Enhanced Nearest Neighbour Search on the R-tree

King Lum Cheung and Ada Wai-chee Fu
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Email: adafu@cse.cuhk.edu.hk

Abstract

Multimedia databases usually deal with huge amounts of data and it is necessary to have an indexing structure such that efficient retrieval of data can be provided. R-Tree with its variations, is a commonly cited indexing method. In this paper we propose an improved nearest neighbor search algorithm on the R-tree and its variants. The improvement lies in the removal of two hueristics that have been used in previous R-tree work, which we prove cannot improve on the pruning power during a search.*

1 Introduction

Multi-media data is being generated at an enormous rate by a lot of applications. The traditional database can deal with text data and provides mechanisms for exact information retrieval. Multimedia data, such as image data, on the other hand, is quite different from text data. Many projects on multimedia databases have been reported e.g. [6, 13, 8]. For such a database, *content-based retrieval* is typically useful. One major advantage of content-based retrieval is that it bypasses the difficult problem of specifying the desired multimedia objects in terms of formal query languages. A popular form of content-based queries employs the query-by-example paradigm. For example, in a collection of images, users can use existing images as query templates and ask the system for images similar to the query images. This is the so-called "like-this" query. Alternatively, user can sketch a picture that serves as the query template.

To support content-based retrieval, often we have to rely on feature extraction capabilities to map each domain object into a point in some k-dimensional space where each object is represented by k chosen features. An example feature vector may be the color components of an image or shot cuts of a video clip. Hence, processing content-based queries typically requires some measurement of similarity between k-dimensional points. The similarity (or distance) between two objects is measured using some metric distance function over the k-dimensional space. The most common metric distance function used is probably the Euclidean distance $d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$. The entire problem is then formulated as storing and retrieving k-dimensional points. In general, these methods are called *Multidimensional Indexing* or *Spatial Access Methods* (**SAMs**) [15].

Some examples of SAMs are [10, 16, 1, 7, 5, 4, 12, 17]. Evidence that nearest-neighbor search in high-dimensional space has inherently high complexity can be found in [9, 3]. In view of the inefficiency, there are attempts to parallelize the processing to speed up the search [2]. Also there are attempts to reduce the number of dimensions effectively [11].

For many indexing methods, the search structure is built in the form of a tree. Inefficiency arises because a lot of tree nodes have to be accessed in order to get the desired objects. In this paper, we discover an enhancement on the nearest neighbor search algorithm for the R-tree and its variants that can speed up the CPU processing, while not increasing the amount of disk I/O. The enhanced algorithm eliminates one computationally expensive step from the previously known algorithm used in nearest neighbor search [14], while preserving the same pruning power.

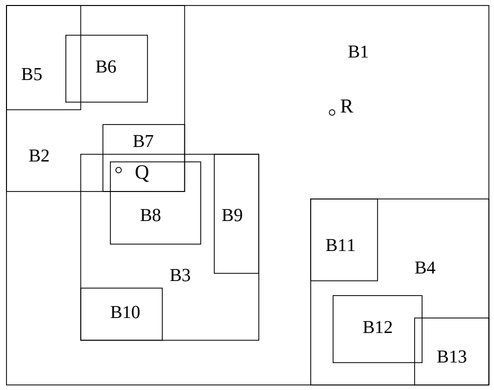

Figure 1. Example of exact search

2 Searching in R-Trees and R*-Trees

In this section, we give a brief overview of exact search and nearest neighbor search on R-trees and R*-trees. For exact search, we determine whether a given query object exists in the index tree or not. If it does, then the object is returned else a failure message is returned. We assume data objects are points in the multi-dimensional search space. In an R-tree or R*-tree, if the minimum bounding rectangle (**MBR**) of a node encloses the query object, then the node may contain the query object. Therefore, beginning from the root, a child node will be accessed if its MBR contains the query object, and the top-down traversal will be iterated until either the leaf level is reached or all minimum bounding rectangles of child nodes do not enclose the query. The process will be terminated when the query object is found, or all nodes whose minimum bounding rectangles enclose the query object are searched but the required object is not found. For example, in Figure 1, for an exact search for a query point Q, the minimum bounding rectangles for B_1, B_2, B_7, B_3, B_8, may be searched. On the contrary, if query point R is to be searched in the same example, the searching will only examine the root of tree, B_1.

N-Nearest Neighbor search aims at searching for N objects which are the nearest ones to the query object among all data objects. The meaning of nearest usually corresponds to the shortest Euclidean distance. Roussopoulos, Kelley and Vincent in [14] suggested an efficient nearest neighbor search algorithm on R-tree. In this algorithm, pruning heuristics are used to discard candidates subtrees, so that less nodes will be accessed while the correct result can be guaranteed at the same time. Two metrics are introduced for the pruning. The first metric, $MINDIST_A$, is the minimum distance from node A to the query $Q = \{q_1, q_2, ..., q_n\}$. It serves as a lower bound on the distance from the nearest neighbor within the MBR of node A to the query. That means, if an object P (note that P is at a leaf node in the R-tree) is nearest to the query among all objects in node A, then $MINDIST_A \leq DIST_P$ must be true where $DIST_P$ is the distance from P to the query. The second metric, $MINMAXDIST_A$, is the minimum of maximum possible distances from a point P to a face of the minimum bounding rectangle A. $MINMAXDIST_A$ serves as an upper bound of distance of the nearest neighbor in MBR of node A to the query. Therefore, if P is an object nearest to the query among all objects in A, then $DIST_P \leq MINMAXDIST_A$ must be true.

Based on these metrics, [14] developed three heuristics to discard nodes which do not contain the nearest neighbor. We shall adopt the following symbols in our discussion.

Symbols	Definition
$MINDIST_A$	minimum distance from node A to the query
$MINMAXDIST_A$	minimum of maximum possible distances from the query point to a face of the MBR A.
NN_DIST_N	distance from the N-th nearest neighbor among searched objects to the query
$DIST_P$	distance from the object P to the query

Heuristic 1 *If* $MINDIST_A > MINMAXDIST_B$, *then node* A *will be discarded.*

Heuristic 2 *If* $DIST_P > MINMAXDIST_B$, *then the object* P *will be discarded.*

Heuristic 3 *If* $MINDIST_A > NN\_DIST_N$, *then node* A *will be discarded.*

Let us call the single nearest neighbor search algorithm in [14] **Algorithm NN**. In this algorithm, pruning depends on Heuristics 1, 2 and 3 in which NN_DIST_N is restricted to NN_DIST_1. A generalization to N-nearest neighbor search is also described in [14]. The generalized algorithm will make use of Heuristics 1, 2 and 3 in the pruning process.

3 An Improved Nearest Neighbor Search Algorithm for R-Tree

Although the pruning heuristics described in the previous section can reduce the number of node access on an R-tree, extra CPU time overhead is introduced by the process of calculating the two metrics. The calculation of *MINMAXDIST* is computationally expensive and has a complexity of $O(d)$ where d is the number of dimensions. Heuristics 1 and 2 make use of *MINMAXDIST*. The overhead is large especially when a large amount of high dimensional data has to be dealt with. It turns out that these two heuristics do not actually increase the pruning power, and so calculation of $MINMAXIST$ is indeed not necessary. In this section, an improved algorithm that does not make use of *MINMAXDIST* and Heuristics 1, 2 will be proposed. The new algorithm will be shown to be as least as powerful as the original one in pruning so that the number of disk accesses during the searching will not be increased.

As discussed in Section 2, we denote by **Algorithm NN** the original single nearest neighbor search algorithm in [14] using Heuristics 1, 2 and 3, and using MINDIST ordering in the *Active Branch List* (see [14]). The improved nearest neighbor search algorithm is given as **Algorithm INN** and is shown in Figure 2. Algorithm INN is similar to Algorithm NN. The major difference between the new and the original search algorithms is that the use of Heuristics 1, 2 and 3 have been replaced by using only Heuristic 3 in the new algorithm.

In the following subsection, we show that the number of node accesses will not be increased by the new algorithm by showing that if a node is pruned by the old algorithm, then it will also be pruned by the new algorithm. We assume that node access corresponds to disk access. Once this is established, we can see that with the new algorithm, the computational cost can be decreased without increasing the amount of disk accesses.

```
ALGORITHM INN:
Procedure NN_Search
Input : NODE      /* node to be visited */
        NN_DIST_temp     /* distance from temporary
                  nearest neighbor to the query */
Begin
    If current node P is at leaf level
    Then
        If DIST_P < NN_DIST_temp
           Set current node to be nearest neighbor
           Update NN_DIST_temp
    Else
        Generate Active Branch List of NODE
        Calculate MINDIST
        Sort the Active Branch List by ascending
             ordering of MINDIST
        For i := 1 to no. of entries in the Active Branch List
            Apply Heuristic 3 to do pruning
            Call NN_Search
End
```

Figure 2. New nearest neighbor search algorithm for R-tree

3.1 Efficiency of Algorithm INN

The following lemmas help to establish the efficiency of the new Algorithm INN.

Lemma 1 *If P is the nearest neighbor among all objects in node A to the query Q, then* $MINDIST_A \leq DIST_P \leq MINMAXDIST_A$.

Proof: By definition, $MINDIST_A$ is the minimum distance from A to the query Q. From the minimal bounding region face property shown in [14], if P is an object nearest to the query among all objects in A, then $DIST_P \leq MINMAXDIST_A$. Therefore, $MINDIST_A$ and $MINMAXDIST_A$ serve as a lower bound and a upper bound to the distance from the nearest neighbor in node A to the query respectively. □

Lemma 2 *If A is an ancestor node of B in a R-tree, then* $MINDIST_A \leq MINDIST_B$.

Proof: This follows from the definition of $MINDIST$. □

We assume that in both algorithms, a tightest upper bound on the distance to the nearest object discovered so far is kept in a variable NN_DIST_{temp}. Next we show that if a node is pruned by the heuristics in Algorithm NN, then it will also be pruned by Algorithm INN.

The first heuristic to be considered is Heuristic 2. Heuristic 2 says that if $DIST_P$ is greater than $MINMAXDIST_A$, then the object P will be discarded. Note that no node access is saved in this case, since the discarded object P is already searched. (Effectively, if P is the nearest object discovered so far, then the NN_DIST_{temp} is updated to be $MINMAXDIST_A$.) Since we are interested here only in the reduction of node access, Heuristic 2 need not be considered. The second heuristic we consider is Heuristic 3. Heuristic 3 says that if NN_DIST_1 is smaller than $MINDIST_A$, then node A will be discard.

Lemma 3 *If a node is pruned by Heuristic 3 using Algorithm NN, it can be also be pruned by Algorithm INN.*

Proof: Assume that during the execution of Algorithm NN, there is a node A such that $MINDIST_A > NN\_DIST_{temp}$, so that node A will be pruned by Heuristic 3. Note that NN_DIST_{temp} is obtained either from a searched object P so that $NN\_DIST_{temp} = DIST_P$, or from the $MINMAXDIST$ of a MBR that contains an object P such that $NN\_DIST_{temp} \geq DIST_P$. That is, we know that $MINDIST_A > DIST_P$. Next suppose Algorithm INN is used, there are three possibilities:

Case 1: Node P is pruned (we say that a node P is pruned if either it is pruned or an ancestor node of P is pruned). Since P and A have a common root, and A is not the root, then an ancestor of P must be searched before A, let this ancestor be C. By Lemma 2, the ancestor C must have a $MINDIST$ smaller than $DIST_P$, and also the nodes in the path in the tree from C to P must all have $MINDIST$ smaller than $DIST_P$. If P is pruned before being searched, then A would also be pruned since the pruning is via Heuristic 3, and A has a greater $MINDIST$ than P's ancestors.

Case 2: Node P is searched after node A is searched. A basic depth-first traversal with $MINDIST$ ordering is followed in the nearest neighbor search for both algorithms, and since P is searched before A in Algorithm NN, it is not possible that A is searched before P in Algorithm INN. Therefore, this case cannot happen.

Case 3: The object P is searched before node A is either searched or pruned. Hence P has been considered as a possible candidate for the temporary nearest neighbor. Let NN_DIST_{temp} be the distance of the nearest neighbor discovered immediately before the search of node A. Since updates in the temporary nearest neighbor can only get closer to the query point, $NN\_DIST_{temp} \leq DIST_P$ must be true. Since $MINDIST_A > DIST_P$, has been given, $NN\_DIST_{temp} < MINDIST_A$ can be derived and the node A will be pruned by Heuristic 3. □

It remains to show that every node which is pruned by Heuristic 1 in Algorithm NN will also be pruned by Algorithm INN. In order to do so, we would make use of the following lemmas.

Lemma 4 *If there are two nodes A and B with the condition $MINDIST_A \leq MINDIST_B$, then $MINMAXDIST_B \not< MINDIST_A$.*

Proof: From Lemma 1, $MINDIST_B \leq MINMAXDIST_B$ must be true for all nodes B. Since the precondition $MINDIST_A \leq MINDIST_B$ is provided, we have $MINDIST_A \leq MINDIST_B \leq MINMAXDIST_B$. Hence $MINMAXDIST_B \not< MINDIST_A$. □

Lemma 5 *If a node B is searched before a sibling node A using Algorithm INN, and $MINMAXDIST_B < MINDIST_A$, then the distance of the temporary nearest neighbor, NN_DIST_{temp}, just before A is either searched or pruned is less than or equal to $MINMAXDIST_B$.*

Proof: Let α be the set of nodes that are searched after B and before the search or pruning of A. (We say that A is pruned when either it is pruned or an ancestor node of A is pruned.) Let B_C be the object in B that is closest to the query point. There are two possible cases:

Case 1: B_C is in α. In this case, B_C has been considered as a candidate for the temporary nearest neighbor, then since we know that its distance is less than or equal to $MINMAXDIST_B$, hence $NN\_DIST_{temp} \leq MINMAXDIST_B$.

Case 2: B_C is not in α. Since $DIST_{B_C} \leq MINMAXDIST_B < MINDIST_A$, by Lemma 1 and 2, all ancestor nodes of B_C have $MINDIST < MINDIST_A$. As B_C is not in α, one ancestor node of B, let it be B', must have been in α and has been pruned by Heuristic 3. That is, node B' is discarded because $MINDIST_{B'} > NN\_DIST'_{temp}$. Hence $NN\_DIST_{temp} < MINDIST_{B'} \leq MINMAXDIST_A$. □

Lemma 6 *If a node is pruned by Heuristic 1 using Algorithm NN, it will be pruned by Algorithm INN.*

Proof: Heuristic 1 says that if $MINDIST_C$ is greater than $MINMAXDIST_D$ then node C is discarded. Without loss of generality, suppose there are two nodes A and B so that Node A is discarded by Heuristic 1 because of Node B in Algorithm NN. Hence A and B are sibling nodes (in the same active branch list) and $MINMAXDIST_B < MINDIST_A$. There are three cases to consider:

Case 1: $MINDIST_A \leq MINDIST_B$.

According to Lemma 1, we have inequalities $MINDIST_A \leq MINMAXDIST_A$ and $MINDIST_B \leq MINMAXDIST_B$. Since $MINDIST_A \leq MINDIST_B$, by Lemma 4, we have $MINMAXDIST_B \not< MINDIST_A$. Therefore, it is impossible that $MINMAXDIST_B < MINDIST_A$ so that node A is pruned by Heuristic 1.

Case 2: $MINDIST_A > MINDIST_B$, and Node A is searched before node B in Algorithm INN. This is not possible since the search is ordered by the values of $MINDIST$.

Case 3: $MINDIST_A > MINDIST_B$, and node B is searched before node A. Let NN_DIST_{temp} be the distance of the temporary nearest neighbor just before A is either searched or pruned. By Lemma 5,

$$NN\_DIST_{temp} \leq MINMAXDIST_A$$

Since the condition $MINMAXDIST_B < MINDIST_A$ is given, the relation

$$NN\_DIST_{temp} < MINDIST_A$$

can be derived from the above inequalities. Therefore, node A will be pruned by Heuristic 3.

The above show that all nodes pruned by Heuristic 1 in Algorithm NN will be pruned by Algorithm INN. □

Theorem 1 *If node access corresponds to disk access, then Algorithm INN requires no extra disk access compared to Algorithm NN.*

Proof: Under our assumption, for a given R-tree, disk access is required if a node is searched for the first time. Hence the theorem follows directly from Lemmas 3 and 6. □

3.2 N-Nearest Neighbor Search

An improved N-nearest neighbor algorithm for the R-tree can be derived based on the new single nearest neighbor search algorithm, Algorithm INN. In the algorithm, we shall store a list of up to N nearest neighbors. The search is started from the root node. The current node will first be checked to see whether it is at the leaf level or not. If it is a leaf, then its distance to the query will be calculated, and if the distance is less than the distance from the N^{th} nearest neighbor discovered so far, NN_DIST_N, we insert the object into the nearest neighbor list and then NN_DIST_N is updated if necessary. On the other hand, if the current node is not at the leaf level, then the Active Branch List for further search will be generated. The Active Branch List is a list which contains all child nodes of current node that will be accessed in order to get the nearest neighbors. The Active Branch List is sorted by ascending order of $MINDIST$. Next, it iterates through the Active Branch List and recursively access child nodes by calling NN_Search. In NN_Search pruning will be performed by applying Heuristic 3. Therefore, the pseudo-code for the algorithm will be very similar to that of Algorithm INN, except we would replace NN_DIST_{temp} by NN_DIST_N, which is the distance of the query point to the N-th nearest neighbor that have been found so far.

The proof of the efficiency of the modified N-nearest neighbor search algorithm will be similar to that for the single nearest neighbor case.

4 Conclusion

The commonly used content-based index structures of R-tree and R*-tree are studied. An enhanced nearest neighbor search algorithm have been derived. It is shown that the new algorithm can preserve the pruning power of the original algorithm while reducing computational cost.

Acknowledgements We thank the anonymous referee and the Editor-in Chief who have given us helpful advices. This research work was supported by the RGC (the Hong Kong Research Grants Council) grant UGC REF.CUHK4176/97E.

References

[1] N. Beckmann, Hans-Peter Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pages 322–331, May 1990.

[2] S. Berchtold, C. Bohm, B. Braunmuller, D. Keim, and H.P. Kriegel. Fast Parallel Similarity Search in Multimedia Databases. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, 1997.

[3] S. Berchtold, C. Bohm, D. Keim, and H.P. Kriegel. A Cost Model for Nearest Neighbor Search in High Dimensional Data Space. In *Proceedings of the ACM PODS Symposium on Principles of Database Systems*, 1997.

[4] S. Berchtold, D. A. Keim, and Hans-Peter Kriegel. The X-tree: an index structure for high-dimensional data. In *Proceedings of the 22nd International Conference on VLDB*, 1996.

[5] S. Brin. Near neighbor search in large metric space. In *Proceedings of the 21st International Conference on VLDB*, pages 574–584, 1995.

[6] A. Brink, S. Marcus, and V.S. Subrahmanian. Heterogeneous Multimedia Reasoning. *IEEE Computer*, 28(9), September 1995.

[7] Tzi-cker Chiueh. Content-based image indexing. In *Proceedings of the 20th VLDB Conference*, pages 582–593, 1994.

[8] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker. Query by image and video content: the QBIC system. *IEEE Computer*, 28(9):23–32, September 1995.

[9] J.H. Friedman, J.L. Bentley, and R.A. Finkel. An Algorithm for Finding Best Matches in Logarithmic Expected Time. *ACM Transactions on Mathematical Software*, 3(3):209–226, September 1977.

[10] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pages 47–57, June 1984.

[11] King-ip Lin and C. Faloutsos. Fastmap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proceedings of ACM SIGMOD*, 1995.

[12] King-ip Lin, H. V. Jagadish, and C. Faloutsos. The TV-tree - an index structure for high-dimensional data. *VLDB Journal*, 3:517–542, October 1994.

[13] V. E. Ogle. "Chabot: Retrieval from a Relational Database of Images. *IEEE Computer*, 28(9), September 1995.

[14] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pages 71–79, June 1995.

[15] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, 1989.

[16] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R$^+$-tree: a dynamic index for multidimensional objects. In *Proceedings of the 13th International Conference on VLDB*, pages 507–518, 1987.

[17] D. A. White and R. Jain. Similarity indexing with the SS-tree. In *Proceedings of the 12th IEEE International Conference on Data Engineering*, pages 516–523, February 1996.