attributes from a relation is equivalent to simply eliminating all but the attributes retained by the last projection

$$\Pi^e_A(\Pi^e_{A \cup B}(R)) \Leftrightarrow \Pi^e_A(R)$$

Rule 4: [*Pushing Projections Below a Join*] A projection can be pushed below a join if it retains all the attributes used in the join.

$$\Pi^e_A(R \bowtie^e S) \Rightarrow (\Pi^e_{A_1}(R)) \bowtie^e (\Pi^e_{A_2}(S))$$

where A_1 and A_2 are the attributes in R and S retained by the projection.

Rule 5: [*Lifting Projections Up a Join*] A Projection can not always be lifted up a join. The following transformation rule can be applied only when the top Π^e operator in the resulting plan satisfies the Eq. 1 of Thm. 3.

$$(\Pi^e_A(R)) \bowtie^e S \Rightarrow \Pi^e_{A \cup Attrs(S)}(R \bowtie^e S)$$

Theorem 9 *The transformation rules described above are sound.*

The soundness of rules 1 and 2 can be verified easily, rules 3 and 4 follow from Lemma 1 and Lemma 3 respectively, while Rule 5 is sound by definition. We haven't shown the rules involving σ^e operator, but it behaves exactly like the traditional select operator and commutes with project, join and other select operators.

Next, we study the completeness of the rules. Ideally, we would like to traverse the space of all safe plans using the above transformation rules. Given two plans $\mathcal{P}_1$ and

[9] Contrast this with the intensional operators σ^i, Π^i and $\bowtie^i$ defined in Sec 4.1. It is shown in [12] that these operators satisfy all the equivalences that traditional operators satisfy

[10] A sound transformation rule applied to an unsafe plan may result in a plan that is not equivalent to it. We only require that the rule produce equivalent plans when applied to safe plans

$\mathcal{P}_2$, let $\mathcal{P}_1 \Rightarrow^* \mathcal{P}_2$ denote the statement that $\mathcal{P}_2$ can be obtained from $\mathcal{P}_1$ by a sequence of the above transformation rules. Note that rules 1, 2 and 3 can be applied in either direction. Also, if $\mathcal{P}_2$ can be obtained from $\mathcal{P}_1$ by applying rule 4, then $\mathcal{P}_1$ can be obtained back from $\mathcal{P}_2$ by applying rule 5. The rule will be applicable because $\mathcal{P}_1$, being a safe plan, will satisfy the conditions of rule 5. Hence, we have the following result.

Lemma 5 *For two safe plans $\mathcal{P}_1$ and $\mathcal{P}_2$, if $\mathcal{P}_1 \Rightarrow^* \mathcal{P}_2$, then $\mathcal{P}_2 \Rightarrow^* \mathcal{P}_1$.*

This makes $\Rightarrow^*$ an equivalence relation. To emphasize this, we will use the notation $\mathcal{P}_1 \Leftrightarrow^* \mathcal{P}_2$ to denote that either plan be transformed into the other. We next prove our main result for query optimization which says that the transformation rules are complete, i.e. any safe plan can be reached from any other safe plan using these transformations.

Theorem 10 *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two safe plans for a query q. Then, $\mathcal{P}_1 \Leftrightarrow^* \mathcal{P}_2$.*

Proof First, using rule 3 in the reverse direction in both $\mathcal{P}_1$ and $\mathcal{P}_2$, we ensure that every Π^e operator removes exactly one attribute. Now, we use induction on the size of the plans.

Suppose the topmost operator in $\mathcal{P}_1$ is a Π^e operator that removes the attribute A. Let q_A be the query obtained from q by adding A to the head variables. Then, $\mathcal{P}_1 = \Pi^e_{Heads(q)}(\mathcal{P}'_1)$, where $\mathcal{P}'_1$ is a safe plan for query q_A. Consider the operator in $\mathcal{P}_2$ that removes the attribute A (it need not be the top operator in $\mathcal{P}_2$). Let $\mathcal{P}'_2$ be the plan obtained by removing this operator from $\mathcal{P}_2$. As shown in the proof of Lemma 2, $\mathcal{P}'_2$ is a safe plan for the query q_A. By induction hypothesis, $\mathcal{P}'_1 \Leftrightarrow^* \mathcal{P}'_2$ and hence,

$$\mathcal{P}_1 = \Pi^e_{Heads(q)}(\mathcal{P}'_1) \Leftrightarrow^* \Pi^e_{Heads(q)}(\mathcal{P}'_2)$$

Note that $\Pi^e_{Heads(q)}(\mathcal{P}'_2)$ is a plan that looks exactly like $\mathcal{P}_2$ except that it removes the attribute A as a final operation. Using successive applications of rule 4, we can push this operation down to its original place in $\mathcal{P}_2$ (its always safe to push a projection down). Thus, $\Pi^e_{Heads(q)}(\mathcal{P}'_2) \Leftrightarrow^* \mathcal{P}_2$, which proves that $\mathcal{P}_1 \Leftrightarrow^* \mathcal{P}_2$.

If the topmost operator in $\mathcal{P}_2$ is a Π^e, we can again apply the same argument. So lets assume that both of them have a $\bowtie^e$ as the top operator. Let the top join in $\mathcal{P}_1$ split the relations into sets S and T, and let the corresponding subplans be $\mathcal{P}_S$ and $\mathcal{P}_T$. Thus, $\mathcal{P}_1 = \mathcal{P}_S \bowtie^e \mathcal{P}_T$. S and T must form a separation for q since there are no projections after the join. Similarly, let the top join in $\mathcal{P}_2$ split the relations into $S_1 \cup T_1$ and $S_2 \cup T_2$, where $S_1, S_2 \subseteq S$ and $T_1, T_2 \subseteq T$. Again, $S_1 \cup T_1$ and $S_2 \cup T_2$ form a separation. Let $\mathcal{P}_2 = \mathcal{P}_{S_1T_1} \bowtie^e \mathcal{P}_{S_2T_2}$. Denote $q_{S_1T_1}$ the query represented by the plan $\mathcal{P}_{S_1T_1}$. S_1 and T_1 form a separation for $q_{S_1T_1}$, since S and T form a separation. By Thm. 5, $q_{S_1T_1}$ has an equivalent safe plan of the form $\mathcal{P}_{S_1} \bowtie \mathcal{P}_{T_1}$, where $\mathcal{P}_{S_1}$ only refers to relations in S_1 and $\mathcal{P}_{T_1}$ only refers to relations in T_1. By induction hypothesis, $\mathcal{P}_{S_1T_1} \Leftrightarrow^* \mathcal{P}_{S_1} \bowtie \mathcal{P}_{T_1}$. Using similar argument, we get $\mathcal{P}_{S_2T_2} \Leftrightarrow^* \mathcal{P}_{S_2} \bowtie \mathcal{P}_{T_2}$ for some plans $\mathcal{P}_{S_2}$ and $\mathcal{P}_{T_2}$. We have

$$\begin{aligned}
\mathcal{P}_2 &= \mathcal{P}_{S_1T_1} \bowtie^e \mathcal{P}_{S_2T_2} \\
&\Leftrightarrow^* (\mathcal{P}_{S_1} \bowtie^e \mathcal{P}_{T_1}) \bowtie^e (\mathcal{P}_{S_2} \bowtie^e \mathcal{P}_{T_2}) \\
&\Leftrightarrow^* (\mathcal{P}_{S_1} \bowtie^e \mathcal{P}_{S_2}) \bowtie^e (\mathcal{P}_{T_1} \bowtie^e \mathcal{P}_{T_2})
\end{aligned}$$

The last equivalence uses the rules 1 and 2 to reorder the joins. We again use induction hypothesis on these subplans to get $(\mathcal{P}_{S_1} \bowtie^e \mathcal{P}_{S_2}) \Leftrightarrow^* \mathcal{P}_S$ and $(\mathcal{P}_{T_1} \bowtie^e \mathcal{P}_{T_2}) \Leftrightarrow^* \mathcal{P}_T$. This proves that $\mathcal{P}_1 \Leftrightarrow^* \mathcal{P}_2$.

7 Unsafe Plans

When a query's data complexity is $\#P$-complete, then Safe-Plan fails to return a plan. Since this can indeed happen in practice, we address it and propose two solutions.

7.1 Least Unsafe Plans

Here we attempt to pick a plan that is less unsafe than others, i.e. minimizes the error in computing the probabilities. Recall from Eq.(1) that $\Pi^e_{A_1,\ldots,A_k}$ is safe in $\Pi^e_{A_1,\ldots,A_k}(q)$ iff $A_1, \ldots, A_k, R^p.E \rightarrow Head(q)$ for every R^p. Let $\bar{B} = \{A_1, \ldots, A_k, R^p.E\} \cap Attr(R^p)$ (hence $R^p.E \in \bar{B}$) and $\bar{C} = Head(q) \cap Attr(R^p)$. Define R^p_{fanout} to be the expected number of distinct values of $\bar{C}$ for a fixed value of the attributes $\bar{B}$. In a relational database system, it is possible to estimate this value using statistics on the table R^p. Define the degree of unsafety of $\Pi^e_{A_1,\ldots,A_k}$ to be $max_{R^p \in PREL(Q)}(R^p_{fanout} - 1)$. Thus, a safe project has degree of unsafety 0. Also, the higher the degree of unsafety, the higher is the expected error that would result from using the extensional semantics for that project operator.

We modify Algorithm 1 to cope with unsafe queries. Recall that the algorithm tries to split a query q into two subqueries q_1, q_2 s.t. all their join attributes are in $Head(q)$. Now we relax this: we allow joins between q_1 and q_2 on attributes not in $Head(q)$, then project out these attributes. These projections will be unsafe, hence we want to minimize their degree of unsafety. To do that, we pick q_1, q_2 to be a minimum cut of the graph, where each edge representing a join condition is labeled with the degree of unsafety of the corresponding project operation[11]. The problem of finding minimum cut is polynomial time solvable as a series of network flow problems or using the algorithm of Stoer and Wagner [33].

[11] The estimator of R^p_{fanout} should make sure that the estimated value is 0 only when the FD holds, otherwise the algorithm may favor 'expected' safe plans over truly safe plans.

7.2 Monte-Carlo Approximations

As an alternative, we present now an algorithm based on a Monte-Carlo simulation, which runs in polynomial time and approximates the probabilities to arbitrary precision.

Given a conjunctive query q over probabilistic relations $R_1^p, R_2^p \cdots R_k^p$, let q' be the query obtained from q by making it return all the variables in its body, i.e. $Head(q') = Attr(q') = Attr(q)$ and $q = \Pi_{Head(q)}(q')$. Also, let q' return all event attributes $\bar{E} = R_1^p.E$, ..., $R_k^p.E$. Evaluate q' over the database (without any probability calculations) and group the tuples in the answer based on the values of their attributes $Head(q)$. Consider one such group, and assume it has n tuples $t_1, \ldots, t_n$. The group defines the following complex event expression: $\bigvee_{i=1}^{n} C_i$, where each C_i has the form $e_1 \wedge \ldots \wedge e_k$. We need to compute its probability, since this will be the probability of one tuple in $q^{rank}(D^p)$. We are back to the problem of evaluating the probabilities of complex events, but now these events C_i are in disjunctive normal form (DNF). Before we describe the techniques to evaluate the probabilities of DNF formulas, let us look at an example.

Consider the probabilistic database given in Fig. 2 and the query in Fig. 3 (b). We have

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

We create a new query q' that returns 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 we evaluate it over the probabilistic database, we get the following result:

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

Observe that since the new query returns all the variables in its body, there are no projections and every event is a conjunction of atomic events. We do a final projection at the end, giving us a single tuple $\{'p'\}$ whose event is the DNF expression $(s_1 \wedge t_1) \vee (s_2 \wedge t_1)$.

The problem of evaluating the probability of a boolean expression, even when restricted to DNF formulas, is $\#P$-complete [36]. However, it can be approximated efficiently using the Monte Carlo algorithm described by Karp [18]: 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 the error being greater that ϵ is less than δ.

In our case, N for a given output tuple is the number of tuples that got merged during the final projection to produce the output tuple. The simulation algorithm runs in time linear in N and hence, linear in the size of the intermediate result before the final projection. As a final note, if N is very small, an exact algorithm may be applied in place of the simulation. This choice can be made independently for each output tuple.

8 Extensions

8.1 Relations with Repeated Events

The various results and the query evaluation technique we have described so far assume that all the events in probabilistic relations are distinct. However, there are several scenarios that require multiple tuples sharing a common event. Consider a relation `Casts(actorname, filmname)` and a query with an approximate predicate `Casts.actorname` $\approx \cdots$. Given two tuples with the same actor name, the user either wants both of them or none of them. Thus, a common event should be associated with both the tuples, rather than each one of them independently satisfying the predicate. This choice also affects the probabilities of the resulting tuples. Suppose the user simply wants to return a list of actors (with the predicate `Casts.actorname` $\approx \cdots$). If an actor that approximately matches the predicate appears 100 times in the relation with independent events, its final probability in the answer would be very high. On the other hand, if the same event is given to all these tuples, the final probability will not depend on how many times the actor appears in the database.

Fortunately, handling repeated events is easy in our framework. A user can specify them by using functional dependencies involving the event attributes. In the above example, the query predicate induces the following dependency: `Casts.actorname` $\rightarrow$ `Casts.E`. For the query evaluation algorithms to work, we can use the following four-step procedure:

1. **Normalize the schema**: we create a virtual schema that represents the same data in normalized form, such that no probabilistic table has repeated events. This is achieved as follows: for each probabilistic table T^P where the events in $T^P.E$ are not unique, we decompose it into two tables, a deterministic table T_1 and a probabilistic table T_2^P. T_2^P stores all the distinct events of T^P. It also has an attribute EID that stores a unique identifier for each distinct event. T_1 is the deterministic part of T^P along with an extra attribute EID that refers to the corresponding events in T_2. Thus, joining T_1 and T_2^P gives us back the original table T^P. This decomposition achieves our objective: the new schema represents the same data and further, every probabilistic table in it has unique events. Note that this is only a virtual schema: we do not transform the data into the new schema.
2. **Translate original query into new schema**: this is easy, every occurrences of T^P in the query is replaced by the join $T_1 \bowtie_{EID} T_2^P$ (this is done for each T^P that is decomposed in step 1)

3. **Find a safe plan**: using our query evaluation algorithm, we find a safe plan P' for the translated query over the new schema.
4. **Translate back to original schema**: we translate the safe plan P' into a plan P over original schema. We replace each occurrence of T_1 and T_2^P with a plan that compute them from T^P using projections. These projections will not be probabilistic projections but normal projections without any probability calculations.

We illustrate this procedure with an example. Consider two probabilistic relations $R^P(A, B)$ and $S^P(C, D)$. R^P has all distinct events while S^P has a distinct event for each value of D. Thus, we have $\{S^P.E \rightarrow S.C\}$ and $\{S^P.C \rightarrow S.E\}$. Now, consider the query

$$q(x) :- R^P(x, y), S^P(y, z)$$

To find a safe plan, first we create a new schema. We decompose S^P into two relations, $S_1(C, D, EID)$ and $S_2^P(EID)$. Thus, the following holds in the new schema: $\{S_1.EID \rightarrow S_1.C\}$ and $S_1.C \rightarrow S_1.EID\}$. Next we translate q into a query q' over the new schema. We have

$$q'(x) :- R^P(x, y), S_1(y, z, eid), S_2^P(eid)$$

Using the SAFE-PLAN algorithm given in Sec 4.3, we get the following plan for q'

$$P' = \Pi_A(R^P \bowtie^P_{B=C} (\Pi_{B,EID}(S_1) \bowtie_{EID} S_2^P))$$

Finally, we substitute S_2^P with a plan that projects S^P on E and substitute S_1^P with a plan that projects S^P on A,B and E.

Note that repeated events provide, in a limited way, support for specifying dependencies between tuples. Supporting arbitrary dependencies between tuples in an efficient manner is beyond the scope of this paper.

8.2 Additional operators

So far, we have limited our discussion to conjunctive queries, or, equivalently to the algebra consisting of σ, Π and $\times$. We show now how to extend these techniques to $\cup, -, \gamma$ (union, difference, groupby-aggregate). A large fragment of SQL queries, including queries with nested sub-queries, aggregates, group-by and existential/universal quantifiers can be expressed in this logical algebra [35]. (We omit δ (duplicate elimination) since we only consider queries with set semantics, i.e. δ is implicit after every projection and union.) Fig. 9 describe the extensional semantics for these operators, using the functional notation.

The treatment of union and set difference operators with intensional semantics is given in [12]. Similarly, aggregate queries have been considered by Sadri [30] and Ross et al. [28] using possible worlds approach. Our aim, as with conjunctive queries, it to study when the cheap extensional evaluation can be used in place of the expensive possible worlds approach. The following theorem gives sufficient conditions under which it is safe to use the extensional semantics for these operators.

$$\begin{array}{lcl}
Pr_{p \cup^e p'}(t) & = & 1 - (1 - Pr_p(t))(1 - Pr_{p'}(t)) \\
Pr_{p -^e p'}(t) & = & Pr_p(t) \times (1 - Pr_{p'}(t)) \\
Pr_{\gamma^e_{\bar{A},\min(B)}(p)}(t, v) & = & Pr_{\Pi^e_{\bar{A}}(\sigma^e_{B=v}(p))}(t) \times \\
 & & (1 - Pr_{\Pi^e_{\bar{A}}(\sigma^e_{B<v}(p))}(t)) \\
Pr_{\gamma^e_{\bar{A},\max(B)}(p)}(t, v) & = & Pr_{\Pi^e_{\bar{A}}(\sigma^e_{B=v}(p))}(t) \times \\
 & & (1 - Pr_{\Pi^e_{\bar{A}}(\sigma^e_{B>v}(p))}(t))
\end{array}$$

Fig. 9 Extensional Semantics for Union, Set difference, Min, Max

Theorem 11 *Let q, q' be conjunctive queries.*

1. *$\cup^e$ is safe in $q \cup^e q'$ if $PRels(q) \cap PRels(q') = \phi$.*
2. *$-^e$ is safe in $q \cap^e q'$ if $PRels(q) \cap PRels(q') = \phi$.*
3. *$\gamma_{\bar{A},agg(B)}$ is safe in $\gamma_{\bar{A},agg(B)}(q)$ if $\Pi_{\bar{A}}(q)$ is safe, where agg is* min *or* max*, i.e. they have the same condition for safety as the projection operator.*

The conditions are intuitive. For instance, in $q \cup^e q'$, if $PRels(q)$ and $PRels(q')$ do not contain a common relation, then the complex events in q and q' will be independent, and hence, the extensional semantics will work. The conditions are not necessary: q and q' may contain a common relation, but with disjoint SELECT clauses on that relation, so that the common relation does not contribute the same events to q and q'. While a more detailed condition can be worked out, there is a much bigger and challenging open problem: is there a dichotomy result, analogous to Thm. 8, for queries with these operators? In other words, if there is no extensional plan for a query, does it mean that the query has high complexity?

In the case of SUM, the aggregated attribute may take values that are not in the input table. For instance, consider a simple SUM query over a single probabilistic table with 50 tuples. There are 2^{50} possible worlds defined by this table and each world can potentially have a distinct value of the sum. In the true spirit of the possible worlds semantics, all of these 2^{50} sums should be returned as the answer along with their probabilities. Instead, it may be more useful to the user if we simply compute the expected value of the sum. Both of these two semantics has been considered in the literature. Computing the expected value of the sum is much easier, using the linearity of expectations. According to the linearity of expectations, if there are tuples $t_1, \ldots, t_k$ with probabilities $p_1, \ldots, p_k$ and values $v_1, \ldots, v_k$, then the expected value of the sum is $p_1v_1 + \ldots + p_kv_k$. This holds irrespective of whether the tuples are independent or not, so it also applies to tuples from an intermediate result. Thus, once the input to the SUM operator has been evaluated, using

either a safe plan or simulation, the expected value of sum can easily be computed. The COUNT aggregate can be treated in a similar fashion.

Having clause A HAVING clause with MIN or MAX aggregate does not pose additional difficulty, because once the aggregate has been computed for each group, the HAVING clause gets translated into a simple SELECT condition. This does not work for SUM and COUNT, since we only compute their expected values. Processing a HAVING clause with SUM or COUNT efficiently is a much harder problem and beyond the scope of this work.

Self-joins All of our techniques apply in the presence of self-joins involving deterministic tables. However, they do not work when there is a self-join involving a probabilistic table. We argue that a query $q^{\approx}$ with uncertain predicates rarely results in self-join, even if the same table R occurs twice in $q^{\approx}$. For instance, consider the query

$$R(x, y, z), R(x', y', z'), x = x', y \approx' ABC', z' \approx' DEF'$$

The query would have a self join between two occurrences of R, each being probabilistic because of the uncertain predicates. However, because of the different uncertain predicates on the two occurrences, the system will make two probabilistic "copies": R_1^p and R_2^p. Thus, there is no self-join in the resulting query.

Nevertheless, self-joins are important from the perspective of probabilistic databases in general. A self-join does not rule out a safe extensional plan. A self-join is safe when tuples are guaranteed not to join with themselves. For instance, if a query contains a self-join between R^P renamed as R_1 and R_2, conditions like $(R_1.id < R_2.id)$ or $(R_1.type = A$ and $R_2.type = B)$ makes sure that no tuple in R^P joins with itself. Even when tuples join with themselves, queries can have a safe plan. For instance, consider the query

$$q(x, y, z) :- R^P(x, y), R^P(x, z)$$

It can be written as the union of following two queries:

$$q_1(x, y, y) :- R^P(x, y)$$
$$q_2(x, y, z) :- R^P(x, y), R^P(y, z), y \neq z$$

Now, both the queries can be evaluated using a safe plan (q_2 has safe plan because tuples do not join with themselves), and the results can be combined. Such a decomposition is not always possible. A complete dichotomy result for queries with self-joins seems challenging and we leave it as an open problem.

Of course, the Monte-Carlo simulation algorithm works fine even in the presence of self-joins.

Extending the optimization algorithm SAFEPLAN is extended to handle each block of conjunctive queries separately. As an example, the query in Section 1, asking for an actor whose name is like 'Kevin' and whose first 'successful' movie appeared in 1995, has a safe plan as shown below:

$$\Pi_{name}(A \bowtie_{actorid} (\sigma_{year=1995}(\gamma_{actorid,\min(year)}(\Pi_{actorid,year}C))))$$

9 Atomic Predicates

Our main motivation is executing a query with uncertain predicates $q^{\approx}$ on a deterministic database D. As we saw, our approach is to apply the uncertain predicates first, and generate a probabilistic database D^p, then evaluate q (without the uncertain predicates). We discuss here briefly some choices for the uncertain predicates proposed in the literature. All proposals have a notion of closeness between two data values. This is domain dependent and can be classified into three categories:

Syntactic closeness This applies to domains with proper nouns, like people's names. Edit distances, q-grams and phonetic similarity can be employed. The excellent surveys on string matching techniques by Zobel and Dart [39] and Navarro [22] describe more than 40 techniques and compare them experimentally. Navarro also has a discussion on the probability of string matching. In our system, we used the 3-gram distance, which is the number of triplets of consecutive letters common to both words.

Semantic closeness This applies to domains that have a semantic meaning, like film categories. A user query for the category 'musical' should match films of category 'opera'. Semantic distance can be calculated by using TF/IDF or with ontologies like Wordnet [37]. We do not support them in our system currently.

Numeric closeness This applies to domains like *price* and *age*. Distance can be just the difference of the values.

Depending on the semantics of the attributes, the distance metric to use can be specified a priori for each attribute. Once distances are defined between two values of an attribute, they need to be meaningfully converted into probabilities. One way of converting is to fit a distribution curve on the distances. An example is a Gaussian curve centered around the distance 0 where it takes value 1. The variance of the Gaussian curve, which reflects the importance given to the match on that attribute, can be set depending on the domain or user preferences. In our experiments, we used fixed, query independent values, for the variances. An ideal system should use metrices based on user studies or learn from relevance feedbacks but developing techniques for these is beyond the scope of this work.

Finally, one issue is when to generate new probability events. For example consider the uncertain predicate `Product.category` $\approx$ ... and assume there are two products with the same category. Should they result in two independent probabilistic events with the same probabilities, or in the same probabilistic events? Both choices are possible in our system. In the first case the functional dependency is $\texttt{Product}^p.\texttt{key} \rightarrow \texttt{Product}^p.E$ while in the second the FD is $\texttt{Product}^p.\texttt{category} \rightarrow \texttt{Product}^p.E$. In the latter case, we will have a relation with repeated events and will need to use the techniques of Sec. 8.1 to generate safe plans.

10 Prototype

Based on the theory of probabilistic databases that we have developed, we return to the problem of building a system that can support complex SQL queries with uncertain predicates.

Our query language is an extension of standard SQL that has an $\approx$ operator. Figure 1 shows an example. Our system for supporting these queries have several salient features. First, it is implemented as a middleware. It works on top of any off-the-shelf database engine containing relational data. Secondly, it does not require altering either the schema of the data in the database. Finally, given an uncertain query, it does not create a new instance of probabilistic database. Rather, it rewrites the query into a new standard SQL query that is (i) safe, (ii) contains all the probability calculations embedded in the query using various aggregates and (iii) can be directly executed by any engine to return the tuples along with probabilities.

We describe the above process with the help of an example. Consider the following SQL query

```
SELECT  Films.year
FROM    Films, Casts
WHERE   Films.filmid = Casts.filmid
  and   Films.name ≈ 'FILM'
  and   Casts.actor ≈ 'ACTOR'
```

Using the uncertain predicate `Films.name` $\approx$ `'FILM'`, we need to convert the `Films` table into a probabilistic table. Assume we have a function MATCH stored in the database that given two film names outputs a probability score of their match. Then, the following SQL query represents the probabilistic `Films` relation

```
SELECT  Films.year,
        MATCH(Films.name, 'FILM') as prob
FROM    Films
```

Similarly, we can create a SQL statement for a probabilistic `Casts` relation. Now, the following plan is a safe plan for the query:

$$\pi_{year}(\\ (\pi_{Films.year,Films.filmid}) \bowtie_{filmid} (\pi_{Casts.filmid}))$$

We convert this plan back into a SQL query with probability calculations as shown in Fig 10. Note that although standard database engines do not have a `PRODUCT` aggregate, it can be implemented using the following transformation:

$$\texttt{PRODUCT(A)} \equiv \texttt{POWER(10, SUM (LOG (A)))}$$

This technique of SQL rewriting can be applied to any query plan in general. Recursively, a SQL statement is generated for each node where the last attribute refers to the probability of the tuples and parent nodes nest the SQL queries of their children.

```
SELECT  P1.year,
        1 - PRODUCT(1 - P1.prob * P2.prob) as prob
FROM
(SELECT  year, filmid,
1 - PRODUCT(1 - MATCH(name, 'FILM')) as prob
FROM     Films
GROUP BY year, filmid
) as P1
(SELECT  filmid,
1 - PRODUCT(1 - MATCH(actor, 'ACTOR')) as prob
FROM     Casts
GROUP BY filmid
) as P2
WHERE   P1.filmid = P2.filmid
GROUP BY  P1.year
```

Fig. 10 The final SQL rewriting

11 Experiments

We performed some preliminary evaluation of our probabilistic query evaluation framework, addressing four questions. How often does the Safe-Plan optimization algorithm fail to find a plan? What is the performance of safe plans, when they exists? Are naive approaches to query evaluation perhaps almost as good as a safe plan? And how effectively can we handle queries that do not have safe plans?

We did not modify the relational engine, but instead implemented a middleware. SQL queries with approximate predicates were reformulated into "extensional" SQL queries, using the techniques described in this paper, and calls to a TSQL function computing 3-gram distances. These queries were then executed by the relational engine and returned both tuples and probabilities. We used Microsoft SQL Server.

We used the TPC-H benchmark, with a database of 0.1GB. We modified all queries by replacing all the predicates in the WHERE clause with uncertain matches. The constants in the queries were either misspelled or made vague. For instance, a condition like `part.container = 'PROMO PLATED GREEN'` was replace with `part.container` $\approx$ `'GREEN PLATE'`. When executed exactly, all modified queries returned empty answers.

All of the following experiments were carried on the first 10 of the 22 TPC-H queries. We found other queries to be not very interesting for applying uncertain predicates, since most of them involve complex aggregates.

1. Frequency of unsafe queries In our first experiment, we wanted to see how many queries do not have safe plans. Out of the 10 TPC-H queries, 8 turned out to have safe plans. Q_7 and Q_8 were the only query that were unsafe. These also become safe if not all of their predicates are uncertain.

2. Performance Next, we measured the running times for the eight queries that have safe plans, shown in Figure 11. All times are wall-clock. The first column is the running time of the safe plan. The second column represents an optimization where at each intermediate

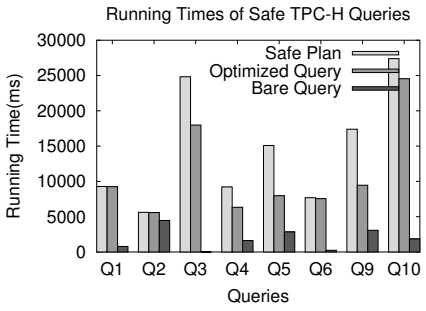

Fig. 11 TPC-H Query Running Times

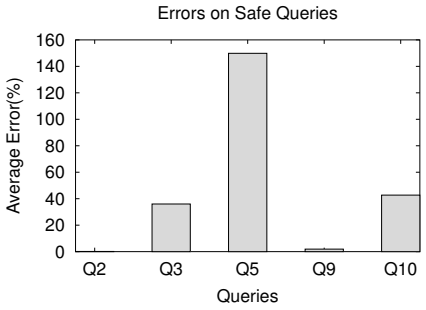

Fig. 12 Errors on Safe TPC Queries

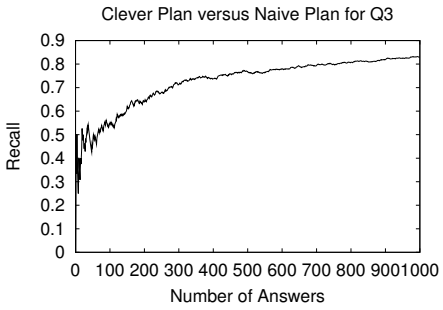

Fig. 13 Recall Plot for Q_3

stage, tuples with zero probability are discarded. This optimization does not affect the final answer and as we can see from the graph, it brings about considerable savings for some queries. This also suggests the use of other optimizations like an early removal of tuples with low probabilities if the user is only interested in tuples with high probability. The third column in the graph shows the time for running safe queries without taking into account the computation time for the uncertain predicate, which, in our case, is the 3-gram distance. The graphs show that most of the time is spent in computing the uncertain predicate (for Q_3, this accounts for almost all of the running time). It suggests that significant improvements can be achieved if the predicates are supported by the engine itself.

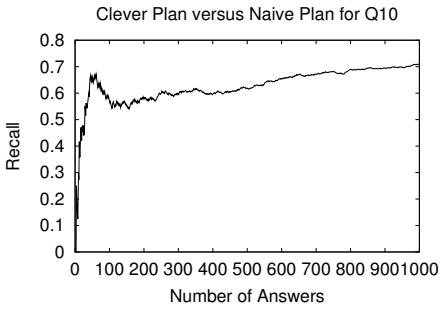

Fig. 14 Recall Plot for Q_{10}

3. Naive Approaches In the next experiment we calculated the error produced by a naive extensional plan. We considered the naive plan that leaves all project operators (and the associated duplicate elimination) at the end of the plan, which are typical plans produced by database optimizers. The error was calculated as below: for each tuple, we measured the percentage error in its probability relative to the correct probability, and we took the average over all tuples. Figure 12 shows the percentage relative error of naive plans. We only considered the 8 queries that have safe plans. The naive plans for Q_1, Q_4, Q_6 were already safe, hence had no errors (and SAFE-PLAN indeed returned the same plan): these queries are not shown. Queries Q_3, Q_5 and Q_{10} had large errors with Q_5 showing an average error of 150% in the tuple probabilities. Queries Q_2 and Q_9 had negligible errors. Thus, while some naive plans were bad, others were reasonable. But, in general, naive plans *can* be arbitrarily bad. However, we argue that the low extra complexity of searching for a safe plan is a price worth paying in order to avoid the (admittedly rare) possibility of arbitrarily large errors.

However, since we are only interested in *ranking* the results, not in the actual probabilities, it is worth asking whether high errors in the probabilities translate into high ranking results. We plotted the recall graphs for queries Q_3 and Q_{10} (for which the naive plan produced only medium errors). We defined recall as the fraction of answers ranked among top N by the naive plan that should actually have been in top N. We plotted this as a function of N. Figures 13 and 14 show the recall graphs. By definition, the recall approaches to 1 when N approaches the total number of possible tuples in the answer. However, as the graphs show, the recall was bad for small values of N. A user looking for top 50 or 100 answers to Q_3 would miss half of the relevant tuples. For smaller values of N (say, 10) the naive approach misses 80% of the relevant tuples.

4. Unsafe Queries Finally, we tested our approach to handle queries with no safe plans on Q_7 and Q_8. We ran the Monte Carlo simulation to compute their answer probabilities and used them as baseline. Figure 15 shows the errors in evaluating them with a naive plan and the

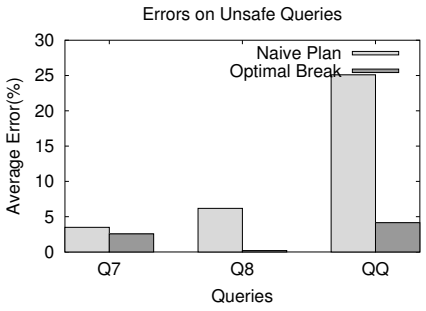

Fig. 15 Errors on Unsafe Queries

least unsafe plan (using min-cut, Sec. 7). The graphs show that the plan chosen by the optimizer was better, or significantly better than a naive one. Still, from two data points it is hard to judge the improvement over a naive plan. To see a third data point we wrote a new unsafe query, QQ, where the relation `lineitem` is joined with `orders` and `suppliers`. Here the fanout is larger, and the difference between the naive plan and the optimal break is more pronounced.

12 Related Work

There are various probabilistic systems for relation databases that have been proposed in the literature and they can primarily be classified into two classes: extensional and intensional. The extensional systems [5,4,8,19] are very efficient when they work, but they have to make simplifying assumptions or impose some restrictions to get around the problem of high complexity. For instance, Cavallo and Pittarelli [5] start by assuming that tuples in the same relations represent disjoint events. Barbara et. al. [4] generalize Cavallo and Pittarelli's model, where tuples are independent and attributes may be inaccurate (leading to disjoint events). However, their system has a requirement that every relation must have a set of deterministic attributes forming the key of that relation. Dey and Sarkar [8] improve upon this model allowing arbitrary keys, but allow only those projections that contain the key. Thus, none of these systems can correctly handle arbitrary conjunctive queries. The Probview system [19] takes a different approach: it does not assume independence while combining probabilities, but requires strategies from users to combine them. Also, it works with interval probabilities instead of point probabilities.

The intensional systems [29,31,32,30,12,38] manipulate symbolic events rather than raw probabilities, and are based on the *possible worlds semantics*. Originally put forward by Kripke for modal logics, possible worlds semantics is commonly used in AI for representing knowledge with uncertainties. Detailed discussions on extensional and intensional systems can be found in the book by Pearl [26]. In context of relational databases, intensional semantics have been used by Sadri [29–32] to compute the reliability of answers to the queries, where the information in the database comes from multiple information sources of varying reliability. In this framework, the system stores a vector with each tuple that identifies the sources contributing to the tuple, and these vectors are manipulated as the query is evaluated. Fuhr and Rolleke [12] define a probabilistic relational algebra that generalizes the standard relational algebra for tuples with probabilistic events. Zimanyi [38] formalize probabilistic databases by means of logic theories based on a probabilistic first-order language. In knowledge representation, Halpern et al. [10,2] have shown the use of possible worlds semantics to assign degrees of beliefs to statements based of the probabilities in the knowledge base. Note that intensional semantics are impractical to use when the number of sources of uncertainties is very large, which is the case with approximate queries where every tuple can be viewed as an independent source of uncertainty. Our work, on the other hand, gives a characterization of queries where intensional semantics can be replaced by the cheaper extensional semantics.

There is also work on supporting probabilities in other models of databases. Ng and Subrahmaniam [23] extend deductive databases with probabilities and give fixed point semantics to logic programs annotated with probabilities, but they use absolute ignorance to combine event probabilities. An alternate model for uncertainty is considered by Cheng et al.[7] that is more suitable for sensor databases. They consider attributes (which are typically sensor readings) whose values is given by a probability distribution function and show how simple queries can be efficiently evaluated over such databases. Probabilistic models have also been considered for XML data under independence [24], under arbitrary distributions [17] and with interval probabilities [16]. There is also work on probabilistic object bases [9] and probabilistic logics for schema mappings [25].

There are also several non-probabilistic approaches to imprecise queries. Keyword searches in databases are discussed in [15,6,14]. Fagin [11] gives an algorithm to rank objects based on its scores from multiple sources: this applies only to a single table. The VAGUE system [20] supports queries with vague predicates, but the query semantics are ad hoc, and apply only to a limited SQL fragments. Chaudhuri et al. [1] consider ranking query results automatically: this also applies to a single table. Theobald and Weikum [34] describe a query language for XML that supports approximate matches with relevance ranking based on ontologies and semantic similarity.

13 Conclusions and Future Work

In this paper, we considered the problem of evaluating queries over probabilistic databases according to the *possible worlds semantics*.. We showed that by choosing

suitable execution plans for queries, extensional semantics can be used to evaluate a certain class of queries. We further showed that this class is precisely the class of queries that have polynomial time data complexity. Our theoretical results capture the fundamental properties of query complexity on probabilistic databases, and lead to efficient evaluation techniques. We showed how this approach can be used to evaluate arbitrarily complex SQL queries with uncertain predicates.

There are several problems that emerge from this work and remain open. Given any conjunctive query that is allowed to have self joins, can we decide if its data complexity is polynomial time? Is there still a dichotomy of queries into PTIME and $\#P$-complete classes when self joins are allowed? What is the complexity of query evaluation with aggregates like SUM, COUNT, MIN and MAX and with HAVING clauses? Apart from these questions, there are engineering issues that need to be resolved. We need to examine the implications for a relational engine: what functionality does a relational engine need to provide for an efficient implementation of probabilistic databases? Another interesting problem is to develop algorithms for top-K answers to the probabilistic queries.

References

1. Agrawal, S., Chaudhuri, S., Das, G., Gionis, A.: Automated ranking of database query results. In: CIDR (2003)
2. Bacchus, F., Grove, A.J., Halpern, J.Y., Koller, D.: From statistical knowledge bases to degrees of belief. Artificial Intelligence **87**(1-2), 75–143 (1996)
3. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. Addison-Wesley (1999)
4. Barbará, D., Garcia-Molina, H., Porter, D.: The management of probabilistic data. IEEE Trans. Knowl. Data Eng. **4**(5), 487–502 (1992)
5. Cavallo, R., Pittarelli, M.: The theory of probabilistic databases. In: VLDB, pp. 71–81 (1987)
6. Chaudhuri, S., Das, G., Narasayya, V.: Dbexplorer: A system for keyword search over relational databases. In: Proceedings of the 18th Int. Conf. on Data Engineering, San Jose, USA (2002)
7. Cheng, R., Kalashnikov, D.V., Prabhakar, S.: Evaluating probabilistic queries over imprecise data. In: SIGMOD, pp. 551–562 (2003)
8. Dey, D., Sarkar, S.: A probabilistic relational model and algebra. ACM Trans. Database Syst. **21**(3), 339–369 (1996)
9. Eiter, T., Lu, J.J., Lukasiewicz, T., Subrahmanian, V.S.: Probabilistic object bases. ACM Trans. Database Syst. **26**(3), 264–312 (2001)
10. Fagin, R., Halpern, J.Y.: Reasoning about knowledge and probability. In: Theoretical Aspects of Reasoning about Knowledge, pp. 277–293. San Francisco (1988)
11. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. In: PODS, pp. 102–113 (2001)
12. Fuhr, N., Rolleke, T.: A probabilistic relational algebra for the integration of information retrieval and database systems. ACM Trans. Inf. Syst. **15**(1), 32–66 (1997)
13. Gradel, E., Gurevich, Y., Hirch, C.: The complexity of query reliability. In: PODS, pp. 227–234 (1998)
14. Guo, L., Shao, F., Botev, C., Shanmugasundaram, J.: Xrank: Ranked keyword search over xml documents. In: SIGMOD, pp. 16–27 (2003)
15. Hristidis, V., Papakonstantinou, Y.: Discover: Keyword search in relational databases. In: Proc. 28th Int. Conf. Very Large Data Bases, VLDB (2002)
16. Hung, E., Getoor, L., Subrahmanian, V.S.: Probabilistic interval xml. In: ICDE (2003)
17. Hung, E., Getoor, L., Subrahmanian, V.S.: Pxml: A probabilistic semistructured data model and algebra. In: ICDE (2003)
18. Karp, R., Luby, M.: Monte-carlo algorithms for enumeration and reliability problems. In: STOC (1983)
19. Lakshmanan, L.V.S., Leone, N., Ross, R., Subrahmanian, V.S.: Probview: a flexible probabilistic database system. ACM Trans. Database Syst. **22**(3), 419–469 (1997)
20. Motro, A.: Vague: a user interface to relational databases that permits vague queries. ACM Trans. Inf. Syst. **6**(3), 187–214 (1988)
21. Movie database: *http://kdd.ics.uci.edu/databases/movies/movies.html*
22. Navarro, G.: A guided tour to approximate string matching. ACM Computing Surveys **33**(1), 31–88 (2001)
23. Ng, R.T., Subrahmanian, V.S.: Probabilistic logic programming. Information and Computation **101**(2), 150–201 (1992)
24. Nierman, A., Jagadish, H.V.: ProTDB: Probabilistic data in XML. In: VLDB (2002)
25. Nottelmann, H., Fuhr, N.: Combining DAML+OIL, XSLT and probabilistic logics for uncertain schema mappings in MIND. In: ECDL (2003)
26. Pearl, J.: Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1988)
27. Provan, J.S., Ball, M.O.: The complexity of counting cuts and of computing the probability that a graph is connected. SIAM J. Comput. **12**(4), 777–788 (1983)
28. Ross, R., Subrahmanian, V., Grant, J.: Aggregate operators in probabilistic databases. Journal of the ACM **52**(1), 54–101 (2005)
29. Sadri, F.: Reliability of answers to queries in relational databases. TKDE **3**(2), 245–251 (1991)
30. Sadri, F.: Aggregate operations in the information source tracking method. Theor. Comput. Sci. **133**(2), 421–442 (1994)
31. Sadri, F.: Information source tracking method: Efficiency issues. TKDE **7**(6), 947–954 (1995)
32. Sadri, F.: Integrity constraints in the information source tracking method. IEEE Transactions on Knowledge and Data Engineering **7**(1), 106–119 (1995)
33. Stoer, M., Wagner, F.: A simple min cut algorithm. Algorithms–ESA '94 pp. 141–147 (1994)
34. Theobald, A., Weikum, G.: The xxl search engine: ranked retrieval of xml data using indexes and ontologies. In: SIGMOD, pp. 615–615 (2002)
35. Ullman, J.D., Widom, J.: First Course in Database Systems, 2nd ed. Prentice Hall (1997)
36. Valiant, L.: The complexity of enumeration and reliability problems. SIAM J. Comput. **8**, 410–421 (1979)
37. Wordnet 2.0: A lexical database for the english language: *http://www.cogsci.princeton.edu/ wn/* (2003)
38. Zimanyi, E.: Query evaluation in probabilistic databases. Theoretical Computer Science **171**(1-2), 179–219 (1997)
39. Zobel, J., Dart, P.W.: Phonetic string matching: Lessons from information retrieval. In: Research and Development in Information Retrieval, pp. 166–172 (1996)