AVR-Tree: Speeding Up the NN and ANN Queries on Location Data

Qianlu Lin, Ying Zhang, Wenjie Zhang, and Xuemin Lin

The University of New South Wales Australia
{qlin,yingz,zhangw,lxue}@cse.unsw.edu.au

Abstract. In the paper, we study the problems of nearest neighbor queries (NN) and all nearest neighbor queries (ANN) on location data, which have a wide range of applications such as Geographic Information System (GIS) and Location based Service (LBS). We propose a new structure, termed AVR-Tree, based on the R-tree and Voronoi diagram techniques. Compared with the existing indexing techniques used for NN and ANN queries on location data, AVR-Tree can achieve a better trade-off between the pruning effectiveness and the index size for NN and ANN queries. We also conduct a comprehensive performance evaluation for the proposed techniques based on both real and synthetic data, which shows that AVR-Tree based NN and ANN algorithms achieve better performance compared with their best competitors in terms of both CPU and I/O costs.

1 Introduction

With the emergence of location-aware mobile device technologies, communication technologies and GPS systems, the location-aware queries have attracted great attentions in many important applications such as location based service (LBS) and geographic information system (GIS). Particularly, the nearest neighbor search and its variants play an important role among these queries since it is very natural to find various facilities with close spatial proximity. In the paper, we focus on the nearest neighbor (NN) and all nearest neighbor (ANN) queries on location data. Given an object (e.g., a user location) and a set of facilities (e.g., warehouses, petrol stations and restaurants), the nearest neighbor search identifies the closest facility regarding an object. All nearest neighbor (ANN) query aims to retrieve nearest neighbors for a set of objects regarding a set of facilities. Both queries are fundamental in spatial queries and have a wide spectrum of applications. Below is a motivating example.

Fig. 1 shows a map consisting of seven supermarkets and three warehouses. To deliver stocks from a warehouse to a supermarket with the minimized delivery cost and time, it is desirable to put the stocks in the nearest warehouse of each supermarket. For simplicity, the distance between two spatial locations is calculated as Euclidean distance. By issuing a nearest neighbor query on supermarket s_1, it is reported that w_3 is its nearest warehouse. On the other hand,

© Springer-Verlag Berlin Heidelberg 2013

we may issue an all nearest neighbor query against all supermarkets and warehouses. Then we can find w_1 is the closest warehouse to supermarket s_2, w_2 is the closest to s_4, s_6 and w_3 is the closest to s_1, s_3, s_5 and s_7.

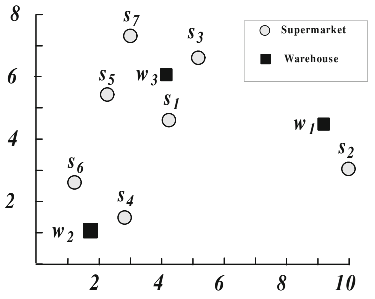


Fig. 1. Motivation Example

Since the number of objects and facilities might be massive in many applications, it is desirable to develop an indexing technique to facilitate the nearest neighbor query and its variants. R-tree technique has been widely employed to support various nearest neighbor queries which relies on the simple rectangular grouping principle, where objects or facilities are grouped by minimal bounding rectangles (MBRs) in a hierarchical way. Efficient NN and ANN algorithms [12,6] have been proposed based on R-tree structure, where the algorithms utilize various distance metrics (e.g., minimal distance and minimal maximal distance) against MBRs. Observe that the R-tree technique may lead to some unnecessary traversals in nearest neighbor query because MBR only provides a coarse granule for various NN queries, Sharifzadeh *et al.* [13] propose the VOR-Tree which incorporates voronoi diagrams into R-tree such that users can quickly identify the search regions with R-tree technique and then provide effective pruning for various NN queries based on the voronoi diagram techniques.

Although VOR-Tree can take advantage of voronoi diagram technique, the benefit is offset by the following facts. Firstly, the intermediate entries of VOR-Tree cannot take advantage of the voronoi diagram technique since they only keep the MBRs of the facilities. This implies that we cannot derive more effective pruning metric compared with the MBR based pruning techniques in [6]. Secondly, since all the vertexes of the voronoi cell and delaunay neighbors are kept for each facility, the capacity (i.e., fan-out) of the VOR-Tree is small. As shown in the empirical study, this results in high I/O cost in NN and ANN queries.

To address the above issues, in the paper we propose a new index approach, termed AVR-Tree, to achieve a better trade-off between the pruning effectiveness and the space usage of the index structure when voronoi diagram technique is applied. More specifically, our index structure is also built on R-tree. But instead of keeping the voronoi cell vertexes and delaunay neighbor information, we maintain the minimal bounding box of a voronoi cell (VBR) and the maximal nearest neighbor distance[1] (MND) for each facility. Moreover, we also maintain the aggregate information of VBRs and MNDs in the intermediate entries. Then effective pruning techniques are proposed for NN and ANN queries based on AVR-Tree structure.

Contributions. Our contributions can be summarized as follows.

- We propose a new data structure, termed AVR-Tree, that effectively utilizes the advantage of the voronoi diagram and the R-tree techniques.
- Efficient nearest neighbor (NN) and all nearest neighbor (ANN) queries are proposed based on AVR-Tree.
- Comprehensive experiments demonstrate the effectiveness and efficiency of our techniques.

Organization of the Paper. The remainder of the paper is organized as follows. Section 2 formally defines the NN and ANN query. In Section 3, we propose our new data structure, AVR-Tree. Efficient NN and ANN algorithms are proposed based on AVR-Tree in Section 4 and Section 5 respectively. Results of comprehensive performance studies are presented in Section 6. Section 7 introduces the related work. Finally, Section 8 concludes the paper.

2 Preliminary

In this section, we will formally define the problems we study in this paper in Section 2.1. In Section 2.2 we briefly describe the voronoi diagram. Table 1 below summarizes the notations frequently used throughout the paper.

Table 1. The summary of notations

Notation	Definition
o ($\mathcal{O}$)	object (a set of objects)
f ($\mathcal{F}$)	facility (a set of facilities)
$NN(o)$	the nearest neighbor of an object o in facilities $\mathcal{F}$
VBR	the voronoi minimal bounding rectangle for facility entries (data entries and intermediate entries)
MND	the maximal nearest neighbor distance for data entries and intermediate entries

2.1 Problem Definition

A point (location) p referred throughout the paper, by default, is in d-dimensional numerical space. Let $\delta(p, f)$ denote the Euclidean distance between two points (locations) p and f. For simplicity, we use Euclidean distance in the paper. Nevertheless, the techniques proposed in the paper can be easily extended to other metric distances.

[1] The maximal distance if the facility is the nearest neighbor.

The NN query is formally defined as follows.

Definition 1. ***Nearest Neighbor(NN) Query.*** *Given an object o and a set $\mathcal{F}$ of facilities, the nearest neighbor of o, denoted by $NN(o)$, is the facility $f \in \mathcal{F}$ with the closest distance to o; that is, $\delta(o, f) \leq \delta(o, f')$ for any facility f' in $\mathcal{F}$.*

Then we have the definition of all nearest neighbor query in which we aim to find the nearest neighbors for a set of objects.

Definition 2. ***All Nearest Neighbor (ANN) Query*** *Given a set $\mathcal{O}$ of objects and a set $\mathcal{F}$ of facilities, for each object $o \in \mathcal{O}$, we retrieve its nearest neighbor from $\mathcal{F}$.*

Example 1. In Fig. 2, there are three objects $\{o_1, o_2, o_3\}$ and three facilities $\{f_1, f_2, f_3\}$ in $\mathcal{O}$ and $\mathcal{F}$ respectively. Given the object o_1, its nearest neighbor regarding $\mathcal{F}$ is f_2, i.e., $NN(o_1) = f_2$. Fig. 2 shows the nearest neighbors for all objects, which can be obtained by issuing the all nearest neighbor query against $\mathcal{O}$ and $\mathcal{F}$.

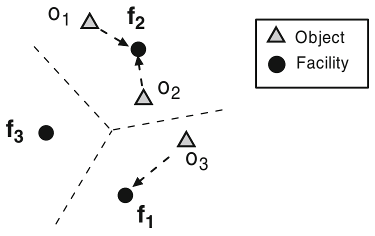


Fig. 2. NN and ANN Example

2.2 Voronoi Diagram

Voronoi diagram is a data structure that consists of a set of voronoi cells where each cell is a polygon. Each cell has an owner which is usually called as seed. Voronoi diagram provides us a nice feature for nearest neighbor search, which is if a point is in a voronoi cell, this point is guaranteed to be the nearest neighbor of the cell's seed. Although the cells are constructed in a way that guarantees the seed is the nearest neighbor of all the points fall in the cell, determining which cell a point falls in can be time consuming.

3 AVR-Tree

In this section, we will introduce a new data structure, termed AVR-Tree, to efficiently facilitate the NN and ANN queries. In Section 3.1, we will show the motivation of the AVR-Tree and then followed by the detailed description of the new indexing structure, including the maintenance of the AVR-Tree.

3.1 Motivation

As the facilities are usually organized by some hierarchical spatial indexes such as R-tree and the NN search follows the *branch-and-bound* travel paradigm, it is critical to estimate the minimal and maximal possible distances between a query object o and a minimal bounding rectangle (MBR) such that some non-promising entries can be excluded from computation at high level to significantly reduce the I/O and CPU costs.

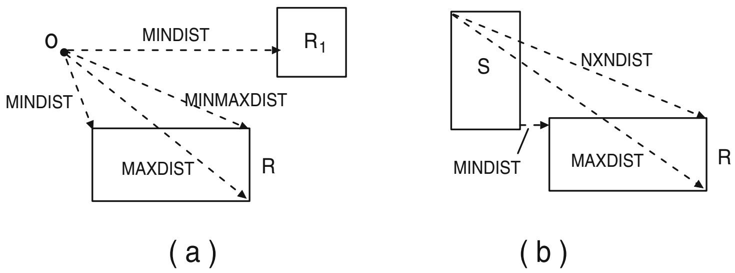


Fig. 3. Example for upper and lower bounds for NN distance

For a query object o, we use nearest neighbor distance, denoted by $nnd(o)$, to represent the distance between o and its nearest neighbor. Given a minimal bounding rectangle (MBR) R of a set of facilities, we use $nnd_{min}(o, R)$ ($nnd_{max}(o, R)$) to denote the lower (upper) bound of the nearest neighbor distance of o regarding facilities located in R. As shown in Fig. 3(a), an immediate approach is to use the minimal distance (MINDIST) and maximal distance (MAXDIST) between o and R as $nnd_{min}(o, R)$ and $nnd_{max}(o, R)$. See detailed calculation in [12]. Observe that each MBR's face is touching at least one facility within R, the concept of MiniMax Distance (MINMAXDIST) is proposed in [12] which can provide a tighter upper bound for $nnd_{max}(o, R)$ as shown in Fig. 3(a).

Example 2. In the example of Fig. 3(a), MINMAXDIST between o and R is smaller than MINDIST between o and R_1. This implies that none of the facilities in R_1 can be the nearest neighbor of o and hence R_1 is pruned without exploring its facilities. On the other hand, we cannot prune R_1 if MAXDIST is used as $nnd_{max}(o, R)$.

With similar rationale, a new pruning metric NXNDIST is proposed in [6], which can provide a tighter upper bound of the nearest neighbor distance between two MBRs. More specifically, as shown in Fig. 3(b), S and R represent MBRs of a set of objects and facilities respectively. For any object o within S, its nearest neighbor distance regarding the facilities within R is bounded by NXNDIST. The empirical study in [6] shows that NXNDIST is much more effective than MAXDIST between two MBRs.

Observe that the MBRs based pruning technique is less effective compared with the Voronoi technique, in [13], VOR-Tree combines the R-tree and Voronoi diagram techniques to speed up the nearest neighbor search. Nevertheless, since all the vertexes of the voronoi cell and delaunay neighbors are kept for each facility, the node capacity (i.e., fan-out) of VOR-Tree is small. For instance, the node capacity of VOR-Tree and R-tree is 40 and 204 respectively when page size is 4K. As shown in the empirical study, this results in expensive I/O costs in NN and ANN queries. Moreover, the intermediate entries of VOR-Tree cannot take advantage of the Voronoi diagram technique since they only keep the MBR of the facilities. This implies that they cannot derive more effective pruning metric compared with the MBR based pruning techniques.

Motivated by the above facts, in the paper we propose a new index approach, termed AVR-Tree, to achieve a better trade-off between the pruning capability and the overhead of the index. Before going to the detail, we first introduce two notations:

Definition 3. ***Voronoi Bounding Rectangle (VBR).*** *Given a facility f, the voronoi bounding rectangle (VBR) of a facility is the minimum bounding box of its voronoi cell, denoted by $f.VBR$. Given a set of facilities F, the VBR of F is the minimal bounding rectangle of the VBRs of all facilities in F, denoted by $F.VBR$.*

Besides the VBR, we also maintain the maximum nearest neighbor distance (MND) for each facility, which is defined as follow.

Definition 4. ***Maximum Nearest Neighbor Distance (MND).*** *Given a facility f, the Maximum Nearest Neighbor Distance (MND) of f , denoted by $f.MND$, is the furthest distance between f and all locations within its voronoi cell; that is, we have $\delta(o, f) \leq f.MND$ if f is the nearest neighbor of an object o. Similarly, we use $F.MND$ to denote the maximal MND values among the facilities in F.*

Since each voronoi cell is a convex polygon, it is immediate that we can derive $f.MND$ by finding the furthest voronoi cell vertex regarding the facility f.

Example 3. In Fig. 4(a), the polygon (v_1, v_2, v_3, v_4, v_5) is the voronoi cell of facility p_1. The VBR of f_1 ($f_1.VBR$) is shown in the example, as well as $p_1.MND$, which is the distance between p_1 and v_2.

Unlike the VOR-Tree which keeps all voronoi cell vertexes and delaunay neighbors for each facility, we only maintain the VBR and MND of a facility so that we can utilize the nice property of the voronoi cell with smaller index size (i.e., larger node capacity). Moreover, we can naturally keep the VBR and MND for a set of facilities and apply VBR and MND based pruning for intermediate entries in AVR-Tree.

In Section 3.2, we will introduce the data structure of AVR-Tree which integrates VBR and MND of the facilities to R-tree structure. Moreover, effective pruning techniques are proposed for NN and ANN queries in Section 4 and Section 5 respectively.

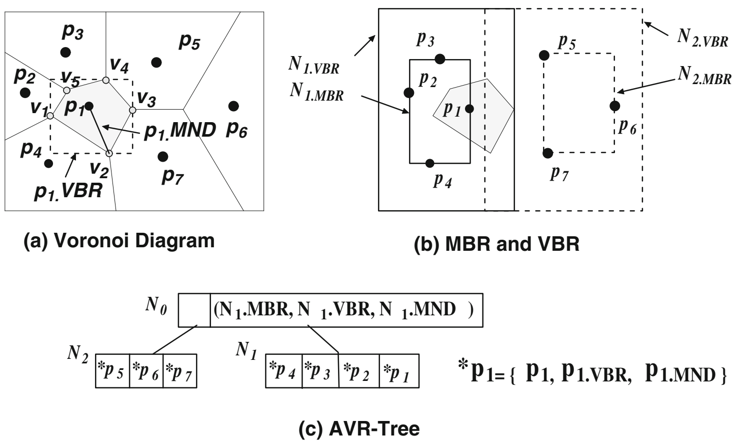


Fig. 4. AVR-Tree

3.2 AVR-Tree Structure

The Aggregate Voronoi R-tree (AVR-Tree for short) is an R-tree like structure which enriches with VBRs and MNDs of the facilities. A data entry of the AVR-Tree consists of a facility location and its VBR and MND. As to an intermediate entry, the MBR, VBR and MND are obtained from its descendant entries.

Fig. 4 illustrates an example of AVR-Tree for a set $\mathcal{F}$ of facilities $\{p_1, \ldots, p_7\}$, where there are 7 data entries $\{^*p_1, \ldots, ^*p_7\}$ and 3 intermediate entries $\{N_0, N_1, N_2\}$. Fig. 4(a) shows the Voronoi diagram of $\mathcal{F}$ and the data entries (e.g., $^*p_1$) can be constructed based on the corresponding voronoi cells. Fig. 4(b) depicts the MBRs and VBRs of the intermediate entries N_1 and N_2 where facilities $\{p_1, p_2, p_3, p_4\}$ are contained by N_1 and others are assigned to N_2. Note that the MNDs are also kept for N_1 and N_2. Fig. 4(c) shows the structure of the AVR-Tree.

3.3 AVR-Tree Construction

Construction of an AVR-Tree consists of two phases. In the first phase, we build an R-tree against the facilities following the standard R-tree algorithm [8]. In the second phase, the Voronoi diagram is constructed and the VBRs and MNDs information of the AVR-Tree entries are obtained in a bottom up fashion.

3.4 AVR-Tree Update

The insertion and deletion of a facility in an AVR-Tree is exactly the same as the R-tree except that we also need to recalculate the VBRs and MNDs of the facilities affected by the update, and then propagate the changes to their parent entries in a bottom up fashion.

4 Nearest Neighbor (NN) Query

In this Section, we introduce the AVR-Tree based nearest neighbor query. We first introduce the pruning technique, then the detailed algorithm.

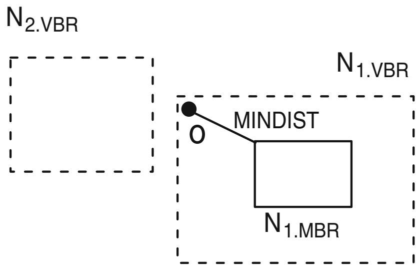


Fig. 5. Pruning Example for NN search

Theorem 1. *Given an object o and an AVR-Tree entry F (data entry or intermediate entry), F can be excluded from NN of o if* $F.VBR \cap o = \emptyset$ *or* $F.MND < MINDIST(o, F.MBR$ [2]*).*

Proof. $F.VBR \cap o = \emptyset$ implies that $f.VBR \cap o = \emptyset$ for all facilities f within F since $F.VBR$ is the minimal bounding box of $\{f.VBR\}$ for all facilities f from F. On the other hand, an object o must fall in the voronoi cell of f if f is the NN of o. Since the voronoi cell of f is contained by $f.VBR$, we can safely claim that none of the facilities in F can be the nearest neighbor of o if $F.VBR \cap o = \emptyset$.

We prove the second case by the contradiction. Suppose a facility f from F is the NN of o, and then we have $\delta(o, f) \leq f.MND$. Consequently, we have $\delta(o, f) \leq F.MND$ since $f.MND \leq F.MND$. Clearly we have $MINDIST\ (o, F.MBR) \leq \delta(o, f)$ since $f \in F.MBR$, and hence $MINDIST(o, F.MBR) \leq F.MND$. This is against the assumption $MINDIST(o, F.MBR) > F.MND$. □

Example 4. Regarding the example in Fig. 5, we can immediately prune the entry N_2 since $N_2.VBR \cap o = \emptyset$. If $MINDIST(o, N_1.MBR) > N_1.MND$, N_1 can also be eliminated from NN candidate of the object o.

In Algorithm 1 we illustrate the details of the NN algorithm assuming the facilities $\mathcal{F}$ are organized by an AVR-Tree, represented by $\mathcal{F}_{AVR}$. The algorithm follows the *branch-and-bound* paradigm. A priority queue Q is employed to access the AVR-Tree entries where the key of an entry is its closest distance to the query object o. Consequently, algorithm can terminate when the first object is popped from Q (Line 1). For each intermediate entry popped from Q, Line 1 eliminates its non-promising child entries based on Theorem 1. Our empirical study demonstrates the efficiency of Algorithm 1 since a large number of non-promising entries can be pruned at low cost.

[2] If F is a data entry (i.e., facility), then the location of F is $F.MBR$.

Algorithm 1. $NN(o, \mathcal{F}_{AVR})$

```
   Input   : $o$ : an object $o$ ,
             $\mathcal{F}_{AVR}$ : a set $\mathcal{F}$ of facilities organized by AVR-Tree
   Output  : the nearest neighbor of $o$ in $\mathcal{F}$
1  $Q \leftarrow$ root of $\mathcal{F}_{AVR}$ ;
2  while $Q \neq \emptyset$ do
3      $E \leftarrow$ top element popped from $Q$ ;
4      if $E$ is a data entry then
5          return the facility $f$ associated with $E$
6      else
7          for each child entry $e$ of $E$ do
8              if $e.VBR \cap o \neq \emptyset$ and $MINDIST(o, e.MBR) \leq e.MND$ then
9                  Push $e$ into $Q$ with key value $MINDIST(o, e.MBR)$ ;
```

5 All Nearest Neighbor (ANN) Query

In this section, we investigate the problem of all nearest neighbor (ANN) query. We first introduce the AVR-Tree based pruning techniques, then the details of the ANN algorithm.

Theorem 2. *Given the MBR of a set of objects, denoted by $O.MBR$, and an AVR-Tree entry F (data entry or intermediate entry), F can be excluded from NN calculation of any object $o \in O.MBR$ if $F.VBR \cap O.MBR = \emptyset$ or $MINDIST\ (O.MBR, F.MBR) > F.MND$.*

Proof. The proof of this Theorem is similar to that of Theorem 1. We omit the details due to the space limitation.

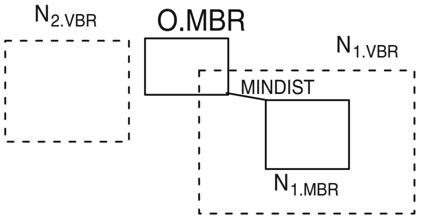


Fig. 6. Pruning Example for ANN query

Example 5. In Fig. 6 we can safely prune N_2 from the nearest neighbor computation for the objects in $O.MBR$ since $O.MBR \cap N_2.VBR = \emptyset$. Similarly, N_1 can be eliminated if $MINDIST\ (N_1.MBR, O.MBR) > N_1.MND$.

Algorithm 2 illustrates the details of the all nearest neighbor (ANN) query, where we assume a set $\mathcal{O}$ of objects and a set $\mathcal{F}$ of facilities are organized by R-tree $\mathcal{O}_R$ and AVR-Tree $\mathcal{F}_{AVR}$ respectively. Similar to spatial join [5,10], we apply the synchronized R-tree traversal paradigm to process ANN query in a level by level fashion. In Algorithm 2, we use a tuple to maintain the object R-tree entry (intermediate or data entry), denoted by $T.O$, as well as a set of facility AVR-Tree entries (intermediate or data entries), denoted by $T.F$, which may contribute to the nearest neighbors for the objects within $T.O$. A FIFO queue, denoted by Q, is employed to maintain the tuples and is initialized as $<$ root of $\mathcal{O}_R$, root of $\mathcal{F}_{AVR}$ $>$ at Line 2. Line 2-2 iteratively process each tuple in the queue until it is empty. When a tuple T is popped from Q, we can come up with the nearest neighbor of the object if all entries in $T.O$ and $T.F$ are data entries (Line 2-2). Otherwise, Line 2 retrieves all child entries of $T.O$ to set $\mathcal{L}$. Similarly, the set $\mathcal{R}$ keeps all child entries of the entries in $T.F$ [3]. For each object entry $o_e \in \mathcal{L}$, we identify the facilities from $\mathcal{R}$ which may be the nearest neighbor of an object within o_e. Specifically, Theorem 2 is applied at Line 2 to eliminate non-promising facility entries regarding o_e. When Algorithm 2 terminates, the nearest neighbors of all objects in $\mathcal{O}$ are retrieved.

Algorithm 2. ***ANN***$(\mathcal{O}_R, \mathcal{F}_{AVR})$

```
   Input  : $\mathcal{O}_R$ : the objects organized by R-tree,
            $\mathcal{F}_{AVR}$ : the facilities organized by AVR-Tree
   Output : $\mathcal{N}$ : the nearest neighbors for all objects in $\mathcal{O}$
1  $Q \leftarrow$ tuple $<$ root of $\mathcal{O}_R$, root of $\mathcal{F}_{AVR}$ $>$;
2  while $Q \neq \emptyset$ do
3      $T \leftarrow$ the tuple popped from $Q$ ;
4      if both $T.O$ and $T.F$ are data entries then
5          $o \leftarrow$ object associated with $T.O$ ;
6          Calculate distances between $o$ and the facilities in $T.F$;
7          $\mathcal{N} \leftarrow < o$, the nearest neighbor of $o >$ ;
8      else
9          $\mathcal{L}$ := child entries of $T.O$ ;
10         $\mathcal{R}$ := child entries of the entries in $T.F$ ;
11         for each entry $o_e \in \mathcal{L}$ do
12             $\mathcal{C} = \emptyset$ ;
13             for each entry $f_e \in \mathcal{R}$ do
14                 if $f_e.VBR \cap o_e.MBR \neq \emptyset$ and MINDIST$(f_e.MBR, o_e.MBR)$
                   $\leq f_e.MND$ then
15                     $C := C \cup f_e$;
16             push $< o_e, C >$ into $Q$ ;
```

6 Performance Evaluation

We present results of a comprehensive performance study to evaluate the efficiency and scalability of the proposed techniques in the paper. Following algorithms are evaluated.

[3] Note that we have $L = T.O$ if $T.O$ is a data entry and facility entries are intermediate entries. The set $\mathcal{R}$ is handled in the same way.

R-tree the algorithms in which facilities are organized by R-tree. The NN and ANN queries are implemented based on the techniques proposed in [12] and [6] respectively.

VOR-Tree the algorithms in which facilities are organized by VOR-Tree [13].

AVR-Tree the algorithms in which facilities are organized by AVR-Tree. Particularly, the NN and ANN queries are implementation as described in Algorithm 1 and Algorithm 2 in the paper.

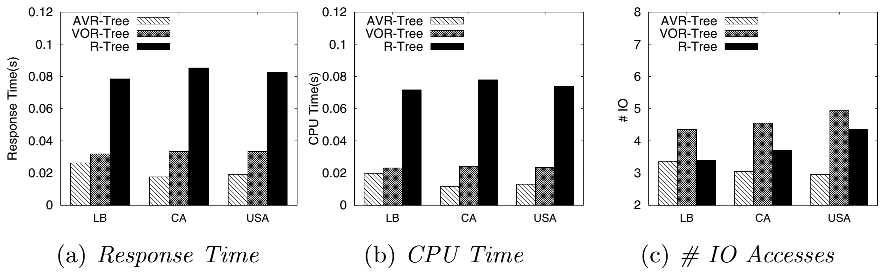
All algorithms in this paper are implemented in standard C++ with STL library support and compiled with GNU GCC. Experiments are performed on a PC with Intel Xeon 2.50GHz dual CPU and 2G memory running Debian Linux. The disk page size is fixed to 4096 bytes. The node capacity of the R-tree, AVR-Tree and VOR-Tree are 204, 102 and 40 respectively. Besides the MBR, we also keep the VBR (voronoi bounding rectangle) and MND (maximal nearest neighbor distance) information in each node of AVR-Tree. Therefore, the node capacity of AVR-Tree is smaller than that of R-tree. Similarly, we need to keep the voronoi neighbor vertexes and delaunay neighbor identifications for each location in VOR-Tree, and VOR-Tree has smallest node capacity. We use an LRU memory buffer with size 512K bytes.

Real Datasets. Three real spatial datasets, *LB*,*CA* and *USA*, are employed in the experiments. The facilities in *CA* are 45,671 facility locations in California (`http://www.cs.fsu.edu/~lifeifei/SpatialDataset.htm`). Similarly, the facilities in *USA* are obtained from the U.S. Geological Survey (USGS) and consists of 205,323 facility locations (`http://www.geonames.usgs.gov`). Meanwhile, the object locations of *CA* and *USA* are public available from `http://www.census.gov/geo/www/tiger`, where there are 62,556 and 1M objects in *CA* and *USA* respectively. In *LB*, both facility and object locations are obtained from the Long Beach dataset from `http://www.census.gov/geo/www/tiger`, in which there are 53,145 locations. In the above three datasets, all dimensions are normalized to domain $[0, 10000]$.

In the paper, the processing time, which includes the CPU time and I/O latency, is used to measure the efficiency of the algorithms. We also record the number of I/O accesses in the algorithms.

6.1 Evaluate Nearest Neighbor Search

In the first set of experiments, we evaluate the performance of three NN search algorithms based on R-tree, VOR-Tree and AVR-Tree against dataset *LB*, *CA* and *USA*, where 1,000 query points are randomly chosen from $[0, 10000]^2$. The average query response time, CPU time and the number of I/O accesses for the three algorithms are reported in Fig. 7. It is shown that AVR-Tree outperforms R-tree and VOR-Tree on both CPU time (Fig. 7(b)) and I/O cost (Fig. 7(c)), and hence has the fastest query response time (Fig. 7(a)). As shown in Fig. 7(c) VOR-Tree incurs the largest number of I/O accesses due to its small node capacity. Nevertheless, its overall performance (i.e., query response time) is better

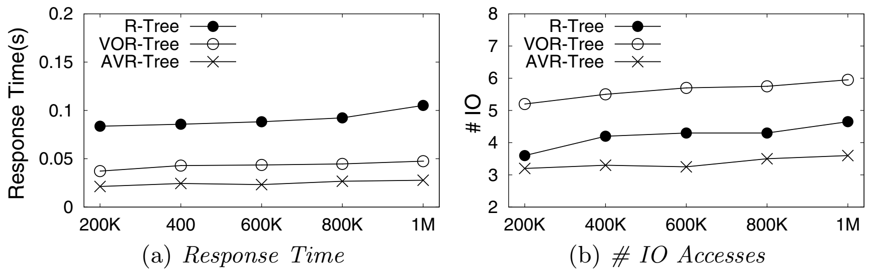


(a) *Response Time* (b) *CPU Time* (c) *# IO Accesses*

Fig. 7. NN Query on Different Datasets

than that of R-tree since it can take advantage of voronoi diagrams and use much less CPU time as shown in Fig. 7(b).

In the second set of experiments, we evaluate the scalability of the three algorithms where the facility locations are randomly chosen from *USA* object locations, with size ranging from 200K to 1M. Fig. 8 shows that the performance of the three algorithms are scalable towards the growth of the number of facilities.

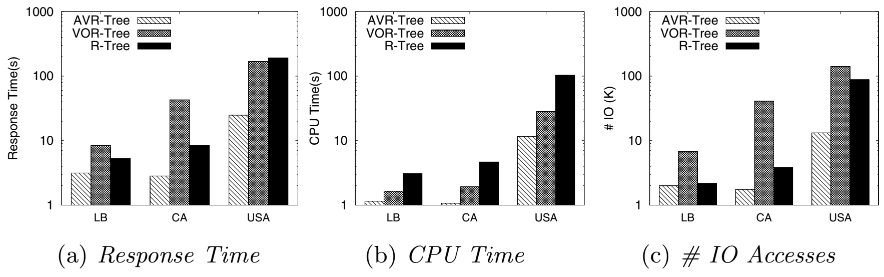


(a) *Response Time* (b) *# IO Accesses*

Fig. 8. Effect of the number of facilities

6.2 Evaluate All Nearest Neighbor (ANN) Query

Fig. 9 evaluates the performance of R-tree, VOR-Tree and AVR-Tree Algorithms for all nearest neighbor (ANN) query, where three datasets *LB*, *CA* and *USA*



(a) *Response Time* (b) *CPU Time* (c) *# IO Accesses*

Fig. 9. ANN Query on Different Datasets

are employed. It is shown that the performance of AVR-Tree significantly outperforms R-tree and VOR-Tree. In *USA*, the query response time of AVR-Tree is 8 times faster than that of R-tree and VOR-Tree. Similar to NN query in Section 6.1, Fig. 9(b) and Fig. 9(c) show that VOR-Tree is more time efficient compared with R-tree, but invokes more I/O accesses due to its small node capacity. AVR-Tree demonstrates superior performance in comparison with R-tree and VOR-Tree because it can achieve a good trade-off between the pruning effectiveness and indexing overhead (e.g., node capacity).

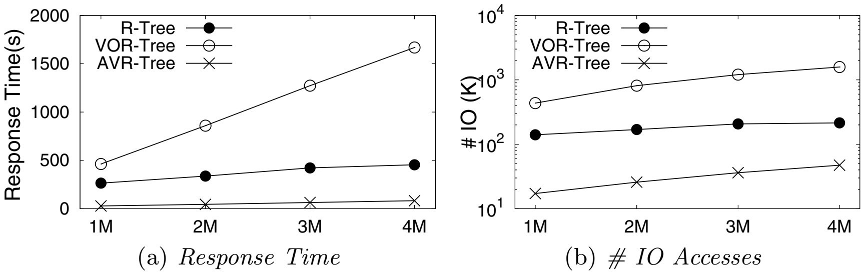


Fig. 10. Effect of the number of objects

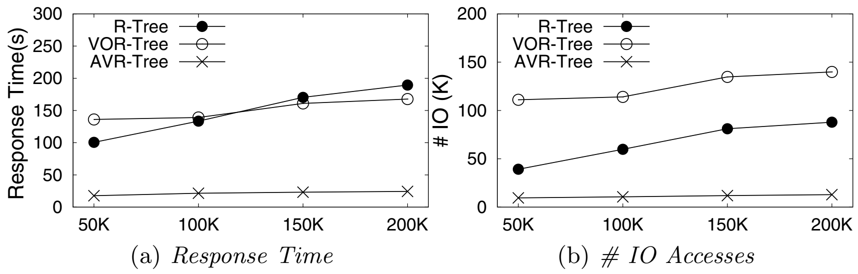


Fig. 11. Effect of the number of facilities

We also study the scalability of the three algorithms towards the growth of the number of facilities and objects in the ANN query. In Fig. 10, a set of object locations are randomly chosen from $[1, 100000]^2$ with size ranging from 1M to 4M. It is reported that the scalability of VOR-Tree is poor because it can only apply the MBR based pruning on facility and object entries at the intermediate level and the node capacity is small due to a large amount of information kept for each facility. As expected, AVR-Tree always ranks first when the number of objects grows. Similar observation is reported in Fig. 11, where a number of facilities are randomly chosen from *USA*.

6.3 Index Construction

In this subsection, we evaluate the index size and index construction time of R-tree, on the facilities of the three datasets *LB*, *CA* and *USA*. Fig. 12 reports the size of the three indexing structures on *LB*, *CA* and *USA*. As expected, VOR-Tree has the largest index size, followed by AVR-Tree and R-tree. The construction time of the three index structures is illustrated in Fig. 13. As the dominant cost of VOR-Tree and AVR-Tree construction is the R-tree construction, their performance is slightly slower than that of R-tree. Note that the voronoi diagram and delaunay graph computation algorithms are public available from Qhull (`http://www.qhull.org/`).

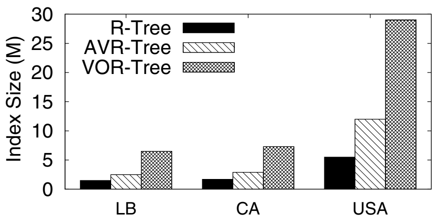


Fig. 12. Index Size

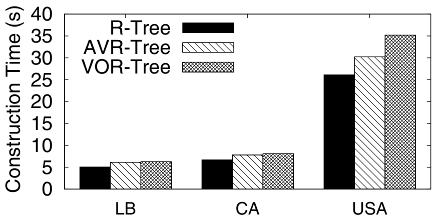


Fig. 13. Construction Time (s)

7 Related Work

The family of the nearest neighbor search problems consists of nearest neighbor (NN) search [14,8] or top k nearest neighbor search [12,3], all nearest neighbor (ANN) search [15,6], reverse nearest neighbor [11] and approximate nearest neighbor [1] and etc. Our work focuses on the NN and ANN queries on location data. The problem of NN query on location data has a wide spectrum applications and has been extensively studied in the literature. Roussopoulos *et al.* [12] propose to use R-tree [8] as the index structure for nearest neighbor search in low dimensional space. [3] proposes a tree-based index to solve nearest neighbor search in high dimensional space. Recently, M. Sharifzadeh *et al.* [13] propose a new structure, VOR-Tree, where it is an R-tree with voronoi cells stored in its data entries. The data entries also store the information of the neighboring cells of each cell.

Our work also relates to ANN search. One of the earliest work proposed by Braumuller *et al.* [4] is to perform one NN search on the facility data set for each data in the object data set. Zhang *et al.* [15] propose an R-tree based approach to search the nearest neighbors as a batch. Corral et al. [7] and Hjaltason et al. [9] propose a technique based on R*-tree structure [2]. Corral *et al.* [7] also propose a pruning rule and two updating strategies to efficiently solve the search problem. Recently, Y. Chen et al. [6] propose a tighter upper bound distance between two MBRs so that the pruning is more effective in order to retrieve all nearest neighbors more efficiently.

8 Conclusion

In this paper, we introduce a new structure, termed AVR-Tree, to organize location data, which utilizes the R-tree and Voronoi diagram techniques. Efficient nearest neighbor (NN) and all nearest neighbor (ANN) algorithms are proposed in the paper and comprehensive experiments are conducted to demonstrate the effectiveness and efficiency of the algorithms.

Acknowledgement. Ying Zhang is supported by ARC DP110104880, ARC DP130103245 and UNSW ECR grant PS27476. Wenjie Zhang is supported by ARC DP120104168 and ARC DE120102144. Xuemin Lin is supported by ARC DP0987557, ARC DP110102937, ARC DP120104168 and NSFC61021004.

References

1. Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R., Wu, A.Y.: An optimal algorithm for approximate nearest neighbor searching fixed dimensions. J. ACM 45(6), 891–923 (1998)
2. Beckmann, N., Kriegel, H.-P., Schneider, R., Seeger, B.: The r*-tree: An efficient and robust access method for points and rectangles. In: SIGMOD Conference, pp. 322–331 (1990)
3. Berchtold, S., Ertl, B., Keim, D.A., Kriegel, H.-P., Seidl, T.: Fast nearest neighbor search in high-dimensional space. In: ICDE, pp. 209–218 (1998)
4. Braunmüller, B., Ester, M., Kriegel, H.-P., Sander, J.: Efficiently supporting multiple similarity queries for mining in metric databases. In: ICDE, pp. 256–267 (2000)
5. Brinkhoff, T., Kriegel, H.-P., Seeger, B.: Efficient processing of spatial joins using r-trees. In: SIGMOD Conference, pp. 237–246 (1993)
6. Chen, Y., Patel, J.M.: Efficient evaluation of all-nearest-neighbor queries. In: ICDE, pp. 1056–1065 (2007)
7. Corral, A., Manolopoulos, Y., Theodoridis, Y., Vassilakopoulos, M.: Algorithms for processing k-closest-pair queries in spatial databases. Data Knowl. Eng. 49(1), 67–104 (2004)
8. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: SIGMOD Conference (1984)
9. Hjaltason, G.R., Samet, H.: Incremental distance join algorithms for spatial databases. In: SIGMOD Conference, pp. 237–248 (1998)
10. Huang, Y.-W., Jing, N., Rundensteiner, E.A.: Spatial joins using R-trees: Breadth-first traversal with global optimizations. In: VLDB 1997 (1997)
11. Korn, F., Muthukrishnan, S.: Influence sets based on reverse nearest neighbor queries. In: SIGMOD Conference, pp. 201–212 (2000)
12. Roussopoulos, N., Kelley, S., Vincent, F.: Nearest neighbor queries. In: SIGMOD Conference, pp. 71–79 (1995)
13. Sharifzadeh, M., Shahabi, C.: Vor-tree: R-trees with voronoi diagrams for efficient processing of spatial nearest neighbor queries. PVLDB 3(1), 1231–1242 (2010)
14. Weber, R., Schek, H.-J., Blott, S.: A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In: VLDB, pp. 194–205 (1998)
15. Zhang, J., Mamoulis, N., Papadias, D., Tao, Y.: All-nearest-neighbors queries in spatial databases. In: SSDBM, pp. 297–306 (2004)