Boosting MBR Based kNN Search Over Multimedia Data by Approximate Pruning Metric

Yang Shuguo
School of Mathematics and Physics , Qingdao Science and Technology University
Qingdao, China

Li Chunxia
School of Mathematics and Physics, Qingdao Science and Technology University
Qingdao, China

***Abstract*—MBR (Minimum Bounding Rectangle) has been widely used to represent multimedia data objects in R*-Tree family indexing techniques. In this paper, in order to improve the performance of kNN searching over multimedia data, we propose an approach to reduce the computation cost of *MINMAXDIST* by using its approximate upper bound instead of its precise value, and then we use it to construct two stronger heuristics for kNN pruning, which are helpful to avoid visiting unnecessary data objects and MBRs. The experimental results show that the proposed approach can reduce the computation cost and boost the overall performance in R*-Tree based kNN searching tasks.**

Keywords- MBR; kNN search; Multimedia indexing; Multidimensional pruning

I. INTRODUCTION

The similarity queries on feature vector have been widely used to perform the content-based retrieval of multimedia data [1]. A set of multimedia data objects similar to the query objects can be retrieved by searching feature vectors which are close to the given query objects. To apply the similarity query technique to large-scale repositories, it is urgent to develop multidimensional index structure nearest neighbor queries.

R*-Tree [2] and its variations use MBR (Minimum Bounding Rectangle) to represent multimedia data objects and conduct efficient similar search by efficiently organizing these MBRs. The MBR based index structures has been employed in various domains. In the MBR based similar queries task, especially kNN queries which return the most similar k objects, ***MINDIST*** and ***MINMAXDIST*** [3] remain the most popular pruning metrics. However, in multimedia database, when the dimensionality is high, the actual distance computations, distance-based pruning metrics, especially the ***MINMAXDIST***, will become expensive. Then the I/O cost becomes a minor performance factor and the computations costs become the dominant component during the search.

Previously works can be divided into two categories: The first type of work often employ MBR based objet approximation to accelerate the search performance. For example, rectangles are employed for approximating the complex spatial objects such as polygons or CAD models because operations on the exact complex object representation are usually much more expensive than on the object's approximate representation. Their typical application scenarios for MBR approximations include GIS [4] and uncertain data retrieval [5]. The second type of rectangular approximations focuses on commonly integrated into spatial pruning methods to speed up the similar queries. In this scenario, the pruning techniques use similar searching over multi-dimensional vector spaces, time series data and spatial-temporal data [6-7].

In order to improve the performance of kNN search over multimedia data, we develop an approach to quickly determine a tightly approximate upper bound value of ***MINMAXDIST***, and then we use this approximate upper bound instead of its precise value to prune and save the computation cost. The remainder of this paper is organized as follows. Section II details the definitions of ***MINDIST***, ***MINMAXDIST*** and how to use them to conduct pruning in the kNN search tasks. We present our approach that how to quickly determine an upper bound of ***MINDMAXDIST*** in section III. Then we evaluate its performance in section IV.

II. MINDIST AND MINMAXDIST BASED KNN SEARCH

In R*-Tree like MBR based index structures, a rectangle ***R*** in n-dimensional Euclidean space E(n) will be defined by two end points ***S*** and ***T*** of its major diagonal. Thus a rectangle ***R*** often denotes as following:

$$\boldsymbol{R=(S, T)}$$

Where $S=[s_1, s_2, \dots, s_n]$, $T=[t_1, t_2, \dots, t_n]$ and $s_i \le t_i$ for $1 \le i \le n$.

This definition of rectangle ***R*** will be referred in the definition of ***MINDIST*** and ***MINMAXDIST***.

A. *MINDIST and MINMAXDIST*

MINDIST(Q, R)is the minimum distance between a query point ***Q*** and a bounding rectangle ***R***. It could be denoted as following:

$$\boldsymbol{MINDIST(Q, R)}= \sum_{i=1}^{n} | q_i - r_i^2 | \qquad (1)$$

Where $r_i = \begin{cases} s_i & if\ q_i < s_i \\ t_i & if\ q_i > t_i \\ q_i & otherwise \end{cases}$.

Notice that if ***Q*** is inside rectangle ***R*** or ***Q*** is on the perimeter of ***R***, then ***MINDIST(Q, R)*** = 0. Otherwise, it is equal to the square of the minimal Euclidean distance from ***Q*** to any point on the perimeter of ***R***. It is easy to know that the computation complexity of ***MINDIST*** is linear to the number of dimensionality, denoted as *O(n)*. ***MINDIST (Q, R)*** is used to determine the closest object to ***Q*** from all those enclosed in

This paper is fund by ShanDong Province Nature Science Found (Grants No:ZR2009GM017)

R. During the kNN search, it can be employed to decide which MBR to search first and produced a sorted queue of MBRs waiting for further processing. In order to avoid visiting unnecessary MBRs in the above sorted queue, we need an upper bound of the (***k***th) nearest neighbor distance to any object inside an MBR. This upper bound guarantees there is an object within the MBR at a distance less than or equal to it. So we call this upper bound as ***MINMAXDIST***.

Given a query point and a rectangle ***R***=(***S***, ***T***), ***MINDIST(Q, R)*** can be defined as following:

$$MINMAXDIST(Q, R) = \min_{1 \le k \le n}(|q_k - rm_k|^2 + \sum_{1 \le i \le n, i \ne k} |q_i - rM_i|^2) \quad (2)$$

Where: $rm_k = \begin{cases} s_k & if\ q_k \le \frac{s_k + t_k}{2} \\ t_k & otherwise \end{cases}$, $rM_i = \begin{cases} s_i & if\ q_i \ge \frac{s_i + t_i}{2} \\ t_i & otherwise \end{cases}$.

According to the definition of ***MINMAXDIST***, we know that ***MINMAXDIST*** is the minimum distance that guarantees the presence of an object ***O*** in ***R*** whose distance from ***Q*** is within this distance. ***MINMAXDIST*** (***Q, R***) is the distance from ***Q*** to the closet corner of ***R*** that is adjacent (i.e., connected with an edge of ***R***) to the corner that is farthest from ***Q***. A value larger than or equal to ***MINMAXDIST*** will be bound to catch some objects inside an MBR, but a smaller distance could miss some objects. Figure 1 is an example of ***MINDIST*** and ***MINMAXDIST*** in 2-D space.

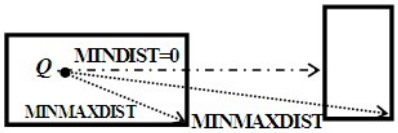


Figure 1. Example of ***MINDIST*** and ***MINMAXDIST*** in 2-D spaces

The construction of **MINMAXDIST** is more complicate and costly than ***MINDIST***, it could be described as follows: for each ***k*** select the hyperplane H_k=rm_k which contains the closer of the tow surfaces of the MBR orthogonal to the ***kth*** space axis. One of the surface could be represented as H_k=S_k, while the other one is denoted as H_k=T_k. The point V_k=(rM_1, rM_2, … , rM_{k-1}, rm_k, rM_{k+1}, …, rM_n), is the farthest vertex from ***Q*** on this surface. Then ***MINMAXDIST*** could be characterized as the minimum of the squares of the distance to each of these vertexes. If we take the distance from ***Q*** to the furthest vertex on the MBR as $F = \sum_{1 \le i \le n} |q_i - rM_i|$, then the ***MINMAXDIST*** will be obtained by iteratively selecting the minimum of $F - |q_k - rM_k|^2 + |q_k - rm_l|$ for $l \le k \le n$.

B. Pruning heuristics based on MINDIST and MINMAXDIST

Since the ***MINDIST*** and ***MINMAXDIST*** provide the lower bound and upper bound from the query point ***Q*** to the MBR, the following 3 most popular heuristics [3] have been proposed to prune the search based on R*-Tree like index structures. The following description is target to nearest neighbor (1NN) search. It is easy to extend it to kNN search by: (1) Keeping a sorted buffer of at most ***k*** current nearest neighbors; (2) Pruning is done according to the distance of the furthest nearest neighbor in this buffer.

H1: An MBR M with ***MINDIST (Q, R)***, which is greater than the ***MINMAXDIST(Q, R')*** of another MBR R', is discarded because it cannot contain the NN. This is used in *downward pruning*. An example of this heuristics is shown as figure 2.

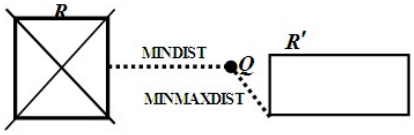


Figure 2. Example of Heuristics 1

H2: An actual distance from ***Q*** to a given object ***O***, which is greater than the ***MINMAXDIST(Q, R)*** for MBR R, can be discarded because M contains an object ***O***' which is near to ***Q***' that is nearer to ***Q***. This is also used in *downward pruning*. An example is illustrated in figure 3.

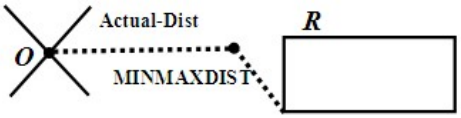


Figure 3. Example of Heuristics 2

H3: Every MBR M with ***MINDIST***(***Q, R***) which is greater than the actual distance from ***Q*** to a given object ***O***, is discarded because it cannot enclose an object nearer than ***O***. This is used in *upward pruning*. An example is depicted in figure 4.

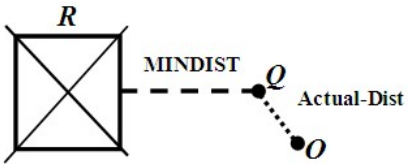


Figure 4. Example of Heuristics 3

III. INMAXDISTUP AND ITS PRUNING HEURISTICS

A. Quickly Obtaining Approximate Upper Bound of MINMAXDIST

According to the analysis in section II, we conclude that if a distance from query point ***Q*** to the MBR is larger than or equal to ***MINMAXDIST***, then it will catch some objects inside an MBR as the candidate results, but a smaller distance could miss some objects. So if we want to approximate the ***MINMAXDIST*** without the loss of effectiveness/precision of kNN search, we must obtain and use its (tightly) approximate upper bound in the pruning. Since the main reason that we use an approximate ***MINMAXDIST*** to replace its original precise value in heuristics ***H1*** and ***H2*** is to save the costly computation, thus the required approximate upper bound of ***MINMAXDIST*** can be quickly determined.

In section II, we know that ***MINMAXDIST*** is obtained by iteratively selecting the minimum of $|q_k - rm_k|^2 + \sum_{1 \le i \le n, i \ne k} |q_i - rM_i|^2$. When the dimension is high,

if we maximize the Euclidean distance computation component of $|q_i - r_i|$ in every dimension, we will get an approximate upper bound of ***MINMAXDIST***. Based on this idea, we denote this upper bound of ***MINMAXDIST*** as $\boldsymbol{MINMAXDIST_{up}}$, its definition is as following:

$$MINMAXDIST_{up}(Q, R) = \sum_{1 \le i \le n} |q_i - r_i|^2 \qquad (3)$$

Where: $r_i = \begin{cases} s_i & |q_i - s_i| > |q_i - t_i| \\ t_i & |q_i - t_i| > |q_i - s_i| \end{cases}$.

Recall the definition of ***MINDIST***, its computation requires comparing $\boldsymbol{q_i}$ with $\boldsymbol{s_i}$ or $\boldsymbol{t_i}$, and $|q_i - r_i|$ (where $\boldsymbol{r_i}$ is either equal to $\boldsymbol{s_i}$ or $\boldsymbol{t_i}$) is also computed. Since they are common computation costs in $\boldsymbol{MINMAXDIST_{up}}$, thus we can cache the needed intermediate results when we compute ***MINDIST***, and then reuse it when computing $\boldsymbol{MINMAXDIST_{up}}$.

For example, when computing the $|q_i - r_i|$ for ***MINDIST*** in ***ith*** dimension, we can know $|q_i - s_i| > |q_i - t_i|$ or $|q_i - t_i| > |q_i - s_i|$, if $|q_i - s_i| > |q_i - t_i|$ are true, then $|q_i - r_i|$ will be computed for ***MINDIST***, this intermediate result is also required by $\boldsymbol{MINMAXDIST_{up}}$, thus we store it in a buffer array and then reuse it when computing $\boldsymbol{MINMAXDIST_{up}}$.

The new obtained approximate upper bound $\boldsymbol{MINMAXDIST_{up}}$ forms two strong heuristics ***H1'*** and ***H2'***, which could be used to replace ***H1*** and ***H2*** to achieve faster pruning.

H1': A rectangle ***R*** is discarded if there is another ***R***' and ***MINDIST*** (***Q, R***) is greater than an approximate upper bound of $\boldsymbol{MINMAXDIST_{up}}$ (***Q, R***).

H2': An object ***O*** is discarded if there exists an ***R*** such that ACTUAL-DIST(***Q, O***) is greater than $\boldsymbol{MINIMAXDIST_{up}}$ (***Q, R***).

B. MINMAXDISTup based kNN Search Algorithm

The (k) nearest neighbor search algorithm is described as follows:

- Initializing the nearest distance as infinite distance.
- Traversing down the tree from the root. At each newly visited non-leaf node, we sort its MBR into an Active Branch List (ABL) by the MBR's ordering metric (***MINDIST*** or $\boldsymbol{MINMAXDIST_{up}}$).
- Applying pruning heuristics ***H1'*** and ***H2'*** to the ABL to remove unnecessary branches. In this step, $\boldsymbol{MINMAXDIST_{up}}$ is quickly obtained by reusing part of intermediate computations of ***MINDIST***.
- Iterating on this ABL until it is empty. For each iteration, we apply the step 3 recursively to its children node.
- At a leaf node, we compute each object's distance from the query point, compare it with the distance of the (***k***th) nearest distance so far, and update it if necessary.
- At the return from recursion, we apply pruning heuristics ***H3*** to remove unnecessary branches.
- When the ABL is empty, (***k***th) the nearest neighbor is returned.

IV. EXPERIMENTAL RESULTS

A. Implementation and Experimental Settings

We implement the proposed $\boldsymbol{MINMAXDIST_{up}}$ based pruning heuristics into SR-Tree to conduct efficient kNN search for high-dimensional multimedia data, we name this revised index structure as **ASR-Tree** (*Approximate SR-Tree*). Its performance is evaluated on the large-scale MSRA-MM 2.0 multimedia data set and compared with R*-Tree and SR-Tree.

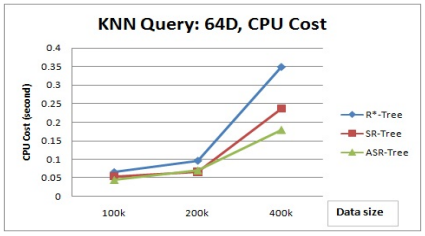
MSRA-MM 2.0 data set contains both images and video shot data. In this study, we use part of image data set to estimate the kNN search efficiency of our proposed ASR-Tree. The size of the involved data set is from 100k to 400k, the number of its dimensional varied from 16 to 256. In all the results, the page sizes of the three index structures are fixed to 8KB. We use two metrics to measure the kNN search performance. They are *CPU time* and *number of page* access. In the kNN search evaluation, the queries objects is 100 images including animals, tools, sports and landscapes, etc. They have been depicted in figure 5. In each experiment, we issue all the 100 query images to conduct kNN search tasks, and repeated it 5 times to get the average value as the final results.

Figure 5. 100 Queries Images for kNN Search Task

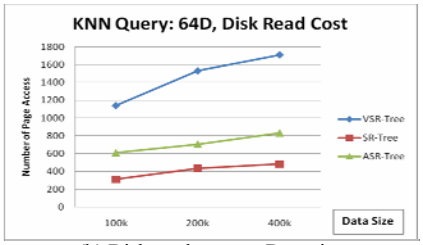
The experiments are conducted on a computer with Ubuntu Linux 10.04. Its CPU is Intel i3 M330 CPU (2.13 GHz), it has 2GB memory and equipped with a HITACHI SATA 5400rpm disk.

B. Effect of Data Size

In the first experiment, we study the performance of kNN search over R*-Tree, SR-Tree and ASR-Tree when data size increases from 100k to 400k. The figure 6 shows the performance of query processing in terms of CPU cost and disk read cost. In the experiment, the ***k*** of kNN is set to be 10 and the three indexes are constructed in the same optimal bulk load manner.



(a) CPU cost vs. Dimensionality



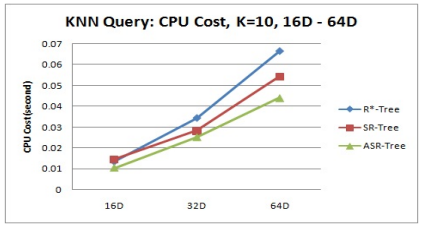
(b) Disk read cost vs. Data size

Figure 6. Effect of Data Size

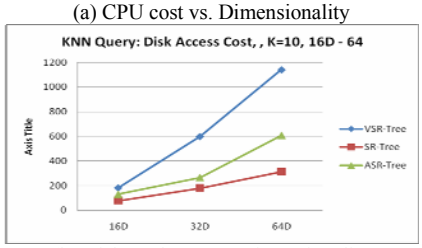
From above figures, it is evident that ASR-Tree outperforms the other two methods in terms of the response time for the three datasets. The graph shows when data size is less than 400k, the page access overhead of SR-Tree and ASR-Tree hold a big gap but their total CPU costs nearly are the same. This phenomenon is considered as the evidence that the CPU cost become the dominant component in the overall query processing cost in MBR based index structures when dimensionality increases.

C. *Effect of Dimensionality*

In our second experiment, we measure the impacts of the dimensionality in MBR based kNN search task. The detailed experimental results are presented in figure 7. As the experiments in last section, ***k*** is also set to be 10 and indexes are constructed in the optimal bulk load manner. In this experiment, data size is 100k.



(a) CPU cost vs. Dimensionality



(b) Disk read cost vs. Dimensionality

Figure 7. Effect of Dimensionality

From the experimental results, we can see that although ASR-Tree has more page access than other two approaches, it still have less response time than VSR-Tree and SR-Tree. Because of the only difference between the SR-Tree and ASR-Tree is its pruning strategies, they are same in index construction and maintenance, thus we can owe the superiority of ASR-Tree to the MINMAXDIST_{up} metric and the two enhanced pruning strategies ***H1***' and ***H2***'.

D. *Effectiveness of Enhanced Pruning Heuristics*

To evaluate the effectiveness of enhanced pruning heuristics ***H1'*** and ***H2'***, we devise this experiment to evaluate the ration of (visited leaves node / all the leaves nodes). The experimental settings are: ***k*** of kNN is 1000, data size is 400k and the dimensionality is up to 256. The results are plot into figure 8.

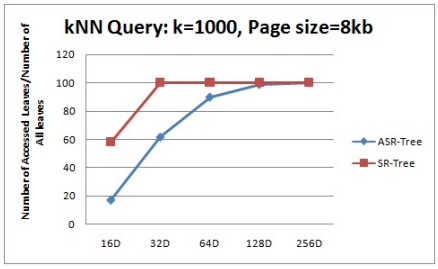


Figure 8. Effect of Enhanced Pruning Heuristics

From section C, we conclude that the differences between ASR-Tree and SR-Tree are the pruning heuristics ***H1'*** and ***H2'*** during query processing. The results are clearly shown in figure 8. When dimensional is less than 64, the effectiveness of ***H1'*** and ***H2'*** is evident, and many nodes visited in SR-Tree will be filtered in ASR-Tree. That is why ASR-Tree outperform R*-Tree and SR-Tree in the first two measurements. However, with the increase of dimensionality, the intrinsic of R*-Tree family techniques make them suffered and visit almost all the leaves node, then the performance decrease rapidly, this is the so-called *curse of dimensionality*. In this case, traditionally R-Tree based pruning techniques only aims to reduce the I/O cost may become impractical, the high-dimensional kNN search pruning techniques should consider reduce both computation cost and I/O cost to improve the overall performance.

In this paper, we propose an approach to quickly obtain a tightly approximate upper bound value of ***MINMAXDIST***, and then we use this approximate upper bound instead of its precise value to improve the search performance.

REFERENCES

[1] Christian Böhm, Stefan Berchtold, Daniel A. Keim: Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. ACM Comput. Surv. (CSUR) 2001.33(3), pp. 322-373.

[2] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, Bernhard Seeger, The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. SIGMOD 1990, pp. 322-331.

[3] Nick Roussopoulos, Stephen Kelley, Frédéic Vincent. Nearest Neighbor Queries. SIGMOD 1995, pp. 71-79.

[4] Jinchuan Chen, Reynold Cheng. Efficient Evaluation of Imprecise Location-Dependent Queries. ICDE 2007, pp. 586-595.

[5] George Beskales, Mohamed A. Soliman, Ihab F. Ilyas. Efficient search for the top-k probable nearest neighbors in uncertain databases. PVLDB, 2008, 1(1), pp. 326-339.

[6] Marios Hadjieleftheriou, George Kollios, Vassilis J. Tsotras, Dimitrios Gunopulos: Indexing spatiotemporal archives. VLDB J. (VLDB), 2006, 15(2), pp. 143-164.

[7] Yufei Tao, Dimitris Papadias, Qiongmao Shen. Continuous Nearest Neighbor Search. VLDB 2002, pp. 287-298.