Integrating OLAP and Ranking: The Ranking-Cube Methodology[*]

Dong Xin Jiawei Han
Department of Computer Science
University of Illinois at Urbana-Champaign

Abstract

OLAP (On-Line Analytical Processing) and Ranking are currently separate technologies in the database systems. OLAP emphasizes on efficient multi-dimensional data analysis and ranking is good for effective data exploration in massive data. In this paper, we discuss the problem of integrating OLAP and ranking, such that ranking serves as a function block for data analysis and exploration in the OLAP environment. Towards this goal, we present the ranking cube*: a semi off-line materialization and semi online computation model. This paper discusses the framework, the implementation issues and the possible extensions of the ranking cube.*

1 Introduction

OLAP (On-Line Analytical Processing) and Ranking are currently separate technologies in the database systems. OLAP refers to a set of data analysis tools developed for analyzing data in data warehouses since 1990s [7]. A data warehouse stores a multi-dimensional, logical view of the data, and supports management's decision-making process. A point in a data cube stores a consolidated measure of the corresponding dimension values in a multi-dimensional space. OLAP operations, such as drill-down, roll-up, pivot, slice, and dice, are the ways to interact with the data cube for multi-dimensional data analysis. Ranking is a way to filter the query results and retain partial but high-quality answers. With the increasing integration of the database systems with Web search, information retrieval, multimedia, and data mining applications, database query processing has been evolving from finding the complete set of answers to finding top-k answers. This leads to the popularity in research into ranked query processing [4, 3, 8, 10, 13, 11, 12]. A *ranked query* (or top-k query) asks for the best k results according to a user-specified preference.

The traditional OLAP provides the high-level data statistics, whereas an equally important task is to view and manipulate the low-level featured data records with respect to multi-dimensional group-bys. With the mounting of an enormous amount of data in business, ranking becomes prominent for effective data analysis and exploration. Example application scenarios are illustrated as follows.

Example 1. (Multi-dimensional data exploration) Consider an online used car database (*e.g.*, car.com) that maintains the following information for each car: (*type, maker, color, price, milage*). Users may want to explore the data with respect to their own preferences among subsets of data confined by multi-dimensional values. For instance, a user may ask for top-k answers, among a subset of cars with *type* = *"sedan"* and *color* = *"red"*, by the ranking function $f = (price - 10k)^2 + (milage - 10k)^2$ (*e.g.*, the *price* is close to $10k$ and the *milage* is close to $10k$). ■

Example 2. (Multi-dimensional data analysis) Consider a notebook comparison database (*e.g.*, bizrate.com) with schema (*brand, price, CPU, memory, disk*). Suppose a function f is formulated on CPU, memory and disk to evaluate the market potential of each notebook. An analyst who is interested in dell low-end notebooks may first issue a top-k query with *brand* = *"dell"* and *price* ≤ 1000, and then rolls up on the *brand* dimension and checks the top-k low-end notebooks by all makers. By comparing two sets of answers, the analyst will find out the position of dell notebooks in the low-end market. ■

To meet the requirement of online analytical processing, the database system has to return the user-preferred answers from any data groups, in a very efficient way. The dynamic nature of the problem imposes a great challenge for the database research community. In this paper, we address this problem from an inte-

[*]Work supported in part by the U.S. National Science Foundation NSF IIS-05-13678/06-42771 and NSF BDI-05-15813. Any opinions, findings, and conclusions or recommendations expressed here are those of the authors and do not necessarily reflect the views of the funding agencies.

grated viewpoint. On the one hand, OLAP requires off-line pre-computation so that multi-dimensional analysis can be performed on the fly; on the other hand, the ad-hoc ranking functions prohibit full materialization. A natural proposal is to adopt a *semi off-line materialization* and *semi online computation* model. This paper discusses the design principles, implementation issues and possible extensions.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 presents our proposal for the semi off-line materialization and semi online computation model: ranking cube. We discuss the extensions, as well as the challenges in Section 4, and conclude the study in Section 5.

2 Related Work

Our work is closely related to the data cube and ranked query processing. Data Cube has been playing an essential role in implementing fast OLAP operations [7]. The measures in the cube are generally simple statistics (*e.g.*, sum). Some recent proposals introduce more complex measures, such as linear regression model [6] and classification model [5]. The multi-dimensional ranking analysis using data cube was proposed by our recent work [15]. This paper extends the framework and points out several future directions.

Recent studies on efficient processing rank-aware queries include (1) the rank-aware materialization approach [3, 10, 13], which maintains views or special indexes to speed up the response time for ranked queries; (2) the rank query transformation approach [4], which maps the k nearest neighbor queries to range queries; and (3) the rank-aware query optimization approach [11, 12], which hacks a traditional query executer with a rank aggregation algorithm [8]. Our work is different from previous work in that we study the ranking problem in the OLAP environment.

3 Ranking Cube

A preliminary study of the ranking cube can be found in [15]. Here we extend the original proposal by abstracting the framework and presenting various implementation issues.

3.1 Framework

We classify the dimensions in a relation R into *boolean dimensions* $A_1, A_2, \dots, A_b$, and *ranking dimensions* $N_1, N_2, \dots, N_r$. The two sets of dimensions are not necessarily exclusive. OLAP is performed on the boolean dimensions and the ranking analysis is conducted on the ranking dimensions.

OLAPing ranked queries is essentially a task to efficiently find top answers (according to a function f) with a set of multi-dimensional boolean predicates, B. To do this, an algorithm can first filter data tuples by B and then compute the top-k results, or first search data tuples according to f and verify B on each candidate. Both approaches may retrieve data that will be pruned by the other criterion later, and thus are not efficient. Ranking cube is a way to simultaneously combine both ranking and boolean pruning. The whole framework consists of three components.

1. To facilitate rank-aware data retrieval, data is partitioned into n blocks according to ranking dimensions. We refer the data partition as $P = \{b_1, b_2, \dots, b_n\}$, where $b_i = \{t_i^1, t_i^2, \dots, t_i^l\}$ is the i^{th} data block, and t_i^j is the j^{th} tuple in b_i.
2. For each B, we compute a measure $M(P|B) = \{m(b_1|B), m(b_2|B), \dots, m(b_n|B)\}$, where $m(b_i|B) = \{\delta(t_i^1|B), \delta(t_i^2|B), \dots, \delta(t_i^l|B)\}$ and $\delta(t_i^j|B) = 1$ if the tuple t_i^j satisfies B. The ranking cube C pre-computes and stores the measure M for all possible dimensional values.
3. Guided by C, the query processing algorithm S searches for top answers over P such that a block b_i is retrieved if and only if b_i may contain tuples better than the current top-k results, and $m(b_i|B) \neq 0$.

3.2 Implementation Issues

In the framework presented above, the data partition P and ranking cube C contain the semi off-line materialization, and the search algorithm S conducts the semi online computation. In this subsection, we discuss two typical implementations: the *grid partition with neighborhood search* and the *hierarchical partition with top-down search*. In each implementation, we will discuss the partition scheme, the measure composition, and the query algorithm.

3.2.1 Grid Partition

Partition Scheme: We demonstrate the grid partition by the method used in [15]. For each ranking dimension (*e.g.*, X and Y in Table 1), we partition the domain into L bins by equi-depth partitioning. In our sample database (Table 1), suppose A and B are boolean dimensions, and X and Y are ranking dimensions. Each ranking dimension is partitioned into 4 bins, and the data is partitioned into 16 blocks (Figure 1). We store the tuples with the values on ranking dimensions, cell by cell, in a *block table*. The bin boundaries of the equi-depth partition are stored in the *meta table*.

tid	A	B	X	Y
t1	a1	b1	0.0	0.4
t2	a2	b2	0.2	0.6
t3	a1	b1	0.3	0.7
t4	a3	b3	0.5	0.4
t5	a4	b1	0.6	0.0
				...

Table 1. A Sample Database

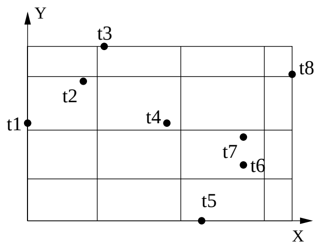

Figure 1. Grid Partition

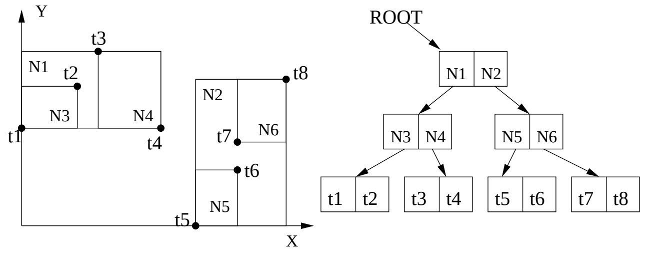

Figure 2. Hierarchical Partition

Measure Composition: Following the above example, suppose the block on the i^{th} row and j^{th} column in Figure 1 is b_{4i+j} $(i, j = 0, 1, 2, 3)$. Given a boolean predicate $B = a_1 b_1$, only $b_1 = \{t_3\}$ and $b_4 = \{t_1, t_2\}$ contain tuples satisfying B (*i.e.*, t_1, t_3). Consequently, $m(b_1|B) = \{1\}$, $m(b_4|B) = \{10\}$ and $m(b_i|B) = 0$ for all the other b_i. We point out two important issues in implementing $M(P|B)$. First, by merging all $m(b_i|B)$, $M(P|B)$ is basically a bit-array and can be compressed by many compression methods. Secondly, $M(P|B)$ can be decomposed into several smaller parts (while preserving neighboring b_i together), each of which is retrieved from the ranking cube only when necessary.

Query Algorithm: Given a ranked query with f and B, the query algorithm searches for blocks that are (1) promising with respect to f, and (2) containing tuples satisfying B. To begin with, we define $f(b_i)$ as the minimal value of f over the region covered by block b_i[1]. Given the bin boundaries stored in the *meta table*, the search algorithm is able to sort all b_i according to $f(b_i)$, and retrieves b_i one by one. The algorithm skips a b_i if $m(b_i|B) = 0$ since it contains no tuple satisfying B. To avoid enumerating all blocks, one may first locate the blocks that contain the extreme points, and progressively search over their neighboring blocks. Among the neighboring blocks, the algorithm will first examine the block with minimal value of $f(b_i)$. We refer this search method as *neighborhood search*, and it assumes that the ranking functions are *convex*.

3.2.2 Hierarchical Partition

Partition Scheme: For hierarchical partition, we use R-Tree as an example [9]. R-Tree splits space with hierarchically nested and possibly overlapping boxes. Each node stores the pointers to child nodes and the bounding box of child nodes. The leaf node stores the *tid*s and the values on ranking dimensions. Figure 2 shows a sample R-Tree, where the root node contains pointers to child nodes N_1 and N_2, and so on.

[1]In this paper, we assume minimal top-k is requested.

Measure Composition: Each node N_i in the R-Tree partition corresponds to a block in the ranking cube framework, and we define $m(N_i|B) = \{\delta(n_i^1|B), \delta(n_i^2|B), \ldots, \delta(n_i^l|B)\}$, where n_i^j is a child node or a tuple. $\delta(n_i^j|B) = 1$ if and only if n_i^j contains a tuple satisfying B.

Query Algorithm: Given the hierarchical partition, the algorithm follows the branch-and-bound principle to progressively retrieve data nodes. Specially, the search process first inserts the root node into a heap h. At each step, the algorithm fetches the node N appearing at the top of h (*i.e.*, with minimal value of $f(N)$), and inserts all child nodes n_j of N to h if $m(n_j|B) \neq 0$. The query processing halts when $f(N)$ is no less than the current top-k results. We refer this search method as *top-down search*.

3.2.3 Comments on Partition Schemes

We have presented a general framework for ranking cube, and demonstrated it by both grid and hierarchical partitions. The grid partition is simple and easy to implement. However, the query performance may be sensitive to data distribution, since the there may be many dead (i.e., empty) cells for skewed data. The hierarchical partition is more robust with respect to data distribution. But it may incur additional cost to build and traversal over the partition. In real application, one may choose different partition scheme accordingly.

4 Extensions and Challenges

4.1 High-Dimensional Ranking Cube

In some applications, the number of dimensions is large. With high boolean dimensions, a full materialization of the ranking cube is too space expensive. Observing that many real life ranked queries are likely to involve only a small subset of attributes, we can carefully select the cuboids which needs to be materialized. A baseline solution is to materialize only those *atomic* cuboids that contain single dimensions. Given a set of arbitrary boolean predicate B that consists of B_1,

$B_2, \ldots, B_n$ atomic predicates, one can online assemble $m(b_i|B) = \cap_{j=1}^{n} m(b_i|B_j)$ for each block b_i.

With high ranking dimensions, the partitioning methods are not effective. A possible solution is to create multiple data partitions, each of which consists a subset of ranking dimensions. The query processing may need to conduct search over a joint space involving multiple data partitions. There exist two challenges: (1) the search space for f grows exponentially with the number of partitions, and (2) the satisfactory of B for a joint block is difficult to determine.

4.2 Multi-Relational Ranking Cube

When multiple relations exist in the database, ranking cube can also be built crossing relations. A multi-relational ranking cube is built by a join condition and consists of a set of boolean dimensions and ranking dimensions from each participating relation. The partition is performed on all ranking dimensions, and the ranking cube is built with respect to all boolean dimensions. For space efficiency, the measure can be computed in a way similar to join indices [14]. That is, for each block b_i, $m(b_i|B) = \{m(b_i|B_1, R_1), m(b_i|B_2, R_2), \ldots, m(b_i|B_n, R_n)\}$, where $B_j \subseteq B$ is the boolean dimension values in R_j, and $m(b_i|B_j, R_j)$ is the original measure in relation R_j (see Section 3.1). Unfortunately, the above method only builds a ranking cube that works for one particular join condition. It remains a challenging topic to have a solution which works for dynamic join conditions.

4.3 Incremental Maintenance

In dynamic web environments, data changes frequently. It is desirable to have an incremental methodology to efficiently maintain the ranking cube with the data insertion or deletion. We illustrate some high-level principles as follows. For grid partition, one can temporally allocate new data according to pre-computed blocks, and re-partition the data periodically. For hierarchical partition, incremental data update on R-trees can be first applied, and the structure changes can be propagated to ranking cube. It is still a challenge problem to design and implement an efficient incremental update method.

4.4 Extended Functionalities

Top-k queries are related to several other preference queries, such as skyline query [2] and convex hulls [1]. Skyline query asks for the objects that are not dominated by any other object in all dimensions. A convex hull query searches a set of points that form a convex hull of all the other data objects. The ranking cube framework is also applicable to these queries. For instance, the convex hull query algorithm developed by [1] progressively retrieves R-tree blocks until the final answers are found. Ranking cube can be integrated into this search procedure by pruning some blocks that do not contain tuples satisfying B.

It is also an interesting but challenging task to support top-k aggregate queries that consists of grouping, ranking and boolean selections.

5 Conclusions

With the enormous amount and fast growth of data in business, Web, and scientific applications, the integration of the OLAP and ranking provides an interesting direction for exploring and analyzing this overwhelming data. Towards this goal, we presents the ranking cube methodology in this paper. Our future work includes: (1) extending the framework to high dimensional data, multi-relations, and other ranking related functionalities; and (2) building a complete solution for efficiently computing and maintaining the ranking cube.

References

[1] C. Bohm et al. Determining the convex hull in large multidimensional databases. In *DaWaK'01*.

[2] S. Borzsonyi et al. The skyline operator. In *ICDE'01*.

[3] Y. Chang et al. Onion technique: Indexing for linear optimization queries. In *SIGMOD'00*.

[4] S. Chaudhuri and L. Gravano. Evaluating top-k selection queries. In *VLDB'99*.

[5] B. Chen et al. Prediction cubes. In *VLDB'05*.

[6] Y. Chen et al. Multi-dimensional regression analysis of time-series data streams. In *VLDB'02*.

[7] S. Churdhuri and U. Dayal. An overview of data warehousing and data cube. *SIGMOD Record*, 26:65–74, 1997.

[8] R. Fagin et al. Optimal aggregation algorithms for middleware. In *PODS'01*.

[9] A. Guttman. R-tree: A dynamic index structure for spatial searching. In *SIGMOD'84*.

[10] V. Hristidis et al. Prefer: A system for the efficient execution of multi-parametric ranked queries. In *SIGMOD'01*.

[11] I. F. Ilyas et al. Rank-aware query optimization. In *SIGMOD'04*.

[12] C. Li et al. Ranksql: Query algebra and optimization for relational top-k queries. In *SIGMOD'05*.

[13] P. Tsaparas et al. Ranked join indices. In *ICDE'03*.

[14] P. Valduriez. Join indices. *ACM TODS*, 12:218–246, 1987.

[15] D. Xin et al. Answering top-k queries with multidimensional selections: The ranking cube approach. In *VLDB'06*.