Efficient Processing of Aggregates in Probabilistic Databases

Lixia Chen and Alin Dobra
University of Florida
{lixchen,adobra}@cise.ufl.edu

Abstract

Computing expectations of aggregates over probabilistic databases is believed to be a hard problem. Moreover, since providing confidence intervals for aggregation is strictly necessary, the problem is even harder. At least the expectation and variance need to be evaluated to provide usable confidence intervals. In this paper, we present efficient algorithms for computing moments of SUM-like aggregates over non-correlated multi-relation queries on databases with tuple-uncertainty. The core idea in our work is to statistically analyze moments of the aggregates by making extensive use of linearity of expectation. The theoretical formulas can be expressed in SQL and evaluated using a traditional DBMS. Our experiments indicate that only a small overhead – usually below 10% – is needed to evaluate probabilistic aggregations using a traditional DBMS. Furthermore, the distribution of probabilistic aggregates tends to be normal and well approximated by the normal distribution with the mean and variance computed using our method. This means good quality confidence bounds based on normality can be readily computed. When the normal approximation is not appropriate, conservative Chebychev bounds can be used instead, bounds that are based on mean and variance as well.

1 Introduction

Probabilistic databases have attracted much attention in recent years because of the increasing demand for processing uncertain data in practice. Computing aggregates over probabilistic data is useful in situations where analytical processing is required over uncertain data. Since errors are inevitably introduced in the data, or the nature of certain kinds of data is uncertain, in a natural manner the need arises to process aggregates over probabilistic databases. While the need is there, the speed of processing aggregates over probabilistic data has to be comparable with the speed current system achieves when computing aggregates over non-probabilistic data. Should the former be much slower, the appeal of probabilistic processing is significantly diminished.

The estimation of aggregates over probabilistic databases is perceived as a harder problem than determining probabilities for result tuples. As a consequence, dealing with aggregate queries has been mostly postponed until enough progress is made with probabilistic databases. Moreover, when it comes to aggregation of probabilistic data, it is not enough to compute aggregates over the top-k most probable tuples since there can be tuples with low probabilities but large values of aggregate that have a major influence on the aggregate value. Even more, computation of the average/expected value of the aggregate is not enough, actual confidence intervals need to be provided. Confidence intervals that indicate the region in which the value of the aggregate can be, not only the average behavior. For example, completely different decisions would be made by a manager if the total revenue is $\$10,000,000 \pm 10,000$ or if the revenue is $\$10,000,000 \pm 100,000,000$. In both cases the average is the same but the latter situation has the potential to be very risky whereas the first is safe. By computing the variance of the aggregate, good confidence bounds can be provided.

The type of probabilistic database we consider in this paper allows uncertainty only in the existence of the tuples in the database. This tuple uncertainty model is widely used in previous work [8, 9, 7, 18]. The actual information in the tuple is not probabilistic and the existence of each tuple is independent of the existence of all other tuples. While considering extensions of this model is interesting, the problem needs to

be tackled first for this simpler model in order to have a chance to solve the more complicated extensions. A natural line of attack for the computation of aggregates under this probabilistic model is to use the possible world paradigm in [16, 13, 2, 17, 22, 4, 8, 15]. While this helps in understanding the semantics, it does not lead to practical methods for computation of the moments of aggregates, and thus of the confidence intervals. A better approach, used successfully in the analysis of sampling methods in [20, 21], is to make extensive use of the linearity of expectation. Other key ingredients are expressing the aggregates in terms of simple random variables and making use of the Kronecker delta symbol. These techniques allow us to characterize `SUM` and `AVERAGE`-like aggregates for non-correlated, distinct free, multi-relation queries. More precisely, we make the following contributions:

- We derive formulas for the moments of `SUM` aggregates, which allows computation of confidence intervals.
- We indicate how these formulas can be evaluated using SQL and a traditional DBMS.
- We optimize the formula for the variance to remove the need to perform computation over the cross-products of matching tuples and indicate how these optimized formulas can be evaluated using SQL.
- We extend the analysis to nonlinear aggregates like `AVERAGE` and for queries containing `GROUPBY` clauses.
- We evaluate the performance of our algorithms on TPC-H data set and observe that the distribution of probabilistic aggregates tends to be normal and well approximated by the normal distribution with the mean and variance computed using our method.

Except for requiring the aggregate query not to have subqueries or duplicate elimination, we put no restrictions on the `WHERE` condition or on the aggregate functions to be computed. Indeed, the treatment in this paper is fully general for the types of queries we consider. Interestingly, as our experiments with an implementation of our theoretical results show, confidence intervals for aggregate queries over probabilistic data can be efficiently provided. As we will see in Section 6, in most situations the overhead is below 10% (in the worst case a factor of 2.26) when the optimized formulas for the variance are used. Such performance was obtained using query rewriting alone without any change to the database system. This might suggest that probabilistic aggregate queries can be easily integrated in existing DBMS's without the need for major redesign.

This paper is organized as follows. In Section 2, we introduce the probabilistic data model, the type of queries and techniques used in this paper. In Section 3, we analyze the first two moments of aggregates over one relation. This analysis serves as a warm-up for the general case. Section 4 studies the moments of aggregate over multiple relations and how to evaluate them efficiently using SQL queries. Section 5 extends the analysis to non-linear aggregates and queries with a GROUPBY clause. We present experimental results in Section 6. Related work is discussed in Section 7 and conclusions are drawn in Section 8.

2 Preliminaries

In this section we introduce formally the queries we support in this work, the formal definition of the probabilistic model we use (tuple-uncertainty model) and useful basic techniques used throughout this paper: obtaining confidence intervals from moments and the use of Kronecker δ_{ij} to encode the cases in order to streamline analysis of random variables.

2.1 Queries and Equivalent Algebraic Expressions

The basic type of query we consider in this paper is:

$$\texttt{SELECT SUM}(F(t_1 \bullet \cdots \bullet t_n))\texttt{FROM } R_1 \texttt{ AS } t_1, \ldots, R_n \texttt{ AS } t_n\texttt{WHERE SIMPLE\_CONDITION};$$

where $R_1, \ldots, R_n$ are n relations, $F(\cdot)$ is an aggregate function that can depend on tuples from each relation and
`SIMPLE_CONDITION` is a condition involving only tuples from relations $R_1, \ldots, R_n$. Arbitrary selections and

join conditions are allowed but no subqueries of any kind or *DISTINCT* are allowed (no queries with set semantics).

In order to perform the statistical analysis in this paper, we need to translate the SQL query above into algebraic formulas. To this end, we unify the `WHERE` condition and the aggregate function $F(\cdot)$ into a single function $f(t_1 \bullet \cdots \bullet t_n)$ in the following way: (a) introduce a *filtering* function $I_C(t)$ that takes value 1 if the tuple t satisfies the condition C and 0 otherwise, and (b) define $f(t) = F(t)I_C(t)$. With this, the value of the aggregate we consider is simply:

$$A = \sum_{t_1 \in R_1} \cdots \sum_{t_n \in R_n} f(t_1 \bullet \cdots \bullet t_n)$$

that is, the sum of the function $f(\cdot)$ over the cross-product of the relations involved.

2.2 Confidence Intervals from Moments

As we argued in the introduction, from user's perspective it is not enough to provide the expected value of an aggregate over the probabilistic database. A confidence interval would provide a lot more information. The standard way to obtain confidence intervals for random variables is to compute the first two central moments, $E[X]$ and $\mathrm{Var}(X)$ and then to use either a distribution dependent or independent bound. The distribution dependent bounds assume the type of distribution is known and is one of the two-parameter *nice* distributions. The most common situation is the application of the Central Limit Theorem to argue that the distribution is asymptotically normal. The two parameters of the distribution are computed from $E[X]$ and $\mathrm{Var}(X)$ and then the $\frac{\alpha}{2}$ and $1-\frac{\alpha}{2}$ quantities are determined, with $1-\alpha$ the desired confidence (α is the allowed error). The distribution independence bounds use the Chebychev inequality to provide conservative bounds (bounds that are correct irrespective of the distribution but might be unnecessarily large). This bound requires quantities $E[X]$ and $\mathrm{Var}(X)$ as well.

The two types of bounds we discussed above require the computation of $E[X]$ and $\mathrm{Var}(X)$. Usually, $E[X]$ is easy to compute but $\mathrm{Var}(X)$ poses significant problems. Unfortunately, it is not possible to avoid the computation of $\mathrm{Var}(X)$ and still obtain reasonable confidence intervals. If only $E[X]$ is known, only Markov's inequality of Hoeffding bounds can be produced. Both can be reasonably efficient if multiple copies of the random variable are available and averaged but both are completely inefficient if this is not the case. As we will see in the next section, we have only one copy of the random variable that characterizes the aggregates, thus $\mathrm{Var}(X)$ is strictly required if reasonable confidence bounds are to be produced. An alternative to consider is to obtain multiple independent instances of X using Monte-Carlo simulation. For the error to be reasonable at least 100 such samples are required which results in a 100 fold increase in the running time over non-probabilistic aggregates – a less than ideal scenario.

2.3 Probabilistic Database as a Description of a Probability Space

In this paper we use probabilistic databases with the following properties: (a) we are given a set of base relations $R_1, \ldots, R_n$, (b) to each tuple t_i in a relation R_i we associate a probability p_{t_i} of inclusion into instance R'_i of R_i, (c) the inclusion of a tuple is independent of the inclusion of all other tuples. The instances $R'_1, \ldots, R'_n$ of relations $R_1, \ldots, R_n$ form a possible world or database instance. Since tuple inclusions are independent, the probability of any such possible world is the product of probabilities of the tuples that appear in the world. We denote the possible worlds by w. This is essentially the tuple-uncertainty probabilistic model.

Since a possible world is a database instance, aggregates of the type described in Section 2.1 over the database are well defined. If we denote by $\mathcal{A}$ such aggregates, then

$$\mathcal{A} = \sum_{t_1 \in R'_1} \cdots \sum_{t_n \in R'_n} f(t_1 \bullet \cdots \bullet t_n)$$

i.e. the aggregate is the sum of applications of function $f(\cdot)$ over tuples from the possible world. Since relations R'_i are random (obtained according to the probabilistic model above), $\mathcal{A}$ is a random variable. The

moments of $\mathcal{A}$, by definition, are:

$$E\left[\mathcal{A}^k\right] = \sum_{w \in W} \mathcal{A}(w)^k p(w) \tag{1}$$

where $p(w)$ is the probability of the possible world w and W is the set of all possible worlds. This immediately gives formulas for $E[\mathcal{A}]$ and $\mathrm{Var}(A) = E\left[\mathcal{A}^2\right] - E[\mathcal{A}]^2$, the two moments that are needed to compute confidence intervals. Unfortunately, computing the moments of $\mathcal{A}$ using this formula is impractical since the number of possible worlds is exponential in the size of relations. This paper is mostly concerned with deriving computation methods that avoid the enumeration of all possible worlds.

2.4 Use of Kronecker δ_{ij}

Cases appear naturally in the analysis of complex random variables since the simple random variables that they are constructed from interact differently with themselves and with other random variables (for example when the variance of the complex random variable is performed). An elegant method used in [20, 21] is to make use of the Kronecker symbol to encode cases. The Kronecker symbol is defined as:

$$\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

Assume that we are given a quantity Q_{ij} that takes value a if $i = j$ and b if $i \neq j$. We can express Q_{ij} in terms of δ_{ij} by observing that we can multiply a by δ_{ij}, b by $(1 - \delta_{ij})$ and obtain:

$$\begin{aligned} Q_{ij} &= \begin{cases} a & i = j \quad (\times \delta_{ij}) \\ b & i \neq j \quad (\times (1 - \delta_{ij})) \end{cases} \\ &= b + (a - b)\delta_{ij} \end{aligned}$$

This is correct since, if $i = j$, then $\delta_{ij} = 1$. Thus $Q = b + (a - b) \cdot 1 = a$ (and the symmetric argument for $i \neq j$).

The following simplification rule is useful for removing δ_{ij} from expressions:

$$\sum_i \sum_j \delta_{ij} \mathcal{F}(i, j) = \sum_i \mathcal{F}(i, i)$$

i.e. the double sum *collapses* into a single sum because of δ_{ij}. Even if δ_{ij} is not removed, it is easy to evaluate.

3 SUM aggregates over one relation

In this section we analyze the SUM aggregate over a single relation. As we will see, the analysis is intuitive and makes explicit use of the independence of the selection of tuples in the database instance. For this reason, the proof is direct and poses no problems. The resulting formulas can be rewritten in SQL, which gives a straightforward way to evaluate them using a DBMS. While the analysis is straightforward for the one-relation case, as we will discuss at the end of the section, it cannot be generalized to the multi-relation case. The main reason is that independence is lost the moment two or more relations are involved. Nevertheless, part of the technique is useful for the generalization and it is better exemplified by this simpler case.

For the one-relation case, the aggregate to be estimated over the instance R' of the relation R is:

$$\mathcal{A} = \sum_{t \in R'} f(t)$$

In order to be able to compute $E[\mathcal{A}]$ and $\mathrm{Var}(A)$, for each tuple $t \in R$, we introduce the $0, 1$ random variable X_t that indicates whether t is selected in R' or not. With this,

$$\mathcal{A} = \sum_{t \in R} X_t f(t)$$

Writing $\mathcal{A}$ in this manner is key for analysis since the dependence on a random range is replaced by a dependence on random variables so linearity of expectation commutes with the sum. This technique is useful in general case as well. Using the linearity of expectation, we have:

$$E[\mathcal{A}] = \sum_{t \in R} E[X_t] f(t) = \sum_{t \in R} p_t f(t)$$

where we used the fact that $E[X_t] = P[X_t = 1] = p_t$.

To compute $\mathrm{Var}(\mathcal{A})$, we use the fact that, according to our probability model, the selection of tuples into R' is independent, thus X_t is independent of $X_{t'}$. This means that the variance of the sum is the sum of variances, thus

$$\mathrm{Var}(\mathcal{A}) = \sum_{t \in R} \mathrm{Var}(X_t) f(t)^2 = \sum_{t \in R} p_t(1 - p_t) f(t)^2$$

Above we used the fact that, for a Bernoulli r.v.

$$\mathrm{Var}(X_t) = p_t(1 - p_t)$$

and the fact that constants are squared when taken out of variances.

In order to evaluate these formulas in SQL, we observe that

$$f(t) = F(t) I_C(t)$$

with C the selection condition, thus $f(t)^2 = F(t)^2 I_C(t)$. In order to compute the formulas for $E[\mathcal{A}]$ and $\mathrm{Var}(\mathcal{A})$ using SQL, we will add a `P` attribute to relations R that specifies the probability of the tuple. The SQL statement that computes the expectation and variance is:

```
SELECT SUM(F(t) × P), SUM(F(t)^2 × P × (1-P))
FROM R as t WHERE C;
```
(2)

It is immediately apparent that the computation on the expectation and variance is as easy as the computation of the non-probabilistic aggregate. The most surprising fact is that variance can be estimated so efficiently.

The above results for the one-relation case are encouraging and might suggest that similar results should exist for multi-relation case. Unfortunately, the above proof technique cannot be extended to the two-relation case. To see this, let t_1 be a tuple in R_1 and t_2 and t_2' be tuples in R_2. Tuples (t_1, t_2) and (t_1, t_2') can both be part of the aggregate computation, but their existence in $R_1' \times R_2'$ is not independent since the t_1 part is common.[1] This means that variance cannot be pushed inside the summations, thus a much more complicated computation is required.

4 Analysis for Multi-relation SUM Aggregates

As we have seen in Section 2, the computation of the moments of an aggregate using the possible worlds is computationally infeasible. Some more progress can be made if $0, 1$ random variables are introduced to indicate whether a tuple is included in a relation instance or not, as we have seen in Section 3. We reached an impasse with this method since we depended on independence to do the derivation, independence is lost the moment two relations are involved in the aggregate. The approach in this section to overcome this problem is to *not make* explicit use of independence between tuples of the same relations by introducing a proof that carries out the full computation and does not use the *independence* shortcut. It might seem that, when this technique is used, significantly longer proofs could result. It turns out that this is not the case and, when

[1] This can be checked by observing that $P[t_1 \in R_1' \wedge t_2 \in R_2' \wedge t_2' \in R_2'] = p_{t_1} p_{t_2} p_{t_2'}$, which differs by a factor p_{t_1} when compared to $P[t_1 \in R_1' \wedge t_2 \in R_2'] P[t_1 \in R_1' \wedge t_2' \in R_2'] = p_{t_1}^2 p_{t_2} p_{t_2'}$.

pairing up this style of proofs with extensive use of linearity of expectation, compact and general derivations result. The proof technique we use is similar to [20, 21] and makes use of the Kronecker delta symbol.

The first key ingredient for the analysis is to use the $0, 1$ indicator variables X_t for each tuple in each of the relations.[2] Each X_t has a Bernoulli distribution parametrized by probability p_t. The second key ingredient is to express the interaction between two random variables X_t and $X_{t'}$ in terms of Kronecker delta. This allows a pure algebraic manipulation without the need to deal with cases that significantly complicate the formulas. When these two ingredients are used together, the derivations become straightforward.

With the above comments in mind, the analysis of the aggregate $\mathcal{A}$ will consist in: (a) express $\mathcal{A}$ in terms of the data and r.v. X_t, (b) use linearity of expectation to compute $E[\mathcal{A}]$ and $\mathrm{Var}(\mathcal{A})$ in terms of X_t, and (c) interpret the resulting formulas from a database point of view and express them in SQL. With the notation in Section 2.3 and using the same idea, the aggregate A can be expressed in terms of X_t as:

$$\begin{aligned}\mathcal{A} &= \sum_{t_1 \in R'_1} \cdots \sum_{t_n \in R'_n} f(t_1 \bullet \cdots \bullet t_n) \\ &= \sum_{t_1 \in R_1} \cdots \sum_{t_n \in R_n} \left(\prod_{i=1}^{n} X_{t_i} \right) f(t_1 \bullet \cdots \bullet t_n)\end{aligned} \tag{3}$$

The following properties of X_t, that follow directly from the fact that it has a Bernoulli distribution, are needed in the rest of the paper:

$$E[X_t] = P[t \in R] = p_t \tag{4}$$

$$\begin{aligned}E[X_t X_{t'}] &= \begin{cases} P[t \in R] & t = t' \\ P[t \in R \wedge t' \in R] & t \neq t' \end{cases} \\ &= \begin{cases} p_t & t = t' \quad (\times \delta_{tt'}) \\ p_t p_{t'} & t \neq t' \quad (\times (1 - \delta_{tt'})) \end{cases} \\ &= p_t p_{t'} + p_t(1 - p_t)\delta_{tt'}\end{aligned} \tag{5}$$

where we used the fact that X_t and $X_{t'}$ are independent (if $t \neq t'$) and the technique explained in Section 2.4 to express cases using the Kronecker delta.

Since the formulas will be too large with the current notation (too many summation signs), as in [21], we introduce more compact notation for summations:

$$\sum_{t_1 \in R_1} \cdots \sum_{t_n \in R_n} f(t_1 \bullet \cdots \bullet t_n) = \sum_{\{t_i \in R_i | i \in \{1:n\}\}} f(\{t_i | i \in \{1:n\}\})$$

Sums that are subscripted by the set $\{t_i \in R_i | i \in S\}$ are equivalent to sums over each of the indexes in the set. We use the same compact notation for the arguments of $f(\cdot)$.

With this, we have the following result:

Theorem 1 *The moments of $\mathcal{A}$, the aggregate defined by Equation 3 are:*

$$E[\mathcal{A}] = \sum_{\{t_i \in R_i | i \in \{1:n\}\}} \left(\prod_{i=1}^{n} p_{t_i} \right) f(\{t_i | i \in \{1:n\}\})$$

$$E\left[\mathcal{A}^2\right] = \sum_{\{t_i \in R_i | i \in \{1:n\}\}} \sum_{\{t_{i'} \in R_{i'} | i' \in \{1:n\}\}} \left(\prod_{i=1}^{n} (p_{t_i} p_{t_{i'}} + p_{t_i}(1 - p_{t_i})\delta_{t_i t_{i'}}) \right) f(\{t_i | i \in \{1:n\}\}) f(\{t_{i'} | i' \in \{1:n\}\})$$

$$Var(\mathcal{A}) = E\left[\mathcal{A}^2\right] - E[\mathcal{A}]^2$$

[2] There is no need to index the random variables by the relation as well, as is done in [21], since it will be clear from the context what relation we are referring to and all random variables interact the same way.

Proof. Using Equation 4, linearity of expectation and the fact that X_t r.v. are independent for different relations, thus the expectation of products is product of expectations, we have:

$$\begin{aligned} E[\mathcal{A}] &= \sum_{\{t_i \in R_i | i \in \{1..n\}\}} \left(\prod_{i=1}^{n} E[X_{t_i}] \right) f(\{t_i | i \in \{1:n\}\}) \\ &= \sum_{\{t_i \in R_i | i \in \{1..n\}\}} \left(\prod_{i=1}^{n} p_{t_i} \right) f(\{t_i | i \in \{1:n\}\}) \end{aligned}$$

Creating two instances for $\mathcal{A}$ with the indexes primed for the second one, using Equation 5 and linearity of expectation, we have:

$$\begin{aligned} E[\mathcal{A}^2] &= E\left[\sum_{\{t_i \in R_i | i \in \{1..n\}\}} \sum_{\{t_{i'} \in R_{i'} | i' \in \{1..n\}\}} \left(\prod_{i=1}^{n} X_{t_i} X_{t_{i'}} \right) f(\{t_i | i \in \{1:n\}\}) f(\{t_{i'} | i' \in \{1:n\}\}) \right] \\ &= \sum_{\{t_i \in R_i | i \in \{1..n\}\}} \sum_{\{t_{i'} \in R_{i'} | i' \in \{1..n\}\}} \left[\left(\prod_{i=1}^{n} E[X_{t_i} X_{t_{i'}}] \right) f(\{t_i | i \in \{1:n\}\}) f(\{t_{i'} | i' \in \{1:n\}\}) \right] \\ &= \sum_{\{t_i \in R_i | i \in \{1..n\}\}} \sum_{\{t_{i'} \in R_{i'} | i' \in \{1..n\}\}} \left[\left(\prod_{i=1}^{n} (p_{t_i} p_{t_{i'}} + p_{t_i}(1 - p_{t_i}) \delta_{t_i t_{i'}}) \right) f(\{t_i | i \in \{1:n\}\}) \right. \\ &\qquad \left. \times f(\{t_{i'} | i' \in \{1:n\}\}) \right] \end{aligned}$$

□

Making the same observations as in Section 3, the formulas in Theorem 1 can be efficiently evaluated using a DBMS. Remember that $f(t) = F(t) I_C(t)$ where $F(t)$ is the aggregate function and $I_C(t)$ is the indicator function of the condition C. To compute $E[\mathcal{A}]$ using SQL we observe that the query is the same except the aggregate function is multiplied by the product of probabilities:

```
SELECT SUM(F(t_1 • ··· • t_n) × t_1.P × ··· × t_n.P)
FROM R_1 AS t_1, ..., R_n AS t_n
WHERE SIMPLE_CONDITION;
```

In order to compute $E[\mathcal{A}^2]$, thus $\mathrm{Var}(\mathcal{A})$, we have to compute the aggregate over the cross product of the result tuples. To translate this efficiently into SQL, we put the unaggregate value and the probabilities into a temporary table and then compute the aggregate over the cross-product of this table with itself. It is possible to write this as a single query but here, for ease of exposition, we prefer two queries. We will assume that a function `Delta(i,j)` that implements the Kronecker delta is available (functions can be added to all major database systems). In order to be able to apply the function, we need the ids of the tuples (primary keys are ideal ids; we will assume that the attribute ID exists). With this, the SQL code to compute $E[\mathcal{A}]$ is:

```
SELECT F(t_1 • ··· • t_n) AS F, t_1.P AS P_1, ..., t_n.P AS P_n, t_1.ID AS ID_1, ..., t_n.ID AS ID_n
INTO unaggregated
FROM R_1 AS t_1, ..., R_n AS t_n
WHERE SIMPLE_CONDITION;
```
(6)

```
SELECT SUM(U.F × V.F × (U.P_1 × V.P_1 + U.P_1 × (1 − U.P_1) × Delta(U.ID_1, V.ID_1)) × ... )
FROM unaggreggated AS U, unaggreggated AS V;
```
(7)

Clearly, except for a slightly more complicated aggregate function, the effort to compute $E[\mathcal{A}]$ is the same as the effort to compute the original aggregate A. Furthermore, the query plan can remain the same and still be efficient. The computation of $E[\mathcal{A}^2]$ involves the cross-product of all result tuples, with no possibility of further optimization at the database level. There are no conditions to take advantage of to replace the cross product by a join. This evaluation is equivalent to the method described in Section 4.1 if the probabilistic database is highly optimized for this particular type of query. The fact that SQL can be used directly instead of redesigning the database helps but this solution is not *fundamentally* better. As we will see in Section 4.2, further optimizations can be applied to this solution in order to significantly reduce the time complexity.

4.1 Connection with existing probabilistic databases

Let us consider a more general type of query in this section: aggregation queries in which `DISTINCT` and set operators are allowed. The only restriction is for the aggregate to be applied as the last operation. Essentially, this means that the queries have the same structure as described in Section 2, but the relations $R_1, \ldots, \mathcal{R}_n$ can be views instead of stored relations. The algebraic formula of the aggregate is:

$$A = \sum_{t \in R_M} F(t)$$

where R_M is the relation containing matching tuples and the aggregate is `SUM(`$F(t)$`)`. If the database is probabilistic, the value of the aggregate over an instance R'_M is:

$$\mathcal{A} = \sum_{t \in R'_M} F(t) = \sum_{t \in R_M} X_t F(t)$$

where, as we did in Sections 3 and 4, we introduced a $0,1$ random variable X_t that indicates whether the matching tuple t is included in the instance (world) R'_M or not. The introduction of X_t is crucial since we can use the linearity of expectation to prove the following result:

Proposition 1

$$E[\mathcal{A}] = \sum_{t \in R_M} P[t \in R'_M] F(t)$$
$$E[\mathcal{A}^2] = \sum_{t \in R_M} \sum_{t' \in R_M} P[t \in R'_M \wedge t' \in R'_M] F(t)F(t')$$

Proof. First, by linearity of expectation:

$$\begin{aligned} E[\mathcal{A}] &= E\left[\sum_{t \in R_M} X_t F(t)\right] \\ &= \sum_{t \in R_M} E[X_t] F(t) \\ &= \sum_{t \in R_M} P[t \in R'_M] F(t) \end{aligned}$$

where we used the fact that X_t is $0,1$ random variable thus

$$\begin{aligned} E[X_t] &= 1 \cdot P[X_t = 1] + 0 \cdot P[X_t = 1] \\ &= P[X_t = 1] \\ &= P[t \in R'_M] \end{aligned}$$

Similarly,

$$\begin{aligned} E\left[\mathcal{A}^2\right] &= E\left[\sum_{t\in R_M}\sum_{t'\in R_M} X_t X_{t'} F(t)F(t')\right] \\ &= \sum_{t\in R_M}\sum_{t'\in R_M} E\left[X_t X_{t'}\right] F(t)F(t') \\ &= \sum_{t\in R_M}\sum_{t'\in R_M} P\left[t\in R'_M \wedge t'\in R'_M\right] F(t)F(t') \end{aligned}$$

where we used the fact that

$$E\left[X_t X_{t'}\right] = P\left[t\in R'_M \wedge t'\in R'_M\right]$$

(using the same reasoning as for $E\left[X_t\right]$) and linearity of expectation. □

The formulas in Proposition 1 can be evaluated using a probabilistic database in the following manner. To compute $E\left[\mathcal{A}\right]$ go over all possible matching tuples and multiply the probability to see the tuple and the value of the aggregate function (all these terms are accumulated in the total sum). To compute $E\left[\mathcal{A}^2\right]$ go over the cross product of the possible matching tuples and multiply the probability to simultaneously see the two tuples and the product of values of the aggregate function applied to the two tuples.

Interestingly, the above method to compute $E\left[\mathcal{A}\right]$ and $\mathrm{Var}\left(\mathcal{A}\right)$ can be seen as a way to implement, using existing probabilistic databases, the results in Theorem 1. For the queries we consider in this paper, since we were able in the previous section to write SQL statements to carry out their computation, the fact that the same formulas can be computed using existing probabilistic databases is not very useful since executing the SQL statements using an traditional DBMS will be significantly more efficient. The usefulness comes in when we observe that the same technique can be applied to queries that contain distinct and set operators, case in which a probabilistic database can compute the probability to see a matching tuple. Potentially existing probabilistic databases can be modified to compute the probability of simultaneous inclusion of two matching tuples, thus allow the computation of $E\left[\mathcal{A}^2\right]$ which gives $\mathrm{Var}\left(\mathcal{A}\right)$.

An even tighter connection can be established with the *safe plans* of Dalvi and Suciu[8]. If we relate the formulas for $E\left[\mathcal{A}\right]$ in Theorem 1 and Proposition 1, we immediately have:

$$P\left[(t_1,\ldots,t_n)\in R'_M\right] = \prod_{i=1}^{n} p_{t_i}$$

This is precisely the probability, as computed by the method in [8] of matching tuple $t=(t_1,\ldots,t_n)$. It is important to notice that, for the non-aggregate part of the queries we consider in this paper any plan is a safe-plan since they do not contain `DISTINCT`. It is no surprise then that the same formula should be used for $P\left[t\in R'_M\right]$. Since the safe-plans cannot contain the same relation multiple times, the method in [8] is not general enough to allow the computation of $P\left[t\in R'_M \wedge t'\in R'_M\right]$, which is required according to Proposition 1 in order to compute $E\left[\mathcal{A}^2\right]$ and $\mathrm{Var}\left(\mathcal{A}\right)$.

The above observations suggest that the moment based analysis of SUM-like aggregates that we consider in this paper can be used to compute confidence intervals for aggregates in the most general scenarios as well. The fact that the cross product of matching tuples has to be considered, might be a bigger problem than the fact that complex events have to be dealt with if the queries are relatively simple. For complex queries or queries with large number of matching tuples, both the method described in this section and the method in the previous section seem impractical. A significantly better method is described in the next section but it works only for the queries considered in this paper, not for general queries.

4.2 Reducing the time complexity

When we derived the formulas for $\mathrm{Var}\left(\mathcal{A}\right)$ for the one-relation case in Section 3, the resulting formula could be evaluated directly over the set of matching tuples; there was no need to go over the cross-product of the matching tuples for the computation of the $E\left[\mathcal{A}^2\right]$ part of the variance. An interesting question to ask is whether such a reduction is possible for the multi-relation case.

A fundamental question that can be asked about the formula for $E\left[\mathcal{A}^2\right]$ in Theorem 1 is whether we can remove the Kronecker deltas, for example by using the simplification rule in Section 2.4. While it is not clear why this might reduce the complexity, it would be a necessary step in that direction since otherwise there is no way to proceed.

To see where the opportunity lies, let us consider the two relation case. In this case, the formula for $E\left[\mathcal{A}^2\right]$ can be written as:

$$
\begin{aligned}
E\left[\mathcal{A}^2\right] &= \sum_{t\in R_1}\sum_{t'\in R_1}\sum_{v\in R_2}\sum_{v'\in R_2}((p_tp_{t'}+p_t(1-p_t)\delta_{tt'})(p_vp_{v'}+p_v(1-p_v)\delta_{vv'})f(t,v)f(t',v')) \\
&= \sum_{t\in R_1}\sum_{t'\in R_1}\sum_{v\in R_2}\sum_{v'\in R_2}((p_tp_{t'}p_vp_{v'}+p_t(1-p_t)\delta_{tt'}p_vp_{v'}+p_tp_{t'}p_v(1-p_v)\delta_{vv'}+p_t(1-p_t)\delta_{tt'}p_v(1-p_v)\delta_{vv'}) \\
&\qquad\qquad \times f(t,v)f(t',v')) \\
&= \sum_{t\in R_1}\sum_{t'\in R_1}\sum_{v\in R_2}\sum_{v'\in R_2}p_tp_{t'}p_vp_{v'}f(t,v)f(t',v') + \sum_{t\in R_1}\sum_{t'\in R_1}\sum_{v\in R_2}\sum_{v'\in R_2}p_t(1-p_t)\delta_{tt'}p_vp_{v'}f(t,v)f(t',v') \\
&\quad + \sum_{t\in R_1}\sum_{t'\in R_1}\sum_{v\in R_2}\sum_{v'\in R_2}p_tp_{t'}p_v(1-p_v)\delta_{vv'}f(t,v)f(t',v') \\
&\quad + \sum_{t\in R_1}\sum_{t'\in R_1}\sum_{v\in R_2}\sum_{v'\in R_2}(p_t(1-p_t)\delta_{tt'}p_v(1-p_v)\delta_{vv'}f(t,v)f(t',v')) \\
&= \sum_{t\in R_1}\sum_{v\in R_2}p_tp_vf(t,v)\sum_{t'\in R_1}\sum_{v'\in R_2}p_{t'}p_{v'}f(t',v') + \sum_{t\in R_1}p_t(1-p_t)\left(\sum_{v\in R_2}p_vf(t,v)\sum_{v'\in R_2}p_{v'}f(t,v')\right) \\
&\quad + \sum_{v\in R_2}p_v(1-p_v)\left(\sum_{t\in R_1}p_tf(t,v)\sum_{t'\in R_1}p_{t'}f(t',v)\right) + \sum_{t\in R_1}\sum_{v\in R_2}p_t(1-p_t)p_v(1-p_v)f(t,v)f(t,v) \\
&= \left(\sum_{t\in R_1}\sum_{v\in R_2}p_tp_vf(t,v)\right)^2 + \sum_{t\in R_1}p_t(1-p_t)\left(\sum_{v\in R_2}p_vf(t,v)\right)^2 \\
&\quad + \sum_{v\in R_2}p_v(1-p_v)\left(\sum_{t\in R_1}p_tf(t,v)\right)^2 \qquad + \sum_{t\in R_1}\sum_{v\in R_2}p_t(1-p_t)p_v(1-p_v)f(t,v)^2
\end{aligned}
$$

Now we can see why such a derivation leads to better evaluation algorithms for $E\left[\mathcal{A}^2\right]$. The first and the last terms straightforwardly require just aggregates over the matching tuples. To see how to efficiently evaluate the second term (the third term is symmetric), we observe that the inner square needs to be computed for each tuple $t \in R_1$. Thus, we can compute this squares using a `GROUPBY` with the correct aggregates. Once the squares are computed, one for each $t \in R_1$, we simply compute the rest of the aggregate. The query in SQL is:

```
SELECT SUM(SQ × P1 × (1 − P1))
FROM ( SELECT SUM(F × P2)^2 AS SQ , P1
       FROM unaggreggated
       GROUPBY ID1, P1 );
```

The grouping on P_1 is needed since otherwise the information cannot be passed up in the outer query. Thus, it seems that each term can be evaluated almost as efficiently as an aggregate over the relation `unaggregated` (an extra `GROUPBY` is required).

The above derivation suggests that we might be able to compute efficiently $\mathrm{Var}(\mathcal{A})$ in the general case. Indeed this is possible as we show in the rest of this section. In order to provide the result, we need the following technical lemma. The result will be used in Section 5 as well:

Lemma 1 *Let $a_{t_i}, b_{t_i}, i \in \{1:n\}, t_i \in R_i$ be arbitrary values. Furthermore, let $f(\{t_i\})$ and $g(\{t_i\})$ be arbitrary functions of tuples $\{t_i\}$. Then*

$$\sum_{\{t_i \in R_i | i \in \{1n\}\}} \sum_{\{t_{i'} \in R_{i'} | i' \in \{1n\}\}} \left(\prod_{i,i' \in \{1n\}} \left(a_{t_i} a_{t_{i'}} + b_{t_i} \delta_{t_i,t_{i'}} \right) \right) f(\{t_i\}) g(\{t_{i'}\})$$

$$= \sum_{S \in \mathcal{P}(n)} \sum_{\{t_i \in R_i | i \in S\}} \left[\left(\prod_{k \in S} b_{t_k} \right) \left(\sum_{\{t_j \in R_j | j \in S^C\}} \left(\prod_{l \in S^C} a_{t_l} \right) f(\{t_i, t_j\}) \right) \left(\sum_{\{t_{j'} \in R_{j'} | j' \in S^C\}} \left(\prod_{l' \in S^C} a_{t_{l'}} \right) g(\{t_i, t_{j'}\}) \right) \right]$$

with $\mathcal{P}(n)$ the power-set of $\{1:n\}$. S is a subset of $\mathcal{P}(n)$ and S^C is the complement of S w.r.t. $\mathcal{P}(n)$.

The lemma above provides a recipe to remove Kronecker deltas and factorize the result. Using this lemma, we can prove the following result:

Theorem 2 *Let $\mathcal{A}$ be the sampling estimate defined by Equation 3. Then,*

$$E\left[\mathcal{A}^2\right] = \sum_{S \in \mathcal{P}(n)} \sum_{\{t_i \in R_i | i \in S\}} \left[\left(\prod_{k \in S} p_{t_k}(1 - p_{t_k}) \right) \left(\sum_{\{t_j \in R_j | j \in S^C\}} \left(\prod_{l \in S^C} p_{t_l} \right) f(\{t_i, t_j\}) \right)^2 \right]$$

Proof. The result follows directly from the expression of $E[\mathcal{A}]$ in Theorem 1 and the Lemma 1 with $a_{t_i} = p_{t_i}$, $b_{t_i} = p_{t_i}(1 - p_{t_i})$ and $g(\cdot) = f(\cdot)$. □

The above result indicates that, for each set $S \in \mathcal{P}(n)$, 2^n in all, we have to perform a computation that generalizes the 2-relation case we saw before. Using the same reasoning, the SQL query that computes the term corresponding to set S is:

$$\begin{aligned} &\texttt{SELECT SUM}(SQ \times \prod_{i \in S} P_i \times (1 - P_i)) \\ &\texttt{FROM (SELECT SUM}(F \times \prod_{j \in S^C} P_j)\texttt{\^{}2 AS SQ }, \{P_i, i \in S\} \\ &\qquad \texttt{FROM unaggreggated} \\ &\qquad \texttt{GROUPBY } \{ID_i, P_i, i \in S\} \texttt{);} \end{aligned} \tag{8}$$

The computation is exponential in n since 2^n different aggregates need to be computed, but, in the worse case, it just requires a sort and a linear scan of relation `unaggregated` for each of the sets S. When the size of `unaggregated` is large, we expect this method to compute $E\left[\mathcal{A}^2\right]$ to be much faster that the method in Section 4. The unoptimized method would be comparable only when the size number of matching tuples is smaller than $n2^n$.

5 Extensions

The analysis and computation for `SUM` aggregates can serve as the basis for computing more complicated aggregates. The two extensions we consider are here: non-linear aggregates and `GROUPBY`.

5.1 Non-linear aggregates

The method described in the rest of the paper can provide confidence intervals only for `SUM` aggregates. In practice, other aggregates such as `AVERAGE` and `VARIANCE`, that are similar to `SUM`, are useful. Such aggregates can be computed from multiple `SUM` aggregates by making a nonlinear combination of them.[3] For example, `AVERAGE` is the ratio of `SUM` and `COUNT`[4]. Since our analysis extensively used the linearity of the `SUM` aggregate, it cannot be applied to non-linear aggregates like `AVERAGE` directly.

[3] If the combination is linear, then the $F(\cdot)$ functions can be combined into a single such function that is used for computation.
[4] `COUNT` is a `SUM` with the aggregate function $F(\cdot) = 1$.

To provide a general treatment, let $A_1, \ldots, A_k$ be `SUM` aggregates (with different aggregation functions). Then, a general non-linear aggregate needs to compute $A = \mathcal{F}(A_1, \ldots, A_k)$. For `AVERAGE` aggregate, A_1 is `SUM (F)`, A_2 is `SUM (1)` and $\mathcal{F}(A_1, A_2) = \frac{A_1}{A_2}$.

Now, let $\mathcal{A}_1, \ldots, \mathcal{A}_k$ be the random variables that give the value of the aggregates for the probabilistic database instance. The value of the aggregate A on the instance would be $\mathcal{A} = \mathcal{F}(\mathcal{A}_1, \ldots, \mathcal{A}_k)$. As we did before, we need to compute (or estimate with good precision) $E[\mathcal{A}]$ and $\mathrm{Var}(\mathcal{A})$ in order to provide confidence intervals for the aggregate. The standard method in Statistics to analyze non-linear combinations of random variables is the *delta method*[5] [26]. The delta method consists in expressing the moments of $\mathcal{A}$ in terms of the moments of $\mathcal{A}_1, \ldots, \mathcal{A}_k$. In particular:

$$E[\mathcal{A}] \approx \mathcal{F}(E[A_1], \ldots, E[A_k])$$

$$\begin{aligned}\mathrm{Var}(\mathcal{A}) \approx & \nabla\mathcal{F}(E[A_1], \ldots, E[A_k])^\top \cdot Var(A_1, \ldots, A_k) \\ & \cdot \nabla\mathcal{F}(E[A_1], \ldots, E[A_k])\end{aligned}$$

Proof. □ where $\nabla\mathcal{F}(\cdot)$ is the gradient of the function $\mathcal{F}$ (i.e. the vector consisting of the partial derivatives w.r.t. each component) evaluated at $E[A_1], \ldots, E[A_k]$. $Var(A_1, \ldots, A_k)$ is the variance matrix, that has $\mathrm{Var}(\mathcal{A}_u)$ on the diagonal and $\mathrm{Cov}(\mathcal{A}_u, \mathcal{A}_v)$ off the diagonal. We can use the technique developed in this paper to compute $E[\mathcal{A}_u]$ and $\mathrm{Var}(\mathcal{A}_u)$ efficiently for each $\mathcal{A}_u$. The only remaining task is to compute

$$\mathrm{Cov}(\mathcal{A}_u, \mathcal{A}_v) = E[\mathcal{A}_u\mathcal{A}_v] - E[\mathcal{A}_u]\,E[\mathcal{A}_v]$$

The only challenge is computing the $E[\mathcal{A}_u\mathcal{A}_v]$ component. The formula is provided by the following result:

Theorem 3 *Let $\mathcal{A}_u, \mathcal{A}_v$ be the sampling estimate defined by Equation 3 for two different sum aggregates given by aggregate functions $f_u(\cdot)$ and $f_v(\cdot)$. Then,*

$$\begin{aligned}E[\mathcal{A}_u\mathcal{A}_v] = \sum_{S\in\mathcal{P}(n)} \sum_{\{t_i\in R_i | i\in S\}} \Bigg[& \left(\prod_{k\in S} p_{t_k}(1-p_{t_k})\right) \times \left(\sum_{\{t_j\in R_j | j\in S^C\}} \left(\prod_{l\in S^C} p_{t_l}\right) f_u(\{t_i, t_j\})\right) \\ & \times \left(\sum_{\{t_{j'}\in R_{j'} | j'\in S^C\}} \left(\prod_{l'\in S^C} p_{t_{l'}}\right) f_v(\{t_i, t_{j'}\})\right)\Bigg]\end{aligned}$$

Proof. First, we observe that the same algebraic manipulations as in the case of $E[\mathcal{A}^2]$ in the proof of Theorem 1, can be used for $E[\mathcal{A}_u\mathcal{A}_v]$. Using linearity of expectation we obtain:

$$\begin{aligned}E[\mathcal{A}_u\mathcal{A}_v] = \sum_{\{t_i\in R_i | i\in\{1:n\}\}} \sum_{\{t_{i'}\in R_{i'} | i'\in\{1:n\}\}} \Bigg[& \left(\prod_{i=1}^{n} \left(p_{t_i}p_{t'_i} + \quad p_{t_i}(1-p_{t_i})\delta_{t_i t'_i}\right)\right) \times f_u(\{t_i | i\in\{1:n\}\}) \\ & \times f_v(\{t_{i'} | i'\in\{1:n\}\})\Bigg]\end{aligned}$$

Now, we can use again the results in Lemma 1 with $a_{t_i} = p_{t_i}$, $b_{t_i} = p_{t_i}(1-p_{t_i})$, $f(\cdot) = f_u(\cdot)$ and $g(\cdot) = f_v(\cdot)$ and we obtain the required result. □

To see how we can evaluate the $E[\mathcal{A}_u\mathcal{A}_v]$ terms using SQL, we first observe that the aggregates are computed using the same `WHERE` condition but different aggregate functions. In particular,

$$f_u(\cdot) = F_u(\cdot)I_C(\cdot)$$
$$f_v(\cdot) = F_v(\cdot)I_C(\cdot)$$

[5]Not to be confused with the Kronecker delta.

This means that the set of matching tuples over which the aggregates are computed are the same. Using the same reasoning as in Section 4.2 where we expressed the computation of $E\left[\mathcal{A}^2\right]$ in SQL, we get the following SQL expression for the computation of the terms corresponding to set S in the expression of $E\left[\mathcal{A}_u\mathcal{A}_v\right]$:

$$\texttt{SELECT SUM}(SQ_u \times SQ_v \times \prod_{i \in S} P_i \times (1 - P_i))$$
$$\texttt{FROM (SELECT SUM}(F_u \times \prod_{j \in S^C} P_j) \texttt{ AS } SQ_u, \texttt{SUM}(F_v \times \prod_{j \in S^C} P_j) \texttt{ AS } SQ_v, \{P_i, i \in S\}$$
$$\texttt{FROM unaggreggated}$$
$$\texttt{GROUPBY } \{ID_i, P_i, i \in S\} \texttt{);}$$

Furthermore, since the SQL queries for computation of terms corresponding to set S for all $E\left[\mathcal{A}_u^2\right]$ and $E\left[\mathcal{A}_u\mathcal{A}_v\right]$ terms are identical except the actual aggregates computed – the `GROUPBY` is the same – they can all be combined into a single query that computes all required aggregates simultaneously. This means that the number of overall SQL queries is 2^n, irrespective of the number of sum aggregates combined by the nonlinear aggregate. The number of aggregates computed by each such query will be $k(k+1)/2$, where k is the number of operations.

5.2 `GROUPBY` queries

If the query contains a `GROUPBY` clause, for each of the groups, confidence intervals for the value of aggregates have to be provided. The computation of the moments for the aggregates for each of these groups would be no different from the computation of the aggregates for the entire relations. With this in mind, one method to obtain the desired confidence intervals/group would be to generate the tuples in each group and then apply the techniques described in this paper for each such group. Fortunately, this can be accomplished using SQL without the need to change the database engine or external computation. Essentially, for each aggregate that is generated we have to add a `GROUPBY` clause to account for the extra grouping and make sure we have the group information where we need it. For example, with grouping on the set of attributes $\mathcal{G}$, the SQL statement at the end of Section 4.2 becomes:

$$\texttt{SELECT SUM}(SQ \times \prod_{i \in S} P_i \times (1 - P_i)), \mathcal{G}$$
$$\texttt{FROM (SELECT SUM}(F \times \prod_{j \in S^C} P_j)\texttt{\^{}2 AS SQ } , \{P_i, i \in S\}, \mathcal{G}$$
$$\texttt{FROM unaggreggated}$$
$$\texttt{GROUPBY } \{ID_i, P_i, i \in S\}, \mathcal{G} \texttt{)}$$
$$\texttt{GROUPBY } \mathcal{G};$$

6 Experiments

The most important question through experiments is what is the overhead of computing the moments of the probabilistic aggregates when compared to the execution of the non-probabilistic query. As we will see in this section, the overhead is usually small – this is especially true for the optimized version – thus the moments of the probabilistic aggregates can be computed with only minimal performance degradation.

A second question to ask is what is the distribution of the probabilistic aggregates. As our experiments will show, the distribution tends to be normal thus the expectation and variance provide complete information about the probabilistic aggregates. This is the best scenario possible since it means that tight confidence intervals can be produced efficiently.

Methodology All the experiments were carried on a 4 processor, 2.4 GHz CPU, and 8GB RAM machine running Linux (Ubuntu 2.6.20 kernel). We used Postgres as a DBMS without any modification. Our algorithms work as a client program that generates the SQL queries to implement the probabilistic aggregates.

Both client and DBMS were running on the same machine. We used TPC-H benchmark with a data set of size 0.1 G, 1G and 10 G. For each tuple, we added a probability attribute that was generated uniformly random in $[0, 1]$ interval. Notice that the query execution time does not depend on the values of the probabilities since the same computations are performed with different numbers.

6.1 Computation of Moments

We implemented both the unoptimized, Algorithm 1, and the optimized, Algorithm 2, versions of our algorithm using query rewriting. Aggregation over one relation was special cased in both algorithms (the specialized algorithm in Section 3 was used). Both algorithms compute first the `unaggregated` table and then process it to compute the moments. The main difference is the fact that the unoptimized algorithm needs to evaluate a single query over the cross product `unaggregated`×`unaggregated` and the optimized algorithm needs to evaluate 2^n `GROUPBY` queries over `unaggregated`.

Algorithm 1 ComputeMomentsUnOpt(q)

1: **if** q is an aggregation over a single relation **then**
2: Generate and execute SQL code(2)
3: **else**
4: Create a temporary table *unaggregated* using SQL code(6)
5: Generate and execute SQL code(7)
6: **end if**

Algorithm 2 ComputeMomentsOpt(q)

1: **if** q is an aggregation over a single relation **then**
2: Generate and execute SQL code(2)
3: **else**
4: Create a temporary table *unaggregated* using SQL code(6)
5: **for** $i = 0$ to $2^{\#relations} - 1$ **do**
6: Map each relation to set S or S^C by i
7: Generate and execute SQL code(8)
8: **end for**
9: Sum up the computed $2^{\#relations}$ items of each group.
10: **end if**

The algorithms are evaluated on queries Q1, Q3, Q5 and Q6 (all the queries in TPC-H without subqueries) for databases of size 1G, Table 1, and 10G, Table 2. To get insight into how the time is spent, for all queries that involve more than one relation, we timed the generation of `unaggregated` table, and the time to evaluate the moments once `unaggregated` table is generated for unoptimized and optimized algorithm. Q3 is executed with the original `GROUPBY` and without it in order to see the impact of large groups for which the aggregates need to be computed. The experimental results reveal as the following:

Queries	# rel.	GroupBy	# matches	time	time probabilistic (ms)			
				nonprob.(ms)	gen. unagg. table	unoptimized	optimized	% inc.
Q1	1	√	6001197	35105			61673	175%
Q3	3	√	30569	20637	21010	2905	1500	9%
Q3	3	×	30569	20270	21056	≈ 4 Hours	711	5%
Q5	6	√	23	5297	5492	86	2422	49%
Q6	1	√	202	21534			21535	.005%

Table 1: Experimental results for TPC-H queries, 1G database

Queries	# rel.	GroupBy	# matches	time	time probabilistic (ms)			
				nonprob.(ms)	gen. unagg. table	unoptimized	optimized	% inc.
Q1	1	√	59986052	529867			1199574	226%
Q3	3	√	301618	714125	737278	65723	32412	7.8%
Q3	3	×	301618	709218	711089	≈ 15 Days	10064	1.7%
Q5	6	√	195	52798	53630	292	2520	6.3%
Q6	1	√	2094	400423			403630	.8%

Table 2: Experimental results for TPC-H queries, 10G database

Aggregation over one relation Queries Q1 and Q6 involve a single relation (for this case, the method in Section 3 was used instead of the general method). For Q6, the time to perform the probabilistic aggregate is virtually the same as the time for the non-probabilistic aggregate for both database sizes. The time to find matching tuples dwarfs the aggregation time. For Q1, most of the time is spent on computing aggregates not finding matching tuples. Since the probabilistic aggregate needs 25 aggregations versus only 8 aggregations for the non-probabilistic aggregate, the running time is approximately $25/8 = 3.125$ times higher. The time spent on Q1 seems to be CPU not I/O bound.

Table `unaggregated` The time to generate the `unaggregated` is almost the same as the execution of the original non-probabilistic query. This is expected since the aggregation is usually the last operation performed thus the table `unaggregated` is generated as an intermediate result.

Comparison of unoptimized and optimized algorithms The advantage of the optimized algorithm is that it does not require the formation of the cross product within each group of tuples in `unaggregated`. Query Q3, as it appears in TPC-H contains a `GROUPBY` that limits the number of tuples within each group. In this case the optimized algorithm is about twice as efficient as the unoptimized algorithm for both 1G and 10G database. To see what happens when the size of groups grows, we removed the `GROUPBY` clause in Q3. For the 1G data set, 30569 tuples are now part of a single group. For 10G data set, 301618 tuples form the single group. When we tried to run these queries in Postgres, the system gave up with a message that the maximum number of tuples allowed was exceeded. We estimated the times based on the rate of processing tuples for the 0.1G data set size as 62500 tuples/second. This results in an estimate of 4 hours for 1G data set and 15 days for the 10G data set. In both situations, all the tuples of relation `unaggregated` can be stored in memory, thus the large running time is due exclusively to computation inefficiency. In comparison, the optimized algorithm can compute the moments in only 10 seconds for the 10G data set. For truly large data sets, it is crucial to use the optimized algorithm to ensure the moments can be computed in reasonable time.

The unoptimized algorithm works faster only when the number of matching tuples is small and the number of joined relations is large, which is happening for Q6. There are only 23 matching tuples for 1G data set and 195 matching tuples for 10G data set for this query but 6 relations are involved. In this case, a large number of queries are posed to the database engine ($2^6 = 64$ queries) in order to run the optimized algorithm. It seems that the time is dominated by the parsing and query plan generation not by the execution itself. Indeed, when the number of matching tuples becomes 9 times bigger (1G vs 10G data sets), the execution time increases only 4%. In such a situation, the inefficiency can be removed by processing the `unaggregated` table in the client rather than using query rewriting and running the query on the server.

Total overhead While the algorithms we proposed, especially the optimized algorithm, might seem to require a lot of effort, the percentage overhead in the experimental result tables paints a different picture. Except for Q1 that is CPU bound, for all other the queries at least one of the versions of the algorithm require less than 10% overhead. The overhead seems to reduce for larger databases which means that for TPC-H type of queries the algorithms are expected to scale to very large databases. Finding matching tuples seems to be much more time consuming than computing moments form the `unaggregated` table.

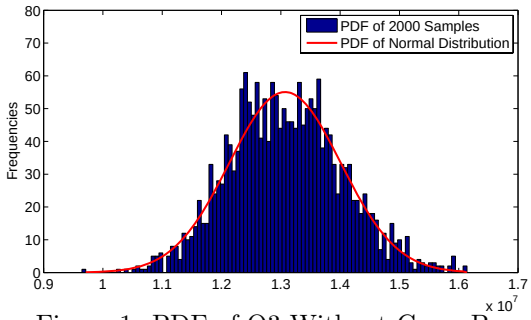

Figure 1: PDF of Q3 Without GroupBy

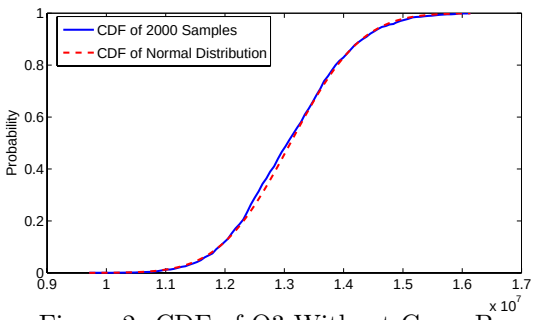

Figure 2: CDF of Q3 Without GroupBy

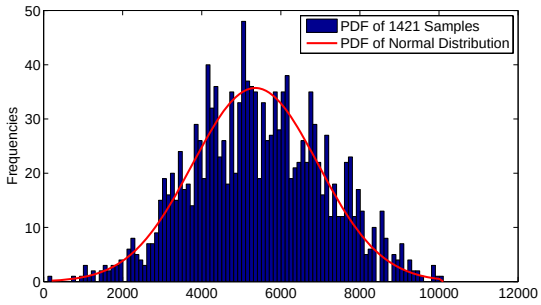

Figure 3: PDF of Q6

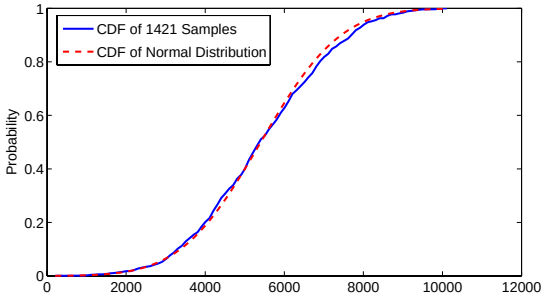

Figure 4: CDF of Q6

6.2 Distribution of Probabilistic Aggregates

In order to examine the distribution of probabilistic aggregates, we performed experiments on TPC-H Q3 without `GROUPBY` and TPC-H Q6 over 0.1G data set. For each query, we generated more than 1000 instances of the base relations according to the probabilistic model and we executed the original SQL query on these instances. The values of the aggregates thus obtained – these are i.i.d. samples from the distribution of the probabilistic aggregate – were used to estimate the PDF and CDF of the distribution.

Figures 1-4 depict the experimental results together with the approximation of these quantities based on approximating the distribution of the probabilistic aggregates with a normal distribution with mean and variance computed using our method. What is immediately apparent from these experiments is that the normal approximation is surprisingly good, especially when it comes to approximation of the CDF. This immediately means that reliable confidence intervals can be computed using the normal approximation – the quality of confidence intervals depends only on the quality of approximating the CDF.

The fact that the distribution tends to be normal is due to the fact that the aggregates are obtained by combining a large number of $0, 1$ random variables with the data, which has a normalizing effect similar to the Central Limit Theorem. We expect the distribution to diverge from the normal distribution when tuples with large contributions to the aggregate appear with small probabilities. In such circumstances, Chebychev bounds can be derived based on the moments, bounds that are correct but conservative.

7 Related Work

A lot of research has been published on probabilistic databases. Below we mostly survey and comment on work that is closest to the current contribution, with a particular emphasis on how our work differs.

In order to allow uncertain data in databases, [6, 5, 12, 11, 23, 14] modeled the uncertain data and extended the standard relational algebra to the probabilistic algebra. [8, 10, 9] further studied the complexity of queries over probabilistic databases and proved that computing the probability of a Boolean query on a disjoint-independent database is a $\#P$ problem. [9] also proved that evaluation of any conjunctive query is either $\#P-$complete or in PTime.

Aggregates over probabilistic databases – perceived as a much harder problem by the community – has attracted attention in recent years. [25] studied the aggregation over probabilistic databases. The focus in [25] is on probabilistic databases with attribute uncertainty and the probability of each attribute is in a bounded interval. [3, 27] developed the TRIO systems for managing uncertainty and lineage of data. Aggregation over TRIO systems is based on the possible worlds model and therefore operations are simple to implement but intractable for most situations. Only expectation seems to be implemented in TRIO (details are scant in the published literature), even though any other moments could be computed as easily (but inefficiently). It is worth mentioning that TRIO can also compute lower and upper deterministic bounds for aggregates but these bounds are likely to be very pessimistic – the probability that the lower or upper bound is achieved is extremely low in most situations.

[18, 19] and [7] studied aggregations over probabilistic data streams. The problem in [18, 19] is to estimate the expected value of various aggregates over a single probabilistic data-stream (or probabilistic relation). As part of this work, they had to derive the expected value formulas for one-relation case that we provide in Section 3. [7] studied the same problem together with the estimation of the size of the join of two relations. The analysis provided in these papers is significantly more restricted than ours: expectation and variance for one-relation case and just expectation for two-relation case. Furthermore, the aggregate is restricted to `COUNT` (the work is only concerned with frequency moments). It is important to note that the problem solved in all these three pieces of work is harder since the estimation has to be performed with small space (data-streaming problem). It would be interesting to investigate how the formulas we derive could be approximated using small space, as well.

Inspired by the same observation that the expected value of aggregations cannot capture the distribution clearly, [24] studied the problem of dealing with HAVING predicates that necessarily use aggregates. The basic problem they consider is: compute the probability that, for a given group, the aggregate α is in relationship θ with the constant k, i.e. $\alpha\theta k$. The types of aggregates considered are `MIN, MAX,COUNT,SUM` and the comparison operator θ is a comparison operator like $>$. Only integer constants k are supported since the operations are performed on the semiring S_{k+1}. The probabilities of events $\alpha < k$ are in fact the *cumulative distribution function*(c.d.f) of aggregate α at the point k. The efficient computation of such probabilities can be readily used to compute confidence intervals for α by essentially inverting the c.d.f. This can be accomplished efficiently using binary search since the c.d.f is monotone. Unfortunately, most of the results in [24] are negative. For most queries, computing exactly the probability of event $\alpha\theta k$ is in $\#P$. Even for the queries for which the computation is polynomial – this is the case for `MIN, MAX,COUNT,SUM(y)` but only for α-safe plans and y a single attribute – the complexity is linear in k, the constant involved. This is especially troublesome for `SUM` aggregates since k can be as large as the product of the size of the domain of the aggregate and the size of the group.

In view of the above comments on the difficulty of computing exact confidence intervals, a fundamental question needs to be asked: how is it possible to have these negative results but at the same time provide efficient algorithms for determining confidence intervals like we do in this paper? The most important observation about the present work is that only the first two moments are computed exactly, not the confidence intervals. The confidence intervals are either pessimistic, if the Chebychev bound is used, or based on extra information about the approximate distribution of the aggregate. The pessimism of the Chebychev bound results only in a small multiplicative constant for the size of the confidence interval – for 95% confidence intervals the constant is about 3. It is important to notice that, from the point of view of the user, the exact confidence interval is not that important; having some idea of the fluctuation of the expected value is the most useful piece of information. It is worth mentioning that the c.d.f. of most discrete distributions is hard to compute efficiently. Even for the Binomial distribution, special functions like the incomplete regularized beta function have to be used to compute the c.d.f. [1]. For this reason, the use of Chebychev bounds of the empirical approximation of the distribution of discrete random variables is standard in Statistics [26]. Users of statistical methods in natural sciences all understand and are comfortable with these *limitations*.

The proof techniques used in this paper were also used in previous work of the authors in a different context: analysis of sampling estimators [20, 21]. The $0, 1$ random variables significantly simplified the analysis of sampling estimators making possible a generic analysis independent of the type of sampling. The

Kronecker δ_{ij} symbol was used to keep under control formulas involving cases, as in the case in the current work. The analysis of sampling estimators in [21] was performed using the similar proof technique as in the current work. Lemma 1 in this paper is in fact a generalization of the formula in [21]. We believe similar techniques can be used for other problems related to probabilistic databases and approximate query processing.

8 Conclusions

In this paper we described a method to efficiently compute confidence intervals for non-correlated, distinct free, aggregates over probabilistic databases. The method requires simple query rewriting and the use of a regular database system to perform the required computations. The core of the method is statistical analysis of the expectation and variance of the aggregates as random variables. We derived both unoptimized, but more general, and optimized formulas for variance of the aggregates and indicated how these formulas can be computed using SQL aggregate queries. As our experimental results indicate, computing moments of probabilistic aggregates using the method we described and existing database technology has a small overhead even without any changes/optimizations to the database system. Moreover, as our experiments showed, the distribution of the probabilistic aggregates tends to be normal and well approximated by the normal distribution with the mean and variance computed using our method. This effectively means that the probabilistic aggregates of the type we considered can be well characterized statistically with small effort without the need to rewrite the database engine.

References

[1] http://mathworld.wolfram.com/.

[2] S. Abiteboul, P. C. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. In *SIGMOD Conference*, 1987.

[3] P. Agrawal, O. Benjelloun, A. D. Sarma, C. Hayworth, S. U. Nabar, T. Sugihara, and J. Widom. Trio: A system for data, uncertainty, and lineage. In *VLDB*, 2006.

[4] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *PODS*, 1999.

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

[6] R. Cavallo and M. Pittarelli. The theory of probabilistic databases. In *VLDB '87: Proceedings of the 13th International Conference on Very Large Data Bases*, pages 71–81, San Francisco, CA, USA, 1987.

[7] G. Cormode and M. N. Garofalakis. Sketching probabilistic data streams. In *SIGMOD Conference*, 2007.

[8] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, 2004.

[9] N. N. Dalvi and D. Suciu. The dichotomy of conjunctive queries on probabilistic structures. In *PODS*, 2007.

[10] N. N. Dalvi and D. Suciu. Management of probabilistic data: foundations and challenges. In *PODS*, 2007.

[11] D. Dey and S. Sarkar. A probabilistic relational model and algebra. *ACM Trans. Database Syst.*, 21(3):339–369, September 1996.

[12] N. Fuhr and T. Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst.*, 15(1):32–66, January 1997.

[13] G. Grahne. Dependency satisfaction in databases with incomplete information. In *VLDB*, 1984.

[14] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, 2007.

[15] T. J. Green and V. Tannen. Models for incomplete and probabilistic information. In *EDBT Workshops*, 2006.

[16] T. Imielinski and W. L. Jr. Incomplete information in relational databases. *J. ACM*, 31(4), 1984.

[17] T. Imielinski, S. A. Naqvi, and K. V. Vadaparty. Incomplete objects - a data model for design and planning applications. In *SIGMOD Conference*, 1991.

[18] T. S. Jayram, S. Kale, and E. Vee. Efficient aggregation algorithms for probabilistic data. In *SODA*, 2007.

[19] T. S. Jayram, A. McGregor, S. Muthukrishnan, and E. Vee. Estimating statistical aggregates on probabilistic data streams. In *PODS*, 2007.

[20] C. Jermaine, A. Dobra, S. Arumugam, S. Joshi, and A. Pol. The sort-merge-shrink join. *ACM Trans. Database Syst.*, 31(4):1382–1416, December 2006.

[21] C. M. Jermaine, S. Arumugam, A. Pol, and A. Dobra. Scalable approximate query processing with the dbo engine. In *SIGMOD Conference*, 2007.

[22] L. Libkin and L. Wong. Semantic representations and query languages for or-sets. In *PODS*, 1993.

[23] M. Pittarelli. An algebra for probabilistic databases. *IEEE Trans. Knowl. Data Eng.*, 6(2):293–303, April 1994.

[24] C. Ré and D. Suciu. Efficient evaluation of having queries on a probabilistic database. In *DBPL*, 2007.

[25] R. B. Ross, V. S. Subrahmanian, and J. Grant. Aggregate operators in probabilistic databases. *J. ACM*, 52(1):54–101, January 2005.

[26] J. Shao. *Mathematical Statistics*. Springer-Verlag, 1999.

[27] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, 2005.