Top-k Queries in Uncertain OLAP

Master Scholarly Paper

Bao N. Nguyen
Department of Computer Science
University of Maryland
College Park, MD 20742, USA
baonn@cs.umd.edu

Abstract

Top-k is the most important query type in OLAP, a 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 remains an unanswered question. This project introduces a solution for top-k query evaluation 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 (*Online Analytical Processing*) [21] is a data warehouse model widely used in industry. Recently, this model has been extended to work with uncertain and imprecise data [3, 13, 15]. In OLAP, *top-k query* is the most dominating type of query. Although the semantics of top-k queries for uncertain tuples were discussed by several authors (e.g., [23]), how to process top-k queries in uncertain OLAP is still unexplored. This project presents a solution to the problem of *answering top-k queries in uncertain OLAP*. The contributions of the project are summarized as follows:

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

In the next section, I provide a brief literature survey on OLAP and probabilistic databases to set the context of the work proposed. I then present 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 is presented to show the effectiveness of proposed model (section 6). I finish with some conclusions and directions for future work.

2 Literature survey

2.1 OLAP

OLAP is a data warehouse model allowing data to be summarized and viewed in *different views* and in an *online* fashion. These multidimensional views are supported by multidimensional database technologies. 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*[1][5]. The data cubes are organized by the star schema [12], where some measures are analyzed with respect to some interesting dimensions, representing different 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, the readers are referred to [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 optimality criterion on dimensional attributes which are 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 the highest aggregation. However, it will be extremely slow as the number of possible cubes in an OLAP may be very large. There are several approaches that have been proposed in the 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.* [18] 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.*[17] 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 [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.*[16], is to partition data cubes into sufficiently small cells and to pre-compute the top-k aggregate values in each partition. 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.

[1]Here "cube" is just used as an abstraction, it doesn't necessarily refer to a three dimension space

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, 13, 15]. In this project, I focus on the model proposed by Doug *et al.* [3]. In their model, the assumption that all facts must be *points* is relaxed to allow facts to 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 in a later work [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[20]. 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 problems are (1) how to represent complex data in a concise way without losing its expressibility, and (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 leads 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 processing. Sen *et al.* [20] propose a method using *graphic models* to capture such correlations. Another approach is to store extra information such as lineage/provenance [24] 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 evaluating the probability distribution over the answer tuples. 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 [11] propose using tuple-level probabilities to present the intensional and extensional query evaluation techniques. Dalvi and Suciu [9] define safe query plans to be those for which extensional and intensional query evaluation produces identical results. Lakshmanan *et al.* [14] attempt to combine these different approaches by associating probability intervals with tuples.

Another open issue is how to generate *query answers*. In many scenarios, when the input data is uncertain or imprecise, an approximations is sufficient. In this case, an approximate strategy should be used to speed up the query processing. Also, a mapping to data sources or some form of meta data [10] might be included to provide the context where the answers are derived from.

2.2.2 Top-k query processing in probabilistic databases

Top-k query processing in uncertain databases is semantically and computationally different from traditional top-k query processing. In the possible world, we have to deal with two variants: *query score* and *uncertainty* at the same time. It is hard to apply traditional techniques in probabilistic databases. 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, the number of work on this area is still limited.

In [23], Soliman *et al.*, with an assumption 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 does not care about possible worlds and fits in the scenarios where we do not need to restrict returned tuples in the same world. A more detailed explanation of $U-kRank$ can be found in [22].

Another approach to evaluate top-k query in uncertain database based on the probability associated with each query answer. In [19], 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.

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. Using the classification of top-k queries semantics for tuples proposed by Soliman et al [23], the top-k queries semantics for OLAP may be interpreted in three different ways:

1. The "top k" query regions[2] (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 straight forward since they are similar to the semantics of top-k queries for probabilistic tuples proposed in previous work [23]. 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 does not 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 $\boldsymbol{PW} = \{PW^1, \ldots, PW^n\}$. Let $\boldsymbol{R} = \{R^1, \ldots, R^m\}$ 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 $\boldsymbol{PW}$. The top-k answer for O is $argmax_{R^i \in R}(\sum_{w \in PW(R^i)}(w))$*

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 [17]. Those values are later used as bounds for branch pruning in the query region search tree.

For the semantics 2 and 3, the task is much 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 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.

[2]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]

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).
- All attributes and entities are treated as *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 [20]. 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 we are trying to solve in this project. For that reason, I just assume 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 do not 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 do not 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 1 explains the details of my approach. The granularity of the OLAP are reflected by the granularity of the dataset D. $D_{certain}$ is the certain part of D and $Top_{k-certain}$ is a set of top-k cells in $D_{certain}$. $Top_{k-certain}$ 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 less one more cell (c_j) become certain and the chance to find the answer becomes higher.

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

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
  $D_{certain} \leftarrow$ The certain part of D
  $Top_{k-certain} \leftarrow Top(k, D_{certain})$
  $T \leftarrow$ 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 $ts \in PT$ **do**
    $D_{T'} \leftarrow D \cup ts \setminus (D_{certain} \cup T)$
    $Top(k - j, D_{T'}) \rightarrow (Top_{(k-j)T'}, prob_{T'})$
    $Top(k, Top_{k-certain} \cup Top_{(k-j)T'}) \rightarrow Top_{kT'}$
  end for
  $Top_k \leftarrow$ Most possible $Top_{kT'}$
  $prob \leftarrow \sum_{Top_{kT'}=Top} prob_{T'}$
end if

5 Data Materialization

Since the cost for calculating cell upper bounders are quite expensive, I propose to materialize and store them 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 cells in different levels, each cell is stored in different levels of dataset granularity. More specifically, 3 more columns $Level$, PW_{ID} and $Prob$ are added 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.

Auto	*Loc*	*Bound*	*Level*	PW_{ID}	*Prob*
Sierra	NY	500	0	1	1
Sierra	MA	200	0	1	1
F150	MA	100	0	1	1
F150	NY	250	0	1	1
F150	MA	0	1	1	0.4
F150	NY	150	1	1	0.4
F150	MA	100	1	2	0.6
F150	NY	250	1	2	0.6

Figure 1: An example of materialize table.

6 Experiments

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

1. What are differences between a probabilistic database and a traditional database?
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 relation between uncertainty level and query characteristics?
4. What is the cost of using materialize data?

6.1 Experiment Setup

Although there are several OLAP API available such as Olap4j or Oracle OLAP, 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. Microsoft Access was used as underlying DBMS. The database API used to connect with the DBMS is JDBC.

In this project, I took the sample data warehouse in Olap4j called Mondrian Food Mart as experimental data. Moreover, because of the limitations of hardware available for experiments, I was only able to pick a small portion of this data warehouse for the experiments.

The experiment process consists of 2 main steps:

1. Re-implement the 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. We need to keep them in mind when evaluating the results.

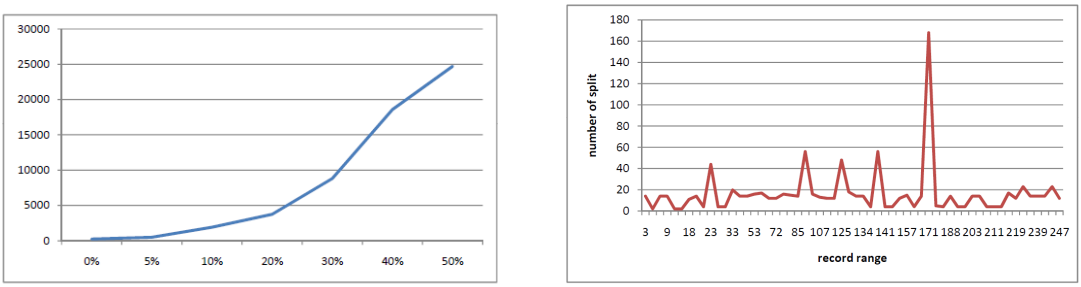
First, the amount data used here was quite small, thus it is hard to evaluate the scalability of my techniques. Also, picking a small portion of data may change the distribution of it. Second, the probabilistic data is synthetically generated and the hierarchy of domain is constructed as a uniform distribution. Normally, it is not the case in the reality. Third, the uncertain OLAP was built based on an existing DBMS. The uncertain queries were evaluated based on the built-in algorithms for

certain top-k query provided by the DBMS. Using this approach leads to some risks as I do not actually know how the top-k queries for certain database is implemented. The similar risk may rise when I used 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 with a limited hardware resource. Normally, the algorithms for database should be run in an independent DB server. Testing the algorithm in a personal computer, I do not 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 10^6cells.

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 questioned 1 about the characteristics of a probabilistic database.



(a) Database size.

(b) Record distribution.

Figure 2: Probabilistic database generation.

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 explored 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

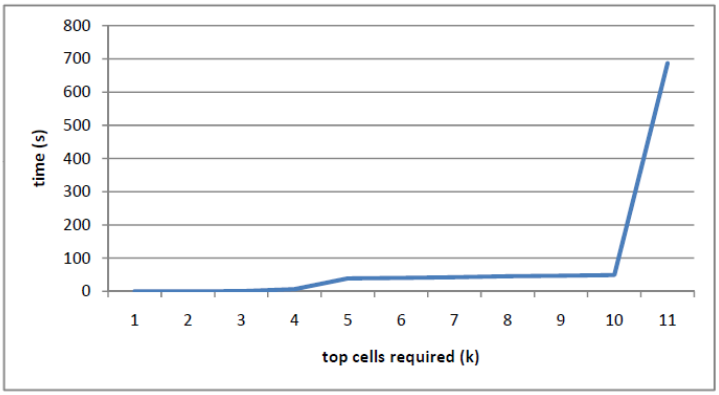


Figure 3: Algorithm effectiveness.

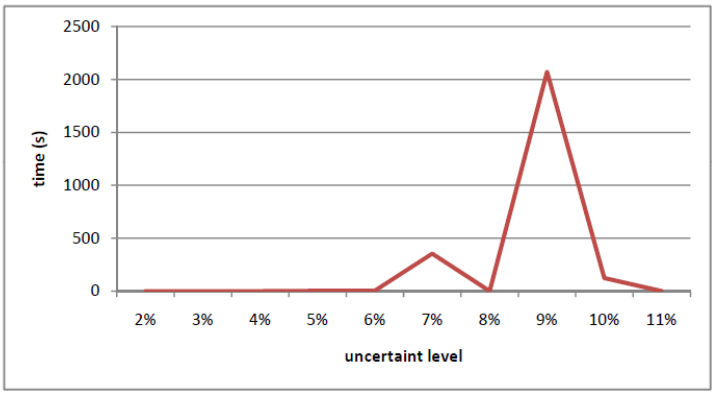
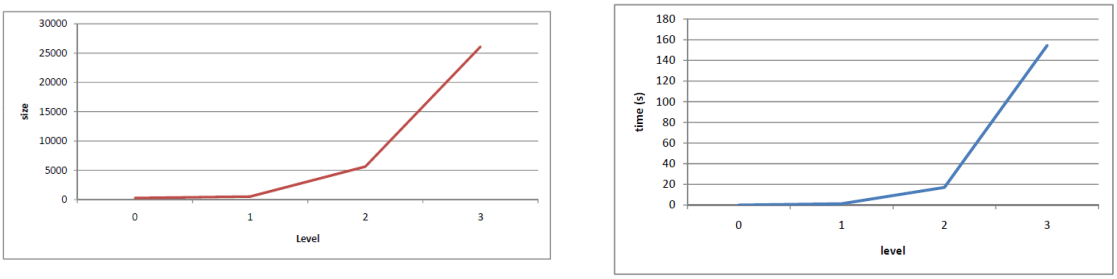


Figure 4: Uncertainty vs. time

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 in to 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 several possible future work for this project. The most obvious one is to make the searching 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 the query processing.

8 Acknowledgements

The author would like to thank Professor Amol Deshpande for his discussion on this project.

References

[1] D. BarbarB, H. Garcia-Molina, and D. Porter, *The Management of Probabilistic Data*, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING **4** (1992), no. 5.

[2] Doug Burdick, Prasad M. Deshpande, T. S. Jayram, Raghu Ramakrishnan, and Shivakumar, *Efficient allocation algorithms for olap over imprecise data*, VLDB '06: Proceedings of the 32nd international conference on Very large data bases, VLDB Endowment, 2006, pp. 391–402.

[3] Doug Burdick, Prasad M. Deshpande, T. S. Jayram, Raghu Ramakrishnan, and Shivakumar Vaithyanathan, *Olap over uncertain and imprecise data*, VLDB '05: Proceedings of the 31st international conference on Very large data bases, VLDB Endowment, 2005, pp. 970–981.

[4] Doug Burdick, AnHai Doan, Raghu Ramakrishnan, and Shivakumar Vaithyanathan, *Olap over imprecise data with domain constraints*, VLDB '07: Proceedings of the 33rd international conference on Very large data bases, VLDB Endowment, 2007, pp. 39–50.

[5] Surajit Chaudhuri and Umeshwar Dayal, *An overview of data warehousing and olap technology*, SIGMOD Rec. **26** (1997), no. 1, 65–74.

[6] R. Cheng, D.V. Kalashnikov, and S. Prabhakar, *Evaluating probabilistic queries over imprecise data*, Proceedings of the 2003 ACM SIGMOD international conference on Management of data (2003), 551–562.

[7] Y.D. Chung, W.S. Yang, and M.H. Kim, *An efficient, robust method for processing of partial top-k/bottom-k queries using the RD-Tree in OLAP*, Decision Support Systems **43** (2007), no. 2, 313–321.

[8] A. Cockburn, A. Karlson, and B.B. Bederson, *A review of focus and context interfaces*, Tech. report.

[9] N. Dalvi and D. Suciu, *Management of probabilistic data: foundations and challenges*, Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (2007), 1–12.

[10] X. Dong, A.Y. Halevy, and C. Yu, *Data Integration with Uncertainty*, Ann Arbor **1001**, 48109.

[11] Norbert Fuhr and Thomas Rlleke, *Hyspirit - a probabilistic inference engine for hypermedia retrieval in large databases*, Proceedings of the 6th International Conference on Extending Database Technology (EDBT, Springer, 1998, pp. 24–38.

[12] R. Kimball et al., *The data warehouse toolkit*, (2002).

[13] J. Kiviniemi, A. Wolski, A. Pesonen, and J. Arminen, *Lazy Aggregates for Real-Time OLAP*, Data Warehousing and Knowledge Discovery: First International Conference, Dawak'99, Florence, Italy, August 30-September 1, 1999, Proceedings (1999).

[14] Laks V. S. Lakshmanan, Nicola Leone, Robert Ross, and V. S. Subrahmanian, *Probview: a flexible probabilistic database system*, ACM Trans. Database Syst. **22** (1997), no. 3, 419–469.

[15] Junqiang Liu, Min Luo, and Xiandi Yang, *Deputy mechanism for olap over imprecise data and composite measure*, CIT '07: Proceedings of the 7th IEEE International Conference on Computer and Information Technology (Washington, DC, USA), IEEE Computer Society, 2007, pp. 65–70.

[16] Zheng Xuan Loh, Tok Wang Ling, Chuan Heng Ang, and Sin Yeung Lee, *Analysis of pre-computed partition top method for range top-k queries in olap data cubes*, (2002), 60–67.

[17] N. Mamoulis, S. Bakiras, and P. Kalnis, *Evaluation of Top-k OLAP Queries Using Aggregate R–Trees*.

[18] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao, *Efficient OLAP Operations in Spatial Data Warehouses*, Proc. of SSTD **280** (2001).

[19] C. Re, N. Dalvi, and D. Suciu, *Efficient top-k query evaluation on probabilistic data*, Proceedings of ICDE (2007).

[20] Prithviraj Sen and Amol Deshpande, *Representing and querying correlated tuples in probabilistic databases*, ICDE, 2007, pp. 596–605.

[21] A. Silberschatz, H.F. Korth, and S. Sudarshan, *Database Systems Concepts*, McGraw-Hill Higher Education, 2001.

[22] M.A. Soliman, I.F. Ilyas, and K.C. Chang, *URank: formulation and efficient evaluation of top-k queries in uncertain databases*, Proceedings of the 2007 ACM SIGMOD international conference on Management of data (2007), 1082–1084.

[23] M.A. Soliman, I.F. Ilyas, and K.C.C. Chang, *Top-k query processing in uncertain databases*, Proceedings of ICDE (2007).

[24] J. Widom, *Trio: A system for integrated management of data, accuracy, and lineage*, Proc. of CIDR **5** (2005).