A VOMR-TREE BASED PARALLEL RANGE QUERY METHOD ON DISTRIBUTED SPATIAL DATABASE

FU. Zhongliang [a b,] *, LIU. Siyuan [a]

[a] School of Remote and Sensing , Wuhan University, Wuhan, China - liusydr@126.com
[b] State Key Laboratory of Information Engineering in Surveying Mapping and Remote Sensing, Wuhan University, China - fuzhl@263.net

Commission II, WG II/3

KEY WORDS: Range Query, Parallel Computing, Distributed Spatial Database, Spatial Index, VoMR-tree

ABSTRACT:

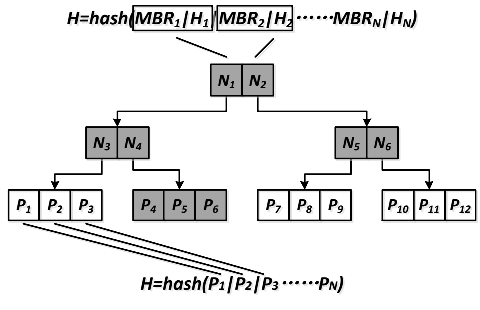
Spatial index impacts upon the efficiency of spatial query seriously in distributed spatial database. In this paper, we introduce a parallel spatial range query algorithm, based on VoMR-tree index, which incorporates Voronoi diagrams into MR-tree, benefiting from the nearest neighbors. We first augments MR-tree to store the nearest neighbors and constructs the VoMR-tree index by Voronoi diagram. We then propose a novel range query algorithm based on VoMR-tree index. In processing a range query, we discuss the data partition method so that we can improve the efficiency by parallelization in distributed database. Just then a verification strategy is promoted. We show the superiority of the proposed method by extensive experiments using data sets of various sizes. The experimental results reveal that the proposed method improves the performance of range query processing up to three times in comparison with the widely-used R-tree variants.

1. INTRODUCTION

Distributed spatial database technology with the advantages of powerful data management capability, expansibility and low location constraints, dominate an important part of geodatabase applications. It improves data process efficiency by separate data into pieces (Yang C, 2009a). However, the following problem in the spatial query of distributed geodatabase is generally occurring. (i) Spatial query relies on spatial relationship of the entities. A part of topological relation information will be lost owing to the distributed storage. (ii) It takes too much hardware resources and time when the mass data is processed by centralization calculation after a complex transmission. (iii) The spatial index of distributed geodatabse cause a large number of data redundancy so that processing efficiency will be reduced. Consequently, a high-efficiency method of range query is studied in this paper.

A range query is a common database operation that retrieves all records where some value is between an upper and lower boundary. As the basic operation of spatial query, range query can be regard as the first step of distributed spatial analysis. R-tree and its variants (the modified R-tree) are widely used in spatial queries. R-tree (Guttman, 1984b, Huang, 2001a), R*-tree (Beckmann, 1990b), R^+-tree (Sellis, 1987b), MB-tree (Li, 2006b) and MR-tree (Yang Y, 2009a) are typical examples. The most suitable index for a range query is MR-tree which augments the standard R-tree by computing hash the concatenation of the binary representation of all the entries in a tree node.

The MR-tree combines concepts from MB-tree and R*-trees. Figure 1 illustrates the tree structure. Leaf nodes are identical to those of the R*-tree: each entry P_i corresponds to a data object. A digest is computed on the concatenation of the binary representation of all objects in the node. Internal nodes contain entries of the form (p_i, MBR_i, H_i), signifying the pointer, minimum bounding rectangle, and digest of the child, respectively. The digest summarizes child nodes' MBRs (MBR_1-MBR_f), in addition to their values (H_1−H_f). When performing a range query Q (shows in Figure 2), the procedure runs a deep-first traversal from root node. If MBR of N_i does not overlap Q, all the children of N_i need not be traverse. This effectively reduces redundant processing.



* Corresponding author. This is useful to know for communication with the appropriate person in cases with more than one author.

Figure 1. Depth travel and structure of *MR-tree*

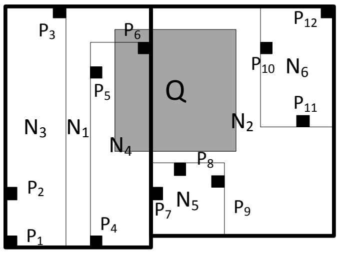


Figure 2. Range query on *MR-tree*

There are still shortcomings in the MR-tree as a result of hash concatenation. The higher the randomness, the more impossible the area size of each N_i is to minimize. It increases the likelihood of overlap between Q and N_i, so that more nodes would be accessed and the efficiency is reduced.

This paper introduces a parallel spatial range query algorithm, based on VoMR-tree index, which incorporates Voronoi diagrams into MR-tree, benefiting from the nearest neighbors. The organization of the paper is as follows. Section 2 briefly introduces the VoMR-tree index and its constructing method. Section 3 proposes an algorithm for range query processing using VoMR-tree in distributed spatial database, and discusses a strategy for verification of result set. Section 4 presents the experimental results estimating the performance of the proposed algorithms. Finally, Section 5 summarizes and concludes the paper.

2. VOMR-TREE INDEX

2.1 Subset Partition

The efficiency of depth-first traversal is impacted seriously by the ratio of overlap. When the query Q overlaps the MBR of nodes frequently, the query process will be costly as a result of a much deeper traversal. To reduce the probability of overlap between Q and MBR, A better idea is to minimize the area of each MBR. The major drawback of MR-tree is summarized as the subset partition depending on a hash sorting. So that the area size of each MBR cannot be restricted as minimal as possible. And it also has not a better way to sort dataset by axes in multi-dimensional space.

An optimal partition method in the range query shows in Figure 3. Compared with the case in Figure 2, the MBRs in Figure 3 are smaller. When MBR of N_2 does not overlap Q, the process will not traverse to N_3, N_5, P_1, P_2, P_4, P_7, P_8, and P_9. In another words, half of the nodes in MR-tree do not need to be traversed. In this method, the nearest neighbors are partitioned into a subset, for instance P_1, P_2, and P_4 into N_3.

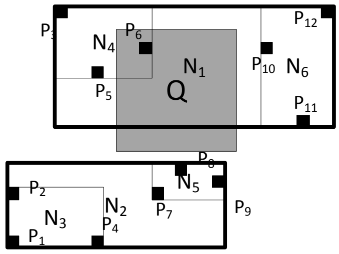


Figure 3. An optimal partition of a dataset

A high-efficiency partition method is based on nearest relationship. The most common pattern of finding the nearest neighbor is calculating Euclidean distance as follow:

$$d(x,y)=\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2} \tag{1}$$

where $d(x, y)$ = Euclidean distance between 2 geometries
x_i, y_i, = coordinates of geometries

To process a nearest neighbor (NN) search, it is not necessary to calculate the Euclidean distance of each pair of objects. A Delaunay triangulation should be available. Assume that each subset contains three objects, the steps of data partition are as follows: Choose a start point P_1, process a 2NN for P_1, record the three objects in a subset N_1. To reduplicate the first step. A better way to process *k*NN search is based on the dual graph of Delaunay triangulation, Voronoi diagram (Sharifzadeh, 2010b).

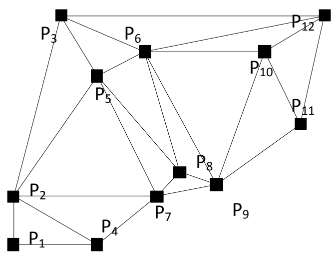


Figure 4. Delaunay triangulation

2.2 Voronoi Diagram

Given a set of distinct objects $P=\{p_1, p_2, \cdots, p_n\}$ in space R, the *Voronoi diagram* of P, denoted as $VD(P)$, partitions the space of R into n disjoint regions, such that each object p_i in P belongs to only one region and every point in that region is closer to p_i than to any other object of P in the Euclidean space. The region around p_i is called the *Voronoi cell* of p_i, denoted as $VC(p_i)$, and p_i is the generator of the Voronoi cell. Therefore, the Voronoi diagram of P is the union of all Voronoi cells $VD(P) = \{VC(p_1), VC(p_2), \cdots, VC(p_n)\}$. If two generators share a common edge, they are Voronoi neighbors. If we connect all the Voronoi neighbors, we get the *Delaunay triangulation DT(P)*, which is the dual graph of $VD(P)$ (Hu, 2010b).

Property 1. Given a set of distinct points $P=\{p_1, p_2, \cdots, p_n\}$ in R, the Voronoi diagram *VD(P)* and the corresponding Delaunay triangulation *DT(P)* of *P* are unique.

Property 2. The average number of Voronoi edges per Voronoi polygon does not exceed six. That is, the average number of Voronoi neighbors per generator does not exceed six.

Property 3. Given the Voronoi diagram of *P*, the nearest neighbor of a query point *q* is *p*, if and only if $q \in VC(p)$.

Property 4. Let $p_1, p_2, \cdots, p_k$ be the k ($k > 1$) nearest neighbors in *P* to a query point *q*. Then, p_k is a Voronoi neighbor of at least one point $p_i \in \{p_1, p_2, \cdots, p_{k-1}\}$

Figure 5 shows a Voronoi diagram constructed on a spatial dataset. Voronoi diagram is extremely efficient in searching a nearest neighbour region. It divides the two-dimensional space into several parts, and each pair of neighbour parts share a unique edge. The nearest relationship is recorded with this edge. It is an effective method to reduce the cost of Euclidean distance calculation. We use a term "VoMR-tree" to present the structure Voronoi diagram on MR-tree.

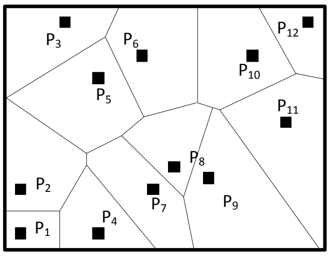


Figure 5. Voronoi diagram

2.3 Distance Judgement

Constructing a Voronoi diagram promotes the queries of nearest neighbours. In a dataset partition process, it is possible that all the nearest neighbours of a spatial point has already partitioned into an exist subset. For example, show in Figure 7, assume each subset contains four leaves, objects P_1 has two nearest neighbours P_2 and P_4, how about the fourth point in this subset. In this case, we should choose a nearest point P_x in neighbours of P_2 and P_4 as the fourth point. To minimize the MBR of this subset, P_x must be close to either P_2 and P_4 or P_1. The problem comes down to a geometric centroid.

Consider identifying the nearest point q_i ($i=1, 2, \cdots, n$) of spatial objects *p*, the geometric centroid of the region *Q* of q_i can computing as follow:

$$x = \frac{1}{n}\sum_{i=1}^{n} q_i.x \qquad y = \frac{1}{n}\sum_{i=1}^{n} q_i.y \tag{2}$$

where x, y = coordinates of geometric centroid
$q_i.x, q_i.y$ = coordinates of the nearest point

The direction vector of geometric centroid can be the auxiliary judgment in nearest point search. The partial derivative of distance between *p* and the geometric centroid *Q* of q_i is computed. According to formula 2, the direction vector is computed as following:

$$\partial x = \frac{\partial adist(q,Q)}{\partial x} = \sum_{i=1}^{n} \frac{(x - x_i)}{\sqrt{(x-x_i)^2 + (y-y_i)^2}}$$
$$\partial y = \frac{\partial adist(q,Q)}{\partial x} = \sum_{i=1}^{n} \frac{(y - y_i)}{\sqrt{(x-x_i)^2 + (y-y_i)^2}} \tag{3}$$

where $\partial x, \partial y$ = the direction vector of geometric centroid
adist(*q*,*Q*) = the Euclidean distance between object *q* and region *Q*
x_i, y_i = coordinates of the q_i
x, y = coordinates of *p*

Computing according formula 3 at point p_1, to get a direction vector d_1. Drawing a ray r_1 originating from p_1 in direction d_1 enters the Voronoi cell of p_2 intersecting its boundary at point x_1. The direction d_2 at x_1 is computed and the same process is repeated using a ray r_2 originating from x_1 in direction d_2 which enters *VC* (*p*) at x_2. Now, as we are inside *VC* (*p*) that includes centroid *q*, all other rays consecutively circulate inside *VC* (*p*). Detecting this situation, we return *p* as the closest point to *q*.

2.4 Initial Index

Given a spatial dataset *P*, the first step to initialize *VoMR-tree* index is constructing a Voronoi diagram. Next, we sort all the objects in dataset by *x* coordinate, and choose the minimum object as the start point P_1. Assume objects' count in each subset is *k*, a *k-1* nearest neighbors search result for P_1 constructs a subset N_1. If the count of this subset less than *k*, we process a distance judgment which mentioned in 3.3. Then remove the objects in N_1 from source dataset *P* and repeat the steps above. When all the objects are removed from source dataset *P*, re-initialization of *P* is promoted as follow: $P=\{N_1, N_2, \cdots, N_n\}$. To repeat the steps above till *P* contain only one object. Finally, a *VoMR-tree* construct to store all the values by leaf node recording the MBR and Voronoi diagram of entity P_i while the parent node recording the MBR of corresponding subset N_i.

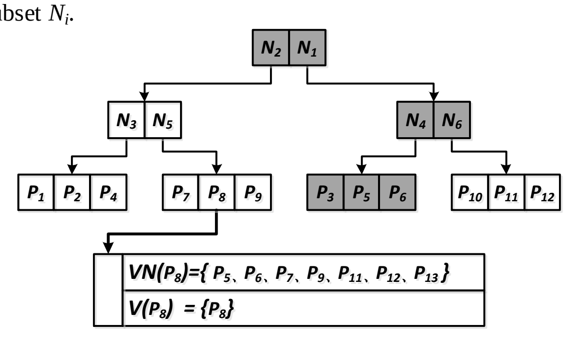
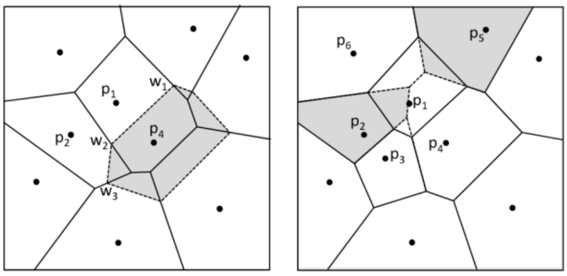


Figure 6. Range Query on VoMR-tree

Figure 6 illustrates the structure of VoMR-tree which is based on MR-tree and augments the storage content to record Voronoi diagram. For a same range query, compared with MR-tree (Figure 1), the deep-first traversal goes through much less nodes (highlight in shadow) in the tree structure.

2.5 Update

As the storage content augmenting, it is necessary to discuss about the update of VoMR-tree because the Voronoi diagram was stored synchronously in database when the index initialized.



Insertion Deletion

Figure 7. Data update on VoMR-tree

While inserting a new object in the database, the procedure employs the *incremental boundary growing* method to compute the updated Voronoi diagram. In Figure 7, for instance, to insert a new object p_4, it first identify (i) the Voronoi cell $VC(p_1)$ containing p_4 and (ii) all the Voronoi neighbors of p_1. Then, a bisector between p_1 and p_4 intersects $VC(p_1)$ at w_1 and w_2. Segment w_1 w_2 becomes the first edge of the new Voronoi cell. The next step is to iterate over all the neighboring objects that share Voronoi edges with p_1 in a counter-clockwise fashion (e.g., starting with p_2). Then, it draws the bisector between p_2 and p_4 and retrieves the next intersection at w_3. To repeat this process until all the edges of the new Voronoi cell (of p_4) are computed. The above process can also be performed in a clockwise manner. Finally, the new object is augmented with authentication and neighborhood information, and server simply inserts it in the corresponding spatial index.

Deleting an object (e.g., p_1) follows a similar approach, as shown in Figure 7. A procedure firstly locates p_1 and its neighbors, and divide $VC(p_1)$ with the bisectors between the neighboring pairs of p_1. Next, it updates the Voronoi neighbors of p_1 with new neighboring information, and transmits all affected objects (with their new signatures) to server. Additionally, the server removes p_1 from the corresponding spatial index.

3. PARALLEL RANGE QUERY

3.1 Range Query Algorithm

For a range query Q, algorithm includes the following steps. Firstly, the process traverses a deep-first path in the tree from the root node. For each node which traversed, if its MBR is contained by Q, traversing all the children of this node are stopped. And if its MBR has not intersected with Q, it is also stopped to travelling the children of this node. When the MBR overlaps Q, access the node's children and repeat the steps above.

Algorithm 1. RangeQuery(*Q*, *Nodes*)

Input: *Query* ***Q***, *VoMR_nodes* ***Nodes***
Output: *Set of Candidate Objects* ***CS***

```
CS={φ};
For i=0; i< Nodes.deeplevels; i++
    NodeCursor=Nodes(i);
    For each entry N in NodeCursor
        If N is leaf
            If Q contains N.MBR
                Append N to CS;
            Else
                Remove N from Nodes;
        Else
            If N.MBR overlaps Q
                Call RangeQuery(Q, N.p);
            ElseIf N.MBR contained by Q
                Append all leafs in N to CS;
            Else
                Remove N from Nodes;
Return CS;
```

Figure 8. Range query algorithm on VoMR-tree

Figure 8 describes the range query algorithm on VoMR-tree. The deep-first travel will stop while N does not overlap Q. hereafter, all the iteration process won't access the children of Node N. Consequently, redundant process in range query is reduced and efficiency of process is improved.

3.2 Data Partition

A single procedure to process range queries of mass data is seemed unreasonable. Distributed computing should be available. In distributed computing environment, the chief issue is to distributed data to each computing node.

Given a set of data points sorted by *x* coordinate, each procedure reads an input split in the format of <*data_point*, *d_value*> (i.e., <*key*, *value*> pair) (Akdogan, 2010b). Note that the d_value does not have any purpose, i.e., it is used to follow the input format of distributed computing. Subsequently, each procedure generates a data unit for the data points in its split, marks the boundary polygons to be later used in the merge phase and emits the generated data unit in the form of <*key*, *value*> pair where key denotes the split number. The constant key is common to all data units, so that all units can be grouped together and merged in the next subsequent step. When the query process completes in computing node, a unique procedure collects all the splits, and merges them into a whole. Figure 9 shows the data partition of a VoMR-tree.

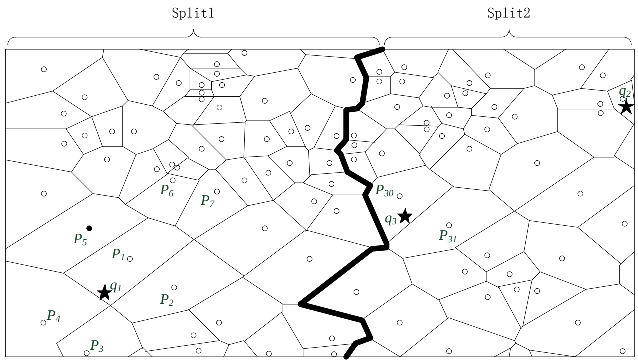


Figure 9. Data partition

3.3 Paralleling Process of Query

To process mass data in highly efficiency, distributed computing could be took into consider. The dataset is partitioned into several parts and sends to each computing node for processing. There was a new problem that how to balance the data distributed. The simplest way is that partitioning the dataset by a single level in VoMR-tree. However, if the count of this level's nodes is more than the count of computing nodes, there will be idle equipment. On the contrary, Equipment is not enough for distribution.

A solution of this problem is attempting a tentative deep-first travel from root node. When it is impossible to balance data distributed in this level of nodes, access the next level of nodes and process a deeper travel. Because several nodes would be stop to visit in each level, data distribution should be balanced in the end. When it balanced, cease the travel process immediately, and then send each unit data to distributed computing node for a parallel process. Figure 10 shows the algorithm of the parallel process.

Algorithm 2. ParallelProcess(Q, Nodes, C)

```
Input:   Query Q,  VoMR_nodes Nodes,
         Computing Node Count C;
Output:  Set of Candidate Objects CS
CS={φ};
For i=0; i< Nodes.deeplevels; i++
    if NodeCursor=Nodes(i);
    For each entry N in NodeCursor
      If N is not leaf
        If N.MBR overlaps Q
          If N.chidrencount<=C
            For j=0; j<C; j++
              Send N.chidren(j) to ComputingNode(j)
              Call RangeQuery(Q, N.chidren(j))
          Else
            Call ParallelProcess(Q, N, C)
      Else
        Call RangeQuery(Q, N.p);
Merged Subresults;
CS=RemoveRepeatObject(CS);
Return CS;
```

Figure 10. The algorithm of the parallel process

3.4 Verification

In VoMR-tree and all the R-tree variants, objects are typically approximated using minimum bounding rectangles, which require less storage space than the full object, resulting in faster processing and less expensive I/O operations(Jacox, 2007a). As the approximation of spatial object, we should discuss the verification strategies when process a range query in VoMR-tree.

VoMR-tree augments the storage content for Voronoi diagram. It is considerable that using a Voronoi diagram to remove the incorrect result from the candidate result set. According to property 3 and property 4 of Voronoi diagram, the nearest neighbor Voronoi cells of each object less than six. Therefore, if all nearest neighbor Voronoi cells are contained by query Q, the object must be the correct result. This lemma helps us to reduce the processing cost because a Voronoi cell constructed at most six points which less expensive I/O operations as MBR. When it is unable to verify the result set, the process finally reads all the points of the object for a topological operator. Figure 11 illustrates the algorithm of verifying a range query result set.

Algorithm 3. VerifyRangeQuery(Q, VNs, CS)

```
Input:   Query Q,  Voronoi Diagram VNs,
         Set of Candidate Objects CS
Output:  Set of Result RS
RS={φ};
For each entry e in CS
    If e.VD is contained by Q
        veryfyweight=0;
        For each VN in e.Neighbors
            If VN overlaps Q
                veryfyweight++;
        If veryfyweight=e.NeighborCount
            Append e to RS;
        Else
            For each point in e.points
                If all points contained by Q
                    Append e to RS;
                Else
                    Remove e from CS;
Return RS;
```

Figure 11. The algorithm of verifying a range query

4. EXPERIMENT AND ANALYSIS

We deploy a simulation of the distributed service system and extract terrain and traffic data in total 739,744 elements with 535 Mbytes for the experiments.

4.1 Cost Models

The important performance metrics for spatial index structures are (i) index construction time, (ii) index size, (iii) query processing cost, (iv) size of the VO, and (v) verification time. Table 1 shows the costs calculated by the above-mentioned equations using the typical values of Table 1. The R-tree incurs about 2 time the overhead of the MR-tree for computing the query process information (in the entire tree), and is 9 times larger. The VoMR-tree is also significantly better in terms of result set and verification cost. The latter is particularly important because the clients are distributed devices with limited computing power. The only aspect where the two structures are similar is in size of index. In the following section we present a test of parallel efficiency

Symbols	Description	*R-tree*	*VoMR-tree*
Ci	Cost of index construction	0.3 h	2 h
Si	Size of index	535 MBytes	535 MBytes
Cq	Cost of query process	40 s	22 s
Srs	Size of result set	1790 KBytes	398 KBytes
Cv	Cost of verification	240ms	91ms

Table 1. Symbols and values in analysis

4.2 Parallel Efficiency

The efficiency of the algorithm is evaluated in four symbols as following:

Object repeat rate:

$$r_o = \frac{\sum_{i=1}^{n} C_i - C}{C} \quad (4)$$

where r_o = object repeat rate
C_i = the count of objects in a subset
C= the count of objects in a spatial dataset

Results repeat rate:

$$r_r = \frac{\sum_{i=1}^{n} M_i - M}{M} \quad (5)$$

where r_r = result repeat rate
C_i = the count of result of each computing node
C= the count of objects in a result set

Task of fluctuation:

$$f = \frac{Max(M-u, u-m)}{u} \quad (6)$$

where f = task of fluctuation
M = maximum count in process of each node
M= minimum count in process of each node
u= average count in process of each node

Transfer ratio:

$$r_t = \frac{\sum_{i=1}^{n} Q_i}{Q_r + Q_s} \quad (7)$$

where r_i = transfer ratio
Q_r = the size of source dataset
Q_s= the size of result set
Q_i= the size of result set in computing node

We extract the data in equal size to evaluate the object repeat rate, and process a unique range query to check result repeat rate of R-tree and VoMR-tree.

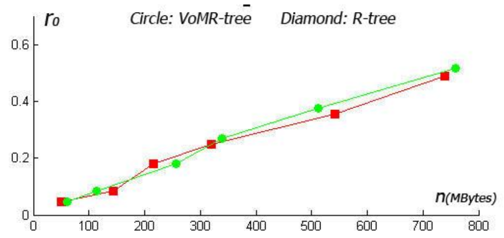


Figure 12. Object repeat rate

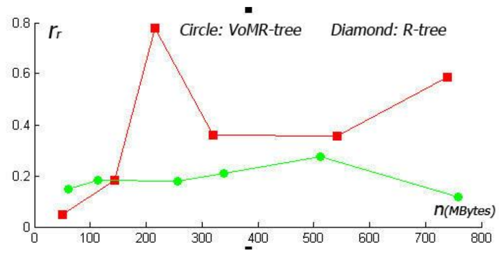


Figure 13. Result repeat rate

The experimental result is showed in Figure 12 and Figure 13. As it show in line graph, the VoMR-tree keep a more stable result repeat rate in value of 0.2 while R-tree and VoMR-tree are close to each other in object repeat rate.

R *-tree has the effect of parallel processing. So we consider to contrast R *-tree with VoMR-tree for redundancy of processing. Figure 14 illustrates the sampling result of the task from 0 to 10000. It is obvious that R*-tree keeps a more stable task of fluctuation. However, fluctuation of VoMR-tree is reduced when the task more than 2000. Thus, VoMR-tree shows a lower redundancy in process of mass data.

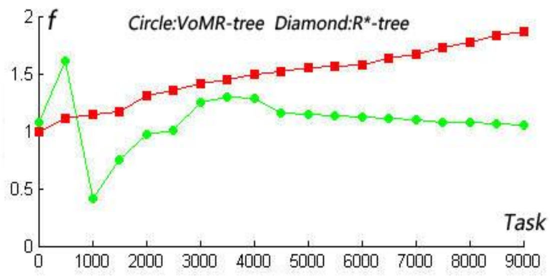


Figure 14. Task of fluctuation

Figure 15 shows the result of testing transfer ratio for VoMR-tree and MR-tree. In the line graph, both VoMR-tree and MR-tree have similar transfer ratio in the value between 1 and 2. When the throughput increased, the transfer ratio of MR-tree begins to float while that of VoMR-tree seems more stable.

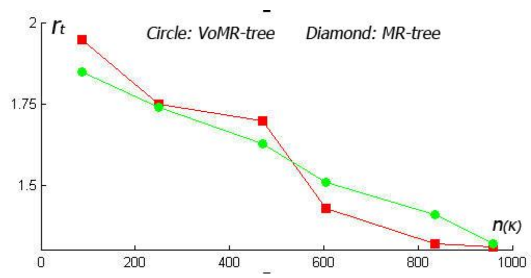


Figure 15. Transfer ratio

5. CONCLUTION

VoMR-tree augments the MR-tree, by computing hash values on the concatenation of the nearest neighbors and MBRs of all the entries in a tree node. The leaf node is sorted by nearest neighbor relationship while the construction of spatial index. In the process of query, a depth-first traverse is executing. While the MBRs overlapping, the procedure replaces the recursive query of the nearest neighbor search in order to dominate the space-cost. VoMR-tree is more advantages in the aspects of processing cost and average size of result set and is high efficient both in time and space parallelization. As future works, we plan our solutions to the problem of other types of spatial queries. We will improve a universal spatial index for a space database.

REFERENCES

Akdogan A, Demiryurek U, Banaei-Kashani F, et al. 2010b. Voronoi-based Geospatial Query Processing with MapReduce, *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pp. 9-16.

Beckmann N, Kriegel H P, Schneider R, et al, 1990b. The R*-tree: an efficient and robust access method for points and rectangles. *Proceedings of the 1990 ACM SIGMOD international conference on Management of data.*

Guttman A, 1984b. R-trees: A dynamic index structure for spatial searching. *Proceedings ACM SIGMOD international conference on Management of Data.*

Hu L, Ku W S, Bakiras S, et al. 2010b. Verifying spatial queries using Voronoi neighbors, *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pp. 350-359.

Huang P, Lin P, Lin H, 2001a. Optimizing storage utilization in R-tree dynamic index structure for spatial databases. *Journal of Systems and Software*, pp. 55: 291-299.

Jacox E H, Samet H, 2007a. Spatial join techniques. *ACM Transactions on Database Systems (TODS)* , pp. 32-37.

Li F, Hadjieleftheriou M, Kollios G, et al, 2006b. Dynamic authenticated index structures for outsourced databases, *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pp. 121-132.

Sellis T, Roussopoulos N, Faloutsos C, 1987b. The R+-tree: A dynamic index for multi-dimensional objects. Proceedings of 13th International Conference on Very Large Data Bases.

Sharifzadeh M, Shahabi C, 2010b. Vor-tree: R-trees with voronoi diagrams for efficient processing of spatial nearest neighbor queries. *Proceedings of the VLDB Endowment*, pp. 1231-1242.

Yang C, Raskin R, 2009a. Introduction to distributed geographic information processing research. *International Journal of Geographical Information Science*, pp. 23: 553-560.

Yang Y, Papadopoulos S, Papadias D, et al. 2009a. Authenticated indexing for outsourced spatial databases. *The International Journal on Very Large Data Bases*, pp. 631-648.