Fast k nearest neighbour search for R-tree family

Joseph Kuan Paul Lewis

Multimedia Research Group
Dept. of Electronic and Comp. Science,
University of Southampton,
Highfield, Southampton,
SO9 5NH,
United Kingdom.

Abstract

A simplified k nearest neighbour (knn) search for the R-tree [6] family is proposed in this paper. This method is modified from the technique developed by Roussopoulos et al. [10]. The main approach aims to eliminate redundant searches when the data is highly correlated. We also describe how MINMAXDIST calculations can be avoided using MINDIST as the only distance metric which gives a significant speed up. Our method is compared with Roussopoulos et al.'s knn search on Hilbert R-trees [7] in different dimensions, and shows that an improvement can be achieved on clustered image databases which have large numbers of data objects very close to each other. However, our method only achieved a marginally better performance of pages accessed on randomly distributed databases and random queries far from clustered objects, but has less computation intensity.

1 Introduction

The R-tree [6] is one of the more popular approaches for spatial access methods and a number of R-tree variants have been developed. The R-tree is simply a hierarchical tree where the higher level node is an MBR (minimum bounding rectangle) that encloses a set of child MBRs or data objects in the lower level.

Beckmann et al. [1] developed the R*-tree which is more efficient in insertion and space utilization than the R-tree. They proposed a force-reinserted technique to cut down the possibility of splitting a full node. It has been used for image applications, such as QBIC [5], and content-based retrieval and navigation in a hypermedia system [9].

Kamel et al. [7] developed an improved R-tree, the Hilbert R-tree, which uses a Hilbert space filling curve for grouping data objects [2] [3]. They show that the Hilbert R-tree outperforms the R*-tree. The Hilbert space filling curve has been shown to give a better space localization than other space filling curves [8].

Although there has been much research on the R-tree and its variants, there is hardly any development in the k nearest neighbour search for the R-tree family. Roussopoulos et al. [10] developed the first knn search for the R-tree. It measures the optimistic (MINDIST) and pessimistic (MINMAXDIST) distance between the query object and the multi-dimensional MBRs with different pruning strategies. However, the main application area motivating our work is for content based retrieval from large image databases, using high dimensional image feature vectors. Since multiple instances of similar images occur, the data tend to be clustered in the feature space. If a clustered database is indexed by the R-tree, then a number of nodes' MBRs will have a large overlap with their siblings. This may cause Roussopoulos et al.'s knn method to search unnecessary rectangles.

A simplified version of knn search is introduced. The method is similar to Roussopoulos et al.'s knn method. All overlapping rectangles with the query point are gathered and processed together in each level of the R-tree. The calulation of Roussopoulos et al.'s pessimistic distance between MBRs and the query point is avoided, speeding up the knn search performance. In section 2, the basic structure of the R-tree is shown. In section 3, Roussopoulos et al.'s knn search method is briefly explained. The simplified version of knn search for R-tree is described in section 4. Finally, comparative experiments are performed between Roussopoulos et al.'s knn search and the simplified knn search in terms of the number of dimensions and the value of k with the Hilbert R-tree at section 5. We used a clustered database for all the experiments (image based feature vectors derived from sub-images).

2 R-tree

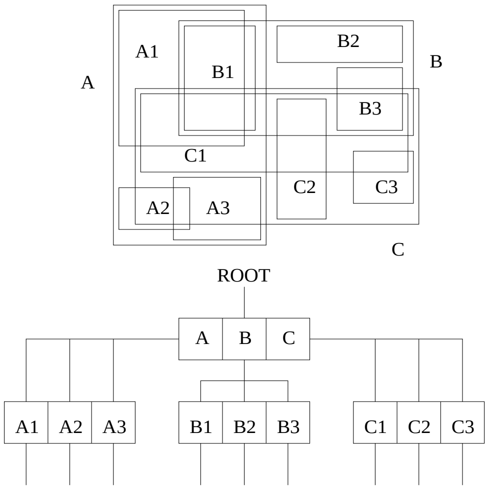

Figure 1: Example of R-tree indexing structure

The R-tree indexing structure is basically constructed of a number of multi-dimensional rectangles, and is balanced dynamically with all the data objects stored in the same level. Each rectangle in the higher level is formed by enclosing multiple child rectangles from a lower level. At the bottom level (Leaf-level), each MBR is generated by enclosing data objects. Each node in the Leaf-level or non-Leaf level is allowed to have a number of children between upper-bound and lower-bound values. There is no minimum number of children required in the root level. The root node can store any number of children up to the maximum child allowance.

3 The k Nearest Neighbour Search

Roussopoulos et al.'s *knn* search used the squares of Euclidean distances as a measurement between objects. They derive MINDIST and MINMAXDIST to measure the optimistic and pessimistic distances between a query point and an MBR. Figure 2 shows the MINDIST and MINMAXDIST in a 2D case.

The MINDIST is the minimum possible distance between a query point and an MBR, such that there is no child rectangle or data point inside the MBR that can

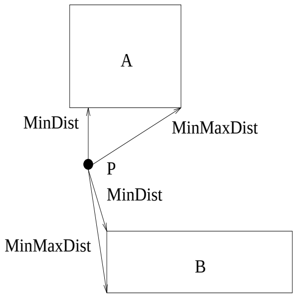

Figure 2: Example of MinDist and MinMaxDist

have a smaller distance value than this. The mathematical calculation for MINDIST between a query point, P, and an MBR, R is:

$$MINDIST(P,R) = \sum_{i=1}^{n} |p_i - r_i|^2$$

where n is the number of dimensions of P and R, p_i is the coordinate of point P in ith dimension, and

$$r_i = \begin{cases} s_i & \text{if } p_i < s_i \\ t_i & \text{if } p_i > t_i \\ p_i & \text{otherwise} \end{cases}$$

where s_i and t_i are the minimum and maximum values of the ith dimension of rectangle R. If P is located inside the rectangle R or on the perimeter of R, then the MINDIST = 0.

The MINMAXDIST is the minimum distance that guarantees a data object can be found within the distance. The MINMAXDIST is more computationally intensive than the MINDIST shown by the calculations in [10]. For the definition of MINMAXDIST, we let the readers refer to the Roussopoulos et al. paper, as our paper concentrates on using MINDIST for *knn* search.

Roussopoulos et al. proposed using a branchlist to browse a node to see whether it has any children within the maximum distance of the *knn*. The branchlist holds the MINDIST and MINMAXDIST of each MBR to the query object. The branchlist is sorted in ascending order based on the MINDIST value. Three pruning strategies were suggested for branchlists and the *knn* distance list. They are used to reduce the search path for *knn*

1. Any rectangle M with a MINDIST that is greater than the MINMAXDIST value of another rectangle M' can be pruned from the branchlist.

2. The furthest data object in the *knn* list can be omitted if its Euclidean distance to the query point is longer than the MINMAXDIST of a rectangle.

3. Any rectangle that has MINDIST greater than the furthest distance in the *knn* list can be discarded.

4 Simplified k Nearest Neighbour Search

In the simplified *knn* search, two main approaches that are different to Roussopoulos et al.'s method are taken. These can achieve a better performance.

1. Collect all overlapping MBRs to the query point and expand all the MBRs in each lower level as a larger branchlist. This is performed recursively until the LEAF level.

2. Avoid calculating MINMAXDIST. Only use MINDIST and the maximum distance value in the *knn* list for pruning (Roussopoulos et al.'s pruning strategy 3).

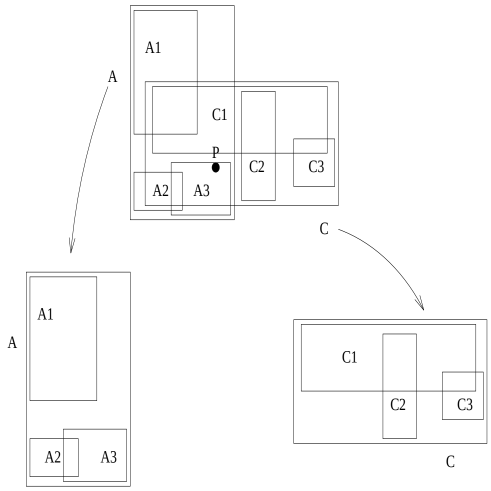

Figure 3: Overlapping rectangles with a Query Point

For *knn* search, the first priority MBRs to search for are those with zero MINDIST. The MBRs (A and C in figure 3) overlap the query point, P, as these MBRs contain most of the closest data objects that can be found (unless the query point is very near to the boundary of an MBR), compared to the non-overlapping MBRs. Figure 3 shows a simple example explaining why the overlapping rectangles should be expanded together and a branch-and-bound search performed. If a non-leaf MBR, A, is browsed through first, then a branchlist of leaf MBRs $A3, A1$, and $A2$ is created (sorted with MINDIST value) and searched for nearest neighbours. However, sometimes the MBR C is browsed first, as both C and A MBRs have zero MINDIST value. With searching MBR C first, some of the real *knn* could be found but probably not all of them. Assume $C1$ and $A3$ both contain all the real *knn*; either the selection of A or the selection of C creates extra searches. If A and C are treated together with a branchlist of MBRs as: $A3, C1, C2, A1, A2$, and $C3$, then $A3$ and $C1$ can be searched in priority, and some of the other MBRs can be pruned by Roussopoulos et al.'s strategy 3 [10]. However, this situation is less likely to happen in randomly distributed databases indexed by R-tree because there is a very small overlap between MBRs and the query point, or there may be none.

In order to obtain a faster *knn* search method, the MINMAXDIST calculation can be avoided by using MINDIST as the only ordering metric. MINDIST is a less computational measurement than MINMAXDIST and it is very useful. It gives information regarding whether a query point is overlapping with an MBR (MINDIST $=0$) or the minimum distance value if it is not overlapping. Roussopoulos et al. also suggested that MINDIST can be used as effectively as MINMAXDIST if the dead space (space in an MBR that contains no data objects or child MBRs) is very small (figure 7 in [10]). Indeed, our view is that the MINDIST can always be effective with or without the MINMAXDIST and no matter what the size of dead space. This is because if there is an MBR which has a MINDIST value smaller than the maximum distance in the nearest neighbours list, then we cannot be confident in considering that no closer child nodes or data objects can be found in the MBR without searching through, despite any factors such as dead space and MINMAXDIST values. The only way to achieve maximum efficiency is if the *real* minimum distance between the query point and a data object inside an MBR can be obtained without exploring down to leaf nodes. As far as we know, there is no possible way of achieving this in high dimensional space. In fact, the pruning strategy 3 in section 3 can be applied individually everytime a *k*nn list has a new closer object, and still have the same result and efficiency as when combined with the other two strategies. Moreover, we have found that Roussopoulos et al.'s knn search can be made up to 30% faster by using solely MINDIST to prune MBRs instead of using MINDIST and MINMAXDIST.

5 Experiments

5.1 Database

A database is constructed from 40,000 texture images (from a CDROM), and each grey level image size is 64×64. Each image is submitted to a texture features extractor, Statistical Geometrical Features [4], which yields a 16 dimensional feature vector (real values). Then each feature is normalized between 0 - 100, and indexed by Hilbert R-tree.

5.2 *knn* query

The number of nearest neighbours to search for (k) is set to 10, 30, and 50. We randomly generate 20,000 data query points for the image database with dimensions between 2 and 16 in steps of 2. As the dimension goes up, the distances between these 20,000 randomly generated query points are more sparsely distributed and the performance results of the query points in high dimensions can be as close as a query outside clustered data objects.

For a clustered database retrieval application where clusters correspond to object classes, it is likely that queries will not be randomly distributed but will probably be close to or inside the cluster regions. For this reason, another 20,000 query points are generated from another set of similar textures which are evenly spread over the regions of the clustered data objects.

The number of nodes accessed to retrieve *knn* is recorded for every query point and the mean is calculated from different values of k and the dimensions.

5.3 Results

Figures 4 and 5 show the performance difference of Roussopoulos et al. and our simplified *knn* search for 20,000 randomly distributed query points and clustered query points respectively.

In figure 4, the simplified *knn* search performance is only slightly improved for the 20,000 randomly generated query points, (not more than 10% fewer nodes accessed). If a query data point is far outside the clustered data, then there are fewer overlaps between MBRs around the clustered boundary. In general, the clustered query points in figure 5 have more nodes accessed than the randomly generated points, except with 16 dimensions. However, the shape of the graph mainly depends on how close the data points are oriented in the database in each dimension, and it is very hard to visualise such cases in high dimensions. In fact, there

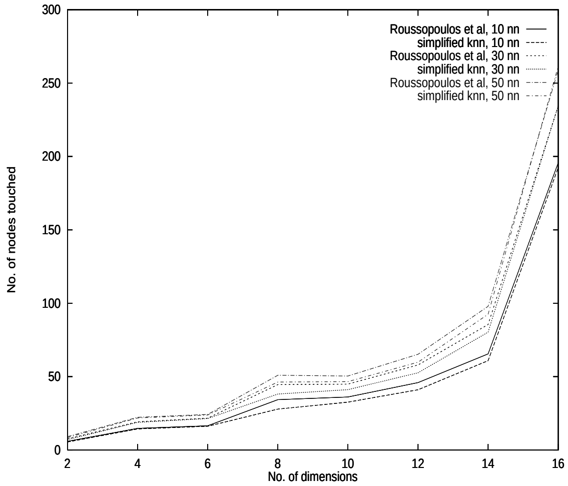

Figure 4: Comparison of Roussopoulos et al. and simplified knn search with Hilbert R-tree and random query data points

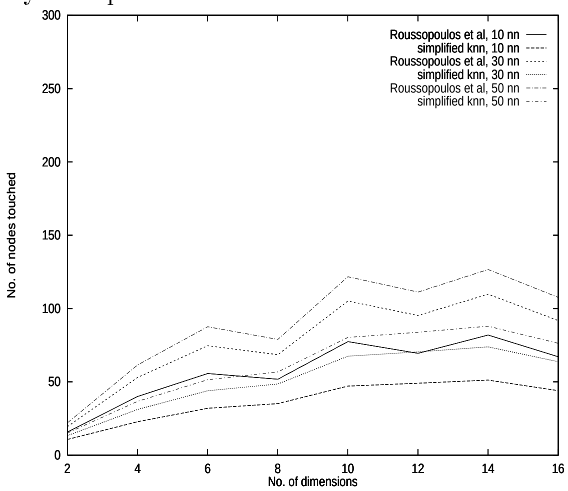

Figure 5: Comparison of Roussopoulos et al. and simplified knn search with Hilbert R-tree and clustered query data points

is a clear performance difference in figure 5; the simplified *knn* search has considerably less nodes accessed to retrieve nearest neighbours compared to Roussopoulos et al.'s search. The results show that the simplified *knn* search can access up to 40% fewer nodes than the other knn search. For the clustered database, our *knn* search can give a reasonable performance even with high dimensions. For our technique, the number of data object comparisons to retrieve 10 nearest neighbours in 2D is about 0.3% of the entire database population, and about 8.0% in 14D for 50 nearest neighbours compared to the other method which was at 1.2% and 12% for 2D and 14D cases respectively.

However, we have also compared both *knn* searches with a synthetic random database, and the improvement of our proposed nearest neighbour search is no better than 3% fewer nodes accessed as expected. There is significantly less overlap between MBRs in the random database than the clustered database, and the simplified *knn* search only performs well when there are several nodes with zero MINDIST value. Therefore, both *knn* search algorithms have almost the same number of nodes accessed on random databases.

Despite the nature of databases, our method can still perform similarity searches with less calculation.

6 Conclusion

A simplified *knn* search is proposed for the R-tree family which reduces the number of nodes accessed when a query point is close to or within the clustered data points in a database. We specified that with only MINDIST values, the *knn* search can be achieved effectively, and giving a faster response than with the MINMAXDIST computation. Roussopoulos et al. *knn* search and our simplied search are compared with Hilbert R-trees indexing an image feature database. Our method shows a consistent improvement with fewer nodes accessed in different numbers of dimensions.

However, our method shows that there is hardly any improvement for synthetic random databases and *knn* query points far from clustered data points. The reason for this is that it is not possible to forsee the real minimum distance between a query point and a data object within an MBR without exploring down to the leaf nodes.

In future work, our algorithm will be presented in more detail with the psuedocode, further reduction on distance computation, and a more generalised *k*nn implementation that can offer the *next k* nearest matches effectively on the basis of the *k*th distance value.

References

[1] N. Beckmann, H.P.Kriegel, R.Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. *ACM SIGMOD*, pages 322 – 331, May 1990.

[2] T. Bially. Space-filling curves: Their generation and their application to bandwidth reduction. *IEEE Trans. on Information Theory*, IT-15(6):658 – 664, November 1969.

[3] Arthur R. Butz. Alternative algorithm for hilbert's space-filling curve. *IEEE Trans on Computers*, C-20(4):424 – 426, April 1971.

[4] Y. Q. Chen, M. S. Nixon, and D. W. Thomas. Texture classification using statistical geometrical features. In *IEEE International Conference on Image Processing*, volume 3, pages 446 – 450, Austin, Texas, November 1994.

[5] Myron Flickner, Harpreet Sawhney, Watne Niblack, Jonathan Ashley, Qian Huang, Byron Dom, Monika Gorkani, Jim Hafner, Denis Lee, Dragutin Petkovic, David Steele, and Peter Yanker. Query by image and video content: The qbic system. *IEEE Computer*, 28(9):23 – 32, September 1995.

[6] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 45 – 57, 1984.

[7] Kamel. I and Faloutsos. C. Hilbert R-tree: an improved R-tree using fractals. In *Proc. of Int. Conf. of Information and Knowledge Management*, pages 490 – 499, 1993.

[8] H. V. Jagadish. Linear clustering of objects with multiple attributes. In *ACM SIGMOD Conf.*, pages 332 – 342, May 1990.

[9] P. Lewis, J. Kuan, S. Perry, M. Dobie, H. Davis, and W. Hall. Navigating from images using generic links based on image content. In *SPIE conference Storage and Retrieval for Image and Video Database*, pages 238 – 248, San Jose, February 1997.

[10] Nick Roussopoulos, Stephen Kelley, and Frédéric Vincent. Nearest neighbor queries. In *Proceedings of the 1995 ACM-SIGMOD Intl.Conf. on Management of Data*, San Jose, CA, June 1995.