

Spatial Joins Based on NA-Trees ¹

Hue-Ling Chen ^{*}, Ye-In Chang

*Dept. of Computer Science and Engineering, National Sun Yat-Sen University
Kaohsiung, Taiwan, ROC*

Key words: design of algorithms, NA-Tree, R-tree, spatial index, spatial join

1 Introduction

Spatial join is one important spatial query in a spatial database system which combines two spatial datasets to retrieve the matched pair of objects based on the spatial predicate. The spatial predicate specifies the geometrical relationship between objects, and the most common one is the overlap of spatial objects. In a GIS system, the spatial join may be used for an efficient implementation of the map overlay. The map overlay constructs a new map from two or more given maps which is important for geographic analysis. Application areas contain city planning, ecological, demographic studies, and so on [1, 5, 6]. However, there is no total ordering of spatial objects that preserves spatial proximity. It is complicated to represent the attributes and relations of spatial objects, even to evaluate the complexity of spatial queries. Since the spatial join constructs the large volume of spatial relations, to evaluate the spatial predicate between objects is time-consuming. Since a spatial index is an accelerator for the spatial query to search an object quickly [1, 5], we expect to take advantage of the spatial index to perform the spatial join efficiently. Recently, an Nine-Areas tree (denoted *NA-tree*) has been proposed to minimize the number of disk accesses during a tree search for spatial queries [3]. It solves the problem of overlaps between MBR's of internal nodes in the *R-tree* and accesses less number of nodes than the *R-tree*[2, 4]. Therefore, we present an algorithm for the spatial join based on *NA-trees*, i.e., the *NA-tree* join. Although several studies have been made on solving the overlapping

^{*} Corresponding author.

Email address: `chenhl@db.cse.nsysu.edu.tw` (Hue-Ling Chen).

¹ This research was supported in part by the National Science Council of Republic of China under Grant No. NSC-93-2213-E-110-027.

5	7	13	15
4	6	12	14
1	3	9	11
0	2	8	10

Fig. 1. An example of the bucket numbering scheme, $O(l, u) = (9, 14)$

problem of the R -tree, e.g., R^* -tree[5], the R^* -tree join will not be better than the R -tree join. Because the spatial join aims at examining overlaps of objects quickly, the R^* -tree join has to make a larger number of nodes comparisons than the R -tree join. For the other techniques which are used to improve the performance of the R -tree join in[5], they also can be helpful to our NA -tree join. Therefore, we compare the performance of the original R -tree join with our NA -tree join. We analyze all overlaps between regions of two nodes in two NA -trees and obtain one correlation table which records the pair of overlapped nodes. Our NA -tree join can execute efficiently by looking up this correlation table, instead of nodes comparisons. Our NA -tree join can access a smaller number of nodes directly based on the correlation table than the R -tree join.

2 An NA -Tree

In an NA -tree[3], a spatial object is specified by its bounding rectangle and represented by two points, $L(X_l, Y_b)$ and $U(X_r, Y_t)$, where L is the lower left coordinate and U is the upper right coordinate of the bounding rectangle. Based on the bucket-numbering scheme[3], the spatial number $O(l, u)$ is calculated for the spatial object O , where l is the bucket number of $L(X_l, Y_b)$ and u is the bucket number of $U(X_r, Y_t)$. For example, in Figure 1, the spatial number of object O is $(9, 14)$. A variable, Max_bucket , is used to record the maximum bucket number (in decimal form) of this area. In Figure 1, the maximum bucket number is 15 (0111), i.e., $Max_bucket = 15$.

An NA -tree is a structure based on data location and organized by the spatial numbers. First, the whole spatial region is decomposed into four regions with corresponding limitations of bucket numbers as shown in Figure 2-(a)[3]. Then, for an object $O(l, u)$, it may be stored as one of the following children of an internal node p in an NA -tree structure, as shown in Figure 2-(b). For example, when $l \in \text{region I}$ and $u \in \text{region II}$, O is the 5th_child of node p . However, based on different types of children, an internal node may have different numbers of children. For example, when it is the 5th_child or the 7th_child, it has only three children: 5th_child, 7th_child, and 9th_child. Finally, an object can only be stored in a leaf node, instead of an internal node. Figure 3 shows an example of an NA -tree structure. For example, object D_6 in Figure 3-(a) is

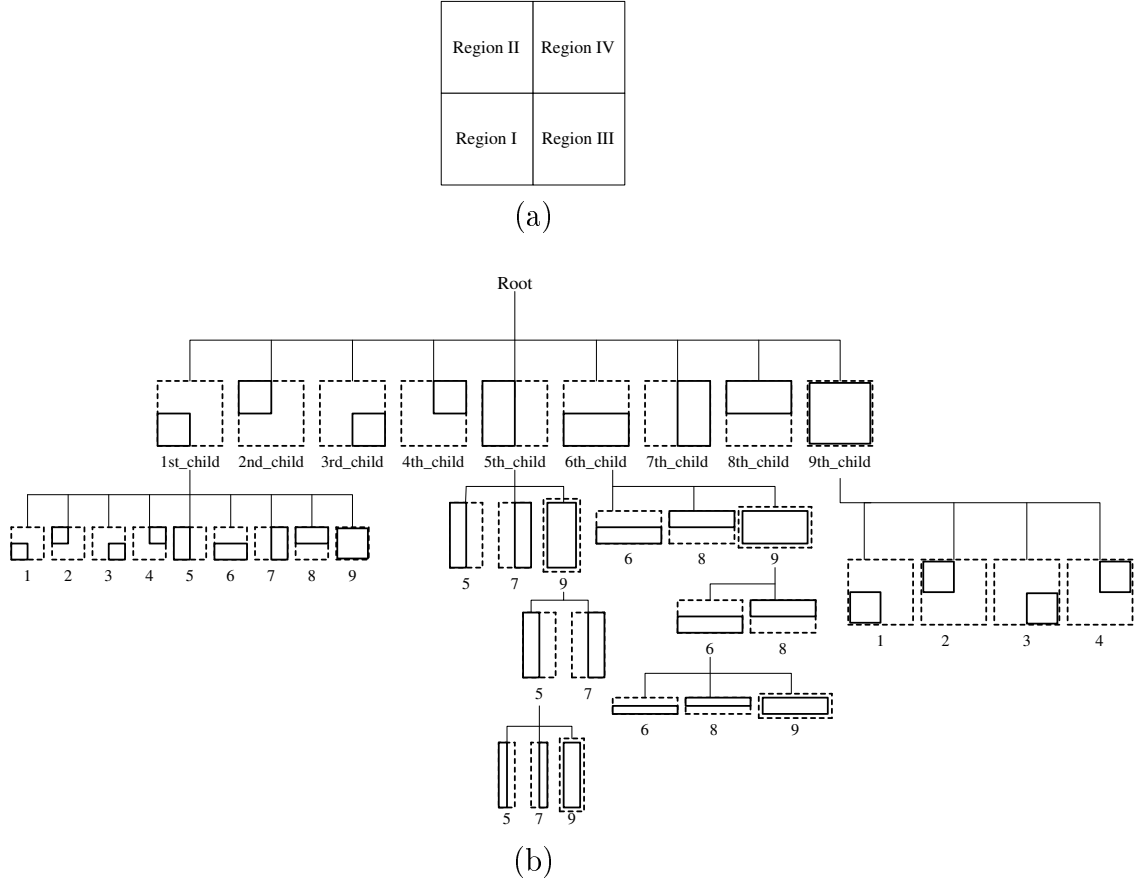


Fig. 2. An NA-tree: (a) four regions; (b) the basic structure.

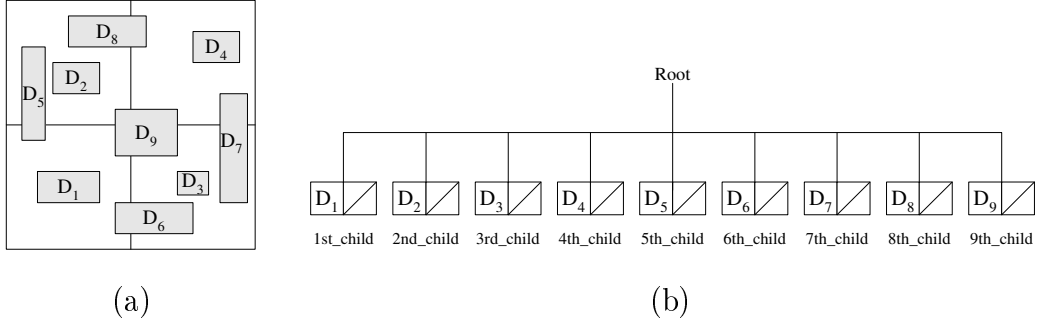


Fig. 3. An example: (a) the data; (b) the corresponding *NA*-tree structure (bucket_capacity = 2).

stored in the 6th_child node in Figure 3-(b). In[3], an *NA*-tree can support arbitrary insertions and deletions of objects without any global re-organization, depending on the spatial numbers of objects. It also can efficiently support exact match queries and range queries.

```

Procedure NASJoin( $P_A, P_B$ );
/*  $N_A$  and  $N_B$  are child nodes of nodes  $P_A$  and  $P_B$ , respectively. */
/*  $O_A$  and  $O_B$  are objects under the leaf nodes  $P_A$  and  $P_B$ , respectively. */
begin
  if ( $P_A$ .leaf = false) and ( $P_B$ .leaf = false) then /* Case 1 */
    for (all  $N_A$  of  $P_A$ ) do
      for (all  $N_B$  of  $P_B$ ) do
        if (Correlation( $N_A$ .uid,  $N_B$ .uid) = true) then
          begin    ReadNode( $N_A$ ); ReadNode( $N_B$ ); NASJoin( $N_A, N_B$ );    end
        else continue
      end if
    end for
  else if ( $P_A$ .leaf = true) and ( $P_B$ .leaf = true) then /* Case 2 */
    ObjectsJoin( $P_A, P_B$ )
  else if ( $P_A$ .leaf  $\neq$   $P_B$ .leaf) then /* Case 3 */
    begin
      if ( $P_A$ .leaf = true) then for (all  $O_A$  of  $P_A$ ) do Range_Query( $P_B, O_A$ )
      else if ( $P_B$ .leaf = true) then for (all  $O_B$  of  $P_B$ ) do Range_Query( $P_A, O_B$ );
    end if;
  end;
end;

```

Fig. 4. Procedure *NASJoin*

3 The *NA*-tree Join

Let A and B be two *NA*-trees which index two datasets. The process of our *NA*-tree join starts on two root nodes, continues on two internal nodes at each level, and stops on leaf nodes in two *NA*-trees. An overlap means that the intersection of regions of two nodes is non-empty. Basically, three cases of the height of both input *NA*-trees will be considered during our *NA*-tree join on pairs of nodes N_A and N_B in *NA*-trees A and B , respectively. These cases are described as follows:

- Case 1: *NA*-trees are of the same height and nodes N_A and N_B are internal nodes.
- Case 2: *NA*-trees are of the same height and nodes N_A and N_B are leaf nodes.
- Case 3: *NA*-trees are of different height; that is, one node is an internal node, and another one is a leaf node.

Procedure *NASJoin* in Figure 4 shows the process of our *NA*-tree join. In this section, we first describe the definition of a correlation pair used in function *Correlation*. Then, we describe the process of our *NA*-tree join on two *NA*-trees of the same height, i.e., Case 1 and Case 2. Finally, we describe the process of our *NA*-tree join on two *NA*-trees of different height, i.e., Case 3.















set	<i>S</i>				<i>R</i>				<i>LS</i>
<i>uid</i>	1	2	3	4	5	6	7	8	9
region									
subregions									

Fig. 5. Three Sets of *uid*'s and Regions of nodes with *uid*'s

3.1 A Correlation Pair

Since the spatial join investigates overlaps, a correlation pair $(N_A.uid, N_B.uid)$ is used to describe the overlap with *uid*'s of nodes N_A and N_B , respectively. An *uid* is used to represent the type of one node N_x based on the NA -tree structure in Figure 2. Let $N_x.uid = 1, 2, 3, 4, 5, 6, 7, 8$, and 9 represent each of nine child nodes N_x under one internal node at each level. (Note that the *uid* for the root is 1.) Because the overlap is examined by relative regions of two nodes, we observe the region of the node N_x first. From Figure 5, the region of the node N_x can be in one of three shapes: Square, Rectangle, and Large Square. Let set $S = \{1, 2, 3, 4\}$, set $R = \{5, 6, 7, 8\}$, $LS = \{9\}$ to record these *uid*'s whose corresponding shapes are square, rectangle, and large square, respectively. From Figure 5, we can observe that the region of $N_x.uid \in R$ or $N_x.uid \in LS$ contains subregions which are smaller than it. For example, the region of $N_x.uid = 5 \in R$ contains two subregions of *uid*'s 1 and 2, respectively.

In our NA -tree join, we use function $Correlation(N_A.uid, N_B.uid)$ to examine whether the overlap is non-empty or not. Based on correlation pairs in nine cases of $\{S, R, LS\} \times \{S, R, LS\}$, we analyze all overlaps in Table $CPairTab$, as shown in Table 6. The correlation pairs (N_A, N_B) in table $CPairTab$ have non-empty overlaps and function $Correlation$ returns True. (Note that the value under the correlation pair is the *uid* of the real region.) On the other hand, the empty slots means empty overlaps and function $Correlation$ returns False. Therefore, table $CPairTab$ can help function $Correlation$ to return True or False value. The result of those non-empty overlaps recorded in table $CpairTab$ is determined by considering two conditions on $N_A.uid$ and $N_B.uid$.

- Condition 1: If $N_A.uid$ is equal to $N_B.uid$, the resulting overlap is $N_A.uid$ (or $N_B.uid$). Otherwise, we have to consider Condition 2.
- Condition 2: Assume that $N_A.uid$ is larger than $N_B.uid$, and vice versa, there are three subconditions.
 - (1) If $5 \leq N_A.uid \leq 8$ ($N_A.uid \in R$) and $1 \leq N_B.uid \leq 4$ ($N_B.uid \in S$), the resulting overlap is $N_B.uid$. The correlation pairs in the light-shaded portion satisfy Condition 2-(1).
 - (2) If $6 \leq N_A.uid \leq 8$ ($N_A.uid \in R$) and $5 \leq N_B.uid \leq 7$ ($N_B.uid \in R$),

$N_{B,uid}$ $N_{A,uid}$		S				R				LS
		1	2	3	4	5	6	7	8	9
S	1	(1,1) 1				(1,5) 1	(1,6) 1			(1,9) 1
	2		(2,2) 2			(2,5) 2			(2,8) 2	(2,9) 2
	3			(3,3) 3			(3,6) 3	(3,7) 3		(3,9) 3
	4				(4,4) 4			(7,4) 4	(4,8) 4	(4,9) 4
R	5	(5,1) 1	(5,2) 2			(5,5) 5	(5,6)		(5,8) 5	(5,9) 5
	6	(6,1) 1		(6,3) 3		(6,5) 1	(6,6) 6	(6,7)		(6,9) 6
	7			(7,3) 3	(7,4) 4		(7,6) 3	(7,7) 7	(7,8) 4	(7,9) 7
	8		(8,2) 2		(8,4) 4	(8,5) 2		(8,7) 4	(8,8) 8	(8,9) 8
LS	9	(9,1) 1	(9,2) 2	(9,3) 4	(9,4) 3	(9,5) 5	(9,6) 6	(9,7) 7	(9,8) 8	(9,9) 9

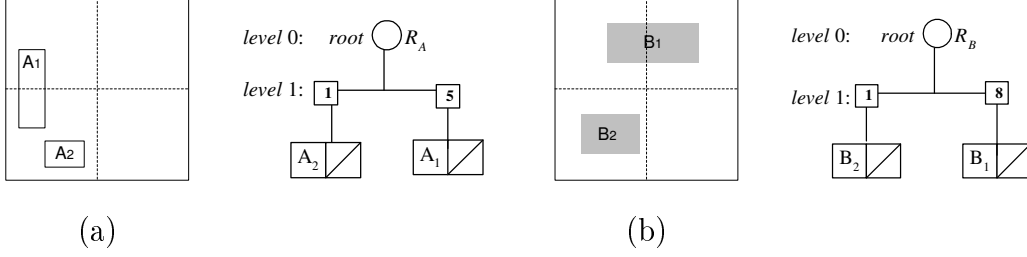


Fig. 8. An example: (a) the first dataset with NA -tree A ; (b) the second dataset with NA -tree B .

3.2 NA -trees of the Same Height

Assume that two NA -trees for our NA -tree join are of the same height. Let's take Figure 8 as an example to illustrate Cases 1 and 2 of Procedure *NASJoin* in Figure 4. Figures 8 -(a) and (b) are input datasets with their NA -trees of the same height. First, because roots R_A and R_B have child nodes (Case 1), correlation pairs $(1,1)$, $(5,1)$ and $(5,8)$ are examined to have overlaps by function *Correlation*. Then, objects are retrieved sequentially from the disk referenced by leaf nodes with their *uid*'s in these correlation pairs (Case 2). The coordinates of these spatial objects are compared. Finally, the last result contains (A_1, B_2) and (A_2, B_2) which have non-empty overlaps between them.

3.3 NA -trees of Different Height

When one NA -tree is higher than another one, objects may be stored in the leaf nodes at different height. Our NA -tree join may be executed synchronously on one internal node and one internal node at the same level. It is a condition of Case 3 in Figure 4. Let's take Figure 9 as an example. After the spatial join is executed at level 1, we can obtain all correlation pairs which have overlaps. Let's take the correlation pair $(N_A.uid, N_B.uid) = (1, 1)$ as an example to illustrate procedure *RangeQuery*. The node N_A is a leaf node, while the node N_B is an internal node at level 1. The regions of $N_A.uid$ and $N_B.uid$, respectively, are shown as the thick dotted-lined regions in the right side of Figure 9-(a) and (b), respectively. First, we retrieve object A_2 from node N_A as a query rectangle and perform a range query on node N_B by procedure *RangeQuery*. Next, object A_2 has overlaps with the region of the existing two child nodes ($uid = 5$ and $uid = 9$) under node N_B at the level 2. Then, we retrieve objects B_4 and B_2 from these two nodes, respectively. We compare their positions with the query rectangle A_2 . Finally, the pair of objects (A_2, B_2) is output as the result.

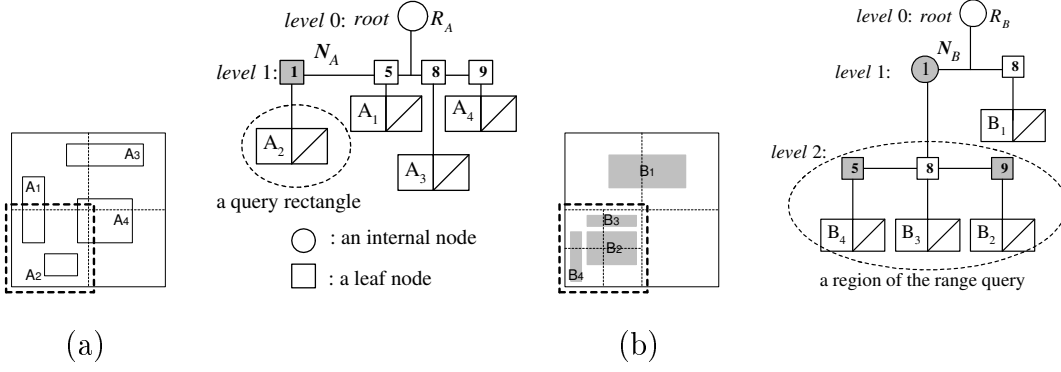
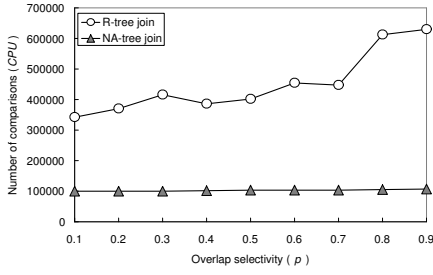


Fig. 9. An example of the correlation pair $(N_A.uid, N_B.uid)=(1, 1)$ at the level 1: (a) the first dataset with NA -tree A ; (b) the second dataset with NA -tree B .

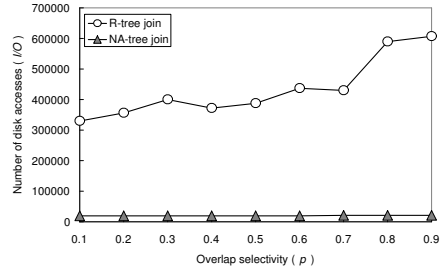
4 Performance

In our simulation, we compare the performance of the R -tree join [2] and our NA -tree join [3]. We consider the *overlap selectivity* ($0 \leq p \leq 1$), which is defined as the number of the data objects with overlap relationship in the total number N of data objects. We assume that the buffers for join operations are infinite large. We took 1000 averages of results after the spatial join on two datasets with $N=10000$ and variable p . Since the spatial join combines two datasets by their spatial relationship, we consider the CPU time (the number of nodes comparisons) and the I/O time (the number of disk accesses) as performance measures [2].

Figure 10-(a) shows the result for the measure of the CPU time. Our NA -tree join requires a smaller number of nodes comparisons than the R -tree join. The reason is that our NA -tree join examines overlaps only by looking up the static number of correlation pairs in table $CpairTab$. However, the number of nodes comparisons of the R -tree join increases as the value of the overlap selectivity p (i.e., the number of overlaps) increases. The reason is that the R -tree join has to compare many related MBR's of the internal nodes due to overlaps between them. Figure 10-(b) shows the result for the measure of the I/O time. Our NA -tree join requires a smaller number of disk accesses than the R -tree join. Because our NA -tree join produces correlation pairs of nodes which have non-empty overlaps, the objects can be directly accessed by these candidate nodes. Since the objects are well ordered in the disk based on the bucket numbering scheme [3], the pairs of objects which have non-empty overlaps can be accessed once by their sequential spatial numbers. However, there is a huge increase in the number of disk accesses of the R -tree join as the value of p increases. Because of no sequential order of these objects in R -trees, the R -tree join has to access objects from the disk more than once to obtain the actual result.



(a)



(b)

Fig. 10. A comparison of the number of nodes comparisons (CPU): $N = 10000$; (b) A comparison of the number of disk accesses (I/O): $N = 10000$.

5 Conclusion

In this paper, we have presented our NA -tree join based on the correlation table. Our NA -tree join simply uses the correlation table to directly obtain candidate leaf nodes on two NA -trees which have non-empty overlaps. Moreover, our NA -tree join accesses objects once from those candidate leaf nodes and returns pairs of objects which have non-empty overlaps.

References

- [1] Arge, L., Procopiu, O., Ramaswamy, S., Suel, T., Vahrenhold, J., Vitter, J. S., A Unified Approach for Indexed and Non-Indexed Spatial Joins, Proc. of the 7th Int. Conf. on Extending Database Technology: Advances in Database Technology (2000) 413–429.
- [2] Brinkhoff T., Kriegel, H. P., Seeger, B., Efficient Processing of Spatial Joins Using R-trees, Proc. of ACM SIGMOD Int. Conf. on Management of Data (1993) 237–246.
- [3] Chang, Y. I., Liao, C. H., Chen H. L., NA -Trees: A Dynamic Index for Spatial Data, Journal of Information Science and Eng. 19 (1) (2003) 103–139.
- [4] Dittrich, J. P., Seeger, B., Data Redundancy and Duplicate Detection in Spatial Join Processing, Proc. of the 16th Int. Conf. on Data Eng. (2000) 535–546.
- [5] Jacox, E. H., Samet, H., Spatial join techniques, ACM Trans. on Database Systems 32 (1) (2007) 7–51.
- [6] Zhu, M., Papadias, D., Zhang, J., Lee D. L., Top-k Spatial Joins, IEEE Trans. on Knowledge and Data Eng. 17 (4) (2005) 567–579.