Top-k Queries in Uncertain OLAP

CMSC 724 Project

Bao Nguyen

May 20, 2008

Abstract

Top-k is the most important query type in OLAP, the most common data warehouses model. Recently, OLAP was extended to represent data ambiguity, specifically imprecision and uncertainty. However, how to deal with top-k queries in uncertain OLAP still remain an unanswered question. This project introduces a complete solution for top-k query in uncertain OLAP including query semantic definition, query processing algorithm and data materialization. An experimental evaluation showed that the proposed solution is feasible.

Keywords: data warehouse, OLAP, top-k, query processing

1 Introduction

OLAP is a database model widely used in industry. Recently, this model has been extended to work with uncertain and imprecise data [3, 12, 13]. In OLAP, top-k query is the most dominated type of query. Although the semantics of top-k queries for uncertain tuples were discussed by several authors [21], how to deal with top-k queries in uncertain OLAP is still unexplored. In this project, I propose to work on the problem of answering top-k queries in uncertain OLAP. The contributions of this project are summarized as follows:

- An extension of top-k query semantics for uncertain OLAP.
- An algorithm to effectively process top-k queries under proposed semantics.
- An method to materialize data to speed up searching.
- An experimental evaluation to evaluate the proposed techniques.

In the next section, I provide a brief literature survey on OLAP and probabilistic database to see the context of the work proposed in this project. I then present my proposed the semantics for top-k query in uncertain OLAP (section 3), an algorithm to effectively process top-k queries (section 4) and a method to materialize data (section 5). An experimental evaluation are presented to show the effectiveness of proposed techniques (section 6). I finish with some conclusions and directions for future work.
2 Literature survey

2.1 OLAP

OLAP is an acronym standing for Online Analytical Processing\cite{19} which describes a class of techniques allowing data to be summarized and viewed in different views in an online fashion. These multidimensional views are supported by multidimensional database technology. Querying in OLAP is fast and easy because data are pre-analyzed in advance and the results are stored as a part of the database. This contrasts to traditional data analysis techniques called OLTP (Online Transaction Processing), in which the aggregations are calculated when queries are invoked (i.e. managed by transactions).

The data in OLAP is typically organized in a multidimensional format called data cubes\footnote{Here “cube” is just used as an abstraction, it doesn’t necessarily refer to a three dimension space}\cite{5}. The data cubes are organized by the star schema \cite{11}, where some measures are analyzed with respect to some interesting dimensions, representing business perspectives. Measures are derived from the records in the fact table and dimensions are derived from the dimension tables. For example, a shoes company may have a sale database with \{number of items\} as a measure attribute and \{color, year, type\} as three dimensional attributes. The OLAP model defines some typical operations including roll-up (increasing the level of aggregation) and drill-down (decreasing the level of aggregation or increasing detail) along one or more dimension hierarchies, slice and dice (selection and projection), and pivot (re-orienting the multidimensional view of data). Those operators help users quickly and easily retrieve the data in their dimensional range of interest. For more details about OLAP, readers are referred to \cite{5}.

2.1.1 Top-k query in OLAP

A common problem in OLAP is the top-k query execution. A top-k query returns k data points satisfying some optimal functions on dimensional attributes which is specified by the users. In the example described above, the user may wish to find the top 3 sales made since 1995 by the number of items.

A straightforward method to evaluate a top-k OLAP query is to compute the aggregate value for each dimension and then select the cubes with the highest aggregates. However, it will be extremely slow as the number of possible cubes in an OLAP may be very large. There are several approaches have been proposed in literature to solve this problem. One of them is to build a spatial index tree to store information which is pre-computed in advance. Papadias et al. \cite{16} use an aggregate R-tree (i.e. aR-Tree) to store aggregate results. The structure of aR-Tree defines the spatial hierarchy of data cubes. Using this structure, Kalnis et al.\cite{15} propose a branch-and-bound algorithm that accesses a minimal number of tree nodes in order to compute the top-k groups. A variant of using spatial index tree is to use RD-Tree \cite{7}, an index structure for sets, instead of R-Tree to model the data hierarchy.

Another approach to efficiently execute top-k query in OLAP, as proposed by Loh et al.\cite{14}, is to partition data cubes into sufficiently small cells and the top-k aggregate values in each partition are pre-computed. These values can then be used to compute the top-k results in query regions that
cover multiple cells. The partition factors and the number of pre-stored values can be estimated through statistical analysis.

2.1.2 OLAP over uncertain and imprecise data

Recently, the traditional OLAP model has been extended to deal with uncertain and imprecise data [3, 12, 13]. In this project, I focus on the lasted model proposed by Doug et al. [3]. In their model, the assumption that all facts must be points is relaxed to facts can be regions (i.e. more ambiguous). The authors formalize the ambiguous OLAP by transferring several concepts in the traditional OLAP to ambiguous versions which are uncertain domain, imprecise domain, hierarchical domain, fact table schemas and instances, regions and cells. A process called allocation is proposed to transform a ambiguous OLAP database into a form, called the Extended Database, that can be used to answer aggregation queries. This can be done by a scalable and efficient algorithm proposed later [2]. In their latest work, they augment their model with a domain constraint language to present correlations between attributes [4].

2.2 Probabilistic Databases

The existing database systems are working with exact “facts” that can be considered certain. However, there are more and more real-world applications today producing large amount of uncertain data[18]. This has led to the development of a new database model that can handle those uncertainty in databases - the probabilistic databases. The database systems are no longer modeled by classical logic but by probabilities to represent the uncertainty and by laws of probability to process the data [9].

There are two levels of uncertainty in probabilistic databases: tuple level and attribute level. In tuple level, all attributes are exactly identified whereas the existence of the tuples is uncertain. In attribute level, on the other hand, the existence of all tuples are certain where as the values of attributes are vague. The uncertain OLAP model mentioned in section 2.1.2 belongs to the attribute level uncertainty class.

2.2.1 Challenges in managing probabilistic data

As probabilistic data is a fairly new research area there are a lot challenges and open problems in the field. The two main problem are (1) how to represent data to take the trade off between expressibility and complexity (2) how to efficiently process query using such data representations.

Data representation In its most general form, a probabilistic database is a probability space over all possible instances of the database, called possible worlds. A tuple is now associated with a possibility of existence in the database. This led to the explosion of possible worlds. A lot of research in probabilistic database now is involving how to choose the most relevant possible worlds. One approach is extracting the correlations between tuples in database and the bring them into query process. Sen et al. [18] propose a method using graphic models to capture such correlations. Another approach is to store extra information such as lineage/provenance [22] along with the data to capture the tuple correlations supporting for its uncertainty. This information may be assigned to data in advance or determined during the data lifetime. Also, from the attribute level uncertainty view point, Cheng et al [6] propose a method to associate each attribute with probability
distributions. However, as of now, there is no consensus on how data should be represented in probabilistic database.

**Query processing** Since in probabilistic database, each tuple is associated with a probability, the answer SQL query is no longer a single answer but rather a set of probabilistic answers. Query processing now involves in selecting the best answers and this problem is an instance of the *probabilistic inference* problem, which has been known to be notoriously hard [1]. To process large scale probabilistic data we need to develop specific probabilistic inference techniques that can be integrated well with SQL query processors and optimizers, and that scale to large volumes of data. Fuhr and Rolleke propose using tuple-level probabilities to present the intensional and extensional query evaluation techniques. Dalvi and Suciu define safe query plans to be those for which extensional and intensional query evaluation produces identical results. Lakshmanan et al attempt to combine these different approaches by associating probability intervals with tuples2.

Another open issue is how to generate *query answers*. In many scenarios, when the input data is uncertain or imprecise, the approximations are sufficient. In this case, an approximate strategy should be used to speed up the query processing. For example, a mapping to data sources or some form of meta data [10] should be maintained to increase in the future processing in large scale database.

### 2.2.2 Top-k query processing in probabilistic databases

Top-k processing in uncertain databases is semantically and computationally different from traditional top-k processing. In the possible world, we have to deal with two variants: *query score* and *uncertainty* at the same time. It makes traditional techniques hard to be applied. The main challenge for top-k query processing in probabilistic database is how to prune un-related returned tuples in query plan. To my awareness, there is still a limit number of work on this area.

In [21], Soliman et at., assuming that there are scoring functions associated with every tuples, propose a measurement for tuple ranking. The semantics of top-k query is represented by two values:

- $U - Topk$: return the most probable top-k answer set that belongs to same possible world. This measure is the combination of possible world and traditional top-k semantics and fits in the scenarios where we need all top-k tuples in the same world.

- $U - kRanks$: return the most probable $ith - ranked$ tuples ($i = 1, 2, \ldots, k$) across all possible worlds. This measure doesn’t care about possible worlds and fits in the scenarios where we don’t need to restrict returned tuples in the same world. A more detailed explanation of $U - kRank$ can be found in [20].

Another approach to evaluate top-k query in uncertain database based on the probability associated with each query answer. In [17], Re et al. run Monte-Carlo simulations for each candidate answer. The top-k tuples are extracted based on the probabilities to satisfy query in all possible worlds.

---

2Those examples are directly taken from the related work section of [18].
3 The Semantic of Top-k Queries

To process top-k queries in uncertain OLAP, we first have to understand the semantic of top-k queries in an uncertain context. Since in uncertain OLAP, there is the intervention of three concepts “most probable”, “top-k” and “summarization”, we have several ways to interpret the meaning of a top-k query. Just basing on the classification of top-k queries semantics for tuples proposed by Soliman et al [21], the top-k queries semantics for OLAP may be interpreted in three different ways:

1. The “top k” query regions (wrt an aggregation) in the “most possible” world
2. The “top k” query regions over all possible worlds
3. The set of “most probable top-i-th” query regions over all possible worlds

The first and the third semantics are quite straightforward since they are similar to the semantics of top-k queries for probabilistic tuples proposed in previous work [21]. However, for the second semantic, since we may either summarize or rank the aggregations first, there are two possible ways to understand the semantic of this options:

2.1 Summarize the uncertain OLAP by calculating the expected aggregate values for all regions first (i.e. the semantic proposed by Burdick [3]) and then take the top-k results
2.2 Summarize aggregate values for regions in each individual possible world first and then take the most possible top-k query regions.

The semantic 2.1 is, on the surface, similar to the semantic of top-k queries in certain OLAP. However, using this semantic, duplicate tuples that refer to the same entity can be counted in several cells at the same time. In the real world, those tuples should be considered mutually exclusive. Thus this semantic doesn’t quite make sense. Especially in the ranking situation, if there is an uncertain tuple having a very large aggregate value, it is most probably that all regions covered by that tuple will be selected as the final top-k answer. For that reason, I propose just to use the semantic 2.2 to represent the semantic 2: “top k” query regions over all possible worlds. Formally, the semantic 2.2 is defined as follow:

**Definition 1.** Let \( O \) be an uncertain OLAP with a set of possible worlds \( PW = \{PW^1, \ldots, PW^n\} \). Let \( R = \{R^1, \ldots, R^m\} \) be a set of k-length region vector where (1) regions in each \( R^i \) are ordered according to a scoring function \( F \) on aggregated values, and (2) \( R^i \) is the topk answer for a non empty set of possible worlds \( PW(R^i) \) in \( PW \). The top-k answer for \( O \) is \( \text{argmax}_{R^i \in R} (\sum_{w \in PW(R^i)} (w)) \)

4 Processing Top-k Queries

The top-k query processing on semantics 1 is similar to that on certain OLAP since we just have to summarize and rank in only one unique world. There are several solutions for this problem such as using R-tree, aR-tree to pre-computed and store some aggregated values in advance [15]. Those values are later used as bounds for branch pruning in the query region search tree.

---

3 Basically, a query region is a set of cubes or cells in uncertain OLAP forming a certain shape. A more formal definition of this concept can be found in the original uncertain OLAP paper [3]
For the semantics 2 and 3, the task is much more harder since we have to consider all possible worlds. A naive approach is to brute force search over all possible worlds and take the most probable answer. However, as the number of possible worlds is exponentially increase with the number of uncertain tuples, it is infeasible to do so. In this section, I propose an effective algorithm to search for top-k answers. Moreover, as the problem for semantics 2 and 3 is essentially similar (although it is a bit harder for semantic 3), I just consider the top-k query processing for semantic 2. The detailed algorithm for semantic 3 is left for future work.

4.1 The algorithm scope

Processing top-k queries in uncertain OLAP in general is a hard problem. In this project, I am just interested in this task with several assumptions:

- The query type I am particularly interested in here is cells searching query. In general, the query regions needed to summarize can be in arbitrary and complex shapes. However, as the number of possible shapes in an OLAP space is exponential, trying to find answers in such a general case is extremely hard. I put this general case outside the scope of this project.

- The aggregate functions considered here are monotonic functions (e.g. SUM, COUNT, etc). This limitation means when you add more tuples in one region, the aggregation will be changed monotonically (either increase or decrease). It will let us use a branch and bound strategy to prune the search tree (described in section 4.2).

- I assume that all attributes and entities are independent. In a recent work, Burdick augmented their model with some domain constraints to capture the attribute dependencies [4]. In another work, Sen et.al. proposed to use a graphical model to represent the tuple correlations [18]. It is reasonable to account for those constraints since the entities falling in one cell may strongly depend on each other. However, it will largely complicate the problem I am trying to solve in this project. For that reason, I just keep the tuples from one entity are mutually exclusive and assume all the entities are independent.

4.2 Searching top-k answers

In this section, I propose an algorithm to process top-k query. For a clearer explanation, I first define some terminologies for uncertain OLAP as follow:

- A dataset $D$ is a set of tuples either certain or uncertain.

- A dataset $D'$ is called a descendant of a dataset $D$ if we can get $D$ by specifying some tuples in $D$. Also $D$ is called an ancestor of $D'$

- A region is a set of cells in the OLAP. A certain region is a region that don’t have any uncertain tuple falling in it and an uncertain region is a region which is not certain.

- A cell $c$ is called certain with regard to a dataset $D$ if there’s no uncertain tuples in $D$ falling in $c$. Also, a cell $c$ is called uncertain w.r.t a dataset $D$ if it is not certain w.r.t $D$.

- The upper bound of a cell $c$ in dataset $D$ with regard to an aggregate function $F$ is $F(t_1, t_2, ..., t_n)$ where $t_i (1 \leq i \leq n)$ are all tuples in $D$ may fall in $c$ (i.e. with a probability greater than 0).
Here, we treat all $t_i$ as certain tuples. Due to the monotonic property of $F$, the upper bound always greater than the true value of $c$ in all possible worlds generated from $D$. Also, if $D'$ is a descendant of $D$ then $ub(c, D') \leq ub(c, D)$. Since the aggregate function is unchanged for all $D$, for simplicity, I denote this upper bound as $ub(c, D)$. This upper bound is also called an abstract aggregation of cell $c$ with regard to a dataset $D$.

Normally, for each dataset $D$ there are always two regions in the corresponding OLAP: certain region and uncertain region. Searching in an uncertain region always much harder than searching in a certain one. The key idea of my approach is to put the priority of searching on certain region first. If all cells needed (i.e. top k cells) can be found in certain region then we don’t have to look into the uncertain one. Otherwise, if no cell or just a part of the answer are in certain region, we will dig deeper to the uncertain region to make a decision. The evaluation of cells here is done by using the real aggregations for certain cells and the abstract aggregation for uncertain cells. This idea is inspired by the of zoom-able interfaces [8], a kind of computer-user interface, developed by HCIL group in UMD. Often, we work on an abstraction of all items and we only zoom in when there is something in doubt and we need to investigate in more details. An interesting point here is when zooming in, we just focus on the in-doubt part and ignore other areas of the interface.

The algorithm 4.2 explains the details of my approach. The granularity of the OLAP are reflected by the granularity of the dataset $D$. $D_{\text{certain}}$ is the certain part of $D$ and $Top_k_{-\text{certain}}$ is a set of top-k cells in $D_{\text{certain}}$. $Top_k_{-\text{certain}}$ will later be used to compete with the cells found in uncertain part. To make a dataset $D$ less uncertain, we select the highest uncertain cell $c_j$ (i.e. the most suspect cell) and splitting all uncertain tuples may fall in it. By this way, in the descendants $D'$ of $D$, there is at least one more cell ($c_j$) become certain and the chance to find the answer becomes higher.

**Algorithm 1** $Top (\text{Rank} \ k, \text{DataSet} \ D) \rightarrow (\{c_1, ..., c_k\}; \text{prob})$

1. Find top k cells in $D$ ranked by upper bounds $\rightarrow \{ub(c_1, D) \geq ... \geq ub(c_k, D)\}$
   - if $\forall i \in \{1, ..., k\}$, $c_i$ is certain in $D$ then
     - $Top \leftarrow (\{c_1, ..., c_k\}; 1)$
   - else
     - Assume $j$ is the min index such that $c_j$ is uncertain and $\forall i < j$, $c_i$ is certain
     - $D_{\text{certain}} \leftarrow$ The certain part of $D$
     - $Top_k_{-\text{certain}} \leftarrow Top(k, D_{\text{certain}})$
     - $T \leftarrow$ Set of all tuples may fall in $c_j$
     - $C \leftarrow$ Set of all cells may affected by $T$
     - $PT \leftarrow$ All possible tuple sets from $T$
     - for all $ts \in PT$
       - $D_{T'} \leftarrow D \cup ts \setminus (D_{\text{certain}} \cup T)$
       - $Top(k - j, D_{T'}) \rightarrow (Top(k - j)_{T'}, \text{prob}_{T'})$
       - $Top(k, Top_k_{-\text{certain}} \cup Top(k - j)_{T'}) \rightarrow Top_{kT'}$
     - end for
     - $Top_k \leftarrow$ Most possible $Top_{kT'}$
     - $\text{prob} \leftarrow \sum_{Top_{kT'} = Top} \text{prob}_{T'}$
   - end if
5 Data Materialization

Since the cost for calculating cell upper bounders is quite expensive, I propose to materialize them and store in disk. Figure 1 shows an example of a materialize table. This table is similar to an OLAP table except that the aggregate values are replaced by the upper bounds of them. To support searching cell in different levels, I propose store each cell in different levels of dataset granularity. More specifically, I add 3 more columns Level, PW_{ID} and Prob in the materialize table as indicated in figure 1. When processing top-k queries, instead of calculating the cell abstract aggregation, we may refer to this table (at different levels) to save time and effort.

6 Experiments

This section present the experiments to evaluate my propose techniques to deal with uncertain OLAP. More specifically, I interested in answering following questions:

1. What is the characteristics of probabilistic databases?
2. Can proposed algorithm effectively process top-k queries? And what is the relations between query characteristics and query processing effort?
3. What is the relations between uncertain level and query characteristics?
4. What is the cost of using materialize data?

6.1 Experiment Setup

Although there are several existing OLAP API such as OLAP4J or Oracle OLAP but it is not easy to tune their source code to make an uncertain OLAP version. For that reason, I chose to develop a new uncertain OLAP API from scratch.

Sine the purpose of this experiment is just to demonstrate my techniques, I used MS Access as an underline DBMS. This DBMS is easy to use with a small amount of data and more importantly I can run it efficiently on my personal computer. The database API I use to connect with the DBMS is JDBC.

It’s not easy to find a real data warehouse. In this project, I took the sample data warehouse in olap4j called Mondrian Food Mart to run my experiment. Moreover, as the number of probabilistic

<table>
<thead>
<tr>
<th>Auto</th>
<th>Loc</th>
<th>Bound</th>
<th>Level</th>
<th>PW_{ID}</th>
<th>Prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sierra</td>
<td>NY</td>
<td>500</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Sierra</td>
<td>MA</td>
<td>200</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>F150</td>
<td>MA</td>
<td>100</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>F150</td>
<td>NY</td>
<td>250</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>F150</td>
<td>MA</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0.4</td>
</tr>
<tr>
<td>F150</td>
<td>NY</td>
<td>150</td>
<td>1</td>
<td>1</td>
<td>0.4</td>
</tr>
<tr>
<td>F150</td>
<td>MA</td>
<td>100</td>
<td>1</td>
<td>2</td>
<td>0.6</td>
</tr>
<tr>
<td>F150</td>
<td>NY</td>
<td>250</td>
<td>1</td>
<td>2</td>
<td>0.6</td>
</tr>
</tbody>
</table>

Figure 1: An example of materialize table.
data may be come very large and I have to run all the experiments in my personal computer, I was only able to pick a small part of this data warehouse in my experiment.

The experiment consists of 2 main steps:

1. Re-implement previous work by Burdick et.al. to get a probabilistic database
2. Evaluate the proposed techniques using generated data.

6.2 Threats to Validity

Using the above experiment setup, there are several threats to validity. I have to keep them in my mind when evaluating the results.

First, the amount data used here was quite small, thus it is hard to evaluate the scalability of my techniques. Also, picking a small portion of data may change the distribution of it. Second, the probabilistic data is synthetically generated and the hierarchy of domain are constructed as a uniform distribution. Normally, it’s not the case in the reality. Third, the uncertain OLAP was built based on an existing DBMS. The way I queried in uncertain database is to utilize top-k queries in certain database can combined the results to get the answer. Using this approach will lead to some risks as I don’t actually know how the top-k queries for certain database is implemented. The same risk may rise when I use JDBC as an intermediate method to communicate with the DBMS. I am also not sure if a Java API can work smoothly with a Microsoft DBMS. The last threat is I tried to run all experiments on a personal computer. Normally, the algorithms for database should be run in an independent DB server. Testing the algorithm in a personal computer, I don’t know if the programs existing in my computer (like Antivirus, MS Office) can affect the results. Also, it is hard to address if there is any issue with parallelism, catch, etc.

6.3 Experimental results

The first step in my experiment is to generate a probabilistic database. Basically, in this step I re-implement what proposed in Burdick’s paper[3]. The data domain used here consists of 3 dimension, each has a 3 level hierarchal domain. Every inner nodes in domain have 10 children. Thus we have in total a space of $10^6$ cells.

To get the uncertain database, I randomly select some records from a certain database and generalize some (i.e. 1,2,3) of their dimensional attributes into a higher level in the domain hierarchy. I started with a small certain database with only 253 records. Figure 2(a) shows the results of probabilistic data generation. As we may see, although the starting database is very small, the uncertain database size may become very large, as large as 1000 times the original one. Figure 2(b) shows the record distribution in for the uncertain database having 10% of uncertainty. In this version, an original record may be splitted up to nearly 200 pieces. This experiment answers the question 1 about the characteristics of probabilistic databases.

Using the synthesized probabilistic data above, my next experiment is to evaluate the effectiveness of my top-k algorithm. I picked the 10% uncertainty database version for my experiment. The experiment was run with different values of $k$. The results are shown on figure 3. The results indicate that for $k$ small, the running time is still acceptable. However, when $k$ is large, the running time increases dramatically. In fact, I tried to run this experiment with $k$ large than 11 but I had to interrupt the experiment as it took too long.
There are two breaking points in figure 3, at $k = 4$ and $k = 11$, that the graph slope is suddenly changed. After examining the data, I figured out that at these points, the queries met some uncertain cells and thus they had to explore deeper (i.e., splitting cells) to get the answer. This experiment answers my research question 2.

To answer research question 3, I tried to run a top-3 query in different levels of database uncertainty. I selected top-3 query in this experiment because in the previous experiment top-3 query didn’t take too long even though $k$ is large enough. So I hoped I wouldn’t have to interrupt in the middle as in the experiment to evaluate the query effectiveness. The results are shown in figure 4. A very interesting observation here is it is not always that querying in a less uncertain database will take less time. As we can see, even a query in the 7% uncertain database can take longer than a query in the 11% uncertain database. This observation is somewhat unusual but after looking into the database I figured out that the query runtime does not only depend on the data uncertainty but also depends on the nature of data. In the case the uncertain portion of a database is large but the values of uncertain tuples are small, it is likely that those uncertain tuples will be pruned by the algorithm (as their abstract aggregations are small). On the other hand, even if the uncertain portion of a database is small the runtime can still be very large large. It may be the case that there is one or two uncertain tuples having large values and the query, in order to find
the final answer, must split cells containing them, search in many different possible worlds.

(a) Database size.  
(b) Generating time.

Figure 5: Material data generation.

Figure 5 presents the size of materialized data and time to generate it. As we can see, from a relatively small database, after several level, the size of materialized data is dramatically increased. It’s because the number of possible words splitted from an “uncertain” cell is very large. This experiment answers question 4 and also suggests that in future, we need to develop a mechanism for selecting possible worlds to materialized.

7 Conclusion and Future Work

This project has presented a complete solution for top-k queries in uncertain OLAP. The experimental work showed that this solution is feasible.

There are a lot of future work remained for this project. The most obvious one is to make the algorithm more scalable. In this project, I just executed the experiments on a very small dataset with a small level of uncertainty. One possible solution is to use a R-tree to group cells and use an abstraction of a region instead of a cell like proposed in this project. Also, some approximation approaches may be useful in speed up query processing.
A Some irrelevant work

This appendix presents some of my unsuccessful effort during the project just to clarify why I have to change my topic.

A.1 Using Graphical Model to Represent Uncertain OLAP

At the beginning of this project, I proposed to use the graphical model in probabilistic database [18] to represent uncertain OLAP. I hoped the relationships between tuples within a cube as well as the domain constrains [4] can be naturally captured as the correlations proposed in [18]. By this way, we can use the query processing mechanisms in probabilistic database to manipulate OLAP queries. However, this idea doesn’t work for the following reasons:

- To process queries, the previous authors convert uncertain OLAP to a probabilistic database. This makes the uncertain OLAP model and graphical model naturally equivalent and actually uncertain OLAP model is just a case of graphical model. For example, the Constraint Hypergraph representing the domain constrains in OLAP can be seen as the correlation edges between tuples in graphical model.

- The most important queries in OLAP are aggregate queries whereas graphical model is inefficient in such kind of queries. The idea doesn’t work because I tried to apply approaches from a more general model (graphical) to deal with a simpler thing (OLAP) and it complicates the process.

A.2 Using Stream Query Processing to Summarize Possible Worlds

Reading the original paper of uncertain OLAP by Burdick et al. [3] (the previous paper of the one suggested on the web) for the first time, I thought that they calculate an aggregation by calculating in each individual possible world and then summarizing the results. This approach is inefficient in that it requires scanning precise tuples multiple time (i.e. for each possible world). For that reason, I tried to improve this approach by using an approach similar to stream query processing to approximate the aggregation. By this way, although we just get an approximation (but it’s acceptable since the database it-self is uncertain), each tuple is seen once. However, after carefully considering Burdick’s algorithm, I figured out that actually their approach is more effective than mine. They divided each uncertain tuple to several certain tuples to build up a probabilistic database called Extended Data Model (EDM) whose tuples having certain attributes. The queries are answered by first filtering in EDM to get a candidate query region. Then within the filtered region, the aggregations are calculated by summing up the allocations (possibilities) of all tuples. By this way, only imprecise tuples are scanned several time (i.e. having multiple copies in EDM).

References


