Nearest Neighbor Queries *

Nick Roussopoulos Stephen Kelley Frédéric Vincent

Department of Computer Science
University of Maryland
College Park, MD 20742

Abstract

A frequently encountered type of query in Geographic Information Systems is to find the k nearest neighbor objects to a given point in space. Processing such queries requires substantially different search algorithms than those for location or range queries. In this paper we present an efficient branch-and-bound R-tree traversal algorithm to find the nearest neighbor object to a point, and then generalize it to finding the k nearest neighbors. We also discuss metrics for an optimistic and a pessimistic search ordering strategy as well as for pruning. Finally, we present the results of several experiments obtained using the implementation of our algorithm and examine the behavior of the metrics and the scalability of the algorithm.

1 INTRODUCTION

The efficient implementation of Nearest Neighbor (NN) queries is of a particular interest in Geographic Information Systems (GIS). For example, a user may point to a specific location or an object on the screen, and request the system to find the five nearest objects to it in the database. Another situation where NN query is useful is when the user is not familiar with the layout of the spatial objects. In the case of an astrophysics database, finding the nearest star to a given point in the sky could involve multiple unsuccessful searches with varying window sizes if we were to use a more traditional 2D range query. Another even more complex query that could be handled by an NN technique is to find the four nearest stars which are at least ten light-years away.

The versatility of k nearest neighbors search increases substantially if we consider all variations of it, such as the k furthest neighbors, or when it is combined with other spatial queries such as find the k NN to the East of a location, or even spatial joins with NN join predicate, such as find the three closest restaurants for each of two different movie theaters.

Efficient processing of NN queries requires spatial data structures which capitalize on the proximity of the objects to focus the search of potential neighbors only. There is a wide variety of spatial access methods [Same89]. However, very few have been used for NN. In [Same90], heuristics are provided to find objects in quadtrees. The exact k-NN problem, is also posed for hierarchical spatial data structures such as the PM quadtree. The proposed solution is a top-down recursive algorithm which first goes down the quadtree, exploring the subtree that contains the query point, in order to get a first estimate of the NN location. Then it backtracks and explores remaining subtrees which potentially contain NN until no subtree needs be visited. In [FBF77] a NN algorithm for k-d-trees was proposed which was later refined in [Spro91].

R-trees [Gutt84], Packed R-trees [Rous85], [Kamel93], R-tree variations [SRF87], [Beck90] have been primarily used for overlap/containment range queries and spatial join queries [BKS93] based on overlap/containment. In this paper, we provide an efficient branch-and-bound search algorithm for processing exact k-NN queries for the R-trees, introduce several metrics for ordering and pruning the search tree, and perform several experiments on synthetic and real-world data to demonstrate the performance and scalability of our approach. To the best of our knowledge, neither NN algorithms have been developed for R-trees, nor similar metrics for NN search. We would also like to point out that, although the algorithm and these metrics are in the context of R-trees, they are directly applicable to all other spatial data structures.

Section 2 of the paper contains the theoretical foundation for the nearest neighbor search. Section 3 describes the algorithm and the metrics for ordering the search and pruning during it. Section 4 has the experiments with the implementation of the algorithm. The conclusion is in section 5.

*This research was sponsored partially by the National Science Foundation under grant BIR 9318183, by ARPA under contract 003195 Ve1001D, and by NASA/USRA under contract 5555-09.

Proceedings of the 1995 ACM-SIGMOD Intl. Conf. on Management of Data, San Jose, CA.

2 NEAREST NEIGHBOR SEARCH USING R-TREES

R-trees were proposed as a natural extension of B-trees in higher than one dimensions [Gutt84]. They combine most of the nice features of both B-trees and quadtrees. Like B-trees, they remain balanced, while they maintain the flexibility of dynamically adjusting their grouping to deal with either *dead-space* or *dense areas*, like the quadtrees do. The decomposition used in R-trees is dynamic, driven by the spatial data objects. And with appropriate split algorithms, if a region of an n-dimensional space includes dead-space, no entry in the R-tree will be introduced.

Leaf nodes of the R-tree contain entries of the form $(RECT, oid)$ where oid is an object-identifier and is used as a a pointer to a data object and $RECT$ is an n-dimensional *Minimal Bounding Rectangle (MBR)* which bounds the corresponding object. For example, in a 2-dimensional space, an entry $RECT$ will be of the form $(x_{low}, x_{high}, y_{low}, y_{high})$ which represents the coordinates of the lower-left and upper-right corner of the rectangle. The possibly composite spatial objects stored at the leaf level are considered atomic, and are not further decomposed into their spatial primitives, i.e. quadrants, triangles, trapezoids, line segments, or pixels. Non-leaf R-tree nodes contain entries of the form $(RECT, p)$ where p is a pointer to a successor node in the next level of the R-tree, and $RECT$ is a minimal rectangle which bounds all the entries in the descendent node.

The term *branching factor* (or *fan-out*) can be used to specify the maximum number of entries that a node can have; each node of an R-tree with branching factor fifty, for example, points to a maximum of fifty descendents or leaf objects. To illustrate the way an R-tree is defined on some space, Figure 1 shows a collection of rectangles and Figure 2 the corresponding tree. Performance of an R-tree search is measured by the number of disk accesses (reads) necessary to find (or not find) the desired object(s) in the database. So, the R-tree branching factor is chosen such that the size of a node is equal to (or a multiple of) the size of a disk block or file system page.

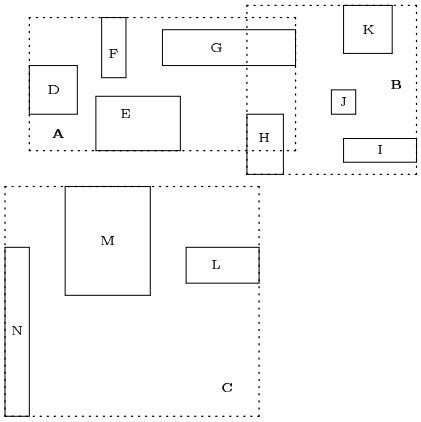


Figure 1: Collection of Rectangles

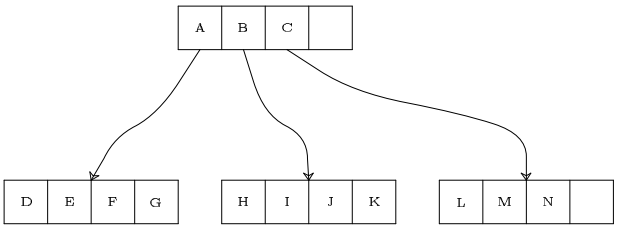


Figure 2: R-tree Construction

2.1 Metrics for Nearest Neighbor Search

Given a query point P and an object O enclosed in its MBR, we provide two metrics for ordering the NN search. The first one is based on the minimum distance (MINDIST) of the object O from P. The second metric is based on the minimum of the maximum possible distances (MINMAXDIST) from P to a face (or vertex) of the MBR containing O. MINDIST and MINMAXDIST offer a lower and an upper bound on the actual distance of O from P respectively. These bounds are used by the nearest neighbor algorithm to

order and efficiently prune the paths of the search space in an R-tree.

Definition 1 *A rectangle R in Euclidean space $E(n)$ of dimension n, will be defined by the two endpoints S and T of its major diagonal:*

$$R = (S, T)$$
$$where\ S = [s_1, s_2, \ldots, s_n]\ and\ T = [t_1, t_2, \ldots, t_n]$$
$$and\ s_i \leq t_i\ for\ 1 \leq i \leq n.$$

Minimum Distance (MINDIST) The first metric we introduce is a variation of the classic Euclidean distance applied to a point and a rectangle. If the point is inside the rectangle, the distance between them is zero. If the point is outside the rectangle, we use the square of the Euclidean distance between the point and the nearest edge of the rectangle. We use the square of the Euclidean distance because it involves fewer and less costly computations. To avoid confusion, whenever we refer to *distance* in this paper, we will in practice be using the square of the distance and the construction of our metrics will reflect this.

Definition 2 *The distance of a point P in $E(n)$ from a rectangle R in the same space, denoted $MINDIST(P, R)$, is:*

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

where

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

Lemma 1 *The distance of definition 2 is less than or equal to the square of the minimal Euclidean distance from P to any point on the perimeter of R.*

Proof: If P is inside R, then MINDIST = 0 which is less than or equal to the distance of P from any point on the perimeter of R. If P is on the perimeter, again MINDIST = 0 and so is equal to the square of minimal Euclidean distance of P from its closest point on the perimeter, namely itself.

If P is outside R and j coordinates, $j = 1, 2, \ldots, n-1$ of P satisfy $s_j \leq p_j \leq t_j$, then MINDIST measures the square of the length of a perpendicular segment from P to an edge, for $j = 1$ or to a plane for $j = 2$, or a hyperface for $j \geq 3$. If none of the p_j coordinates fall between (s_i, t_i), then MINDIST is the square of the distance to the closest vertex of R by the way of selecting r_i.

Notice that computing MINDIST requires only linear in the number of dimensions, $O(n)$, operations.

Definition 3 *The minimum distance of a point P from a spatial object o, denoted by $||(P, o)||$, is:*

$$||(P, o)|| = min(\sum_{i=1}^{n} |p_i - x_i|^2,$$
$$\forall X = [x_1, \ldots, x_n] \in O).$$

Theorem 1 *Given a point P and an MBR R enclosing a set of objects $O = \{o_i, 1 \leq i \leq m\}$, the following is true:*

$$\forall o \in O, MINDIST(P, R) \leq ||(P, o)||$$

Proof: If P is inside R, then MINDIST = 0 which is less than the distance of any object within R including one that may be touching P. If P is outside R, then according to lemma 1, $\forall X$ on the perimeter of R, $MINDIST(P, R) \leq ||(P, X)||$.

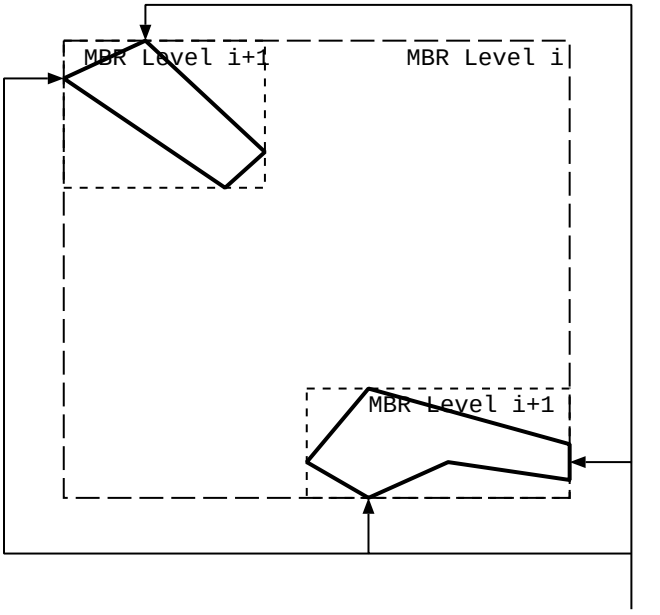
MINDIST is used to determine the closest object to P from all those enclosed in R. The equality in the above theorem holds when an object of R touches the circle with center P and radius the square root of MINDIST.

When searching an R-tree for the nearest neighbor to a query point P, at each visited node of the R-tree, one must decide which MBR to search first. MINDIST offers a first approximation of the NN distance to every MBR of the node and, therefore, can be used to direct the search.

In general, deciding which MBR to visit first in order to minimize the total number of visited nodes is not that straightforward. In fact, in many cases, due to dead space inside the MBRs, the NN might be much further than MINDIST, and visiting first the MBR with the smallest MINDIST may result in *false-drops*, i.e. visits to unnecessary nodes. For this reason, we introduce a second metric MINMAXDIST. But first, the following lemma is necessary.

Lemma 2 *The MBR Face Property: Every face (i.e. edge in dimension 2, rectangle in dimension 3 and 'hyperface' in higher dimensions) of any MBR (at any level of the R-tree) contains at least one point of some spatial object in the DB. (See figures 3 and 4).*

Proof: At the leaf level in the R-tree (object level), assume by contradiction that one face of the enclosing MBR does not touch the enclosed object. Then, there exists a smaller rectangle that encloses the object which contradicts the definition of the Minimum Bounding Rectangle. For the non-leaf levels, we use an induction on the level in the tree of the MBR. Assume any level $k \geq 0$ MBR has the MBR face property, and consider an MBR at level $k+1$. By the definition of an MBR, each face of that MBR touches an MBR of lower level, and



Each edge of the MBR at level i is in contact with a graphic object of the DB. (The same property applies for the MBRs at level i+1)

Figure 3: MBR Face Property in 2-Space

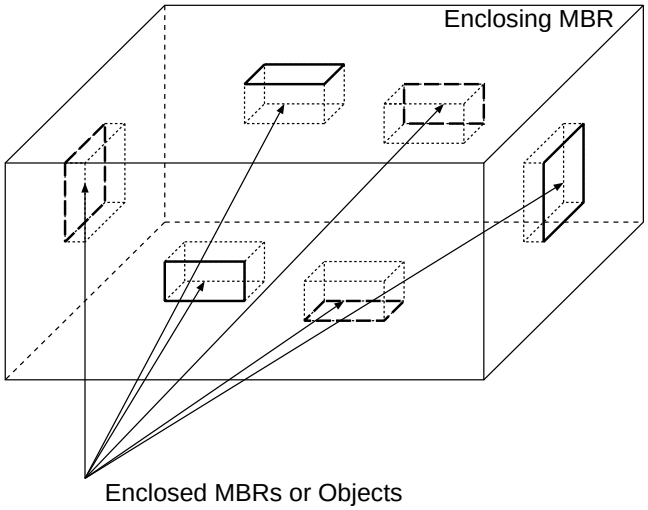


Figure 4: MBR Face Property in 3-Space

therefore, with a leaf object by applying the inductive hypothesis.

Minimax Distance (MINMAXDIST) In order to avoid visiting unnecessary MBRs, we should have an upper bound of the NN distance to any object inside an MBR. This will allow us to prune MBRs that have MINDIST higher than this upper bound. The following distance construction (called MINMAXDIST) is being introduced to compute the minimum value of all the maximum distances between the query point and points on the each of the n axes respectively. The MINMAXDIST guarantees there is an object within the MBR at a distance less than or equal to MINMAXDIST.

Definition 4 *Given a point P in E(n) and an MBR R = (S, T) of the same dimensionality, we define MINMAXDIST(P,R) as:*

$$MINMAXDIST(P,R) =$$

$$\min_{1\le k\le n}\left(|p_k - rm_k|^2 + \sum_{\substack{i\neq k\\ 1\le i\le n}} |p_i - rM_i|^2\right)$$

where:

$$rm_k = \begin{cases} s_k & \text{if } p_k \le \frac{(s_k+t_k)}{2}; \\ t_k & \text{otherwise.} \end{cases} \quad \text{and}$$

$$rM_i = \begin{cases} s_i & \text{if } p_i \ge \frac{(s_i+t_i)}{2}; \\ t_i & \text{otherwise.} \end{cases}$$

This construction can be described as follows: For each k select the hyperplane $H_k = rm_k$ which contains the closer of the two faces of the MBR orthogonal to the k^{th} space axis. (One of these faces has $H_k = S_k$ and the other has $H_k = T_k$). The point $V_k = (rM_1, rM_2, \ldots, rM_{k-1}, rm_k, rM_{k+1}, \ldots, rM_n)$, is the farthest vertex from P on this face. MINMAXDIST then, is the minimum of the squares of the distance to each of these points.

Notice that this expression can be efficiently implemented, in $O(n)$ operations by first computing $S = \sum_{1\le i\le n} |p_i - rM_i|^2$, (the distance from P to the furthest vertex on the MBR), then iteratively selecting the minimum of $S - |p_k - rM_k|^2 + |p_k - rm_k|^2$ for $1 \le k \le n$.

Theorem 2 *Given a point P and an MBR R enclosing a set of objects $O = \{o_i, 1 \le i \le m\}$, the following property holds: $\exists o \in O, \|(P,o)\| \le MINMAXDIST(P,R)$.*

Proof: Because of lemma 2, we know that each MBR's face is touching at least one object within the MBR. Since the definition of MINMAXDIST produces an estimate of the NN distance to an object touching its MBR at the extremity of one of its faces, this guarantees that MINMAXDIST is greater than or equal to the NN distance. On the other hand, a point object located exactly at the vertex of the MBR at distance MINMAXDIST would contradict the proposition that one could find a smaller distance than MINMAXDIST as an upper bound of the NN distance.

Theorem 2 says that MINMAXDIST is the minimum distance that guarantees the presence of an object O in R whose distance from P is within this distance. A value larger or equal to MINMAXDIST would always 'catch' some object inside an MBR, but a smaller distance could 'miss' some object.

Figures 5 and 6 illustrate MINDIST and MINMAXDIST in a 2-Space and 3-Space respectively.

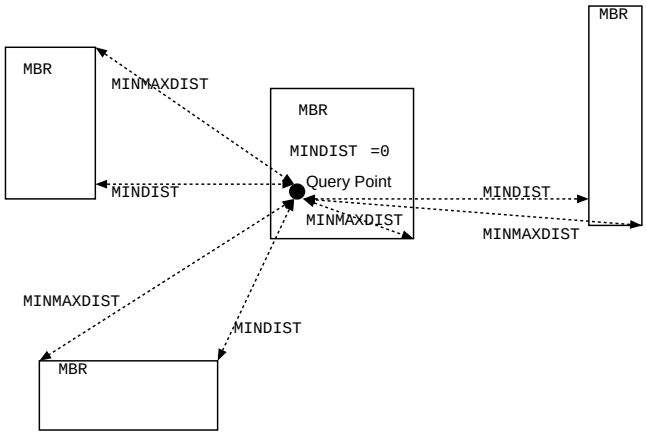


Figure 5: MINDIST and MINMAXDIST in 2-Space

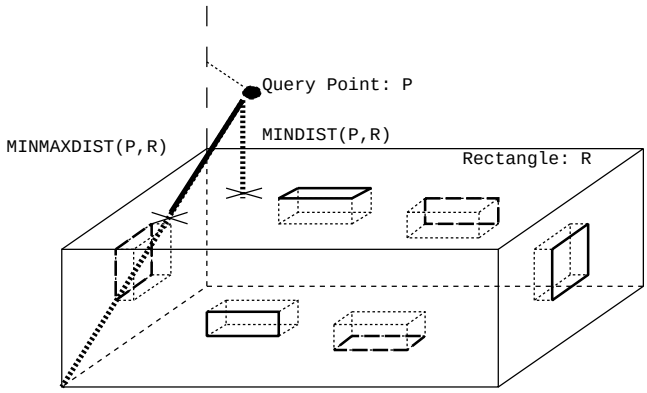


Figure 6: MINDIST and MINMAXDIST 3-Space

3 Nearest Neighbor Algorithm for R-trees

In this section we present a *branch-and-bound* R-tree traversal algorithm to find the k-NN objects to a given query point. We first discuss the merits of using the MINDIST and MINMAXDIST metrics to order and prune the search tree, then we present the algorithm for finding 1-NN and finally, generalize the algorithm for finding the k-NN.

3.1 MINDIST and MINMAXDIST for Ordering and Pruning the Search

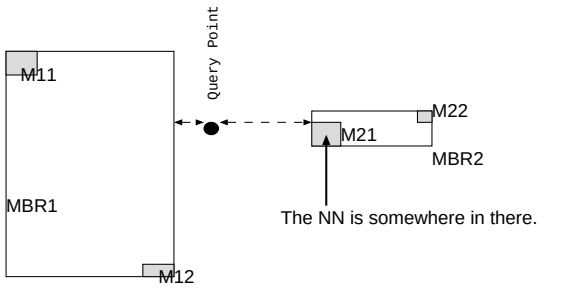
Branch-and-bound algorithms have been studied and used extensively in the areas of artificial intelligence and operations research [HS78]. If the ordering and pruning heuristics are chosen well, they can significantly reduce the number of nodes visited in a large search space.

Search Ordering: The heuristics we use in our algorithm and in the following experiments are based on orderings of the MINDIST and MINMAXDIST metrics. The MINDIST ordering is the optimistic choice, while the MINMAXDIST metric is the pessimistic (though not worst case) one. Since MINDIST estimates the distance from the query point to any enclosed MBR or data object as the minimum distance from the point to the MBR itself, it is the most optimistic choice possible. Due to the properties of MBRs and the construction of it, MINMAXDIST produces the most pessimistic ordering that need ever be considered.

In applying a depth first traversal to find the NN to a query point in an R-tree, the optimal MBR visit ordering depends not only on the distance from the query point to each of the MBRs along the path(s) from the root to the leaf node(s), but also on the size and layout of the MBRs (or in the leaf node case, objects) within each MBR. In particular, one can construct examples in which the MINDIST metric ordering produces tree traversals that are more costly (in terms of nodes visited) than the MINMAXDIST metric.

This is shown in figure 7, where the MINDIST metric ordering will lead the search to MBR1 which would require of opening up M11 and M12. If on the other hand, MINMAXDIST metric ordering was used, visiting MBR2 would result in an smaller estimate of the actual distance to the NN (which will be found to be in M21) which will then eliminate the need to examine M11 and M12. The MINDIST ordering optimistically assumes that the NN to P in MBR M is going to be close to $MINDIST(M, P)$, which is not always the case. Similarly, counterexamples could be constructed for any predefined ordering.

As we stated above, the MINDIST metric produces most optimistic ordering, but that is not always the



1. MINDIST ordering: if we visit MBR1 first, we have to visit M11, M12, MBR2 and M21 before finding the NN.

2. MINMAXDIST ordering: if we visit MBR2 first, and then M21, when we eventually visit MBR1, we can prune M11 and M12.

Figure 7: MINDIST is not always the better ordering

best choice. Many other orderings are possible by choosing metrics which compute the distance from the query point to faces or vertices of the MBR which are further away. The importance of MINMAXDIST(P,M) is that it computes the *smallest* distance between point P and MBR M that guarantees the finding of an object in M at a Euclidean distance less than or equal to MINMAXDIST(P,M).

Search Pruning: We utilize the two theorems we developed to formulate the following three strategies to prune MBRs during the search:

1. an MBR M with MINDIST(P,M) greater than the MINMAXDIST(P,M') of another MBR M' is discarded because it cannot contain the NN (theorems 1 and 2). We use this in *downward pruning*.

2. an actual distance from P to a given object O which is greater than the MINMAXDIST(P,M) for an MBR M can be discarded because M contains a object O' which is nearer to P (theorem 2). This is also used in upward pruning.

3. every MBR M with MINDIST(P,M) greater than the actual distance from P to a given object O is discarded because it cannot enclose an object nearer than O (theorem 1). We use this in *upward pruning*.

Although we specify only the use of MINMAXDIST in downward pruning, in practice, there are situations where it is better to apply MINDIST (and in fact strategy 3) instead. For example, when there is no dead space (or at least very little) in the nodes of the R-tree, MINDIST is a much better estimate of $||(P,N)||$, the actual distance to the NN than is MINMAXDIST, at all levels in the tree. So, it will prune more candidate MBRs than will MINMAXDIST.

3.2 Nearest Neighbor Search Algorithm

The algorithm presented here implements an ordered depth first traversal. It begins with the R-tree root node and proceeds down the tree. Originally, our guess for the nearest neighbor distance (call it *Nearest*) is infinity. During the descending phase, at each newly visited non-leaf node, the algorithm computes the ordering metric bounds (e.g. MINDIST, Definition 2) for all its MBRs and sorts them (associated with their corresponding node) into an Active Branch List (ABL). We then apply pruning strategies 1 and 2 to the ABL to remove unnecessary branches. The algorithm iterates on this ABL until the ABL is empty: For each iteration, the algorithm selects the next branch in the list and applies itself recursively to the node corresponding to the MBR of this branch. At a leaf node (DB objects level), the algorithm calls a type specific distance function for each object and selects the smaller distance between current value of $Nearest$ and each computed value and updates $Nearest$ appropriately. At the return from the recursion, we take this new estimate of the NN and apply pruning strategy 3 to remove all branches with $MINDIST(P,M) > Nearest$ for all MBRs M in the ABL.

See Figure 8 for the pseudo-code description of the algorithm.

3.3 Generalization: Finding the k Nearest Neighbors

The algorithm presented above can be easily generalized to answer queries of the type: Find The k Nearest Neighbors to a given Query Point, where k is greater than zero.

The only differences are:

- A sorted buffer of at most k current nearest neighbors is needed.

- The MBRs pruning is done according to the distance of the furthest nearest neighbor in this buffer.

The next section provides experimental results using both MINDIST and MINMAXDIST.

4 Experimental Results

We implemented our k-NN search algorithm and designed and carried out our experiments in order to demonstrate the capability and usefulness of our NN search approach as applied to GIS type of queries. We examined the behavior of our algorithm as the number of neighbors increased, the cardinality of the data set size grew, and how the MINDIST and MINMAXDIST metrics affected performance.

We performed our experiments on both publically available real-world spatial data sets and synthetic data sets. The real-world data sets included segment based

```
RECURSIVE PROCEDURE
nearestNeighborSearch (Node, Point, Nearest)

NODE        Node        // Current NODE
POINT       Point       // Search POINT
NEARESTN    Nearest     // Nearest Neighbor

    //Local Variables
    NODE            newNode
    BRANCHARRAY     branchList
    integer         dist, last, i

    // At leaf level - compute distance to actual objects
    If Node.type = LEAF
    Then
        For i := 1 to Node.count
            dist := objectDIST(Point, Node.branch_i.rect)
            If (dist < Nearest.dist)
               Nearest.dist := dist
               Nearest.rect := Node.branch_i.rect

    // Non-leaf level - order, prune and visit nodes
    Else
        // Generate Active Branch List
        genBranchList(Point, Node, branchList)

        // Sort ABL based on ordering metric values
        sortBranchList(branchList)

        // Perform Downward Pruning
        // (may discard all branches)
        last = pruneBranchList(Node, Point, Nearest,
               branchList)

        // Iterate through the Active Branch List
        For i := 1 to last
            newNode := Node.branch_branchList_i

            // Recursively visit child nodes
            nearestNeighborSearch(newNode, Point,
               Nearest)

            // Perform Upward Pruning
            last := pruneBranchList(Node, Point, Nearest,
               branchList)
```

Figure 8: Nearest Neighbor Search Pseudo-Code

TIGER data files for the city of Long Beach, CA and Montgomery County, MD, and observation records from the International Ultraviolet Explorer (I.U.E) satellite from N.A.S.A. Our examples will be from the Long Beach data, which consists of 55,000 street segments stored as pairs of latitude and longitude coordinates. For the synthetic data experiments, we generated test data files of 1K, 2K, 4K, 8K, 16K, 32K, 64K, 128K and 256K points (stored as rectangles) in a grid of 8K by 8K. The points were unique and randomly generated using a different seed value for each data set. We then generated 100 equally spaced query points in the 8K by 8K space.

We built the R-tree indexes by first presorting the data files using a Hilbert [Jaga90] number generating function, and then applying a modified version of [Rous85] R-tree packing technique according to the suggestion of [Kamel93]. The branching factor of both the terminal and non-terminal nodes was set to be approximately 50 in all indexes.

In Figure 9 we see the average of 100 queries for each of several different numbers of nearest neighbors for both the MINDIST and MINMAXDIST ordering metrics applied to the Long Beach data. We generated three uniform sets of querys of 100 points each based on regions of the Long Beach, CA data set. The first set was from a sparse (few or no segments at all) region of the city, the second was from a dense (large number of segments per unit area), and the third was a uniform sample from the MBR of the whole city map. We then executed a series of nearest neighbor queries for each of the query points in each of these regions and plotted the average number of nodes accessed against the number of nearest neighbors.

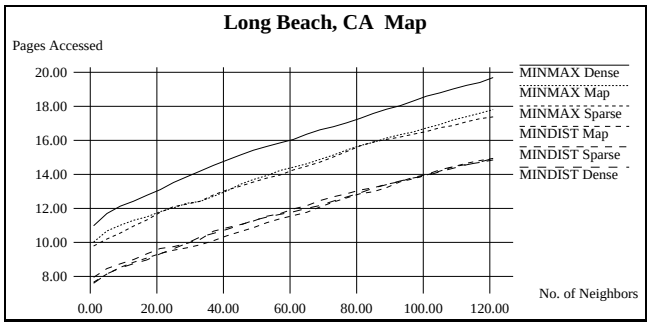


Figure 9: MINDIST and MINMAXDIST Metric Comparison

For this experiment we see that graphs of the MINMAXDIST ordered searches were similar in shape to the graphs of the MINDIST ordered searches but the number of pages accessed was consistently, approximately 20% higher. MINMAXDIST performed the worst in dense regions of the various data sets which was not surprising. It turned out that in all the experiments

we performed comparing the two metrics, the results were similar to this one. Since this occurred with both real world and (pseudo) randomly distributed data, we surmise that for spatially sorted and packed R-trees, MINDIST is the ordering metric of choice. So, for the sake of clarity and simplicity, the rest of the figures will show the results of the MINDIST metric only.

Figure 10 shows the results of an experiment using synthetic data. We ran and averaged the results from the 100 equally spaced query points for 25 different values of k NN (ranging from 1 to 121) on the 1K, 4K, 16K, 64K and 256K data sets. We graphed the results as the (average) number of pages (nodes) accessed versus the number of nearest neighbors searched for.

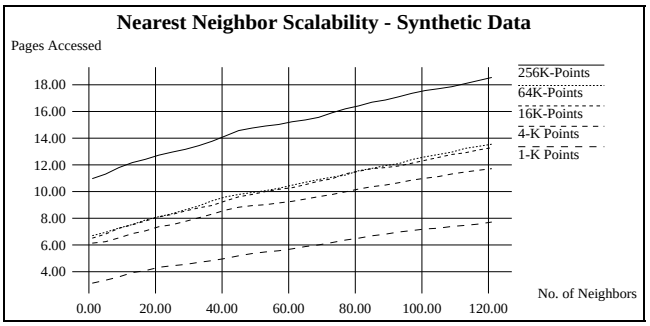


Figure 10: Synthetic Data Experiment 1 Results

From the experimental behavior, we can make two observations. First, as the number of nearest neighbors increased the number of pages accessed grew in a linear ratio with a (small) fractional constant of proportionality. Since all the synthetic data sets were created with the same node cardinality (approximately 50), the degree of similarity of the curves strongly suggests that this is the dominant term in the components of the constant of proportionality (at least in spatially ordered data sets). Second, as the data set size grew, the average number of page accesses grew sublinearly and clustered in three distinct groupings (producing a banded pattern) as the number of neighbors increased. The fact that the 4K, 16K and 64K curves were so close to each other gave us the insight to run the next experiment.

In the experiment of Figure 11, we examined the correlation between the increase in the size of the data set with the number of nodes accessed for a limited number of nearest neighbor queries. We plotted the number of pages accessed against the logarithm (base 2) of the data set size in terms of kilobytes (so $log_2(256K) = 8$) for each of 1, 16, 32, 64 and 128 nearest neighbor queries. We noticed in this graph the curves appeared to be piecewise linear step functions. We then examined the height of the R-trees and observed that the steps appear at the points where the height of the tree increases. The 1K R-tree index has a depth of 1, the 2K index has a depth of 2, the 4K, 8K, 16K, 32K and 64K data sets have a depth of 3, and the 128K and the 256K data sets have a depth of 4.

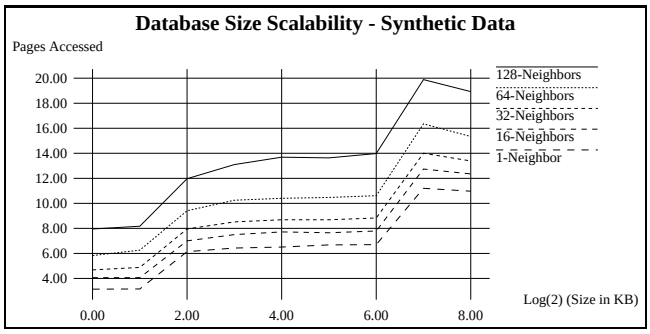


Figure 11: Synthetic Data Experiment 2 Results

This is an important observation because it shows that the algorithm behaves well for ordered data sets and the cost of NN increases linearly with the height of the tree.

5 CONCLUSIONS

In this paper, we developed a branch-and-bound R-tree traversal algorithm which finds the k Nearest Neighbors of a given query point. We also introduced two metrics that can be used to guide an ordered depth first spatial search. The first metric, MINDIST, produces the most optimistic ordering possible, whereas the second, MINMAXDIST, produces the most pessimistic ordering that ever need be considered. Although our experiments have shown that MINDIST ordering was more effective in the case where the data were spatially sorted, other orderings or more sophisticated metrics using a combination of them are possible and might prove useful in the case where the R-tree was either not constructed as well or subject to (many) updates. Nonetheless, these two metrics were shown to be valuable tools in effectively directing and pruning the Nearest Neighbor search.

We implemented and thoroughly tested our k-NN algorithm. The experiments on both real and synthetic data sets showed that the algorithm scales up well with respect to both the number of NN requested and with size of the data sets. Further research on NN queries in our group will focus on defining and analyzing other metrics and how to characterize the behavior of our algorithm in dynamic as well as static database environments.

6 ACKNOWLEDGEMENTS

We would like to thank Christos Faloutsos for his insightful comments and suggestions.

References

[Beck90] Beckmann, N., H.-P. Kriegel, R. Schneider and B. Seeger, "The R*-tree: an efficient and robust access method for points and rectangles," ACM SIGMOD, pp 322-331, May 1990.

[BKS93] Brinkhoff, T., Kriegel, H.P., and Seeger, B., "Efficient Processing of Spatial Joins Using R-trees," Proc. ACM SIGMOD, May 1993, pp. 237-246.

[FBF77] Friedman, J.H., Bentley, J.L., and Finkel, R.A., "An algorithm for finding the best matches in logarithmic expected time," ACM Trans. Math. Software, 3, September 1977, pp. 209-226.

[Gutt84] Guttman, A., "R-trees,: A Dynamic Index Structure for Spatial Searching," Proc. ACM SIGMOD, pp. 47-57, June 1984.

[HS78] Horowitz, E., Sahni, S., "Fundamentals of Computer Algorithms," Computer Science Press, 1978, pp. 370-421.

[Jaga90] Jagadish, H.V., "Linear Clustering of Objects with Multiple Attributes," Proc. ACM SIGMOD, May 1990, pp. 332-342.

[Kamel93] Kamel, I. and Faloutsos, C., "Hilbert R-Tree: an Improved R-Tree Using Fractals," Proc. of Int. Conference of Information and Knowledge Management CIKM, 1993, pp. 490-499.

[Rous85] Roussopoulos, N. and D. Leifker, "Direct Spatial Search on Pictorial Databases Using Packed R-trees," Proc. ACM SIGMOD, May 1985.

[Same89] Samet, H., "The Design & Analysis Of Spatial Data Structures," Addison-Wesley, 1989.

[Same90] Samet, H., "Applications Of Spatial Data Structures, Computer Graphics, Image Processing and GIS," Addison-Wesley, 1990.

[SRF87] Sellis T., Roussopoulos, N., and Faloutsos, C., "The R+-tree: A Dynamic Index for Multi-dimensional Objects," Proc. 13th International Conference on Very Large Data Bases, 1987, pp. 507-518.

[Spro91] Sproull, R.F., "Refinements to Nearest-Neighbor Searching in k-Dimensional Trees," Algorithmica, 6, 1991, pp. 579-589.