ALL_SUM Query Evaluation over unpredictable data

Roslinmary M\textsuperscript{1}, Saravankaumar T\textsuperscript{2} and Addlinshinney R\textsuperscript{3}

\textsuperscript{1}Information Technology, Dr. Sivanthi Aditanar College of Engineering, Tiruchendur, 628215, India
\textsuperscript{2}Information Technology, Dr. Sivanthi Aditanar College of Engineering, Tiruchendur, 628215, India
\textsuperscript{3}Information Technology, Dr. Sivanthi Aditanar College of Engineering, Tiruchendur, 628215, India

Abstract

One of the important Queries in many real-time applications is SUM query, it deals with unpredictable data. In this paper, dealing with the query, called ALL_SUM Query. In general, the SUM query returns only the sum of the values. But the ALL_SUM query returns all possible sum values together with their probabilities. There is no efficient solution for the problem of evaluating ALL_SUM queries used in many applications where the aggregate attribute values are real with small precision. In this paper, evaluating a pseudo-polynomial algorithm called AgrQSUM algorithm which is based on a recursive approach, it efficiently calculate ALL_SUM Query. The proposed AgrQSUM algorithm returns an efficient solution for determining the exact result of ALL_SUM queries. The results of an experimental evaluation over synthetic and real-world data sets show its effectiveness.

Keywords: query processing, Database management systems

1. Introduction

Uncertain Database is “Membership of an item to the database” is a probabilistic event. The value of attributes is a probabilistic variable. Uncertain data streams are important in growing number of environments, such as traditional sensor networks, GPS system for locationing, RFID networks for object tracking, radar networks for severe weather monitoring and the telescope surveys for astrophysical pattern. Aggregate query on those applications is very important.

For Example, E-health Management System. Consider a medical center that monitors key biological parameters of remote patients at their homes, using sensors in their bodies. The sensors periodically send to the center the patients' health data, e.g. blood pressure, hydration levels, thermal signals, etc. For high availability, there are two or more sensors for each biological parameter. However, the data sent by sensors may be uncertain, and the sensors that monitor the same parameter may send inconsistent values.

There are approaches to estimate a confidence value for the data sent by each sensor, e.g. based on their precision. According to the data sent by the sensors, the medical application computes the number of required human resources, e.g. nurses, and equipments for each patient. One important query in this application is “return the sum of required nurses”. Fig. 1 shows an example table of this application. The table shows the number of required nurses for each patient.

<table>
<thead>
<tr>
<th>Patient</th>
<th>Sensor</th>
<th>Blood Pressure</th>
<th>Required Human Resources</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>S1,1</td>
<td>16</td>
<td>3</td>
<td>0.5</td>
</tr>
<tr>
<td>P2</td>
<td>S1,2</td>
<td>13</td>
<td>1</td>
<td>0.4</td>
</tr>
<tr>
<td>P3</td>
<td>S2,1</td>
<td>12</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

The table shows the possible worlds, i.e., the possible database instances, their probabilities, and the result of the SUM query in each world. In this example, there are eight possible worlds and four possible sum values, i.e., 0 to 3.

A Q_PSUM algorithm for evaluating ALL_SUM queries is to return all possible worlds, i.e., all possible database instances, compute sum in each world, and return the possible sum values and their probability. However, response time of these algorithms is high compared to our approach.
In this demonstration, we present a probabilistic database system for managing uncertain data. In particular, we show the efficiency of processing aggregate queries such as \texttt{ALL\_SUM}. Our demonstration application is the E-health Management system application described in above Example.

Table 1. The possible worlds and the results of \texttt{SUM} query in each instances. The rest of the paper is organized as follows. In Section 2, we present some technical basis e.g. the probabilistic data models and some intuitions about our algorithms. In Section 3, we describe our prototype. In Section 4 we present performance evaluation of my paper. Section 5 concludes.

### 2. Technical Basis

In this section we introduce the probabilistic data models we consider. Main objective for using Probabilistic database is to extend data management tools to handle probabilistic data. Then, we define the problem that we address.

#### 2.1 Probabilistic Models

we first introduce the two probabilistic data models that most frequently used in our community.

**Tuple-level model:**

All attributes in a Tuple are known precisely, existence of the Tuple is uncertain. In this model, each uncertain table \( T \) has an attribute that indicates the membership probability (also called existence probability) of each Tuple in \( T \), i.e., the probability that the Tuple appears in a possible world. In this paper, the membership probability of a Tuple \( t_i \) is denoted by \( p(t_i) \). Thus, the probability that \( t_i \) does not appear in a random possible world is \( 1 - p(t_i) \). The database shown in Table 2 is under Tuple-level model.

**Attribute-level model:**

Tuples (identified by keys) exist for certain; an attribute value is however uncertain for example Tomorrow temperature will be somewhere between 50F and 70F.

#### 2.2 Problem Definition

\textbf{ALL\_SUM query:}

It returns all possible sum results together with their probability. Our objective of our paper is to return the result of sum as follows.

**Table 2 Tuple level model**

<table>
<thead>
<tr>
<th>State</th>
<th>Event</th>
<th>( p_S )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p )</td>
<td>( e )</td>
<td>0.4</td>
</tr>
<tr>
<td>( p )</td>
<td>( f )</td>
<td>0.6</td>
</tr>
<tr>
<td>( Q )</td>
<td>( e )</td>
<td>0.5</td>
</tr>
<tr>
<td>( Q )</td>
<td>( f )</td>
<td>0.4</td>
</tr>
</tbody>
</table>

For example, in the above table, the probability that neither of the two tuples \((p,e)\) and \((p,f)\) exists in the database is given by \((1 - 0.4) \times (1 - 0.6) = 0.24\).

**Table 3. Attribute level model**

<table>
<thead>
<tr>
<th>Time</th>
<th>( \text{temp low} )</th>
<th>( \text{temp high} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>2</td>
<td>22</td>
<td>23</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>19</td>
</tr>
</tbody>
</table>

For example, in the above table Tuples exist with certainty. Temperature at time t1 at location 1 etc. But the attribute values (temperatures) are uncertain.

2.2 Problem Definition

\textbf{ALL\_SUM Query:}

It returns all possible sum results together with their probability.
ALL_SUM(Q,D)={p(v,D)\mid v \in V_{D,i}\land p=p(v,Q,D)}

\[ p(v,Q,D) = \sum_{w \in W_{D,\text{and}(w)=v}} p(w) \]

Where D be a uncertain database, W the set of its possible worlds, and P (w) the probability of a possible world w ∈ W. Let Q be a given aggregate query, f the aggregate function stated in Q (i.e., SUM), f (w) the result of executing Q in a world w ∈ W, and \( V_D \) the set of all possible results of executing Q over D. The cumulative probability of having a value v is the result of Q over D, denoted as p(v; Q, D).

2.3 Algorithmic Basis

In Probabilistic database, almost all aggregate functions are processed using recursive algorithms. Below, we discuss the ALLcSUM algorithm that is in charge of executing SUM queries.

Let \( t_1 \ldots t_m \) be the tuples of the given uncertain database which is under the Tuple-level model. Let DB1 be a database involving the tuples \( t_1 \ldots t_m \), and W be the set of all possible worlds in DB1. Let ps(i, j) be the probability of having \( \text{sum} = i \) in DB1. We develop a recursive approach for computing ps (i, j).

2.3.1 Base

Let us consider DB1, i.e., the database that involves only the Tuple1. Let \( p(t_1) \) be the membership probability of \( t_1 \), and \( \text{val}(t_1) \) be the agg value of \( t_1 \). In DB1, there are two Possible worlds: 1) \( w_1 = \emptyset \), in which \( t_1 \) does not exist, so its probability is \( (1 - p(t_1)) \); 2) \( w_2 = \{t_1\} \), in which \( t_1 \) exists, so the probability is \( p(t_1) \). In \( w_1 \), we have \( \text{sum} = 0 \), and in \( w_2 \) we have \( \text{sum} = \text{val}(t_1) \). If \( \text{Val}(t_1) = 0 \), then always we have \( \text{sum} = 0 \) because in both \( w_1 \) and \( w_2 \) sum is zero.

2.3.2 Recursion Step

Now consider DBn, i.e. a database involving the tuples \( t_1 \ldots t_n \). Let Wn be the set of possible worlds for DBn, i.e. set of possible instances for DBn. Let ps(i, n-1) be the probability of having \( \text{sum} = i \) in DBn-1, i.e. the aggregated probability of the DBn-1 worlds in which we have \( \text{sum} = i \). Now, we construct DBn by adding \( t_n \) to DBn-1. Notice that the set of DBn possible worlds, denoted by Wn, are constructed by adding or not adding the Tuple1 to each world of Wn-1. Thus, in Wn, there are two types of worlds: 1) the worlds that do not contain \( t_n \), denoted as Wn1; 2) the worlds that contain \( t_n \), denoted as Wn2. For each world \( w \in W_n \), we have the same world in DBn-1, say \( w' \). Let \( p(w) \) and \( p(w') \) be the probability of worlds \( w \) and \( w' \). The probability of \( w \), i.e. \( p(w) \), is equal to \( p(w') \times (1 - p(t_n)) \), because \( t_n \) does not exist in \( w \) even though it is involved in the database. Thus, in Wn the sum values are the same as in DBn-1, but the probability of \( \text{sum} = i \) in Wn1 is equal to the probability of having \( \text{sum} = i \) in DBn-1 multiplied by the probability of non-existence of \( t_n \). In other words, we have: In Wn1:

\[ \text{(Probability of sum} = i) = \text{ps}(i, n-1) \times (1 - p(t_n)) \] (1)

Let us now consider Wn2. The worlds involved in Wn2 are constructed by adding \( t_n \) to each world of DBn-1. Thus, for each sum value equal to \( i \) in DBn-1 we have a sum value equal to \( i + \text{val}(t_n) \) in Wn2, where \( \text{val}(t_n) \) is the agg value of \( t_n \). The probability of \( \text{sum} = i \) in Wn2 is equal to the probability of \( \text{sum} = i \) in DBn multiplied by the membership probability of \( t_n \). In other words, we have: In Wn2:

\[ \text{(Probability of sum} = i) = \text{ps}(i - \text{val}(t_n), n-1) \times p(t_n) \] (2)

Let \( ps(i, n) \) be the probability of \( \text{sum} = i \) in DBn. This probability is equal to the probability of \( \text{sum} = i \) in Wn, plus the probability of \( \text{sum} = i \) in Wn2. Thus, by using the Equations 1 and 2, and using the base of the recursion, we obtain the following recursive definition for the probability of \( \text{sum} = i \) in DBn, i.e. \( ps(i, n) \):

\[ ps(i, n) = \begin{cases} 
\text{ps}(i, n-1) \times (1 - p(t_n)) + \text{ps}(i - \text{val}(t_n), n-1) \times p(t_n) & \text{if } n > 1 \\
1 - p(t_n) & \text{if } n = 1 \land i = 0 \land \text{val}(t_1) \neq 0 \\
p(t_1) & \text{if } n = 1 \land i = \text{val}(t_1) \land \text{val}(t_1) \neq 0 \\
1 & \text{if } n = 1 \land i = \text{val}(t_1) = 0 \\
0 & \text{otherwise}
\end{cases} \]

Based on the above recursive definition, we developed efficient algorithms for processing SUMQuery.

2.3.3 AgrQSUM Algorithm

In this section, using the dynamic programming technique, we propose an efficient algorithm, called AgrQSUM, designed for the applications where agg values are integer or real numbers with small precisions. It is usually more efficient than the Q_PSUM algorithm.

AgrQSUM processed in two steps. In the first step, it initializes the first column of the matrix. This column represents the probability of sum values for a
database involving only the Tuple t1. AgrQSUM initializes this column using the base of our recursive formula.

In the second step, in a loop, AgrQSUM sets the values of each column j (for j = 2 to n) by using our recursive definition and based on the values in column j – 1 as follows:

\[
PS[i; j] = PS[i, j - 1] \times (1 - p(tj)) + PS[i - val(tj), j - 1] \times p(tj).
\]

Notice that if \(i < val(tj)\), then for the positive aggr values we have \(PS[i - val(tj), j - 1] = 0\), i.e., because there is no possible sum value lower than zero. This is why, in the algorithm only if \(i \geq val(tj)\), we consider \(PS[i - val(tj), j - 1] \times p(tj)\) for computing \(PS[i; j]\).

Execution of DP-SUM over the database works correctly if the database is under Tuple-level model and the aggr attribute are positive integers, and their sum is less than or equal to MaxSum.

Pseudo code of AgrQSUM Algorithm

1. Let MaxSum = \([n \times \text{avg}]\)
2. Let \(PS[\text{MaxSum} + 1, n] : be\) a two Dimensional matrix
3. Step 1: initializing the first row of the matrix
   - \(PS[\text{val}(t1), 1] = p(t1)\)
   - \(PS[0, 1] = (1 - p(t1))\)
   - \(PS[i, 1] = 0\) for \(i \neq 0\) and \(i \neq \text{val}(t1)\)
4. Step 2: compute other rows of the matrix
   - For \(j = 2\) to \(n\) do
     - For \(i = 0\) to \(\text{MaxSum}\) do
     - \(PS[i, j] = PS[i, j - 1] \times (1 - p(tj)) + PS[i - \text{val}(tj), j - 1] \times p(tj)\)

3. Prototype

ProbDB is built on top of a classical Database Management System (DBMS). It adds probabilistic capabilities to the DBMS that are transparent to the user. Instead of directly modifying the DBMS and adding "native" primitives to it, we have chosen to implement ProbDB on top of the DBMS, and thus to be able to change the underlying DBMS with as little programming effort. In its current version, the prototype is built atop PostgreSQL, but could easily be adapted to work on a MySQL database for instance.

When the user sends a query to ProbDB, the query is analyzed and probabilistic keywords are extracted. Then classical (non probabilistic) sub-queries are sent to the DBMS that processes them and returns intermediate results. Then, probabilistic functions are applied to the intermediate results, and the final results are returned to the user. ProbDB is composed of the following components (see the architecture):
Relational DBMS: this is an ordinary (deterministic) relational database management system that, given the Reformulated $Q_1$, executes it over the probabilistic tables, and returns the result to the component that evaluates the Probabilistic parts of the query.

Probabilistic Evaluator: The inputs of this component are the intermediate data generated by the DBMS and the query $Q_2$. According to the probabilistic expressions in $Q_2$, the component chooses the appropriate algorithms and runs them over the intermediate results, and returns the final results to the user.

4. Performance Evaluation

Performance of ALL_SUM Query Evaluated over real-world as well as synthetic datasets.

Based on Uncertain Tuple

Based on the response time of the algorithms vs. the number of uncertain tuples, i.e., $n$, the best algorithm is AgrQSUM, compared to Q_PSUM algorithm. The response time of AgrQSUM is at least four times lower than that of Q_PSUM.

Overall, the problem which we considered in this report, i.e., returning the exact results of ALL_SUM queries, there is no efficient solution in the related work. In this report, we proposed AgrQSUM algorithms that allow us to efficiently evaluate ALL_SUM queries in many practical cases, e.g., where the aggregate attribute values are small integers, or real numbers with limited precisions.

5. Conclusions

One of the important Queries in many real-time applications is SUM query, it deals with unpredictable data. In this paper, dealing with the query, called ALL_SUM Query. There is no efficient solution for the problem of evaluating ALL_SUM queries used in many applications where the aggregate attribute values are real with small precision. In this paper, evaluating a pseudo-polynomial algorithm called AgrQSUM algorithm which is based on a recursive approach, it efficiently calculate ALL_SUM Query. It returns exact result of ALL_SUM queries. The results of an experimental evaluation over synthetic and real-world data sets show its effectiveness of our solution. The performance of AgrQSUM is better than Q_PSUM.

References


