http://ansinet.com/itj

ITJ

ISSN 1812-5638

# INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 9 (7): 1467-1475, 2010
ISSN 1812-5638
© 2010 Asian Network for Scientific Information

# Nearest Neighbors and Continous Nearest Neighbor Queries based on Voronoi Diagrams

[1]Miao Wang and [1,2]Zhongxiao Hao
[1]College of Computer Science and Technology,
Harbin University of Science and Technology, Harbin 150080, China
[2]College of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China

**Abstract:** Voronoi diagram has its advantages in Nearest Neighbors (NN) query. In order to effectively apply Voronoi diagram in NN query and its extension Continuous Nearest Neighbor (CNN) query so as to avoid the pitfalls of the existing approaches based on R-tree for these queries and achieve CNN query with arbitrary query trajectory, in this study, we first devise a data structure VR-tree, in which Voronoi diagram associated with the data space is embeded. Subsequently, an algorithm for nearest neighbor queries using VR-tree is proposed. After thoroughly analyzing the properties of Voronoi diagrams, an algorithm based on Voronoi diagrams for k-nearest neighbors queries is developed. This approach reduces the search space considerably by means of the properties of Voronoi diagram. Finally, we propose an algorithm for continuous nearest neighbor queries using Voronoi diagrams. It is well worth to mention that this approach achieves the continuous nearest neighbor queries with arbitrary query trajectory. The results of both analyzing in theory and experimental evaluation demonstrate that the proposed algorithms significantly outperform the previous ones.

**Key words:** Voronoi diagram, nearest neighbors query, continuous nearest neighbor query, R-tree, VR-tree, k-nearest neighbors query

## INTRODUCTION

Nearest Neighbors (NN) and Continous Nearest Neighbor (CNN) queries are two fundamental problems in the field of spatial databases. These types of queries are widely used in Geographic Information Systems, Database, Intelligent Traffic System *et al*. In the last few years many research contributions have appeared on NN and CNN queries. Roussopoulos *et al.* (1995) and Hjaltason and Samet (1999) proposed a depth first (DF) method and a Best-first (BF) method for nearest neighbors queries by traversing R-tree (Guttman,1984; Beckmann *et al*., 1990) in a branch-and-bound manner, respectively. Many improved methods based on DF method and BF method also has been developed by researchers (Cheung and Fu, 1998; Seidl and Kriegel, 1998; Song *et al*., 2007; Adler and Heeringa, 2008). The first algorithm for CNN query processing, presented in Song and Roussopoulos (2001), employs sampling to compute the result. This method suffers from the drawbacks of sampling. Tao and Papadias (2002) devised an algorithm for CNN query based on the concept of Time-Parameterized (TP) queries. This approach does not incur false misses and thus avoids the drawbacks of sampling, but it is very output-sensitive. The aforementioned two methods are based on the repetitive application of simple NN algorithms traversing R-trees. Tao *et al.* (2002) proposed an algorithm which performs only a single traversal on R-tree for the whole input line segment. This approach avoids the pitfalls of the two algorithms mentioned above, namely, the false misses and the high processing cost. Unfortunately, all these approaches for CNN query can only be applied to the queries with that the query input is a line segment rather than arbitrary trajectory. As a matter of fact, the manner of moving is complex and inconstant for spatio-temporal objects. Moreover, all the existed methods achieve NN and CNN query processing by traversing R-trees. As is known to all, these methods suffer from the drawbacks of unnecessary traversals due to the fact that there are overlapping rectangles in intermediate nodes of R-tree and the performance of them deteriorates rapidly as the overlap between minimum bounding rectangles in the directory of R-tree increases.

It is well known that Voronoi Diagram has many excellent properties for nearest neighbors queries. The application of the Voronoi diagrams to nearest neighbors queries has been theoretically addressed for quite a long time, from the computational geometry perspective. As far as we know, only one related problem that has been solved by Bespamyatnikh and Snoeyink (1999) is that of finding the single NN for the whole line segment. Until

**Corresponding Author:** Miao Wang, College of Computer Science and Technology, Harbin University of Science and Technology, Harbin 150080, China

now, using Voronoi diagrams to nearest neighbors queries has not yet been achieved in spatial databases. It faces the following challenges. First of all, what data structure does it employ? (For example, R-tree or a new data structure that fits to the nearest neighbors queries using Voronoi diagram). This problem has never been addressed in the previous researches. Moreover, how to apply Voronoi diagram to kNN query? Although the application of the Voronoi diagrams to kNN queries has been extensively studied in computation geometry (Lee, 1982; Chazelle and Edelsbrunner, 1987; Meyerhenke, 2005), the approach is based on k-th order Voronoi diagrams. This solution is impractical because it requires that the value of k be predetermined and has a very high complexity due to that k-th order Voronoi polygons have complex shapes.

In order to practically apply Voronoi diagram to nearest neighbors queries, avoid the innate deficiencies of the R-tree traversal algorithms and achieve the CNN queries with arbitrary query trajectory, in this study, we first devise a Voronoi based data structure VR-tree for NN and CNN query. VR-tree integrates the information about the Voronoi diagram associated with the dataset into the data structure so as to give many good properties of Voronoi diagram full play in nearest neighbors queries. Then an algorithm applying Voronoi diagram to 1NN query is proposed. This approach performs only a single traversal on VR-tree where any unnecessary search path is excluded because zero overlap and coverage is achieved in VR-tree and thus avoids the drawbacks of the previous approaches based on R-tree. Furthermore, we propose a novel algorithm for kNN queries on static dataset based on first order Voronoi diagram. This approach uses VR-tree as the underlying data structure and makes full use of the properties of first order Voronoi diagram to reduce the search space from the whole data space to a particular area that involves only some objects which reduces the processing cost considerably, particularly to kNN search on large data space. Therefore this method avoids the pitfalls of previous approaches based on k-th order Voronoi diagram. Finally, we develop an algorithm which, using VR-tree as the underlying data structure, efficiently and accurately implements CNN search by means of Voronoi diagrams. This approach achieved CNN queries with arbitrary query trajectory.

## PRELIMINARIES

The present study, concentrates on NN and CNN queries based on Voronoi diagrams. Here, we review the concept of the nearest neighbors query and the basic concepts and geometric properties of the Voronoi diagrams presented in Sack and Urrutia (2000). Throughout this paper, the Euclidean distance between two arbitrary points a and b is denoted by d (a, b).

The formal definitions of NN query and kNN query are presented as follows.

**Definition 1:** Given a point set S and a query point q, the NN query gives a point set NNS(q) which is a subset of S such that Eq. 1 holds:

$$NNS(q)=\{r \in S|\forall p \in S;\ d(q, r) \le d(q,p)\} \tag{1}$$

**Definition 2:** Given a point set S, a query point q and a positive integer k, the kNN query gives a point set kNNS(q) which is a subset of S such that $d(q,r) \le d(q,p)$ holds for any point $r \in kNNS(q)$ and for any point $p \in (S\text{-}kNNS(q))$.

The formal definition of the Voronoi diagrams is presented as follows:

**Definition 3:** Assume a set of generators $P = \{p_1,\ldots, p_n\} \subset R^2$, where $2<n<\infty$ and $p_i \ne p_j$ for $i \ne j$, $i, j \in I_n = \{1,\ldots,n\}$. The region defined by $V(p_i) = \{p|d(p,p_i) \le d(p, p_j)\}$ for $j \ne i$, $j \in I_n$ where, $d(p, p_i)$ denotes the distance between p and $p_i$, is called the Voronoi Polygon associated with $p_i$ and the set given by: $VD(P) = \{V(p_1),\ldots, V(p_n)\}$ is called the Voronoi Diagram (also called the first order Voronoi diagram) generated by P and the boundaries of the polygons are called Voronoi edges. The Voronoi polygons that share the same edges are called adjacent polygons and their generators are called adjacent generators. Figure 1 shows an example of a Voronoi diagram.

**Definition 4:** Assume a set of generators $P = \{p_1, p_2,\ldots, p_n\} \subset R^2$ and a set $Q \subseteq P$, the region given by $VR(Q) = \bigcup_{p_i \in Q} V(p_i)$, is called the Voronoi Region associated with Q.

Several properties of the Voronoi diagrams used in the following are presented as follows.

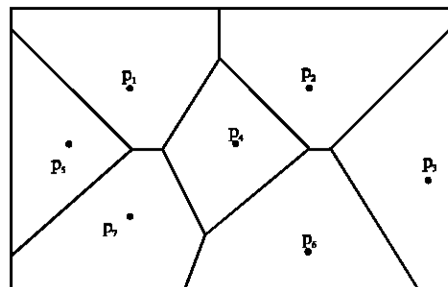


Fig. 1: Example of a voronoi diagram

**Property 1:** A Voronoi diagram divides a space into disjoint polygons where the nearest neighbor of any point inside a polygon is the generator of the polygon.

**Property 2:** Each Voronoi edge is a segment of the perpendicular bisector of a pair of generators.

**Property 3:** Each Voronoi edge is shared by two Voronoi polygons and average number of Voronoi edges per Voronoi polygon is at most 6. This means that each generator has 6 adjacent generators at most.

## VR-TREE

In order to take full advantages of Voronoi diagram in nearest neighbors queries, we devise a novel data structure VR-tree in which Voronoi polygons associated with dataset are stored in leaf nodes and intermediate nodes are built by grouping Voronoi Regions at the lower level nodes and every intermediate node is associated with some Voronoi region which completely encloses all Voronoi regions that correspond to lower level nodes. VR-tree uses Voronoi Regions to group objects similarly with minimum bounding rectangles used in R-trees, to embed the information about the Voronoi diagram associated with dataset in this data structure. An example provided to illustrate how to use Voronoi Regions to group objects in VR-tree associated with these objects presented in Fig. 1 is shown in Fig. 2.

Now, we formally define the VR-tree. Leaf nodes in a VR-tree contain index record entries of the form:

(oid, VP, hash-pointer)

where, oid is an object identifier and is used to refer to an object in the database. VP is used to describe the Voronoi polygon associated with this object. hash-pointer is a pointer to a tuple of a hash table associated with this object. The tuple is of the form:

(adj-set, adj-size)

adj-set is the set of generator points whose Voronoi polygons share same Voronoi edges with the Voronoi polygon associated with this object. adj-size denotes the number of the elements in this set. An intermediate node is of the form:

(child, VR)

where, child is a pointer to a lower level node of the tree and VR is a Voronoi Region that covers all Voronoi

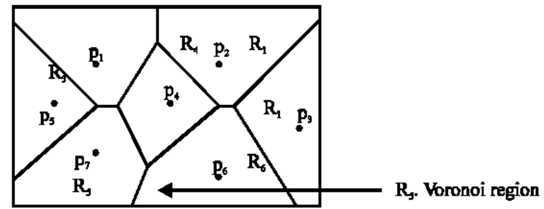


Fig. 2: Some voronoi regions organized into a VR-tree

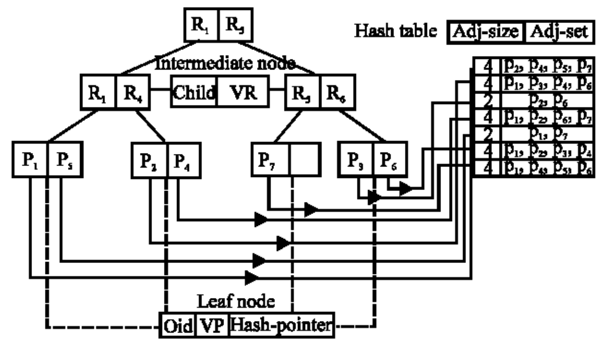


Fig. 3: Example of a VR-tree

Regions in the entries of the lower nodes. Let us assume that M is the maximum number of entries that can fit in a leaf or intermediate node and $m \leq M/2$ is a parameter specifying the minimum number of entries in a node. Figure 3 shows the VR-tree built on these Voronoi Regions shown in Fig. 2.

In essence, VR-tree is equivalent to R-tree in which minimum bounding rectangle is replaced by Voronoi Region to group together objects. This data structure is apt to be used in nearest neighbors queries due to that Voronoi diagram has many good properties. Nearest neighbor queries employ VR-tree without any unnecessary traversal because the Voronoi Regions are mutually exclusive in the directory of VR-tree.

## NEAREST NEIGHBORS QUERIES BASED ON VORONOI DIAGRAMS

**An algorithm for 1NN queries based on Voronoi diagrams:** According to Property 1 of the Voronoi diagram, It merely needs to locate in which polygon the given query point lies, to 1NN queries, namely the generator of this Voronoi polygon is the nearest neighbor of the query point. After this brief discussion, an algorithm, employing VR-tree to 1NN queries, is shown as follows.

```
Algorithm: VNN( n , q )
Input: a query point q, a VR-tree node n
Output: the nearest neighbor of q
begin
  if n is a leaf node then
    for each entry (oid, VP, hash-pointer) in node n do
            if (q∈VP) then
               return p;
else if n is an intermediate node then
  for each entry(child, VR) in node n do
       if (q∈VR ) then
            VNN(child, q);
end
```

**Theorem 1:** Algorithm VNN is correct and has time complexity of O (log n) where n is the number of the objects in the dataset.

**Proof:** This algorithm implements a regular recursive top-down traversal on VR-tree to locate a leaf node containing the query object. The traversal is initiated with the root node of the tree. In the traversal, for an intermediate node, if it does not contain the query object, then this node is pruned according to Property 1 of the Voronoi diagram, else this algorithm is recursively invoked on its child nodes, which is achieved by the second if-then. This algorithm terminates when a leaf node that contains the query object is encountered, namely the generator associated with the leaf node is the nearest neighbor of the query object according to Property 1, which is achieved by the first if-then. Therefore this algorithm is correct.

This algorithm performs only a single depth-first traversal on the VR- tree associated with the dataset to retrieve the nearest neighbor. The height of the VR- tree corresponding to the dataset with n elements is no more than $\lfloor \log n \rfloor + 1$. Therefore the complexity of this algorithm is O (log n) and thus the theorem holds.

Although, the time complexity of VNN is on the same level with previous ones based on R-tree, our approach outperforms the previous ones based on R-tree due to making use of Property 1 of the Voronoi diagram to prune the search space and the fact that there is no any unnecessary search path, in the course of traversing on VR-tree, particularly to large data space.

**An algorithm for kNN queries based on Voronoi diagrams:** In this study, we devise a novel algorithm for kNN queries which makes use of the properties of the Voronoi diagram to pre-compute the candidate set. The search space is reduced to many particular points related to the query point through this pre-computation and thus this algorithm has a lower processing cost. Now, we give several theorems which provide theoretical support to determine the candidate set.

Before the theorems are presented, we give an important concept of k-order adjacent generator of a generator in Voronoi diagrams which is defined formally as follows.

**Definition 5:** Given a Voronoi diagram generated by a set of generators $P = \{p_1,\ldots, p_n\} \subset R^2$, where $2<n<\infty$ and $p_i \neq p_j$ for $i \neq j$, $i, j \in I_n=\{1,\ldots,n\}$ and an integer k, k-order adjacent generator of $p_i \in P$ can be defined:

- 1-order adjacent generators:

$$AG_1(p_i) = \{p_j | V(p_i) \cap V(p_j) \neq \emptyset\}$$

- k-order adjacent generators

$$AG_k(p_i) = \{p_j | \exists p \in AG_{k-1}(p_i), V(p) \cap V(p_j) \neq \emptyset\}$$

For example, $AG_1(p_4) = \{p_1, p_2, p_6, p_7\}$ and $AG_2(p_4) = \{p_3, p_5\}$ in the Voronoi diagram shown in Fig. 1.

**Theorem 2:** In a Voronoi diagram generated by a set of generators $P = \{p_1,\ldots,p_n\} \subset R^2$, where $2<n<\infty$ and $p_i \neq p_j$ for $i \neq j$, $i, j \in I_n=\{1,\ldots,n\}$, assume point $q \in V(p)$ and $p_{k+1} \in AG_{k+1}(p_i)$. Then, there exists a generator $p_k \in AG_k(p_i)$ such that $d(q, p_k) \leq d(q, p_{k+1})$.

**Proof:** Suppose that the nearest neighbor of q is $p_4$ and $p_3$ is some (k+1)-order adjacent generator of $p_4$. The straight line connecting q and the (k+1)-order adjacent generator of the nearest neighbor of q must pass through a Voronoi polygon associated with $p_6$ which is a k-order adjacent generator of the nearest neighbor of q. Hence, The line segment $qp_3$ must intersect the boundary of $V(p_6)$ at points $b_1$ and $b_2$ (Fig. 4). According to Property2 of the Voronoi diagram, we have that $d(p_6, b_2) = d(p_3, b_2)$. In the triangle $\Delta\ b_1p_6b_2$, we know that $d(b_1,b_2)+d(b_2, p_6) \geq d(b_1, p_6)$. We can conclude that $d(b_1,b_2) + d(p_3, b_2) \geq d(b_1, p_6)$ and by adding $d(q,b_1)$ to both sides of the inequality, we have that $d(q, b_1) + d(b_1,b_2) + d(b_2, p_6) \geq d(q, b_1) + d(b_1, p_6)$, namely $d(q, p_6) \leq d(q, p_3)$ and thus, the theorem holds.

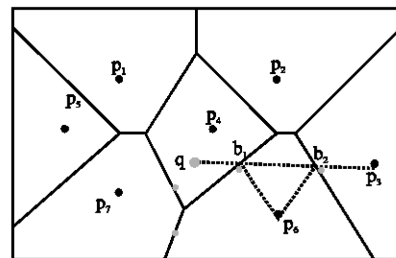


Fig. 4: Proof of Theorem 2

**Theorem 3:** In a Voronoi diagram generated by a set of generators $P = \{p_1,\ldots,p_n\}\subset R^2$, where $2<n<\infty$ and $p_i \neq p_j$ for $I\neq j$, i, $j\in I_n=\{1,\ldots,n\}$, for any point $q\in V(p_i)$, we have that $MinDAG_k(q,p_i)\leq d(q,p)$ for any $p\in AG_{k+1}(p_i)$ where $MinDAG_k(q, p_i)$ denotes the minimum distance between q and the points in $AG_k(p_i)$ and k is an integer.

**Proof:** Assume that point $p_j\in AG_{k+1}(p_i)$ makes that $d(q,p) = MinDAG_{k+1}(q, p_i)$. We can conclude that there exists a point $p_k\in AG_k(p_i)$ such that $d(q,p_k)\leq MinDAG_{k+1}(q, p_i)$ from Theorem 2. We know the fact that $MinDAG_k(q,p_i) = d(q,p_k)$ and $MinDAG_{k+1}(q,p_i)\leq d(q,p)$ for any $p\in AG_{k+1}(p_i)$. We can ultimately conclude that $MinDAG_k(q, p_i)\leq d(q,p)$ for any $p\in AG_{k+1}(p_i)$ and thus the theorem holds.

**Theorem 4:** In a Voronoi diagram generated by a set of generators $P = \{p_1,\ldots,p_n\}\subset R^2$, where, $2<n<\infty$ and $p_i\neq p_j$ for $i\neq j$, i, $j\neq I_n=\{1,\ldots,n\}$, for any point $q\in V(p_i)$, the (k+1)-th nearest neighbors of q is included in $AG_1(p_i)\cup,\ldots,\cup AG_k(p_i)$ where k is an integer and $k\geq 1$.

**Proof:** We can conclude that $MinDAG_k(q, p_i)\leq MinDAG_{k+1}(q, p_i)$ from Theorem 3 where, $MinDAG_k(q, p_i)$ denotes the same as it does in Theorem 3, namely that $MinDAG_j(q, p_i)$ increase with j monotonically where, $j\in I_n = \{1,\ldots, n\}$. Therefore, there exists points $p_i$, $p_{AG1}\in AG_1(p_i)$, $p_{AG2}\in AG_2(p_i),\ldots$, $p_{AGk}\in AG_k(p_i)$, such that $d(q, p_i)\leq d(q, p_{AG1})\leq d(q, p_{AG2})\leq,\ldots,\leq d(q, p_{AGk})$ where $d(q, p_{AG1}) = MinDAG_1(q, p_i),\ldots$ and $d(q, p_{AGk}) = MinDAG_k(q, p_i)$. We also can conclude that $d(q, p)>MinDAG_k(q, p_i)$, for any $p\in(P-AG_1(p_i)\cup,\ldots,\cup AG_k(p_i)-\{p_i\})$. Therefore, we have that the (k+1)-th nearest neighbor of q must be included in $AG_1(p_i)\cup,\ldots,\cup AG_k(p_i)$ and thus the theorem holds.

According to Theorem 4, if one wants to know the (i+1)-th nearest neighbor to a given query point q, there only needs to retrieve in $AG_1(p)\cup,\ldots,\cup AG_i(p)$ where, p is the nearest neighbor of q. Therefore the procedure for kNN queries can be described as following.

**Step 1:** Invoke Algorithm VNN for the query point q to compute the nearest neighbor of q and add it to the result set K
**Step 2:** Search the (i+1)- th nearest neighbor of q in $AG_1(p)\cup,\ldots,\cup AG_i(p)$ where p is the nearest neighbor of q
**Step 3:** Perform Step2, repetitively until the k- th nearest neighbor of q is found

After this brief discussion, a Voronoi-based algorithm, using VR-tree as the underlying data structure to kNN queries, is shown as follows.

```
FINDNN(P, q)
Input: point set P, a query point q
Output: the nearest neighbor of q
begin
  dist←∞/*initialization */
  for each p∈P do
     if (d(p, q)≤dist) then
       {dist←d(p, q);
        MIN←p;}
 return MIN;
end
```

```
Algorithm: kVNN(n, q, k )
Input: a query point q, a VR-tree node n, an integer k
Output: the k nearest neighbors of q
begin
  K←∅; S←∅; /*initialization */
   NNq←VNN(n, q) ; /*Invoke Algorithm VNN to search the first nearest
neighbor of q */
  K←{NNq };
  S←AG1(NNq);
  for each i←1 to k-1 do {
     K←K∪FINDNN(S-K,q); /*search the (i+1)-th nearest neighbor of q for
each i */
    S←S∪AGi(NNq); /*update S*/ }
 return K; /* return the query results*/
end
```

**Theorem 5:** Algorithm kVNN( ) is correct, namely, it returns the k nearest neighbors of the query point q. The running time of this algorithm is $O(\log n+(6^{k+1}-36)/5)$ where n is the total number of objects in the search space

**Proof:** This algorithm invokes VNN( ) for the query point q to search the nearest neighbor $NN_q$ of q and records it into the current results set K, after which it executes a for loop. This algorithm invokes FINDNN( ) to find a point $p\in P-K$ such that $d(q, p)\leq d(q, r)$ for any $r\in P-K$ where, $K = AG_1(NN_q)\cup,\ldots,\cup AG_i(NN_q)$ and records p into K, in the i-th iteration of the for loop. We know that p is the (i+1)-th nearest neighbor of q, according to Theorem 4. Therefore, every iteration a new nearest neighbor of q is found and thus K ultimately records the results of the kNN query after the execution of the for loop. Therefore this algorithm is correct.

We know that the time cost of performing VNN( ) for the query point is O(log n) from Theorem 1. We also know that each generator has 6 adjacent generators at most in the Voronoi diagrams from Property 3. We can conclude that the for loop in FINDNN( ) runs at most $6^i$ times when FINDNN( ) is invoked at the i-th time. Summarizing, the for loop in this algorithm can be executed in $O((6^{k+1}-36)/5)$ time. Therefore, the total time complexity of this algorithm is $O(\log n+(6^{k+1}-36)/5)$ and thus the theorem holds.

During the execution of the algorithm, it executes VNN( ) to search the first nearest neighbor $p_1$ of the query point q. Then the second nearest neighbor $p_2$ of the query point q can be found in $AG_1(p_1)$ which can be obtained directly from the hash table embodied in the VR-tree. This

algorithm searches the i-th nearest neighbor of q in $AG_1(p) \cup, \ldots, \cup AG_i(p)$ which is a subset of P where p is the nearest neighbor of q, rather than the whole data space. Moreover, the search space $AG_1(p) \cup, \ldots, \cup AG_i(p)$ can be swiftly computed by visiting the hash table embodied in the VR-tree directly. Therefore the algorithm can efficiently solve the problem of kNN queries. We know that the time complexity of this algorithm is $O(\log n + (6^{k+1} - 36)/5)$. Then we can conclude that the time complexity converges to O(log n) which means that the time costs mainly depend on the execution of VNN( ) when k is far smaller than n.

## CONTINUOUS NEAREST NEIGHBORS QUERIES BASED ON VORONOI DIAGRAMS

Given a point set P and a query trajectory [s, e] which is a line segment or arbitrary curve with the start point s and the end point e, the CNN query retrieves the nearest neighbor of every point on [s, e], within dataset P. The result contains a set of <R,T> tuples, where R is a point of P and T is an interval which is a segment of a query trajectory [s, e] such that R is the NN of all points in T. Supposing that the query trajectory [s, e] intersects with the boundary of some Voronoi polygon V(p) at two points a and b and for any point $q \in [a, b]$, $q \in V(p)$ holds in the Voronoi diagram associated with point set P, we can conclude that the tuple $<p, [a,b]>$ is an element of the results of the CNN query with dataset P and query point q. From the discussion above, it is clear that the CNN query can be achieved by searching the Voronoi polygons which intersect with the query trajectory and computing the intersection segments between these Voronoi polygons and the query trajectory in the Voronoi diagram associated with data space. Based on the above idea, an algorithm based on Voronoi diagrams for CNN query, using VR-tree as the underlying data structure, is presented as follows.

```
Algorithm:VCNN (n, s, e)
Input: VR-tree node n, start point s of the query trajectory, end point e of the
query trajectory
Output: the results of CNN query with the given query trajectory
begin
T←NN_SEC(n ,s); /*search the nearest neighbor of s*/
S←s;
while (e∉V(T)) do
  {for each edge∈V(T) do
    {if (edge∩[S,e]≠∅) then
        {SL←SL∪<T,[S, edge∩[S,e]]>; /*a new tuple is recorded into results
set SL */
        S←edge∩[S,e];}
      for each p∈AG1(T) do /* update T */
        if (V(T)∩V(p)= edge) then
           T←p;}}
   SL←SL∪<T, [S, e]>;
   return SL;
end
```

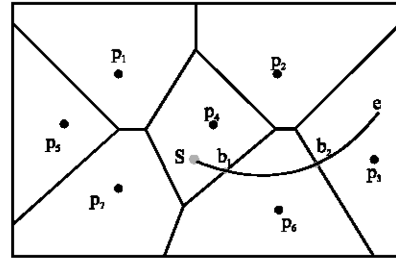


Fig. 5: An example of continuous nearest neighbor query based on Voronoi diagrams

As an example consider Fig. 5 where the dataset $P = \{p_1, p_2, p_3, p_4, p_5, p_6\}$ and the query trajectory q = [s, e]. Let us apply Algorithm VCNN to CNN query for the given dataset P and query trajectory q. The output of the query is $\{<p_4,[s,b_1]>, <p_6,[b_1,b_2]>, <p_3,[b_2,e]>\}$, meaning that point $p_4$ is the NN for interval $[s,b_1]$ and point $p_6$ is the NN for interval $[b_1,b_2]$ and point $p_3$ is the NN for interval $[b_2,e]$ which is also obvious in Fig. 5.

**Theorem 6:** Algorithm VCNN is correct, i.e., it returns the nearest neighbor of every point on the query trajectory [s,e], given the data space P. The running time of this algorithm is O(logn+36i) where n is the total number of points in P and i is the number of the Voronoi polygons which insect with [s,e] in the Voronoi diagram associated with P.

**Proof:** Initially, this algorithm invokes VNN ( ) to find the nearest neighbor T of the start point s of the given query trajectory [s, e]. Then, it executes the while loop starting with recording the tuple <T, [s, p]> where p is the first intersection point between [s, e] and the boundary of V(T). We can conclude that tuple <T, [s, p] > is an element of the results set according to Property1 of Voronoi diagram. Every iteration a new tuple $<p_i, [a, b]>$ is recorded into SL where $p_i$ is the generator of the new Voronoi polygon encountered along the route from point s to point e and a, b are the intersection points between the boundary of $V(p_i)$ and [s, e]. It is clear that this tuple is an element of the results. Therefore SL ultimately records the results of the CNN query after the execution of the while loop and thus this algorithm is correct.

We know that the running time of VNN is O(log n) from Theorem 1. The innermost for loop and the outer for loop all runs at most 6 times due to the fact that each generator has at most 6 adjacent generators as stated in Property 3 of Voronoi diagram. It is clear that the outermost while loop run i times. Therefore the total running time of this algorithm is O(logn+36i) and thus the theorem holds.

## EXPERIMENTAL RESULTS

Here, we experimentally evaluate the performance of the proposed algorithms. All algorithms in our experiments were implemented in Visual C + +. net 2003. All experiments were run on Windows XP with a Pentium IV 3.0GHz CPU, 1GB of memory and Oracle 9.2 as the database server. For our experiments, all tests are done on synthetic datasets produced by Spatial Data Generator. The datasets used consist of at most 1000,000 objects. In VR-tree, the intermediate node and the leaf node sizes were all set to 4K and the maximum capability of a VR-tree node equals 100 entries. Memory buffers based on LRU algorithm were used for caching disk pages. Memory buffers only store the page associated with the root node of VR-tree, initially and can load at most 64 pages.

**Experiment 1:** We compare Algorithm VNN for 1NN query with BF method proposed by Hjaltason and Samet (1999) and the most excellent one of the improved methods for NN queries on R tree we call it RSP proposed by Adler and Heeringa (2008), in query processing time. In this experiment, the measured time cost is the average cost of 100 queries with the same dataset but with different query points randomly generated. Our results are summarized in Table 1.

Table 1 shows that the query processing time of Algorithm VNN is less than that of the other two approaches. It is easy to see that the query processing time of Algorithm VNN grows very slowly, particularly when n<400000 which does not like that of the other two approaches increases sharply as n, the size of dataset, increases from Table 1. The experimental results indicate that Algorithm VNN for 1NN query outperforms the previous methods in running time. This superiority is prominent, particularly to large dataset. This superiority derives from that the excellent data structure VR-tree is used which makes that the role of Voronoi diagram is brought into full play in 1NN query.

**Experiment 2:** For each value of k, we performed 100 kNN queries with the different query point but on the same dataset randomly selected which consists of 100000 objects, using kVNN, BF and RSP, respectively where k varies from 2 to 200. We present the average execution time of the 100 runs of the kNN queries for each value of k and for each algorithm, as shown in Table 2.

From Table 2, it is easy to see that the running time of RSP and BF is about between 1.1 and 2.2 times (respectively, 2.2 and 6.2) as much as that of Algorithm kVNN. We can see that the execution time of Algorithm kVNN grows extremely slowly as k increases when k<20.

Table 1: Execution time of VNN versus BF and RSP for different data size

| Time (sec) | 100000 | 200000 | 400000 | 600000 | 800000 | 100000 |
|---|---|---|---|---|---|---|
| BF | 0.0623 | 0.1229 | 0.2605 | 0.3216 | 0.4117 | 0.5231 |
| RSP | 0.0221 | 0.0439 | 0.0921 | 0.1332 | 0.2039 | 0.3510 |
| VNN | 0.0115 | 0.0115 | 0.0156 | 0.0203 | 0.0275 | 0.0312 |

Table 2: Execution time of kVNN versus BF and RSP for different k

| Time (sec) | 2 | 10 | 20 | 50 | 100 | 200 |
|---|---|---|---|---|---|---|
| BF | 0.0638 | 0.0698 | 0.0752 | 0.0806 | 0.0893 | 0.1096 |
| RSP | 0.0227 | 0.0250 | 0.0296 | 0.0391 | 0.0495 | 0.0538 |
| kVNN | 0.0115 | 0.0115 | 0.0121 | 0.0275 | 0.0396 | 0.0498 |

Table 3: Execution time of kVNN versus BF and RSP for different data size

| Time (sec) | 100000 | 200000 | 400000 | 600000 | 800000 | 1000000 |
|---|---|---|---|---|---|---|
| BF | 0.0752 | 0.1498 | 0.3122 | 0.5721 | 0.7681 | 0.9953 |
| RSP | 0.0296 | 0.0522 | 0.0986 | 0.1475 | 0.2268 | 0.3650 |
| kVNN | 0.0121 | 0.0133 | 0.0201 | 0.0408 | 0.0592 | 0.0755 |

We can also see that the performance of Algorithm kVNN will be more and more modest as k increases when k>50 but it still runs much faster than the other two methods. This phenomenon appears due to the fact that the size of candidate set $AG_1(p) \cup, \ldots, \cup AG_i(p)$ used to search the (i+1)-th nearest neighbor grows rapidly for larger values of i, where p is the nearest neighbor of the query point. The experimental results indicate that Algorithm kVNN outperforms the previous methods in running time, particularly to smaller values of k with respect to the size of the dataset.

Furthermore, we fix k to 20 and perform 100 queries on each dataset where the size of the datasets varies from 100000 to1000000, using kVNN, BF and RSP respectively. We present the average execution time of the 100 runs of the queries for each dataset and for each algorithm, as shown in Table 3.

From Table 3, it is evident that running time of Algorithm kVNN is always less than that of the other two approaches. We can also see that the gap between the execution time of kVNN and that of BF and RSP becomes more and more wide as the size of dataset increases. The experimental results indicate that Algorithm kVNN is superior to the previous methods in execution time. This advantage will be more conspicuous for large data space.

In conclusion, Algorithm kVNN runs faster than the previous methods. It has a better performance for large data space. It is worth to mention that its performance is excellent for small values of k with respect to the size of the data space.

**Experiment 3:** We compare the performance of VCNN with its only competitor, the method we call it TCNN proposed in Tao *et al*. (2002), in execution time. We know that all existed methods for CNN query can only be applied to the case that the query trajectory is line segment. Therefore the query trajectory considered is restricted to line segment in our experiment. In this

Table 4: Execution time of VCNN versus TCNN for different query length

| Time (sec) | 1 | 10l | 30l | 60l | 80l | 100l |
|---|---|---|---|---|---|---|
| TCNN | 0.0697 | 0.0765 | 0.0922 | 0.1339 | 0.2031 | 0.3182 |
| VCNN | 0.0117 | 0.0119 | 0.0131 | 0.0153 | 0.0195 | 0.0219 |

Table 5: Execution time of VCNN versus TCNN for different data size

| Time (sec) | 100000 | 200000 | 400000 | 600000 | 800000 | 100000 |
|---|---|---|---|---|---|---|
| TCNN | 0.0832 | 0.1394 | 0.3522 | 0.4358 | 0.5875 | 0.7231 |
| VCNN | 0.0126 | 0.0128 | 0.0178 | 0.0235 | 0.0336 | 0.0413 |

experiment, for each vale of L ,the length of the query line segment, we performed 100 CNN queries with the different query line segment on the same dataset randomly selected which consists of 100000 objects, using VCNN and TCNN, respectively where, L varies from l to 100l (l is a unit length). We present the average execution time of the 100 runs of the CNN queries for each value of L and for each method, as shown in Table 4.

From Table 4, it is obvious that the execution time of VCNN is far less than that of TCNN. From Table 4 and Table 1, we can see that the execution time of VCNN when L<30l is close to that of a NN query under the same case using VNN. We can also see that the execution time of TCNN grows faster and faster but that of VCNN grows slowly even when L>60l as L increases. The experimental results show that VCNN runs faster then TCNN, the execution time of VCNN much depends on searching the nearest neighbor of the start point of the query segment when the query segment is shorter and VCNN is much less sensitive than TCNN to the length of the query line segment.

Next we fix the query length to 20l and compare the performance of both methods by varying the size of the dataset from 1000000 to 1000000, in execution time. For each dataset, we also perform 100 queries with 100 CNN queries with the different query line segment whose length is 20l and we average the results. Our results are summarized in Table 5.

From Table 5, we have that Algorithm VCNN makes that the query processing time decreases at least 84.9 percent compared with TCNN for any data size. We can also see that the execution time difference between TCNN and VCNN increases continuously from 0.0706 seconds to 0.6818 seconds as the size of dataset increases from 10000 to 1000000. The experimental results show that VCNN outperforms TCNN significantly in all cases and also show that the performance difference increases with the size of the dataset which is explained as follows. A new tuple of the query results is obtained by finding the Voronoi polygon which intersects with the query segment in no more then 6 Voronoi polygons which share the same edges with the last Voronoi polygon encountered along the query segment for any dataset, during the execution of VCNN. However, more and more qualifying points are involved during the execution of TCNN with size of the data space grows which incurs much more time cost.

Here, we experimentally compared our algorithms with the previous ones. The experimental results are in reasonably close agreement with the aforementioned theoretical analysis. Summarizing, our algorithms significantly outperform the previous ones in execution time.

## CONCLUSIONS

In this study, we extensively researched NN and CNN queries by means of Voronoi diagrams. We developed a new data structure VR-tree which fits to the nearest neighbors queries using Voronoi diagrams. Subsequently, we devised an algorithm using VR-tree to 1NN query. Furthermore, an algorithm for kNN query based on Voronoi diagram was proposed. Finally, we proposed an algorithm for CNN query with arbitrary query trajectory employing Voronoi diagrams. Our experimental results indicated that the proposed algorithms significantly outperformed the previous ones in execution time.

The technical contributions of this study can be summarized as follows:

- We originally developed a data structure VR-tree for nearest neighbors queries and efficiently applied Voronoi diagrams to nearest neighbors query
- We applied Voronoi diagrams to CNN queries and originally achieved the CNN queries with arbitrary query trajectory

In the future, we intend to extend our algorithms to NN and CNN queries in mobile environment. There will be a lot of work to do where the most challenging work is how to maintain the Voronoi Diagrams of the moving points.

## ACKNOWLEDGMENT

The authors would like to thank National Natural Science Foundation of China and National Science Foundation of Heilongjiang Province of China, under Grant No.60673136 and No.F200601, for funding this research.

## REFERENCES

Adler, M. and B. Heeringa, 2008. Search space reductions for nearest-neighbor queries. Proceedings of the 5th Annual Conference on Theory and Applications of Models of Computation, April 25-29, Xi'an, China, pp: 554-567.

Beckmann, N., H.P. Kriegel, R. Schneider and B. Seeger, 1990. The R*-tree: An efficient androbust access method for points and rectangles. Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, May 23-25, Atlantic City, New Jersey, pp: 322-331.

Bespamyatnikh, S. and J. Snoeyink, 1999. Queries with segments in Voronoi diagrams. Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, Jan. 17-19, Baltimore, Maryland, USA., pp: 122-129.

Chazelle, B. and H. Edelsbrunner, 1987. An improved algorithm for constructing kth-order voronoi diagrams. IEEE Trans. Comput., 36: 1349-1354.

Cheung, K.L. and A.W. Fu, 1998. Enhanced nearest neighbour search on the R-tree. SIGMOD Record, 27: 16-21.

Guttman, A., 1984. R-Trees: A dynamic index for geometric data. Proceedings of the ACM SIGMOD International Conference on Management of Data, June 18-21, USA., pp: 47-57.

Hjaltason, G.R. and H. Samet, 1999. Distance browsing in spatial databases. ACM Trans. Database Syst., 24: 265-318.

Lee, D.T., 1982. On k-nearest neighbor Voronoi diagrams in the plane. IEEE Trans. Comput., 31: 478-787.

Meyerhenke, H., 2005. Constructing higher order voronoi diagrams in parallel. Proceedings of the 21st European Workshop on Computational Geometry, March 9-11, Eindhoven, Netherlands, pp: 123-126.

Roussopoulos, N., S. Kelley and F. Vincent, 1995. Nearest neighbor queries. Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, May 22-25, San Jose, California, USA., pp: 71-79.

Sack, J.R and J. Urrutia, 2000. Voronoi Diagrams. In: Handbook on Computational Geometry, Sack, J.R. and J. Urrutia (Eds.). Elsevier Science Publishers, Ottawa, pp: 201-290.

Seidl, T. and H.P. Kriegel, 1998. Optimal multi-step k-Nearest neighbor search. Proceedings of the ACM SIGMOD International Conference on Management of Data, June 1-4, Seattle, Washington, USA., pp: 154-165.

Song, M.B., K.J. Park, K.S. Kong and S.K. Lee, 2007. Bottom-up nearest neighbor search for R-trees. Inform. Proc. Lett., 101: 78-85.

Song, Z. and N. Roussopoulos, 2001. K-NN search for moving query point. Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases, July 12-15, Redondo Beach, California, USA., pp: 79-96.

Tao, Y. and D. Papadias, 2002. Time parameterized queries in spatiotemporal databases. Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, June 3-6, Madison, Wisconsin, USA., pp: 334-345.

Tao, Y., D. Papadias and Q. Shen, 2002. Continuous nearest neighbor search. Proceedings of 28th International Conference on Very Large Data Bases(VLDB), Aug. 20-23, Hong Kong, China, pp: 287-298.