An Efficient Algorithm for Searching Nearest Objects in Spatial Database

Jeang-Kuo Chen
Department of Information Management
Chaoyang University of Technolog
Wufeng, Taichung County, Taiwan 413
Republic of China
Email: jkchen@mail.cyut.edu.tw

Yeh-Hao Chin
Institute of Computer Science
National Tsing Hua Universit
Hsinchu 30043, Taiwan
Republic of China
Email: yhchin@cs.nthu.edu.tw

Abstract

Querying the nearest ob jects of a given point is an important function in spatial database systems. The algorithm called RKV [9] solves the requirement incompletely because only one nearest object can be found. Besides, the performance of RKV is low because its depth-first search causes backtracking of re-accessing some visited nodes. Based on breadth-first search, we propose a complete and efficient algorithm called CC in this paper to provide an alternative for high performance. To verify that CC is better than RKV in performan ce, several experiments were conducted on the efficiency of these two algorithms. The experiment results indicate that CC performs better than RKV does about one-and-half to four-and-half-fold as the number of data objects in the database is increased. The more the spatial objects in a database, the better the performance of CC compared with that of RKV.

KEYWORDS: Nearest object query, R-tree, Breadth-first search, Algorithm, Spatial database

1. Introduction

Many data structures, such as Grid file, k-d tree, R-tree, etc. [10], are wildly applied to some non-traditional applications such as GIS, image processing, pattern recognition, and spatial database systems [2,6,7,8] etc. Among them, the R-tree and its variants [1,4,11] are the most popular ones. In applications of astrophysics, astronomy, geography, etc., to find the nearest spatial objects around a given query point in a certain area is a frequent request. For example, the nearest object of the point P in Figure 1 is object A because the distance between P and A is the shortest. Based on [3] and the branch-and-bound technique, the algorithm proposed by Rousopoulos, Kelly, and Vincent [9] (called RKV), was developed to solve the requirement of querying a nearest object of a given point. However, there ar e two drawbacks in RKV. First, RKV solves the requirement incompletely because only one nearest object can be found, while there may have several nearest objects to a given point. Second, the performance of RKV is low since its depth-first search induces backtracking which results in re-accessing some visited nodes. The upper the level of a node, the larger the number of the node been accessed. An example is given below. Assume a nearest object query operation, based on the RKV algorithm, traverses and acce sses the nodes a, b, c, e, g, and h of the R-tree as shown in Figure 2. The accessing sequence of nodes is "*abebacgchca*," as shown in Figure 2. The nodes, a, b, and c are repeatedly accessed due to the backtracking property of depth-first search.

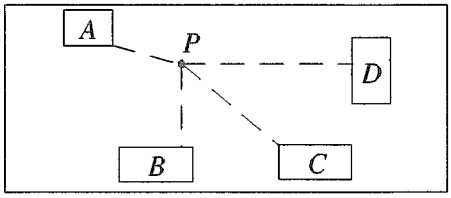


Figure 1. An example of a nearest object query.

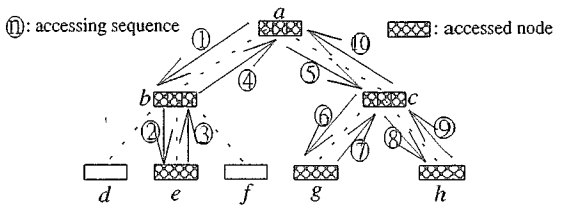


Figure 2. The search sequence of RKV.

Based on breadth-first search, we propose a complete and efficient algorithm, called CC (Chen and Chin), for nearest object queries. For a nearest object query of a given point, CC accesses fewer nodes than RKV does because no backtracking is required in CC. Each node is accessed onl one time by CC whatever the level of the node is upper or lower. Following the case of Figure 2, Figure 3 is an example that details the advantages of CC. A nearest object query operation based on CC descends down the tree, level-by-level and left-to-right, without backtracking. Therefore, the accessing sequence of nodes is "*abcegh*," as shown in Figure 3. Comparing the accessed nodes by CC and RKV, RKV accesses 11 nodes while only 6 nodes are accessed by CC. CC avoids backtracking t o reduce the number of nodes to be accessed. The efficiency of CC is promoted since the cost of node access dominates the total cost of an algorithm. Therefore, our approach is superior t RKV in term of accessed nodes. Another advantage of C is that CC can find out all nearest objects of a query point, while RKV can find out only one. CC completely performs the requirement of the nearest object query, compared with RKV. To verify such observations, several tests were done to compare the performance of CC and RKV. The simulation results indicate that the performance of CC is about one-and-half to four-and-half-fold that of RKV as the number of spatial objects is increased.

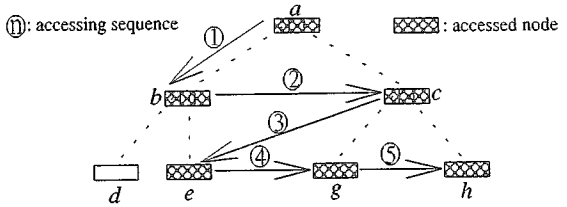


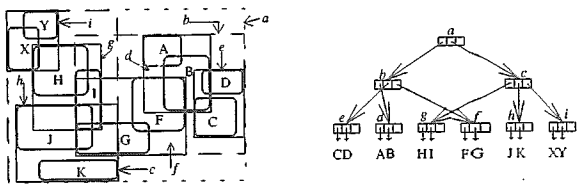
Figure 3. The search sequence of CC.

Section 2 introduces the basic concepts of the nearest object query and the CC algorithm. Section 3 describes the experiments and the analysis of the experiment results. Last Section 4 presents the conclusion.

2. Concurrency Control Algorithm

2.1 Search Method

The R-tree [4] is a dynamic data structure for n-dimensional data objects, as shown in Figure 4. Each node contains index records between m and M ($\lceil M/2 \rceil \leq m$) where M is the maximum number of index records in a node, called *fanout*. Each index record in a leaf node contains a pointer that indicates to the location of an object stored in a storage device and a rectilinear rectangle, called the minimal bounding rectangle (MBR), that tightly binds the indicated object. Each index record in a non -leaf node also has a pointer that indicates to a descendent node and an MBR that tightly binds all the rectangles of the descendent node. To find the nearest object of a query point, the nearest object query operation in an R -tree must descend down from the root to the particular leaf node that contains the desired nearest object of the query point. At each level of the tree, some formulas are used to decide which children of the current node should be visited next. Finally, the desired nearest object can be found when the bottom level of the tree is reached.



(a) Objects distribution space (b) The corresponding R-tree

Figure 4. Data objects and the R-tree.

An object (i.e., the rectangles such as A, B, etc. in Figure 4) is represented by its corresponding MBR in an R-tree. For simplicity, the distance from a point to an object is considered as the distance from that point to the corresponding MBR of that object as defined in [9]. The formula MINDIST [9] (abbreviated to MD hereafter), measuring the minimal distance between a query point and an MBR, determines which child node of a parent node should be visited next. Thus, the suitable candidate among several child nodes of a parent node can be selec ted by measuring the MD of each node with respect to the quer point and the one that has the smallest MD is selected as

the next node to be examined. However, in some situations, the node determined by MD may be not the correct node that contains the nearest object of a query point because of dead space inside the MBR of a node [9]. To compensate the inaccuracy of MD in identifying the real nearest object, another formula MINMAXDIST [9] (abbreviated to MM hereafter) is used to find the appropriate node th at probably contains the real nearest object. The function of MMD is t measure the minimum of the maximum possible distance from a query point to an edge or a vertex of an MBR. The MMD of a node from a point guarantees that the nearest object of the point can be found inside the node at a distance less than or equal to the MMD of the node [9].

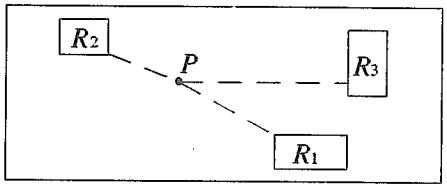
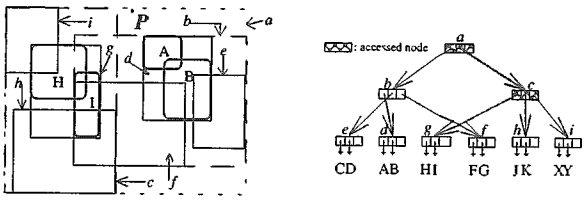


Figure 5. Three cases of MD.

The following examples illustrate the semantics of MD and MMD. The MD of an MBR from a query point denotes the shortest distance between the query point and a cert ain point on one edge of the MBR. Figure 5 shows three MD's of three MBR's. The MD of R_1, R_2, and R_3 from the quer point P are the distance between P and the top-left vertex of R_1, the bottom-right vertex of R_2, and a certain point on the vertical left-edge of R_3, respectively. Sometimes, an incorrect candidate for finding the nearest object is selected if only using MD, as shown in Figure 6. If the operation for finding the nearest object of the query point P is at node a, then node c is selected next because the MD of node c is smaller than the MD of node b. Consequently, object H is found as the nearest object of P. However, the real nearest object of P is object A in node b. The dead space in node c makes the MD of node c smaller than that of node b. Therefore, a node that contains a candidate which is the nearest object/node to a query point should be considered. To find such a node, the MMD of an MBR from a query point is used to denote the shortest distance between the query point and one of the four vertexes of the MBR where at least a child of the MBR can be found within the MMD. Figure 7 shows three MMD's of three MBR's. Here, the MBR R_1 has four vertices, v_1, v_2, v_3, and v_4, and two childre r_1 and r_2. To find a child in R_1 within the least distance, the MMD must be the distance between P and v_1, as shown in Figure 7. Hence, the correct nearest object of a query point can be determined through the values of MD and MMD. In RKV, MMD is used to find the nearest object while MD is used to prune some un necessary branches. In CC, both MD and MMD are concurrently used to focus on nearest objects when descending the tree, level by level.



(a) The location of P (b) Node c is accessed next

Figure 6. An error due to using MD only.

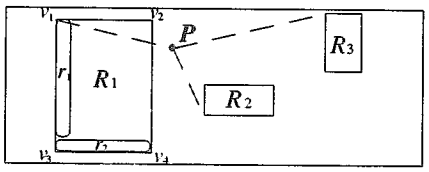


Figure 7. Three cases of MMD.

2.2 The Algorithm

The search technique used in the CC algorithm is breadth-first search that supports a search order of top-down and left-to-right without backtracking. Two formulas for computing MD and MMD respectively are used concurrently to select some non-leaf nodes that contain the desired nearest objects during descending the tree. The concept of the CC algorith can be simpl described in the following steps.

S1. Set and access the root of the R -tree as *Current* and descend the tree.

S2. For each MBR i in *Current*, measure the MD and MMD of i from the query point.

S3. Select the MBR's in *Current* with the smallest MD or MMD and the relative nodes of the selected MBR's as

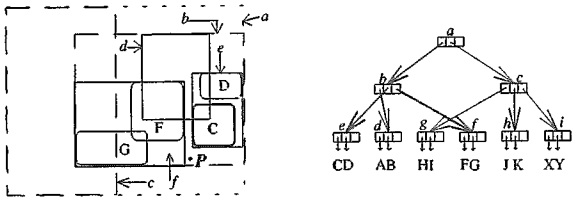
candidates for accessing later in sequence.

S4. Set a selected node as *Current* to be visited next.

S5. if *Current* is a non-leaf node, then go to S2.

S6. For each MBR *i* in the candidate leaf nodes, measure the MD of *i* from the query point.

S7. Select the MBR's with the smallest MD and return the relative objects of the selected MBR's as the nearest objects of the query point.



(a) The location of *P* (b) The search sequence

Figure 8. The behavior of the CC algorithm

To find the nearest neighbors of a given point, CC starts at the root of an R-tree and descends, level by level, toward the bottom of the tree. Initially, the root is accessed and set as the *Current* node. The MD and MMD of each MBR in *Current* from the query point is measured and saved. Then, the MBR's with the smallest MD or MMD a re selected and the nodes corresponding to the selected MBR's are added into a queue in a left -to-right order. Next, an element in the queue is selected and set up as *Current* following the sequence of first-in-first-out. These actions, obtaining, visiting, measuring, and selecting the nodes in the queue, are repeated until *Current* is a leaf node. Finally, the MD' of the objects in the candidate leaf-nodes (*Current* and the nodes in the queue) are measured. The objects with the smallest MD are returned as the nearest objects of the query point. Figure 8 illustrates the searching behavior of CC. Assume *P* is the query point. The MD and MMD of nodes *b* and *c* are measured when the query operation arrives at node *a* (corresponding to steps S1 and S2). Next, node *b* is selected and accessed since its MD and MMD are both smaller than those of node *c* (S3 and S4). Then, the MD and MMD of nodes *d*, *e*, and *f* are measured (S5 and S2) nodes *f* and *e* are found to have the smallest MD and MMD respectively (S3). Therefore, nodes *e* and *f* are accessed as *Current* in sequence (S4). Finally, the MD of objects C, D, E, F, and G are measured (S5 and S6), and the nearest object of *P* is object C (S7). The detailed CC algorithm is listed below.

```
Algorithm  Find_nearest_objects(P, Rtree)
//*********************************************//
// Function.                                    //
// Given a query point P, find the nearest objects of P in  //
// an R-tree Rtree.                             //
//Variables and procedures                      //
// Current : a variable for representing the node currently //
// being visited.                               //
// Dist: a variable for keeping the distance of the  //
// temporary nearest object from P.             //
// Set: a set variable for keeping the found NN (maybe  //
// more than elements).                         //
// Queue: a queue for saving some nodes to be visited  //
// later in breadth-first search.               //
// AddQueue: a procedure for adding a node into a queue. //
// GetQueue: a procedure for getting an element from a  //
// queue.                                       //
//*********************************************//
  Current ← the root of Rtree;  reset Queue;        // S1 //
  while Current is a non-leaf node, do              // S1, S5 //
    for each index record i of Current, compute the MD
    and MMD of the MBR in i;                        // S2 //
    select the index records, say SMD, of Current that have
    the smallest MD;                                // S3 //
    for each index record j in SMD, do              // S3 //
      AddQueue(the pointer in j, Queue);            // S3 //
    endfor;
    select the index records, say SMMD, of Current that
    have the smallest MMD;                          // S3 //
    for each index record k in SMMD, do             // S3 //
      if k ∉ SMD then                               // S3 //
        AddQueue(the pointer in k, Queue);          // S3 //
      endif;
    endfor;
```

Current ← *GetQueue*(*Queue*);

// get a new node to be visited next // // S4 //

endwhile;

// now *Current* & the member in *Queue* are all leaf nodes //

Dist ← ∞; *Set* ← Ø;

while *Current* is not null, do // S6 //

for each object *o* of *Current*, d // S6 //

compute the MD of *o*, say Md_o from *P*; // S6 //

if *Dist* > MD_o, then // S7 //

Dist ← MD_o; // S7 //

discard all elements in *Set*; // S7 //

Set ← *o*; // S7 //

else

if *Dist* = MD_o, then // S7 //

Set ← *Set* + *o*; // S7 //

endif;

endif;

endfor;

if *Queue* is not empty, then

Current ← *GetQueue*(*Queue*);

else *Current* ← null;

endif;

endwhile;

return the objects in *Set*; // S7 //

end Find_nearest_objects.

Table 1

Differences between CC and RKV

Property	CC	RKV
Search order	Breadth-first search	Depth-first search
# of nearest objects can be found	More than one	Only one
Data objects	Raw data objects	Pre-sorted data objects
R-tree structure	Raw R-tree	Pre-packed R-tree

The similarity and difference between CC and RKV are stated as follows. The similarity is that the two formulas, MD and MMD, in RKV are also adopted in CC. The major differences are (1) the search order and (2) the way MD and MMD are used. RKV uses depth -first search, while CC uses breadth -first search. RKV uses MMD durin descending the tree and MD durin descending/backtracking the tree, while CC uses both MD and MMD simultaneously during descending the tree. Table 1 lists the differences of the two algorithms.

3. Experiment Results

To evaluate the perfor mance of CC and RKV, several experiments were conducted to measure the average number of accessed nodes for each algorithm. The more an algorithm accesses nodes, the lower the performance of the algorithm. The average number of accessed nodes for an algorithm is used to represent the performance of the algorithm because the time of accessing nodes for an algorithm dominates the response time of the algorithm. The parameter values of the experiments refer to those values in [9] in order to get the objective results. For each R-tree, the fanout of each node was set at 50 [9]. There were five data sets of object (i.e., 1k, 4k, 16k, 64k, and 256k [9]) for building five R -trees with different heights. The 1k, 4k and 16k, and 64k and 256k data objects were used to produce R-trees with heights of 2, 3, and 4, respectively. In each experiment, the underlying R-tree was built by one of the five data sets, each with the same accessed probability. All data objects were randomly generated and uniformly distributed in the data space. In each experiment, 1,000 operations were performed t access the R-tree uniformly. Likewise, all locations of the query points were randomly generated and uniformly distributed in the data space. To ensure that the number of samples met the r equirement in the *central limit theorem*, each experiment was repeated 30 times to get a 90% confidence interval [5].

The experiment results are analyzed as follows. The curves of the average number of accessed nodes of CC and RK at each different size of the data set are shown in Figure 9. CC has better performance than RKV does because the former accesses fewer nodes than the latter does in each data set. The larger the size of the data set, the larger the difference between the curves of the two algorith ms, as

shown in Figure 9. For example, in the size of the 16k data set, the difference between the two curves is 7.36, while that is 11.31 in the size of the 64k data set. The curve of C ascends slightly as the size of the data set is increased, while the curve of RKV ascends rapidly, as shown in Figure 9. For instance, the curve of CC ascends from 2.81 to 4.95 when the size of the data set is increased from 1k t 256k, while that of RKV ascends from 4.19 to 22.58. Therefore, CC has better performance than RKV does.

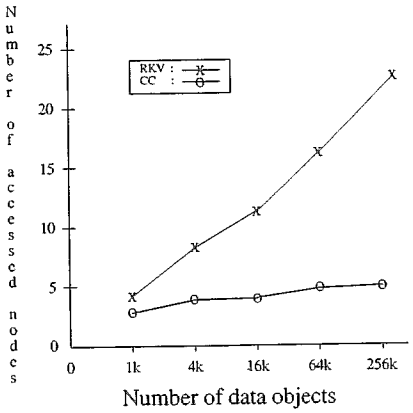


Figure 9. The average number of accessed nodes.

4. Conclusion

To implement the function of querying the nearest objects of a given point, we propose an algorithm that can find out all nearest objects and accesses each node at most one time. CC performs better than RKV does about one -and-half to four-and-half-fold under various data sets of spatial objects in the experiments. The primary reason is that breadth -first search avoids backtracking to re-access some visited nodes. Therefore, the performance of the algorithm, based on depth-first search, is lower than that of the one based on breadth-first search. It is worth while to state two additional comments. First, CC can be easily extended to find the k -th nearest object; the modification is similar to that in [9]. Second, except for uniform data distribution, the same number of experiments based on non -uniform, namely, normal data distribution, were also done and similar results as those for uniform data distribution were obtained. Hence discussion of the non -uniform distribution results is omitted.

References

[1] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles," in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp. 322-331, 1990.

[2] T. Brinkhoff, H. Kriegel, R. Schneider, and B. Seeger, "GENESYS: A System for Efficient Spatial Query Processing," in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp. 519, 1994.

[3] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An Algorithm for Finding the Best Matches in Logarithmic Expected Time, *ACM Trans. on Math. Software*, Vol. 3, pp. 209-226, 1977.

[4] A. Guttman, "R-trees: A Dynamic Index Structure for Spatial Searching," in *Proc. ACM SIGMOD Annual Meeting*, pp. 47-57, 1984.

[5] R. Jain, *The Art of Computer Sys tems Performance Analysis*, MA: John Wiley & Sons, 1991.

[6] F. Korn et al, " Efficient and effective nearest neighbor search in a medical image database of tumor shapes, *Image description and retrieval*, pp. 17-54, 1998.

[7] S. Nene and S. Nayer, " Simple algorithm for nearest neighbor search in high dimensions," *IEE Transactions on Pattern Analysis and Machin Intelligence*, vol. 19, no. 9, pp. 980-1003, 1997.

[8] The Paradise Team, "Paradise: A Database System for GIS Applications," in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp. 485, 1995.

[9] N. Roussopoulos, S. Kelley, and F. Vincent, "Nearest Neighbor Queries," in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp. 71-79, 1995.

[10] H. Samet, *The Design and Analysis of Spatial Data Structure*, MA: Addison Wesley, 1989.

[11] T. Sellis, N. Roussopoulos, and C. Faloutsos, "The R+-tree: A Dynamic Index for Multi -dimensional Objects," in *Proc of the 13th VLDB Conf.*, pp. 507-518, 1987.