Shortest Path Finding Using Spatial Ranking

B.PADMAJA[1], R.SATEESH[2], K.DHANASREE[3]
Assistant Professor, DJRIET [1, 2], Associate Professor, DRKIST[3]

Abstract

The k nearest neighbor object to a point in space is the most regularly used query in finding shortest path of a given network. In this paper we present an efficient pruning method to find the nearest neighbor to a point for finding the shortest path. Finally we present the results of several experiments obtained using the implementation of our algorithm and examine the behavior of the metrics and scalability of the algorithm.

Keyword: *Spatial, Ranking, Nearest Neighbor, Shortest path, MBR, R-Tree.*

I. Introduction

The efficient implementation of Nearest Neighbor (NN) queries is of a particular interest in Geographic Information System (GIS). In this paper the shortest path in a network is obtained by finding the nearest neighbor of nearest neighbors.

Efficient processing of NN queries requires spatial data structure which capitalize on the proximity of the objects to focus the search of potential neighbors only. Finding the Nearest Neighbor of Nearest Neighbor has many applications:

I. In mobile environments, users do not have the accurate knowledge about their locations to specify the query points because all location identification methods have errors. Even if they have such knowledge, they may not want to expose these locations to the service providers for privacy reasons. RNN queries address these issues by allowing users to specify ranges rather than points for NN queries. They are particularly appealing to the large number of 2G/3G mobile subscribers whose devices are incapable of pinpointing locations. While these devices cannot support conventional NN queries, they can issue RNN queries through text messages such as “find the nearest hotels to the City Park?”

II. A user may continuously ask for nearest neighbors while moving around. It is inefficient to submit many PNN queries individually to the server. A better alternative is to submit a single RNN query around the current location to fetch all possible nearest neighbors for this area. Any PNN query issued in this area is then processed locally by a nearest-neighbor search in the prefetched set, saving both computation and communication costs

Section 2 of the paper contains the theoretical foundation for the shortest path finding using nearest neighbor search. Section 3 describes the algorithm for ordering the search and pruning during it. Section 4 has the experiments with the implementation of the algorithm.

II. Shortest Path Finding

USING R-TREES

As with Quad Tree the region is divided into successively smaller rectangles (MBRs). Rectangles need not be of the same size or number at each level. Rectangles may actually overlap. Lowest level cell has only one object, Tree maintenance algorithms similar to those for B-trees.

Leaf nodes of the R-Tree contain entries of the form (RECT, oid) where oid is an object identifier and is used as a pointer to a data object and RECT is an n-dimensional Minimal Bounding Rectangle (MBR) which bounds the corresponding object.

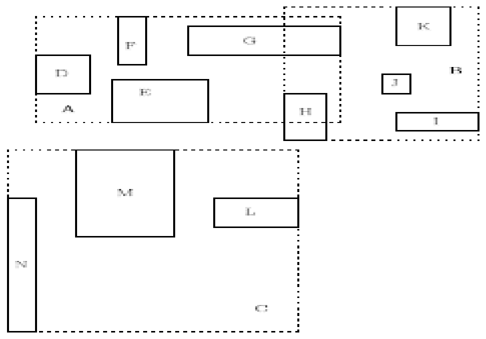

Fig 1: Collection of Rectangles

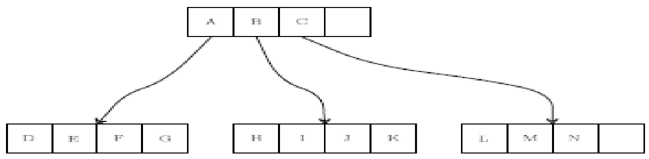

Fig 2: R-Tree Construction

Every MBR have 4 corners which are as shown below

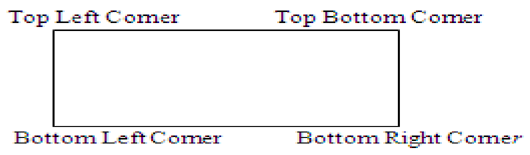

Fig 3 : MBR Corners

Take Two MBR's from the source node as Bottom Right corner in both left and right direction as shown in the following figure Find the nearest neighbor of source in both left and right MBR's and find out the distance between left MBR nearest neighbor and destination node (assumed as ϵ_1), the right MBR nearest neighbor and destination node (assumed as ϵ_2). The smallest ϵ value nearest neighbor is changed as the source, the above procedure repeats until destination is reached.

When the destination reaches all the set of sources traversed becomes the shortest path.

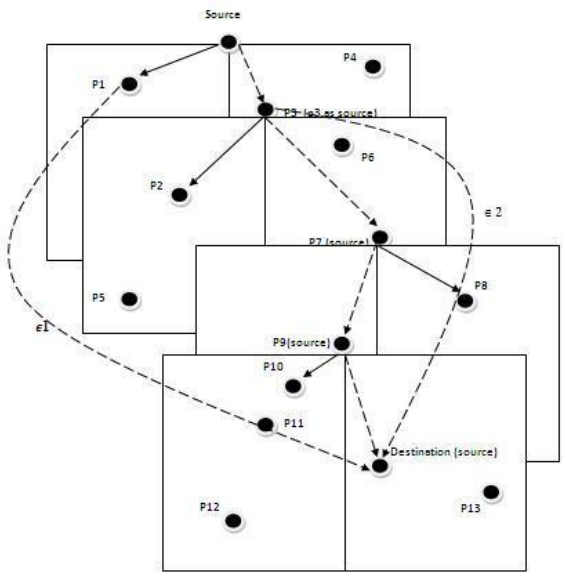

Fig 4: Shortest Path Finding

As in the above figure the Left MBR nearest neighbor to source is node P1 and the Right MBR nearest neighbor to source is node P3. Take the distance between p1 to destination as ϵ_1(Assume it is 25 units) and the distance between p2 to destination as ϵ_2 (Assume it as 20 units). Since ϵ_2 is less than ϵ_1 take the Right MBR's nearest neighbor as source i.e point p3. Now take the source as p3, take two MBR's from the bottom left and bottom right corners and repeat the above procedure. The list of all nodes taken as sources between the source node and destination nodes is the shortest path.

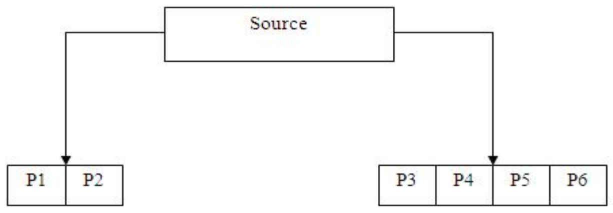

Fig 4(a)

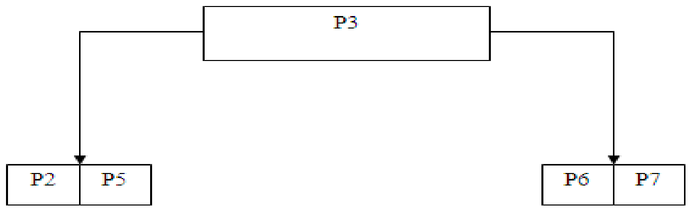

Fig 4 (b)

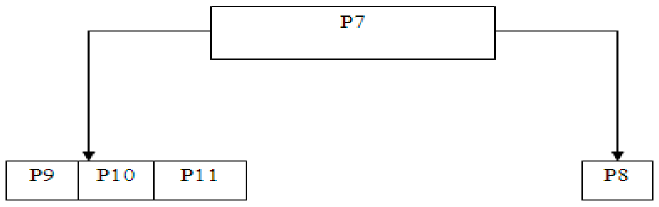

Fig 4 (c)

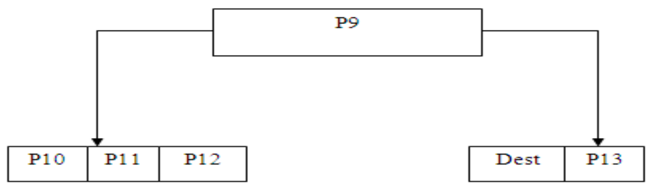

Fig 4 (d)

Above Figures shows both the Left MBR and Right MBR's Nodes.

In Figure 4(a) the root is taken as source node and the Left sub tree is taken as all the Left MBR nodes, the Right sub tree is taken as all the Right MBR nodes. The nearest neighbor to the destination in both the Left sub tree and Right sub tree of Figure 4(a) is taken as the root for the Figure 4(b). This procedure is repeated until the nearest neighbor for the root is Destination. The collection of all the root nodes becomes the shortest path.

III. Shortest Path Finding Algorithm Using R-Trees

Pseudo code for shortest path finding

```
Algorithm: Shortestpath_find (Source, LeftMBR, RightMBR,
Set V, Value ϵ)

L1: Insert Source as root into Hc  //Hc refers Min
      Heap
L2: Nearest Nearest
L3: int  dist
L 4: for each left entry e' of Source do
L 5:     if CN is a non-leaf node then
L 6:           if ∃e' Є V, mindist(e',source) ≤ ϵ then
L 7:                 dist:=objectDIST(e',Source)
L 8:                 if(dist<Nearest.dist) then
L 9:                   Nearest.dist=dist
L 10: insert e' into H α  //H α refers Min Heap
L 11:     else then
L 12:          Perform Pruning
L 13: for each right entry e'' of Source do
L 14:     if CN is a non-leaf node then
L 15:           if ∃e'' Є V, mindist(e'',source) ≤ ϵ
                then
L 16:                 dist:=objectDIST(e'',Source)
L 17:                 if(dist<Nearest.dist) then
L 18:                   Nearest.dist=dist
L 19: Insert e'' into Hβ  //Hβ refers Min Heap
L 20:    else then
L 21:          Perform Pruning
L 22: while |v| >0 and there exists a non-empty
        heap  H α do
L 23: deheap an entry e' from Hα into LNR (Left
           Nearest Neighbor)
L 24: while |v|  >0 and there exists a non-empty eap
          Hβ do
L 25: deheap an entry e'' from H β into RNR (Right
          Nearest Neighbor)
L26: ϵ1=dist(LNR,destination)
L27: ϵ2=dist(RNR,destination)

L28: if  ϵ1< ϵ2 then
L29: Change the source as LNR andCall
        Shortestpath_find (LNR, LeftMBR , RightMBR
        ,Set V,Value ϵ)
L30: else then
L31: Change the source as RNR and Call
        Shortestpath_find (RNR, LeftMBR , RightMBR
       ,Set V, Value ϵ)
```

In this paper, we assume that the object dataset is indexed by an R-tree and each feature dataset is indexed by an MIN R-tree, where each where each non-leaf entry augments the minimum quality (of features) in its sub tree.

The above algorithm shows the procedure for finding the shortest spatial path in networks. In this algorithm the Lines 1-21 are used to take three Min Heaps are used to store the Source Node, its Left MBR nodes, its Right MBR nodes respectively as Hc, H α, Hβ . In H α and Hβ all the nodes of Left MBR are stored in order of minimum distance from the source. All the nodes which are not in the Left and Right MBR's are pruned.

The functions objectDIST and dist are used to calculate the Euclidean distance between the source to destination (or) source to a specific point in MBR. Lines 22-25 shows that the deheap node from H α becomes the nearest neighbor in its Left MBR (i.e LNR), the deheap node from the H β becomes the nearest neighbor in its Right MBR (i.e RNR)

Lines 26-31 shows that the distance between the LNR, RNR and Destination node are taken into respectively ϵ_1 andϵ_2. If $\epsilon 1$ is smaller than $\epsilon 2$ then the source is taken as LNR otherwise the source is taken as RNR for the next recursive call.

IV. Experiment Results

The shortest path is finding by taking two MBR's as any one of the corner based upon the direction of destination from the source. When the destination is below or right to the source as shown in the following figure (5) the Two MBR's are taken as Top Left corner and Top Right corner. If the destination is above or left to the source then the Two MBR's are taken as Bottom Left and Bottom Right corner

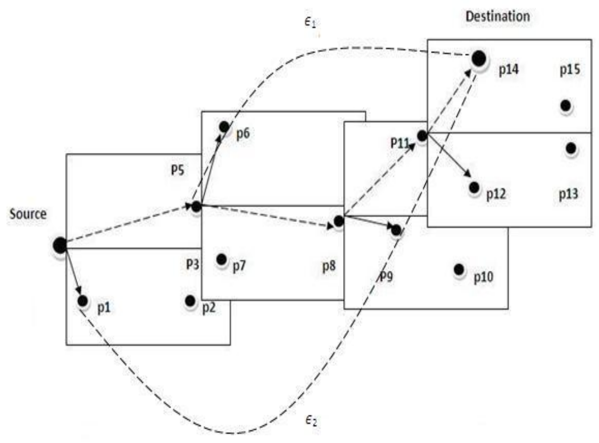

Fig (5)

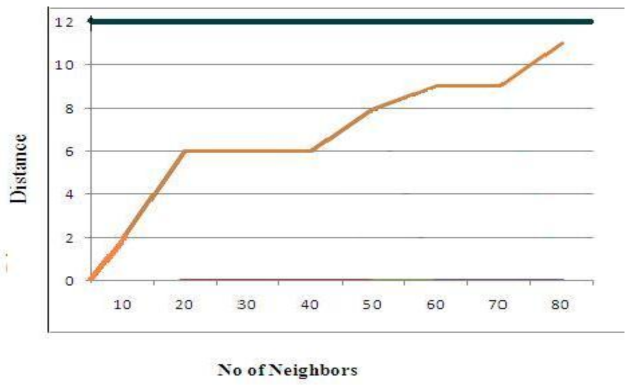

Fig (6)

Fig (6) shows the results of an experiment using the data of Fig (4). From the experimental behavior, we can make two observations. First, in the Traditional system the total number of neighbors accessed to find the shortest path is at constant rate of total number of neighbor nodes. That is at any distance we need to check all the neighbors. Second, in our proposed system the total number of neighbors accessed to find the shortest path is at the rate of 20%. In figure 4, traversed two neighbors in the first 10 units, six neighbor nodes for 20, 30, 40 units, eight neighbor nodes for 50 units, traversed nine neighbor nodes for 60 and 70 units, 11neighbor nodes for 80 units. That is at a specific distance we need to check the Left MBR and Right MBR neighbors only, the remaining neighbor nodes are pruned.

V. Conclusion

In this paper, we developed a Shortest Path finding Algorithm which finds the k Nearest Neighbors of a given source point. We also introduced four corners of MBR that can be used to guide an ordered spatial search. We implemented and thoroughly tested our Shortest Path finding algorithm using k-Nearest Neighbor spatial search. The experiments on both real data sets showed that the algorithm scales up well with respect to both the number of NN requested and with size of data sets.

Further research on shortest path finding using spatial queries will focus on defining and analyzing other metrics and characterization of our algorithm in three space environment.

References

[1] Nearest Neighbor Queries by Nick Roussopoulos Stephen Kelly Frederic Vincent.
[2] Ranking Spatial Data by Quality Preferences Man Lung Yiu, Hua Lu, Nikos Mamoulis, and Michail Vaitis.
[3] Range Nearest-Neighbor Query Haibo Hu and Dik Lun Lee
[4] M. L. Yiu, X. Dai, N. Mamoulis, and M. Vaitis, "Top-k Spatial Preference Queries," in ICDE, 2007.
[5] N. Bruno, L. Gravano, and A. Marian, "Evaluating Top-k Queries over Web-accessible Databases," in ICDE, 2002.
[6] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," in SIGMOD, 1984.
[7] G. R. Hjaltason and H. Samet, "Distance Browsing in Spatial Databases," TODS, vol. 24(2), pp. 265–318, 1999.
[8] R. Weber, H.-J. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high dimensional spaces." in VLDB, 1998.