# Ranking Query Processing in Multimedia Databases

**Byung-Gon Kim*  Joung-Woon Han*  Jaeho Lee**  Haechull Lim***

***Department of Computer Engineering, Hong-Ik University**
**Mapo-Gu Sangsoo-Dong 72-1 Seoul, Korea 121-791**
**Tel) +82-2-320-1673  Fax) +82-2-320-1105**
**E-mail : {bgkim, hanjw, lim}@cs.hongik.ac.kr**

****Department of Computer Education, Inchon National University of Education**
**Gyeyang-Gu Gyesan-Dong San59-12 Inchon, Korea 407-753**
**Tel) +82-32-540-1281  Fax) +82-32-548-0288**
**E-mail : jhlee@mail.inue.ac.kr**

**Abstract:** Among the multi-dimensional query types, ranking query is needed if we want the object one by one until we satisfy for the result. In multi-dimensional indexing structures like R-tree or its variants, not many methods are introduced in this area. In this paper, we introduce new ranking query processing algorithm which use the filtering mechanism in the R-tree variants.

## 1. Introduction

Much of the research on query processing of multi-dimensional data has focussed on two aspects; namely, range query and nearest-neighbor query(nnq). Range queries retrieve information regarding all objects within a certain distance from the query object, while the nearest-neighbor query finds the object closest to the query object. A variation of the nearest-neighbor query(nnq) is the k-nearest neighbor query(k-nnq) where it finds the k neighbors closest to the query object. Another important query type needed for multi-dimensional databases is ranking query. Ranking query is a similarity query that works based on a "give-me-more" facility. Give-me-more facility is frequently used in spatial databases. For example, we may wish to find the closest city from LA that has over 10 million populations in the database. In this case, if we use ranking query processing technique, we can check the population of the city one by one in the order of distance from LA until we meet the city that satisfies the condition. However, in case of existing nnq or k-nnq algorithm[4], though it has ordering and pruning strategies, it have to restart the search from the beginning to get the next city if we do not satisfy for the first result.

In this paper, we describe the existing ranking query processing algorithm in R-tree variants environment and introduce new ranking query processing algorithm to overcome the problem of the existing algorithm.

## 2. Related Works

### 2.1 Ranking query algorithm

The algorithm of ranking query was introduced in the context of 2-dimensional geographic information systems and works on PMR quadtrees[2]. This ranking query algorithm on R-trees used for optimal multi step k-nnq processing[7] was adapted from [2]. The ranking query algorithm used priority queue and used minimum distance(MINDIST) to order the priority queue. In the queue, three kinds of items can exist. Those are directory node, data node, and real object. After popping the first element in priority queue ordered by MINDIST, if the first element is directory node, for each child node of the node, compute the distance from a query object and reinsert to the queue. If the first element is data node, for each object of the node, compute the distance and reinsert objects to the queue in the order of distance. Lastly, if the first element is object, report the object.

However, ranking query algorithm used in this algorithm does not have leaf node filtering ability. To acquire the nearest object from the queue, after popping the leaf node of the queue, distance computation of all objects of the leaf node must be performed.

Next problem of this algorithm is that it uses a priority queue. Priority queue in this algorithm retains objects, leaf nodes, and intermediate nodes according to

their distances to the query point so that it needs heap change whenever the new item is inserted. As the indexing nodes and the objects of the tree grow, the system can be affected by the change overhead. Minimizing the heap management overhead of the priority queue is the important factor of the algorithm.

### 2.2 Tree Structures Using Vantage Point

Uhlmann introduced VP-tree for the k-nearest search[3]. VP-tree basically partitions the data space into spherical cuts around a chosen vantage point(VP). VP is a point chosen among the data points that is used for computing the relative distances from all the other points. At each node, the distances between the VP for that node and data points to be indexed below that node are computed. VP-tree keeps the different VP for each node at the same level. When the median value among the distances is found, the data points are partitioned into two groups. One of them contains the points whose distances to the VP are less than or equal to the median and the other group contains the points whose distances to the VP are larger than to the median. These two groups of data points are indexed separately by the left and right sub-branches below that node, which are constructed in the same way recursively. Although VP-tree can be generalized into a multi-way tree for larger fanouts, when the spherical cuts are very thin, the chances of a search operation descending down to more than one node becomes higher. It can be a overhead in the system.

To overcome this weakness, an interesting approach for finding the k-nearest neighbors in the MVP-tree was proposed by Bozkaya and Ozsoyoglu[6]. To find the k-nearest neighbor in the MVP-tree, like the VP-tree, an index structure is composed based on the distances from a specific vantage point(VP). However, the MVP-tree uses more than one vantage point to partition the space into spherical cuts at each level so that it keeps higher discrimination ability. Especially, in the MVP-tree, the pre-computed distances between the data point and the VPs along the path from the root to the leaf node are contained in the array. These informations are used for leaf node filtering in the query processing. This filtering concept can reduce the object distance computation time in the query processing.

The MVP-tree contains a filtering capability to reduce computations, but, due to its static nature, it is inappropriate for situations where insertions and deletions to the index structure is frequent. Based on these observations, we propose ranking query processing technique using VP filtering that utilizes the filtering technique of MVP-tree, but on the R-tree[1].

## 3. Ranking Query Processing Using Filtering

To perform ranking query with filtering, indexing tree with VP distance value must be constructed. Construction indexing trees (for R-tree variants such as the R-tree, R*-tree, X-tree) that contains vantage point (VP) values is essentially the same as constructing a tree without the VP value. The only extra step that is required is in deciding the VP.

Selecting a VP using one of the many choices may possibly have varying implications on the performance of our method. However, this is an issue that needs to be dealt separately. Hence, in this paper, we simply select the corner point of the domain space as the VP as this saves tree construction time. Once the VP is decided, we compute the distance between the VP and each object, and store their value as they are inserted into the R-trees. Note that these distances are absolute values and only need to be computed once when the objects are inserted. They are not affected by the change of the tree structure. The distance is contained as an entry of the leaf node along with the object pointer. Except for containing the distance to the VP, tree construction method is the same as the original R-tree variant structures.

The main idea behind VP filtering is that objects may be divided into two groups where in one group, distance computations of objects are computed, while for the other group most of these computations are delayed. This filtering is based on the VP values of the objects. This is possible as ranking query, by nature, requests objects in an incremental manner. The two groups are maintained in two separate queues, that we denote as the Return_queue and Delay_queue. Filtering is done as follows.

At first, we use the metric MINMAXDIST[4] mentioned. This notion guarantees the presence of an object in rectangle R whose distance is within MINMAXDIST. As there exists one or more objects within MINMAXDIST distance from query point Q, initially, only those objects within MINMAXDIST are considered for distance computation.

Whether the objects in the leaf node are within the MINMAXDIST or not is determined by two filtering conditions. In the first filtering condition, we use the VP distance retained in the object entry. That is, we perform

distance computations only for those objects that lie between d(VP,Q)-MINMAXDIST(Q,M) and d(VP,Q)+ MINMAXDIST(Q,M) where Q is the source of the query and M is the leaf node being considered. This is the light-shaded area between the two arcs denoted as 'First filtering boundary' in Figure 1.

Among these objects, second filtering condition is considered. The second filtering condition is required to resolve a problem like this. Note that with just first filtering condition there can be situations where objects which lie over MINMAXDIST(Q,M) are included in the distance computation. For example, in Figure 1, the object denoted as 'f', in fact, that does not lie within MINMAXDIST may be included in the result of the first filtering. This second filtering is done using MINMAXDIST(Q,M) and real distance of the object which is computed in the first filtering.

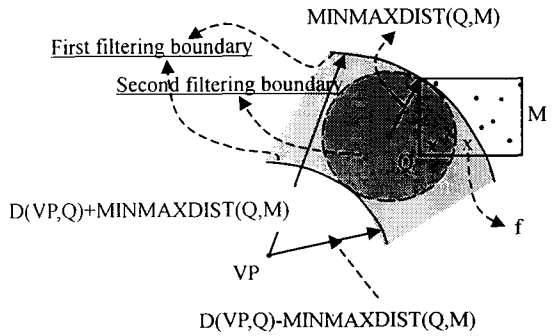


Figure 1. Filtering concept of ranking query algorithm

From the first result, those objects that satisfy the condition $d(f,Q) \leq MINMAXDIST(Q,M)$ which are objects in the area within the circle denoted as the 'Second filtering boundary', are moved to the Return_queue and others are moved to Delay_queue. Hence, only those objects within the dark shaded area in Figure 1 are left in the Return_queue. The objects in this area form an ordering by the distance from the query point Q. The rest of the objects, without most of their distances computed, are put into the Delay_queue.

Aside from the aforementioned two queues, a third Main_queue, which retains only leaf and intermediate nodes is also maintained. This is different from the existing method, which maintains only a single queue. All three queues are maintained via a priority queue though the ordering criteria is all different; the Return_queue is based on the distance from the query point, the Main_queue is based on MINDIST, and the Delay_queue has no special ordering.

Returning an object occurs only from the Return_queue. Upon a request the distance of the first object in the Return_queue is compared with MINDIST of the first node of the Main_queue. If the distance is smaller than MINDIST, the object can be returned in answer to the request. If not, this means that an object of the first node of the Main_queue may be closer to Q, and so, the node is popped in order to find the nearest object. When the popped node of the Main_queue is an intermediate node, each child of that node is inserted to the Main_queue according to their distance to the query point Q. When the popped node of the Main_queue is a leaf node, for each child object of the node, two filtering conditions mentioned above are checked, and the objects are sent to either the Return_queue or the Delay_queue. At this time, to guarantee the consistency of the Return_queue, if the distance of the farthest object of the Return_queue is shorter than the distance of the farthest object of the popped leaf node that satisfy the filtering conditions, only the objects of popped leaf node which are shorter than or equal to the distance of the farthest object of the Return_queue are sent to the Return_queue. If not, to the Delay_queue, objects of the Return_queue are moved which are farther than the distance of the farthest object of the popped leaf node that satisfy the filtering conditions and, to the Return_queue, all objects of the popped leaf node that satisfy the filtering conditions are sent. Next, a response is then returned from the updated Return_queue.

## 4. Experiments

For the ranking query algorithm, we constructed the 12, 27 dimensional indexing R-*tree using 50,000 images. We measure the return time of up to 1000 object, starting from the closest to the furthest, one object at a time. Results are compared with existing ranking query algorithm. For 12 dimensional index tree, when using VP filtering, the return time is ,slightly but always, shorter than without filtering as presented in figure 2.

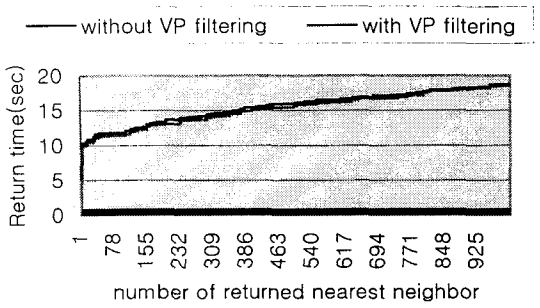


Figure 2. 12 dimensional data return time comparison

As shown in figure 3, in 27 dimensional data, proposed ranking query technique shows much better performance improvement than 12 dimension. That means that as the dimension grows, the gain from the new algorithm is increased.

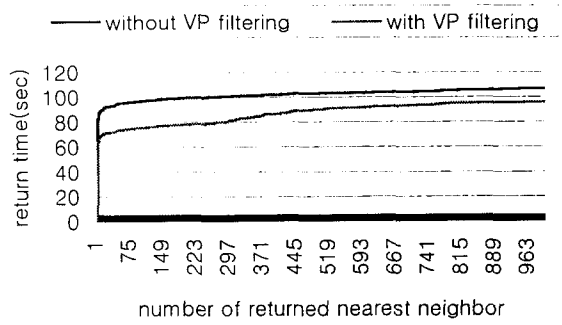


Figure 3. 27 dimensional data return time comparison

## 5. Conclusions

In this paper, we proposed a new ranking query processing technique using VP filtering for R-tree variants. It utilizes the filtering concept of the MVP-tree, but retains the dynamic characteristics of the R-tree. The only overhead of this VP filtering mechanism, which is insignificant and non-existent during query processing time, is the VP distance computation time of objects during tree construction time. This computation is required only once for each of the objects. Experiments show that significant performance gains are obtained compared to previously existing algorithm.

## ACKNOWLEDGEMENTS

This research is supported by KOSEF (Korea Science and Engineering Foundation) under grant no. 98-0102-0901-3.

## References

[1] Antonin Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching", Proceedings of the ACM SIGMOD Conference, pages 47-57, 1984.

[2] Flip Korn, NikolasSidiropoulos, Christos Faloutsos, Eliot Siegel, and Zenon Protopapas, "Fast Nearest Neighbor Search in Medical Image Databases", Proceedings of the VLDB Conference, pages 215-226, 1996.

[3] Jeffrey K. Uhlmann, "Satisfying General Proximity/Similarity Queries with Metric Trees", Information Processing Letters, Vol. 40, pages 175-179, 1991.

[4] Nick Roussopoulos, Stephen Kelley, and Frederick Vincent, "Nearest Neighbor Queries", Proceedings of the ACM SIGMOD Conference, pages 71-79, 1995.

[5] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger, "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles", Proceedings of the ACM SIGMOD Conference, pages 322-331, 1990.

[6] Tolga Bozkaya and Meral Ozsoyoglu, "Distance-Based Indexing for High-Dimensional Metric Spaces", Proceedings of the ACM SIGMOD Conference, pages 357-368, 1997.

[7] Thomas Seidl and Hans-Peter Kriegel, "Optimal Multi-Step k-Nearest Neighbor Search", Proceedings of the ACM SIGMOD Conference, pages 154-165, 1998.