Reverse *k* Nearest Neighbor and Reverse Farthest Neighbor Search on Spatial Networks

Quoc Thai Tran[1], David Taniar[1], and Maytham Safar[2]

[1] Clayton School of Information Technology, Monash University, Australia
qttra2@student.infotech.monash.edu.au,
David.Taniar@infotech.monash.edu.au
[2] Computer Engineering Department, Kuwait University, Kuwait
maytham@me.com

Abstract. One of the problems that arise in geographical information systems is finding objects that are influenced by other objects. While most research focuses on *k*NN (*k* Nearest Neighbor) and RNN (Reverse Nearest Neighbor) queries, an important type of proximity queries called *Reverse Farthest Neighbor* (RFN) has not received much attention. Since our previous work shows that *k*NN and RNN queries in spatial network databases can be efficiently solved using Network Voronoi Diagram (NVD), in this paper, we aim to introduce a new approach to process reverse proximity queries including RFN and R*k*NN/R*k*FN queries. Our approach is based on NVD and pre-computation of network distances, and is applicable for spatial road network maps. Being the most fundamental Voronoi-based approach for RFN and R*k*NN/R*k*FN queries, our solutions show that they can be efficiently used for networks that have a low and medium level of density.

1 Introduction

Due to the rapid growth of information technologies in the twenty-first century, the original use of geographic maps has evolved. Although printed maps are still useful, many people today use electronic maps. People use maps not only for directions from one place to another, but also for decision making. For example, users may ask questions like "*What is the shortest path to go from A to B?*" or "*What is the nearest train station to a shopping centre?*". Taking this into account, numerous algorithms for *k* Nearest Neighbor (*k*NN) queries have been studied in literature (Roussopoulos, 1995; Kolahdouzan and Shahabi, 2005; Safar, 2005). As the result, many navigation systems are now enhanced to support *k*NN searches.

In addition to *k*NN queries, there are other types of proximity queries called *Reverse Nearest Neighbor* (RNN) and *Reverse Farthest Neighbor* (RFN) queries (Korn and Muthukrishnan, 2002; Kumar et al, 2008). An RNN query is, for example, to find residential that consider a restaurant as the nearest restaurant. Therefore, RNN search is used to find other places which are most affected by a given location. An RFN search is, in contrast, is to find places that are least affected by a given location. For example, a real estate company may want to know which properties is least affected by a road construction.

Basic RNN query can be written as R1NN that retrieves interest objects which consider the query points as *the only* nearest neighbor. The generalization of RNN search is termed R*k*NN where $k > 1$. In R*k*NN, the query point is not the only nearest neighbor, but instead, it is one of the nearest neighbors of interest objects retrieved. Likewise, the basic RFN can be generalized to R*k*FN where $k > 1$ and it is used to find objects that consider the query point as one of the farthest neighbors. Therefore, in R*k*NN and R*k*FN, the distance from an object to the query point determines the degree of influence by the query point to that object.

Although these queries are commonly required, finding an efficient way to process these queries has been a problem in geographical information systems and spatial databases. Many researches focus on *k*NN and its variants while the reverse proximity queries are often neglected. It is common that these approaches assume the freedom of movement of objects. Note that when there is an underlying road network, the movement of objects is restricted to pre-defined roads by the underlying network (Papadias, 2003; Jensen et al, 2003). The spatial network distance from *A* to *B* is possibly greater than the Euclidean distance from *A* to *B* (Samet et al, 2008). Therefore, methods that are developed for geometry studies can be significantly wrong on spatial road networks where the network distances between objects must be used instead of their Euclidean distances. Hence, in this paper, we use spatial network, and we aim to introduce an efficient approach to process RNN and RFN queries in spatial network databases.

Figure 1 shows the context of this paper. R1NN has been addressed in our previous work (Safer et al, 2009), in which we have successfully used Network Voronoi Diagram (Okabe et al, 2000), PINE expansion algorithm (Safar, 2005), and the pre-computed network distances between objects (Koulahdouzan and Shahabi, 2004). This paper extends our previous work by focusing on R*k*NN, as well as R1FN and R*k*FN.

	$k = 1$	**$k > 1$**
RNN	Our previous work	Section 4.1
RFN	Section 4.2	Section 4.3

Fig. 1. Reverse Nearest Neighbor (RNN) and Reverse Farthest Neighbor (RFN)

The remainder of this paper is organized as follows: Section 2 summarizes important existing work on *k*NN and RNN queries. Section 3 describes some preliminaries including our previous work on RNN. Section 4 presents our new approaches and algorithms for R*k*NN, RFN and R*k*FN. Section 5 shows and discusses the experimental results of the proposed algorithms. Finally, section 6 concludes the paper and explains possible future work.

2 Related Work

Existing work on proximity query processing can be categorized into two main groups. The first group focuses on *k* Nearest Neighbor (*k*NN) and its variants, and second group concentrates on Reverse Nearest Neighbor (RNN) queries.

A *k*NN query is used to find the nearest neighbors to a given object. Since it is useful in many applications, many approaches have been introduced to support *k*NN. One of the well-known approaches for *k*NN is to use ‘branch-and-bound R-tree traversal’ algorithm developed by Roussopoulos et al (1995). This algorithm was soon followed by another algorithm developed by (Korn et al, 1996) to accelerate the performance of *k*NN search. To achieve even faster performance, Sield and Krigel (1998) produce an algorithm called the ‘multi-step’ algorithm to reduce the number of candidates. While most early algorithms focuses *k*NN in Euclidean spaces, Papadias et al (2003) introduces new methods to process various types of spatial queries including *k*NN, range search, closest pairs and e-distance joins in spatial network databases using both network and Euclidean information of objects. Since Voronoi diagram has been successfully used to solve problems in many applications, it is used as an alternative approach to solve problems in spatial analysis. Using Voronoi diagram and pre-calculated network distances between objects, Kolahdouzan and Shahabi (2004) proposes an approach which was termed ‘Voronoi Network-Based Nearest Neighbor’ (VN^3) to find exact *k*NN of the query object. In this approach, the pre-calculated distances are both network distances within and across the Voronoi polygons. As the pre-computation of network distances can be expensive when the density is high, Safar (2005) proposes a novel approach called ‘Progressive Incremental Network Expansion’ (PINE) algorithm using network Voronoi diagram and Dijkstra’s algorithm (Dijkstra, 1959). Unlike VN^3, PINE stores only network distances between border points of the Voronoi polygons. Since Network Voronoi Diagram has been successfully used to solve problems of *k*NN, we continue using it for RNN/RFN. Note that in the above approaches, the query point is only a single point and thus, the query is sometimes referred as a ‘single-source skyline query’. On the other hand, a ‘multi-source skyline query’ is where the answer is found in response to multiple query points at the same time. An example of this query is to find news agencies that are closest to the train station, the bus stop and the car park. While ‘multi-source skyline queries’ are necessary in many applications, the first approach for processing these queries using network distances is found in (Deng et al, 2007).

In addition to *k*NN, there is a growing number of works on RNN. It is common that these approaches produce approximate results for RNN and they are not applicable to spatial network databases. An early discussion about RNN queries and its variants is found in Korn and Muthukrishnan (2000). This paper proposes a general approach for RNN queries and a method for large data sets using R-tree. Yang et al (2001) introduces a single index structure called ‘Rdnn-tree’ to replace the multiple indexes used in the previous work. This index structure can be used for both RNN and NN queries. A new concept, called ‘Reverse Skyline Queries’, is introduced by Dellis and Seeger (2007) to find RNN of any given query point. It uses the ‘Branch and Bound’ algorithm to retrieve a candidate point set and refines this set to find the exact answers. Kumar et al (2008) introduces another method to process reverse proximity queries in two and three dimensions using a lifting transformation technique. However, the results are only approximate. Although these methods are helpful in geographical and spatial studies, they cannot be used for spatial network problems. Our previous work (Safar et al, 2009) discusses various types of RNN queries and proposes several algorithms to process these queries in spatial network databases. These algorithms are based on Network Voronoi Diagram, PINE algorithm and the pre-computation of network distances.

Similar to nearest neighbor, the basic RNN query can be generalized to find objects that consider the query point as one of the *k* nearest neighbors. We call this an R*k*NN query where *k* can be any number given at the query time. Though there are few methods to find R*k*NN in literature (Achtert et al, 2006; Xia et al, 2005; Tao et al, 2004), methods that use Network Voronoi Diagram to find exact results have not been studied. Thus, in this paper, we focus on using Network Voronoi Diagram for processing R*k*NN queries. In addition, we also introduce other types of reverse proximity queries, termed RFN (Reverse Farthest Neighbor) and R*k*FN.

3 Preliminaries

This section aims to provide some preliminary remarks on various types of queries including Reverse Nearest/Farthest Neighbor and Reverse *k* Nearest Neighbor queries. Since our approaches for these query types are based on Network Voronoi Diagram, the principles of Voronoi diagram are also reviewed in this section. The discussion of Voronoi diagram starts with Voronoi diagram for two dimensional Euclidean spaces. Then, we focus our discussion on Network Voronoi Diagram where real network distances between objects are used in place of Euclidean distances. Next, we highlight some properties of Voronoi diagram to use for our approaches. A brief description of our previous work on RNN queries is also provided at the end of this section.

3.1 Reverse Nearest/Farthest Neighbor Queries

We start this section with the explanation of some terminologies used in this paper. These terminologies include interest point (or object) and query point (or object).

Definition 1. An *interest object* is any object on a network and it is of interest to users. An interest point is where the interest object is located. We use the terms interest objects and interest points interchangeably.

Definition 2. A *query object* is an object on the network and its influence on interest objects is determined as the query is called. A query point is where the query object is located. Therefore, we use the terms query point and query object interchangeably in this paper.

Next, we define the basic Reverse Nearest/Farthest Neighbor and Reverse *k* Nearest/Farthest Neighbor queries using the above terminologies.

- **Type 1.** ***Reverse Nearest Neighbor*** (RNN) query retrieves interest points (or objects) which consider the query point (or object) as their *nearest* neighbor. This query type is used to find places where the query point has *most* impact.

  Example: Consider a restaurant as the query point, an RNN query can be used to retrieve other restaurants which assign the query restaurant as the nearest neighbor.

- **Type 2.** ***Reverse Farthest Neighbor*** (RFN) query retrieves interest points (or objects) which consider the query point (or object) as their *farthest* neighbor. This query type is used to find places where the query point has *least* impact.

  Example: Given a train station *A*, an RFN query can help the train manager to decide which other train stations consider *A* as the farthest train station.

- **Type 3.** ***Reverse k Nearest Neighbor*** (R*k*NN) query is a generalization of basic RNN where the interest points (or objects) retrieved consider the query point as *one of their nearest neighbors* ($k > 1$) rather the only nearest neighbor.

  Example: Given $k = 2$, the R*k*NN query retrieves all restaurants which assign the query restaurant as one of the two nearest neighbors.

- **Type 4.** ***Reverse k Farthest Neighbor*** (R*k*FN) query is a generalization of basic RFN where the query point (or object) is considered as *one of the farthest neighbors* ($k > 1$) by the interest points (or objects) retrieved.

  Example: Given $k = 2$ and a train station *A* as the query point, R*k*FN query tells the train manager which other train stations consider *A* as one of the two farthest train stations.

3.2 Voronoi Diagram and Network Voronoi Diagram

A Voronoi diagram is an exhaustive collection of exclusive Voronoi polygons (Okabe et al, 2000). Each Voronoi polygon (VP) is associated with a generator point (GP). All locations in the Voronoi polygon are closer to the generator point of that polygon than any other generator point in the Voronoi diagram in Euclidean plane. The boundaries of the polygons are called Voronoi edges. Any location on the Voronoi edges can be assigned to more than one generator. Adjacent polygons are the Voronoi polygons that share the same edges and their generators are called adjacent generators. An example of Voronoi diagram is shown in Figure 2.

Voronoi diagram has been used to solve spatial analysis problems (Aggarwal et al, 1990; Patroumpas et al, 2007; Dickerson and Goodrich, 2008). Our proposed algorithms are based on the following property of Voronoi diagram: "The nearest generator point of p_i (e.g. p_j) is among the generator points whose Voronoi polygons share similar Voronoi edges with VP(p_i)." (Safar et al, 2009).

Since movements of objects are based on the spatial road network, the real distance between two objects is not the Euclidean distance but the actual network distance. In order to find the exact answers for those problems, Network Voronoi Diagrams have been used (Okabe et al, 2000). Generally, Network Voronoi Diagram is generated using the actual network distances, not the Euclidean distance between objects. For this reason, the network rather than the space is divided into Voronoi polygons. All nodes and edges in a Voronoi polygon are closer to its generator point than to any other generator point (Safar et al, 2009). A formal definition of Network Voronoi Diagram is found in Papadias et al (2003) and Roussopoulos et al (1995): "A Network Voronoi Diagram, termed NVD, is defined as graphs and is a specialization of Voronoi diagrams, where the location of objects is restricted to the links that connect the nodes of the graph and the distance between objects is defined as their shortest path in the network rather than their Euclidean distance".

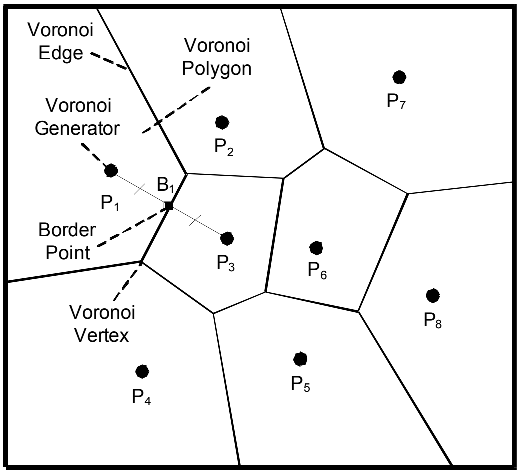


Fig. 2. Voronoi Diagram

An example of Network Voronoi Diagram for three polygons is shown in Figure 3. In this example, '*X*' are the generator points and small squares represent the road network. We use different colored lines (i.e. gray or black) to show different Voronoi polygons. Thus, all nodes and edges in the 'black' network Voronoi polygon are closer to its generator point than to other generator point in the 'gray' network Voronoi polygons. Note that while the Voronoi polygons in the Voronoi diagram have a convex polygon shape, the Voronoi polygons in the Network Voronoi Diagram might have irregular shape because it uses network distance rather than Euclidean distance.

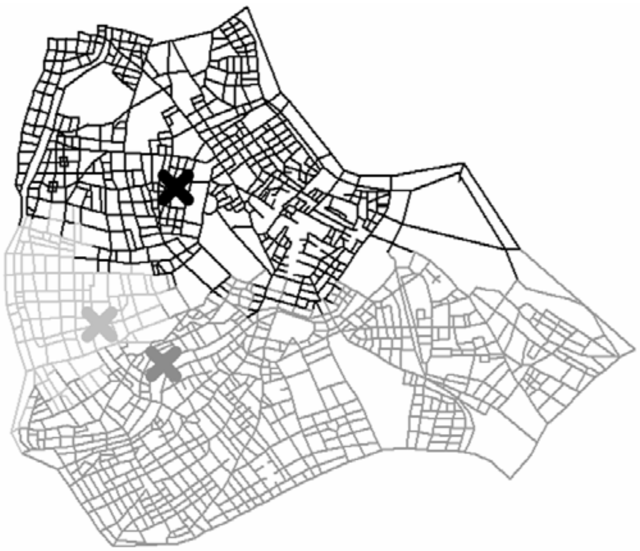


Fig. 3. Network Voronoi Diagram (Graf and Winter, 2003)

3.3 Voronoi-Based Reverse Nearest Neighbor (RNN): Our Previous Work

Our previous work (Safar et al, 2009) has identified four different types of RNN queries that incorporate interest objects, query point, and other static (or quasi-static) non-query objects. In RNN queries, the query point may be a generator point (*GP*) or a non-generator point (~*GP*). Likewise, the interest objects may also be the generator points or non-generator points. Based on this, four types of RNN queries have been classified: $\mathrm{RNN}_{GP}(GP)$, $\mathrm{RNN}_{GP}(\sim GP)$, $\mathrm{RNN}_{\sim GP}(\sim GP)$, and $\mathrm{RNN}_{\sim GP}(GP)$. However, for simplicity, for this work we will only discuss $\mathrm{RNN}_{GP}(GP)$, where both query point and interest objects are generator points. For example, given an NVD whereby the generator points are restaurants, $\mathrm{RNN}_{Restaurant}$ (*Restaurant*) is to find other restaurants that consider the query restaurant as their nearest neighbor (e.g. their nearest restaurant). Our developed algorithms to answer RNN queries depend on the existence of a Network Voronoi Diagram (NVD) and a set of pre-computed data (such as border-to-border, and border-to-generator distances). The system described in our previous work (Safar et al, 2009) creates a set of NVDs, one for each different interest point (e.g., NVD for restaurants, schools, etc), and developed new algorithms to answer RNN queries that utilize the previously created NVDs, pre-computed distances, and the PINE algorithm.

$\mathrm{RNN}_{GP}(GP)$ query type does not need any inner network distance calculations, since we already have pre-computed the NVD for the generator points. All the required information was computed and stored while generating the Network Voronoi Diagram. Both query objects and interest objects are the generator points of the Voronoi diagram, and thus all distances from the generator points to borders are known. The candidate interest objects for RNN belong to the set of the query adjacent polygons RNN ∈ {*QueryAdjacentPolygons*} (see Kolahdouzan and Shahabi (2004); Safar (2005) for details).

The query first starts by using NVD to find the distances from the generator of the polygon and then the distances from those border points to adjacent generators. For example in Figure 2, if the query point "*q*" is the generator point P_3, we need to find $distance(q, B_1) + distance(B_1, P_1)$. Once the neighboring generator points are reached, the algorithm starts a heap list with the *distance*(*q*, *AdjacentGeneratorPoints*) as the initial distance. The distances between all candidates are first measured (i.e. $distance(P_1,P_2)$, $distance(P_1,P_4)$) using NVD generator-to-border and vice versa, thus eliminating the repetition of the calculation among them. To cut down the calculations even further, all distances between the polygons are compared to the shortest distance between the query *q* and its adjacent 1NN. If a path is found that is shorter than the *q*-to-1NN then both interest objects are canceled because they might be considered as the nearest neighbors to each other, or in other words, they are closer to each other than to the query object.

The new candidate interest objects are then set as query points and they start searching for their 1NN. Every newly found distance from the intersection point, border points and generator points are tested and compared to the first entry in the heap. If the distance is larger, then it is heaped as the second entry. However, if the distance is shorter, then the search stops in that direction and we set the polygon as 'not' the RNN to the query point.

While RNN is a specialization of R*k*NN where $k = 1$, in this paper, we propose an R*k*NN algorithm which is based on RNN algorithms. For simplicity, we assume that both query point and interest objects are generator points.

4 Proposed Algorithms

4.1 Reverse *k* Nearest Neighbor (R*k*NN) Search

Consider a Network Voronoi Diagram where both query point q and interest points $\{P_1, P_2, \ldots, P_n\}$ are generator points. A set of interest points which assign q as one of the k nearest neighbors is denoted as RkNN(q), $k > 1$. Here, we highlight some key properties that are used in our approach.

Property 1. If P_x belongs to RkNN(q), then the number of points that are closer to P_x than q is less than or equal to k-1 (Kumar et al, 2008).

Property 2. In Kolahdouzan and Shahabi (2004) and Safar (2005), it has been shown that when $k = 1$, the candidate nearest neighbors of q, Cand_RNN(q), are among its adjacent polygons.

Example. In Figure 4, when $k = 1$, if q is a query point located at P_1, then Cand_RNN(q)=$\{P_2, P_3, \ldots, P_7\}$. Based on this property, our observation shows that when $k = 2$, the candidate reverse nearest neighbors of q is the combination of Cand_RNN(q) and the adjacent polygons of each polygon in Cand_RNN(q). Thus, in Figure 4, we have Cand_2RNN(q)=$\{P_1, P_2, \ldots, P_{19}\}$.

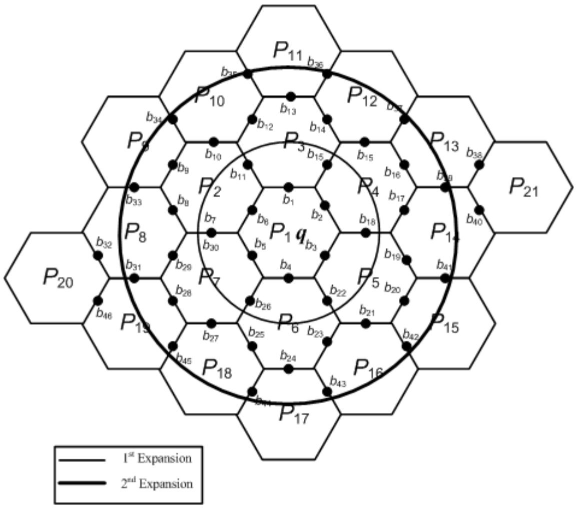


Fig. 4. Example of an R*k*NN query

Property 3. For any number $k > 1$ and a query point q, the candidate set of Reverse k Nearest neighbors of q, Cand_RkNN(q), includes all adjacent polygons resulted from the k expansions from q. Hence, this property is the generalization of property 2.

Proof. This property can be proven by using contradiction. Consider Figure 4 as an example, when $k = 2$, if $P_{20} \in$ Cand_2RNN(q) then q is considered as one of the two nearest neighbors of P_{20}, $q \in$ 2NN(P_{20}) and the maximum number of polygons that are closer to P_{20} than q is one. However, the shortest path from q to P_{20} must pass P_9, P_2, P_7, P_{18}, P_8, P_{19}. This means there are at least two polygons in $\{P_9, P_2, P_7, P_{18}, P_8, P_{19}\}$ which are closer to P_{20} than q and therefore, $q \notin 2NN(P_{20})$ contradicts our initial assumption. As the result, in Figure 4, the candidate R2NNs of q include all adjacent polygons that are found from the two expansions from q.

Taking into consideration the above properties, we develop an efficient approach to process R*k*NN queries in spatial network databases. Our approach includes three main steps which are described as follows:

Step 1: Find polygon P_x that contains the query point q.

Step 2: Assign adjacent polygons of P_x to be the first candidate of R*k*NN(q).

Step 3: For each candidate polygon P_y, find the k nearest neighbors of P_y, *k*NN(P_y), using PINE algorithm. The calculation of network distances between generators is based on: (*i*) the pre-computation of inner network distances and (*ii*) the border-to-border network distance calculation using Dijkstra's algorithm. If *k*NN(P_y) contains P_x, then add P_y to the result. Add adjacent polygons of P_y to the candidate set. Repeat step 3 until k expansions have been made and return the result.

Figure 5 shows the complete algorithm for R*k*NN queries.

```
Algorithm RkNN(k, q)
1.  Voronoi Polygon $P_x$ = contains($q$)
2.  Initialize an empty set to contain result polygons = ∅
3.  Initialize an empty candidate polygon set Cand_RkNN = ∅
4.  Create a new candidate polygon set New_Cand to contain all adjacent polygons
    AP($P_x$)
5.  Initialize a do while loop condition $i$ to zero
6.  do
7.      //Update candidate polygon set in which existing candidates are replaced
8.      //by new polygons in the temporary candidate set.
9.      Cand_RkNN = New_Cand \ (Cand_RkNN ∪ $P_x$)
10.     New_Cand = ∅
11.     for each $P_y$ in Cand_RkNN
12.         //Call PINE algorithm to find the $k$ nearest neighbors of $P_y$
13.         kNN($P_y$) = PINE($k$, $P_y$)
14.         if $P_x \in$ kNN($P_y$) then    //If $P_y$ considers $P_x$ as one of its kNNs
15.             result = result ∪ $P_y$ //Update the result with $P_y$
16.         end if
17.         //Expand the boundary to find new candidate polygons
18.         New_Cand = New_Cand ∪ AP($P_y$)
19.     end for
20.     increase $i$ by 1
21. while $i < k$    //Restrict the expansion
22. return result
```

Fig. 5. R*k*NN Algorithm

To demonstrate our algorithm, we take Figure 4 as an example. Suppose that we have an NVD where the generator points $\{P_1, P_2, \ldots, P_{21}\}$ are hospitals and the query point q located at P_1. An RkNN query is used to find other hospitals that consider q as one of their k nearest neighbors.

First, calling the function `contains`(q) returns P_1 as the Voronoi polygon that contains q. Second, we create three empty sets for result polygons, candidate polygons and newly found polygons, respectively. We denote these sets by *result*, *cand_RkNN*, and *new_cand.*

$result=\varnothing$
$cand\_RkNN=\varnothing$
$new\_cand=\varnothing$

Next, we find adjacent polygons of P_1 and add them to *new_cand*:

$new\_cand=\{P_2, P_3, \ldots, P_7\}$

Create a variable called i and set $i=0$.

$cand\_RkNN= new\_cand \setminus (cand\_RkNN \cup P_1)=\{P_2, P_3, \ldots, P_7\}$
$new\_cand=\varnothing$

For each polygon P_y in *cand_RkNN*, we find the kNN of P_y using PINE algorithm. For example, let us assume:

$kNN(P_2) = PINE(2,P_2) = \{P_3, P_9\}$
…
$kNN(P_7) = PINE(2,P_7) = \{P_1, P_6\}$

Since $P_1 \in k\mathrm{NN}(P_7)$, P_7 is added to the result: $result=\{P_7, \ldots, P_n\}$.

Next, we find adjacent polygons of P_7 and add those polygons to *new_cand* before we go for the next candidate polygon in the *Cand_RkNN.*

$new\_cand= new\_cand \cup AP(P_7) = \{P_2, P_1, P_{19}, P_{18}, P_6, P_8, \ldots, P_n\}$

When every polygon in *cand_RkNN* is explored, we have:

$new\_cand= new\_cand \cup AP(P_7) = \{P_1, P_2, \ldots, P_{19}\}$

Here, we increase i by 1 and thus, i is now equal to 1. Since $i<k$, we repeat step 3 and this time, cand_RkNN=$\{P_8, P_9, \ldots, P_{19}\}$ since we exclude the old candidate polygons from the next refinement step. When $i=k$, the expansion is stopped and the result is returned.

4.2 Reverse Farthest Neighbor (RFN) Search

Given a network Voronoi diagram which contains a query point q and a number of interest points $\{P_1, P_2, \ldots, P_n\}$, an RFN($q$) is a set of interest points which consider q as the farthest neighbor.

According to Korn and Muthukrishnan (2000), the number of RNN(q) is at most six in a two dimensional NVD. The maximum number of RFN(q), however, cannot be pre-determined (Kumar, Janardan and Gupta, 2008). Moreover, in the previous section, we show that the search space in RNN queries can be restricted to adjacent polygons of the query point. However, since there is no such restriction in RFN search, every interest object on the NVD is a candidate RFN(*q*) and therefore, must be examined. However, an important property of RFN queries highlighted in Kumar et al (2008) is given in the following.

Algorithm RFN(*q*)

```
1.  Voronoi Polygon $P_x$ = contains($q$)
2.  result = $\{P_i, \ldots, P_j\} \setminus P_x$ where $P_i, \ldots, P_j \in$ NVD
3.  Old_Cand($P_x$) = $P_x$
4.  New_Cand($P_x$) = AP($P_x$)        //Adjacent polygons of $P_x$
5.  Cand($P_x$) = New_Cand($P_x$) \ Old_Cand($P_x$)
6.  Do                                  //Start the expansion from $P_x$
7.      for each $P_y$ in Cand($P_x$)
8.          Compute d($P_y$, $P_x$)
9.          $d_{max} = 0$
10.         Old_Cand($P_y$)={$P_x$, $P_y$}
11.         New_Cand($P_y$)= AP($P_y$)   //Adjacent polygons of $P_y$
12.         Cand($P_y$)= New_Cand($P_y$) \ Old_Cand($P_y$)
13.         do                          //Start the expansion from $P_y$
14.             for each $P_z$ in Cand($P_y$)
15.                 if d($P_z$, $P_y$)>$d_{max}$ then
16.                     $d_{max}$= d($P_z$, $P_y$)
17.                 end if
18.                 //Add $P_z$ to the old candidate polygons of $P_y$
19.                 Old_Cand($P_y$) = Old_Cand($P_y$) $\cup$ $P_z$
20.                 //Find new candidate polygons $P_y$ in adjacent polygons of $P_z$
21.                 New_Cand($P_y$)=AP($P_z$) $\cup$ New_Cand($P_y$)
22.             end for
23.             //Excl candidate found in new candidate polygons in expansion $P_y$
24.             Cand($P_y$) = New_Cand($P_y$) \ Old_Cand($P_y$)
25.         while $d_{max}$ = d($P_y$, $P_x$) and Cand($P_y$) $\neq \varnothing$
26.         //End the expansion from $P_y$
27.         if $d_{max}$ > d($P_y$, $P_x$) then
28.             //When there exists a point $P_z$ that is farther from $P_y$ than $P_x$
29.             Remove $P_y$ from the result
30.         end if
31.         //Add $P_y$ to the old candidate polygons of $P_x$
32.         Old_Cand($P_x$) = Old_Cand($P_x$) $\cup$ $P_y$
33.         //Find new candidate polygons of $P_x$ in the adjacent polygons of $P_y$
34.         New_Cand($P_x$) = AP($P_y$) $\cup$ New_Cand($P_x$)
35.     end for
36.     //Excl old candidate polygons from new candidate polygons in expansion $P_x$
37.     Cand($P_x$) = New_Cand($P_x$) \ Old_Cand($P_x$)
38.     while Cand($P_x$) $\neq \varnothing$       //End the expansion from $P_x$
39. return result
```

Fig. 6. RFN Algorithm

Property 4. An interest object P_i is called a Reverse Farthest Neighbor of q if the distance from P_i to q is always greater or equal to P_i to any other object $P_j \neq P_i$.

Based on this property, our approach for RFN queries works as follows:

Step 1: Find the polygon P_x that contains the query point q.

Step 2: Assign adjacent polygons of P_x to be first candidate RFN(q).

Step 3: For each candidate polygon P_y, compute the distance from P_y to P_x and P_y's adjacent polygons using Dijkstra's algorithm. Property 4 suggests that if P_y is a RFN(q), then *distance*(P_y, P_x) $\geq$ *distance*(P_y, P_z), for any $P_z \neq P_y$. Therefore, when we find any adjacent polygon of P_y that is farther to P_y than q, *distance*(P_y, P_z) > *distance*(P_y, q), we remove P_y from the result and move to the next candidate polygon. Otherwise, we continue the expansion from P_y until we find any polygon farthest than q or every candidate for the farthest neighbor of P_y is reached. Then, add adjacent polygons of P_y to the candidate RFN(q). Repeat step 3 until every polygon on the NVD is explored. The complete algorithm for RFN is shown in Figure 6.

Here, we take an example of RFN query as shown in Figure 7. In this example, the NVD contains a set of interest points {P_2, P_3, …, P_{13}} and a query point q located at P_1. Note that both interest points and query points are generator points. The query RFN(q) is to find other interest points that consider q as the farthest neighbor.

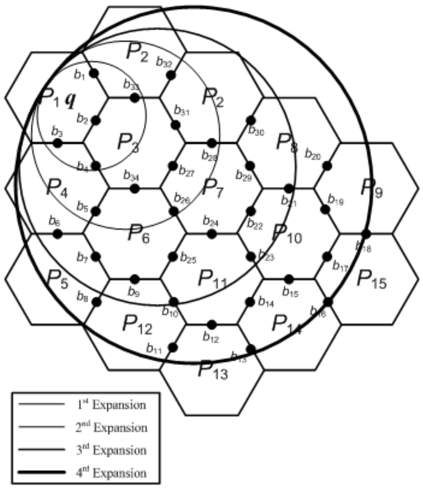


Fig. 7. An example of RFN query

First of all, we retrieve the Voronoi polygon where q is located. In this case, it returns P_1.

Contains(q) = P_1

Secondly, several empty sets are created and we name these sets as *old_cand(P_1), cand(P_1) and new_cand(P_1).*

$old\_cand(P_1)=\varnothing$
$cand(P_1)=\varnothing$
$new\_cand(P_1)=\varnothing$

Next, we add P_1 to $old\_cand(P_1)$ and add adjacent polygons of P_1 to $new\_cand(P_1)$.

$old\_cand(P_1)=\{P_1\}$
$new\_cand(P_1)=\{P_2, P_3, P_4\}$

Now, we have $cand(P_1)= new\_cand(P_1) \setminus old\_cand(P_1) = \{P_2, P_3, P_4\}$

For each polygon P_y in $cand(P_1)$, the distance from P_y to P_1, $d(P_y, P_1)$, is calculated using pre-computed network distances and Dijkstra's algorithm. Also, d_{max} is set to 0.

In the next step, we create several empty sets, namely $old\_cand(P_y)$, $cand(P_y)$ and $new\_cand(P_y)$. We assign P_x and P_y to the $old\_cand(P_y)$ set and assign adjacent polygons of P_y to the $new\_cand(P_y)$ set. For example, when $P_y=P_2$:

$old\_cand(P_2)=\{P_1, P_2\}$
$new\_cand(P_2)=\{P_1, P_3, P_8\}$
$cand(P_2) = new\_cand(P_2) \setminus old\_cand(P_2) = \{P_3, P_8\}$

Then, for each polygon P_z in cand(P_2), the distance from P_z to P_2, d(P_z, P_2), is calculated and compared with the maximum possible network distance d_{max}. If $d(P_z, P_2)>d_{max}$, then $d_{max}= d(Pz, P2)$. For instance, when $P_z=P_3$:

$old\_cand(P_2)=\{P_1, P_2, P_3\}$
$new\_cand(P_2)=\{P_1, P_2, P_8, P_7, P_6, P_4\}$
$cand(P_2) = new\_cand(P_2) \setminus old\_cand(P_2) = \{ P_8, P_7, P_6, P_4 \}$

Suppose $d_{max}=d(P_3, P_2) > distance(P_1, P_2)$ which means P_3 is farther from P_2 than P_1 and thus, P_2 is not regarded as an RFN(q). Next, we add P_3 to $old\_cand(P_2)$ and add adjacent polygons of P_3 to $new\_cand(P2)$ and go for the next polygon in $cand(P_2)$.

When every polygon in $cand(P_2)$ is explored, we add P_2 to $old\_cand(P_1)$ and add adjacent polygons to $new\_cand(P_1)$. We call it a filter/refinement process. This process is done repeatedly until there is no more polygon found in $cand(P_1)$.

4.3 Reverse *k* Farthest Neighbor (R*k*FN) Search

Let $\{P_1, P_2, \ldots, P_n\}$ be interest points and q be a query point on a Network Voronoi Diagram, an R*k*FN(q) is a set of interest points which assign q as one of the k farthest neighbors, $k > 1$. Since our developed approach for R*k*FN is based on R*k*NN and RFN approaches, most of the properties used for R*k*NN and RFN are also applicable to R*k*FN. However, we introduce the following property which is specific to R*k*FN.

Property 5: If P_x belongs to R*k*FN(q), then there are at most k-1 points that are farther from P_x than q (Kumar et al, 2008).

The summary of our approach for R*k*FN is as follows:

Step 1: Find the polygon P_x that contains the query point q.
Step 2: Assign adjacent polygons of P_x to be first candidate RFN(q).

```
Algorithm RkFN(k, q)
1.  Voronoi Polygon $P_x$ = contains($q$)
2.  result = $\{P_i, \ldots, P_j\} \setminus P_x$ where $P_i, \ldots, P_j \in$ NVD
3.  Old_Cand($P_x$) = $P_x$
4.  New_Cand($P_x$) = AP($P_x$) //Adjacent polygons of $P_x$
5.  Cand($P_x$) = New_Cand($P_x$) \ Old_Cand($P_x$)
6.  do //Start the expansion from $P_x$
7.      for each $P_y$ in Cand($P_x$)
8.          //Compute the distance from $P_y$ to $P_x$
9.          Compute d($P_y$, $P_x$)
10.         $d_{max} = 0$
11.         Old_Cand($P_y$)={$P_x$, $P_y$}
12.         New_Cand($P_y$)= AP($P_y$) //Adjacent polygons of $P_y$
13.         Cand($P_y$)= New_Cand($P_y$) \ Old_Cand($P_y$)
14.         do //Start the expansion from $P_y$
15.             $k$FN($P_y$) = $k$ farthest polygons in Cand($P_y$) $\cup P_x$
16.             for each $P_z$ in Cand($P_y$)
17.                 //Add $P_z$ to the old candidate polygons of $P_y$
18.                 Old_Cand($P_y$) = Old_Cand($P_y$) $\cup P_z$
19.                 //Find new candidate polygons of $P_y$ in the
20.                 //adjacent polygons of $P_z$
21.                 New_Cand($P_y$)=AP($P_z$) $\cup$ New_Cand($P_y$)
22.             end for
23.             //Exclude the old candidate found in the new
24.             //candidate polygons found in the expansion of $P_y$
25.             Cand($P_y$) = New_Cand($P_y$) \ Old_Cand($P_y$)
26.         while $Px \in k$FN($P_y$) and Cand($P_y$) $\neq \varnothing$
27.         //End the expansion from $P_y$
28.         if $P_x \notin k$FN($P_y$) then
29.             //When there exist more than $k$-1 points that are farther
30.             // from $P_y$ than $P_x$
31.             Remove $P_y$ from the result
32.         end if
33.         //Add $P_y$ to the old candidate polygons of $P_x$
34.         Old_Cand($P_x$) = Old_Cand($P_x$) $\cup P_y$
35.         //Find  new candidate polygons of $P_x$ in the adjacent polygons of $P_y$
36.         New_Cand($P_x$) = AP($P_y$) $\cup$ New_Cand($P_x$)
37.     end for
38.     //Exclude the old candidate polygons from the new candidate polygons
39.     //found in the expansion of $P_x$
40.     Cand($P_x$) = New_Cand($P_x$) \ Old_Cand($P_x$)
41. while Cand($P_x$) $\neq \varnothing$ //End the expansion from $P_x$
42. return result
```

Fig. 8. R*k*FN Algorithm

Step 3: For each candidate polygon P_y, find the *k* farthest neighbors of P_y, *k*FN(P_y), starting from its adjacent polygons. A *k*FN query is to find a set of *k* interest objects that are considered as the farthest neighbors of the query object. If *k*FN(P_y) does not contain *q*, then P_y is not the R*k*FN(*q*) and jump to the next candidate polygon. Otherwise, continue the search for *k*FN(P_y) until no more polygon is found. Update the candidate RFN(*q*) with adjacent polygons of P_y. Repeat step 3 until every polygon on the NVD is explored. Figure 8 shows the complete algorithm for R*k*FN queries.

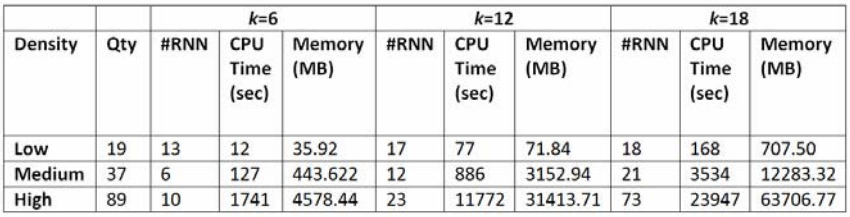
5 Performance Evaluation

To evaluate the performance of our proposed algorithms, we carried out several experiments on a Pentium IV, 2.33GHz processor, 2GB RAM, running Windows XP. For our testing, we use real world data sets for navigation systems and GPS enabled devices from NavTech Inc. These consist of 110,000 links and 79,800 nodes to represent a real road network in downtown Los Angeles. We test our algorithms using different sets of interest points including hospitals, shopping centres, and parks. The experiment for each algorithm is performed as follow. First, we run each proposed query type R*k*NN, RFN and R*k*FN 20 times and for each time, we chose a random query point. Then, we calculated the average numbers of RNN/RFN results, their CPU time and memory accesses and use graphs to show our experimental results. The purpose of our experiments is to show how different factors such as the number of *k* and object density could affect the performance of our algorithms in terms of execution time and disk accesses.

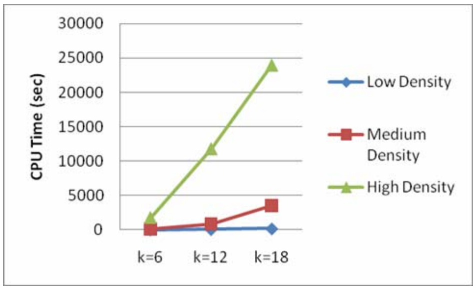
5.1 Performance of Reverse *k* Nearest Neighbor Queries

The performance for R*k*NN query algorithm is tested on different aspects including object density (e.g. from low to high densities) and the value of *k*. In our experiments, we use the term 'Low Density' if the quantity of objects is less than 20 (e.g. Hospitals) while we use 'High Density' if the quantity of objects is greater than 40 (e.g. Parks). In addition, when the term 'Medium Density' is used, we refer to objects' quantity between 20 and 40 (e.g. Shopping Centers). Since we are interested in knowing the average numbers of RNN results, their execution time and memory accesses, we present them in a table as shown in Table 1. Also, Figure 9 is used to depict our experimental results for R*k*NN algorithm.

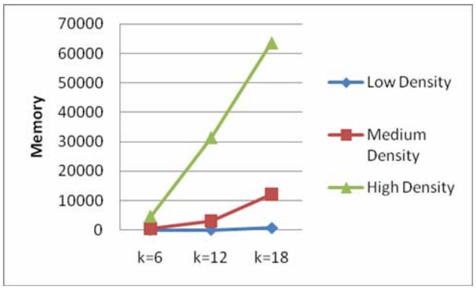
Table 1. Performance of R*k*NN queries



		k=6			*k*=12			*k*=18		
Density	Qty	#RNN	CPU Time (sec)	Memory (MB)	#RNN	CPU Time (sec)	Memory (MB)	#RNN	CPU Time (sec)	Memory (MB)
Low	19	13	12	35.92	17	77	71.84	18	168	707.50
Medium	37	6	127	443.622	12	886	3152.94	21	3534	12283.32
High	89	10	1741	4578.44	23	11772	31413.71	73	23947	63706.77



(a) CPU time (sec) vs. *k*



(b) Memory vs. *k*

Fig. 9. Comparison of *k* and execution time and memory use for R*k*NN queries

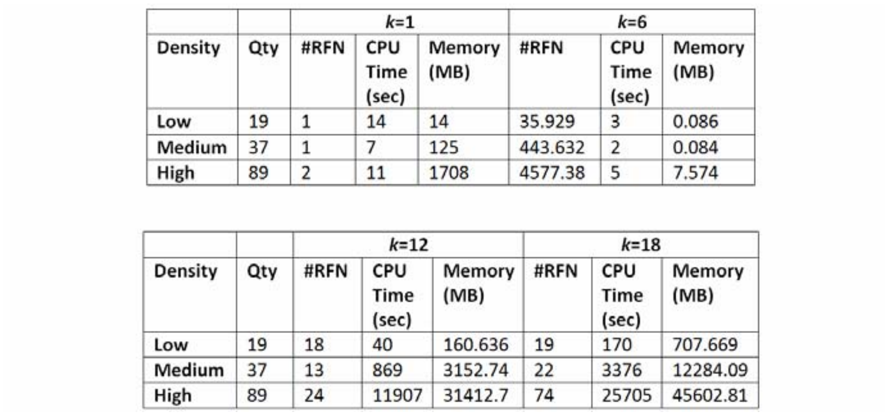
As shown in Figure 9, as the value of *k* increases, the numbers of CPU time and memory accesses also increase. This can be explained as the value of *k* increases, it increases the number of candidate objects for reverse nearest neighbor and the amount of time and memory to process RNN query. However, these changes vary for different object densities. For objects that have low or medium density, there is a slightly increment in the number of execution time and memory whereas for high density objects, this increment is more considerable. Therefore, the higher the object density is, the more CPU time and memory are needed for our algorithm. In summary, although it shows a significant degrade in the performance as the density increases, our algorithm can still have good response time and reasonable use of memory in low and medium object densities.

5.2 Performance of Reverse *k* Farthest Neighbor Queries

Similar to R*k*NN, the performance of RFN and R*k*FN queries is evaluated in terms of number of results, execution time and memory use. Also, our testing is based on using different values of *k* and densities of objects. We run the query 20 times and assign a random query point for each query. Our experimental results are calculated on average and are summarized in Table 2. We also use Figure 10 to depict our results.

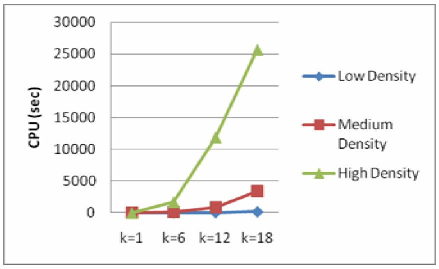
In Figure 10, the amount of execution time and memory is shown on *x* axis while the values of *k* are shown on *y* axis. It shows that on average, the response time and memory used for RFN queries increases as the number of *k* increases. This is because when the value of *k* is small, the number of resources that are needed to run the farthest neighbour query for each candidate is also small. For example, when $k = 1$, the network expansion from each candidate would stop as it could find any object that is farther than the query object. Thus, the execution of RFN query would be performed faster.

Table 2. Performance of RFN and R*k*FN queries

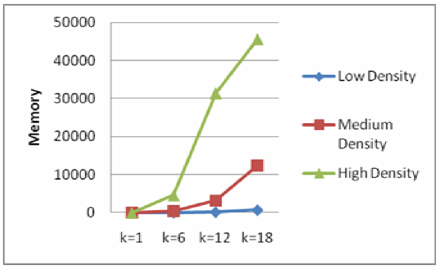


		k=1			*k*=6		
Density	Qty	#RFN	CPU Time (sec)	Memory (MB)	#RFN	CPU Time (sec)	Memory (MB)
Low	19	1	14	14	35.929	3	0.086
Medium	37	1	7	125	443.632	2	0.084
High	89	2	11	1708	4577.38	5	7.574

		k=12			*k*=18		
Density	Qty	#RFN	CPU Time (sec)	Memory (MB)	#RFN	CPU Time (sec)	Memory (MB)
Low	19	18	40	160.636	19	170	707.669
Medium	37	13	869	3152.74	22	3376	12284.09
High	89	24	11907	31412.7	74	25705	45602.81



(a) CPU time (sec) vs. *k*



(a) Memory vs. *k*

Fig. 10. Comparison of *k* and execution time and memory use for R*k*FN queries

Additionally, from our experiments with R*k*FN, we note that the amount of CPU time and memory need rises rapidly as the value of *k* increases when the number of objects is high. As explained in our algorithm, there is no restriction for the search space for R*k*FN as for R*k*NN queries. Every object on the network can be an RFN of the query object and thus, must be explored. This helps to explain why the performance of our algorithm for R*k*FN decreases as the number of objects increases. Nonetheless, our fundamental algorithm for RNN and R*k*FN queries can produce good performance on low and medium density data sets.

6 Conclusion and Future Work

In this paper, we put an emphasis on three types of reverse proximity query, namely (*i*) Reverse *k* Nearest Neighbor, (*ii*) Reverse Farthest Neighbor, and (*iii*) Reverse *k* Farthest Neighbor, and their relations to each other. We also outlined important implication of these query types in geographical planning as opposed to their lack of attention by researchers. Our observation showed that in practice, the possible movement between objects must rely on pre-defined roads by an underlying network. Therefore, existing approaches for reverse proximity queries using Euclidean distances gives only estimated results. Taking this into account, we developed new approaches for R*k*NN, RFN and R*k*FN using Network Voronoi Diagram, PINE network expansion algorithm and pre-computed network distances, so that they can be applied to spatial road networks. Also, we extended the properties of network Voronoi polygons to find new constraints for the network expansion in R*k*NN/R*k*FN searches.

Since location-aware systems are predicted to be widely used in the future, understanding how different spatial analysis problems can be solved using NVD would be an advance. The outcome of this paper would lead up to new interesting field of research in spatial network query processing and new applications to support RNN/RFN queries in the future. While objects in the basic RNN/RFN searches discussed in this paper are of same type, we plan to extend these queries to a biochromatic version, termed biochromatic RNN/RFN in our future paper.

Acknowledgments

This research has been partially funded by the Australian Research Council (ARC) Discovery Project (Project No: DP0987687).

References

Achtert, E., Bohm, C., Kroger, P., Kunath, P., Pryakhin, A., Renz, M.: Approximate reverse k-nearest neighbor queries in general metric spaces. In: Proceedings of the 15th ACM International Conference on Information and Knowledge Management, pp. 788–789 (2006)

Achtert, E., Bohm, C., Kroger, P., Kunath, P., Pryakhin, A., Renz, M.: Efficient reverse k-nearest neighbour search in arbitrary metric spaces. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 515–526 (2006)

Aggarwal, A., Hansen, M., Leighton, T.: Solving query-retrieval problems by compacting Voronoi diagrams. In: Proceedings of the 22nd annual ACM Symposium on Theory of Computing, pp. 331–340 (1990)

Dellis, E., Seeger, B.: Efficient computation of reverse skyline queries. In: Proceedings of the 33rd International Conference on Very Large Data Bases, pp. 291–302 (2007)

Deng, K., Zhou, X., Shen, H.: Multi-source Skyline Query Processing in Road Networks. In: Proceedings of the 2007 IEEE 23rd International Conference on Data Engineering, pp. 796–805 (2007)

Dickerson, M., Goodrich, M.: Two-site Voronoi diagrams in geographic networks. In: Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, vol. 59 (2008)

Dijkstra, E.W.: A Note on Two Problems in Connexion with Graphs. Numeriche Mathematik 1(1), 269–271 (1959)

Goh, J.Y., Taniar, D.: Mobile Data mining by Location Dependencies. In: Yang, Z.R., Yin, H., Everson, R.M. (eds.) IDEAL 2004. LNCS, vol. 3177, pp. 225–231. Springer, Heidelberg (2004)

Goh, J., Taniar, D.: Mining Frequency Pattern from Mobile Users. In: Negoita, M.G., Howlett, R.J., Jain, L.C. (eds.) KES 2004, Part III. LNCS, vol. 3215, pp. 795–801. Springer, Heidelberg (2004)

Graf, M., Winter, S.: Network Voronoi Diagram. Österreichische Zeitschrift für Vermessung und Geoinformation 91(3), 166–174 (2003)

Jensen, C., Kolarvr, J., Pedersen, T., Timko, I.: Nearest neighbor queries in road networks. In: Proceedings of the 11th ACM International Symposium on Advances in Geographic Information Systems, pp. 1–3 (2003)

Korn, F., Muthukrishnan, S.: Influence sets based on reverse nearest neighbor queries. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, vol. 29(2), pp. 201–212 (2000)

Korn, F., Muthukrishnan, S., Srivastava, D.: Reverse nearest neighbor aggregates over data streams. In: Proceedings of the International Conference on Very Large Data Bases, pp. 814–825 (2002)

Koulahdouzan, M., Shahabi, C.: Voronoi-Based K Nearest Neighbor Search for Spatial Network Databases. In: Proceedings of the 30th International Conference on Very Large Data Bases, vol. 30, pp. 840–851 (2004)

Kumar, Y., Janardan, R., Gupta, P.: Efficient algorithms for reverse proximity query problems. In: Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, vol. 39 (2008)

Okabe, A., Boots, B., Sugihara, K., Chiu, N.: Spatial Tessellations, Concepts and Applications of Voronoi Diagrams, 2nd edn. John Wiley and Sons Ltd., Chichester (2000)

Papadias, D., Mamoulis, N., Zhang, J., Tao, Y.: Query Processing in Spatial Network Databases. In: Proceedings of the 29th International Conference on Very Large Data Bases, vol. 29, pp. 802–813 (2003)

Patroumpas, K., Minogiannis, T., Sellis, T.: Approximate order-k Voronoi cells over positional streams. In: Proceedings of the 15th Annual ACM International Symposium on Advances in Geographic Information Systems, vol. 36 (2007)

Roussopoulos, N., Kelly, S., Vincent, F.: Nearest Neighbor Queries. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 71–79 (1995)

Safar, M.: K Nearest Neighbor Search in Navigation Systems. Mobile Information Systems 1(3), 207–224 (2005)

Safar, M., Ebrahmi, D.: eDar Algorithm for Continous KNN queries based on PINE. International Journal of Information Technology and Web Engineering 1(4), 1–21 (2006)

Safar, M., Ibrahimi, D., Taniar, D., Gavrilova, M.: Voronoi-based Reverse Nearest Neighbour Query Processing on Spatial Networks. Multimedia Systems Journal (in press, 2009)

Samet, H., Sankaranarayanan, J., Alborzi, H.: Scalable network distance browsing in spatial databases. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 43–54 (2008)

Sield, T., Kriegel, H.: Optimal Multi-Step k-Nearest Neighbor Search. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 154–165 (1998)

Stephen, R., Gihnea, G., Patel, M., Serif, T.: A context-aware Tour Guide: User implications. Mobile Information Systems 3(2), 71–88 (2007)

Tao, Y., Papadias, D., Lian, X.: Reverse kNN search in arbitrary dimensionality. In: Proceedings of the 30th International Conference on Very Large Data Bases, pp. 744–755 (2004)

Waluyo, A., Srinivasan, B., Taniar, D.: Optimal Broadcast Channel for Data Dissemination in Mobile Database Environment. In: Zhou, X., Xu, M., Jähnichen, S., Cao, J. (eds.) APPT 2003. LNCS, vol. 2834, pp. 655–664. Springer, Heidelberg (2003)

Waluyo, A., Srinivasan, B., Taniar, D.: A Taxanomy of Broadcast Indexing Schemes for Multi Channel Data Dissemination in Mobile Database. In: Proceedings of the 18th International Conference on Advanced Information Networking and Applications (AINA 2004), pp. 213–218. IEEE Computer Society, Los Alamitos (2004)

Waluyo, A., Srinivasan, B., Taniar, D.: Research in mobile database query optimization and processing. Mobile Information Systems 1(4), 225–252 (2005)

Xia, C., Hsu, W., Lee, M.: ERkNN: efficient reverse k-nearest neighbors retrieval with local k-NN distance estimation. In: Proceedings of the 14th ACM International Conference on Information and Knowledge Management, pp. 533–540 (2005)

Xuan, K., Zhao, G., Taniar, D., Srinivasan, B., Safar, M., Gavrilova, M.: Continous Range Search based on Network Voronoi Diagram. International Journal of Grid and Utility Computing (in press, 2009)

Xuan, K., Zhao, G., Taniar, D., Srinivasan, B., Safar, M., Gavrilova, M.: Network Voronoi Diagram based Range Search. In: Proceedings of the 23rd IEEE International Conference on Advanced Information Networking and Applications, AINA (in press, 2009)

Xuan, K., Zhao, G., Taniar, D., Srinivasan, B., Safar, M., Gavrilova, M.: Continous Range Search Query Processing in Mobile Navigation. In: Proceedings of the 14th IEEE International Conference on Parallel and Distributed Systems, ICPADS's 2008, pp. 361–368. IEEE Computer Society, Los Alamitos (2008)

Yang, C., Lin, K.: An Index Structure for Efficient Reverse Nearest Neighbor Queries. In: Proceedings of the 17th International Conference on Data Engineering, pp. 485–492 (2001)

Zhao, G., Xuan, K., Taniar, D., Srinivasan, B.: Incremental K-Nearest Neighbor Search On Road Networks. Journal of Interconnection Networks 9(4), 455–470 (2008)