Multimedia Systems (2009) 15:295–308
DOI 10.1007/s00530-009-0167-z

REGULAR PAPER

Voronoi-based reverse nearest neighbor query processing on spatial networks

Maytham Safar · Dariush Ibrahimi · David Taniar

Published online: 12 September 2009

Abstract The use of Voronoi diagram has traditionally been applied to computational geometry and multimedia problems. In this paper, we will show how Voronoi diagram can be applied to spatial query processing, and in particular to *Reverse Nearest Neighbor* (RNN) queries. Spatial and geographical query processing, in general, and RNN in particular, are becoming more important, as online maps are now widely available. In this paper, using the concept of Voronoi diagram, we classify RNN into four types depending on whether the query point and the interest objects are the generator points of the *Voronoi Polygon* or not. Our approach is based on manipulating *Network Voronoi Diagram* properties and applying a progressive incremental network expansion for finding the polygon inner network distances required to solve RNN queries. Our experimentation results show that our approaches have good response times in answering RNN queries.

Keywords Voronoi diagram · Network Voronoi diagram · Spatial network databases · Nearest neighbor · Reverse nearest neighbor · Query processing

M. Safar (✉) · D. Ibrahimi
Computer Engineering Department,
Kuwait University, Kuwait City, Kuwait
e-mail: maytham.safar@ku.edu.kw; maytham@me.com

D. Ibrahimi
e-mail: darebra@yahoo.com

D. Taniar
Clayton School of Information Technology,
Monash University, Melbourne, Australia
e-mail: David.Taniar@infotech.monash.edu.au

1 Introduction

Voronoi Diagram has been successfully used to solve variety of application problems, including surface reconstruction, optimization, planning, image and signal processing, biometric synthesis, mapping and multimedia problems [10,13,25]. In this paper, we will show how Voronoi Diagram can be used to solve spatial queries, which has a great potential not only in Geographical Information Systems (GIS), as online maps (like Google Maps®) are now widely available to the public, but in several other areas such as multimedia databases indexing.

This paper particularly focuses on one important breed of spatial and geographical information processing, namely *Reverse Nearest Neighbor* (RNN) queries. An RNN query is that given a set of candidate interest points (or objects), and a query point (or object), find the interest points/objects that consider the query object as their nearest neighbors [29]. The efficient implementation of RNN query is of a particular interest in information systems that support user's queries, such as on-line search engines, multimedia search engines, or GIS. For example, a business owner when deciding to open a new grocery store may ask an RNN query similar to: "where is the best location for the grocery store?" This question can be reworded to as "How many buildings consider this possible location as the nearest grocery store?" Each candidate location for the grocery store should start an RNN query and the results are then compared to choose the best location. Hence, RNN plays a major role in not only information dissemination, but also in decision making.

Figure 1 gives an example of an RNN query. Assume the query point is pointed by Q (e.g., proposed student accommodation). There are two existing student accommodations, as pointed by objects O and P. The interest objects in the map are shown by objects A, B, and C (e.g., McDonald).

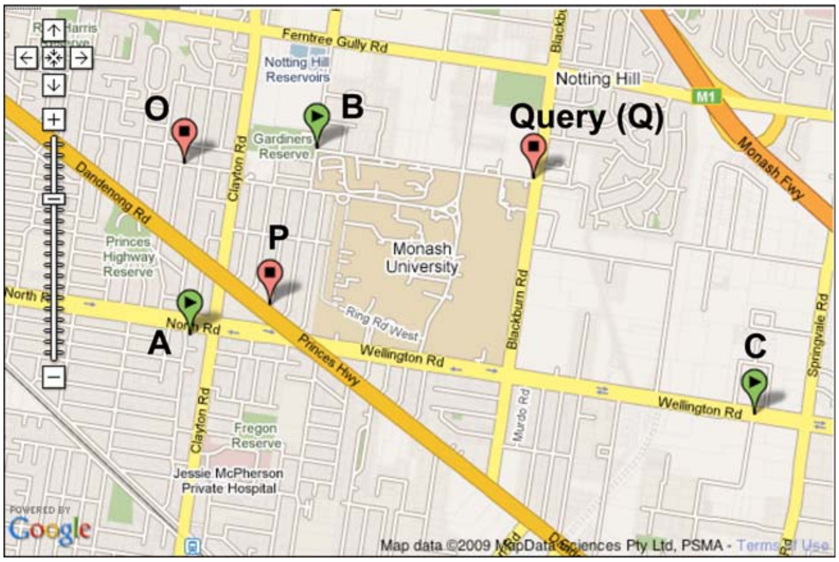


Fig. 1 Example of an RNN query

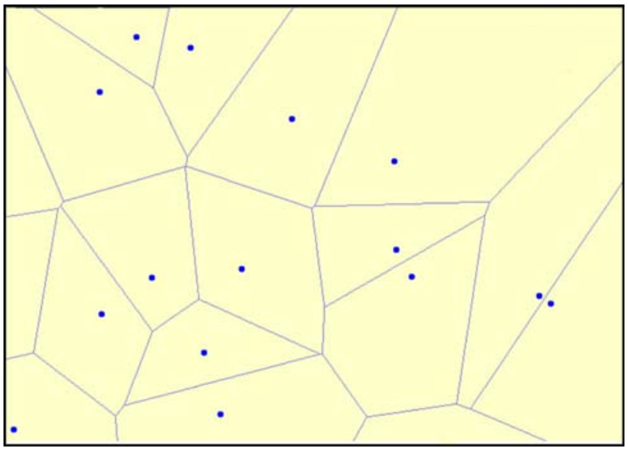


Fig. 2 Voronoi diagram

The RNN query is to find interest objects (either A, B or C) that consider query point Q as its nearest neighbor. In this example, although the distance between interest object B and query point Q is shorter than the distance between C and Q, interest object B does not consider Q as its nearest neighbor, because B is closer to O, instead of to Q. In this case, the answer of the query is interest object C, since C considers Q as its nearest neighbor.

Most existing work focuses on the traditional nearest neighbor (NN) queries [9,12,16,20,21]. These include *Nearest Neighbor* (NN), *k Nearest Neighbor* (kNN), *Continuous-NN* and *kNN* (CNN and CKNN), and RNN. There have been a lot of existing works in NN, kNN, CNN, and CKNN. RNN in the context of spatial network databases, whereby there is an underlying road network, has not been studied in the past. Existing work in RNN assumes a Euclidean plane. In this paper, we focus on RNN on spatial road network, as illustrated in Fig. 2. With *Spatial Road Network Databases* (SNDB), objects are restricted to move on predefined paths (e.g., roads) that are specified by an underlying network.

In this paper, we developed algorithms to answer RNN queries. Our proposed solutions are based on *Network Voronoi Diagram* (NVD) and utilize pre-computed distances such as border-to-border and generator-to-border distances to minimize the network distance computation time [17]. The solution also utilizes the spatial index of NVD to check the polygons for interest data points. Leaving the only unknown network distance from the generator to any interest point inside the polygon. To overcome this, a progressive incremental network expansion, which is based on Dijkstra's algorithm [2], is used for finding the inner network distance in the polygons. The network expansion occurs around the query point and expands from there to find the nearest neighbor.

2 Related work

Spatial queries are also extensively used in mobile multimedia systems and especially in applications related to location-based systems in GIS and spatial databases [11,12,24], and thus were the focus of many researches [4,19]. There are two groups of algorithms proposed to address common spatial queries: one is based on *Euclidean distance*, and the other based on *network distance*.

Existing work on the first group considers *Cartesian* (typically, *Euclidean*) spaces, where the distance between two objects is determined by their relative position in space. The major disadvantage with those approaches, as highlighted in [8,16,20], is that the shortest path calculations are performed based on Euclidean distances. In practice, on the contrary, objects usually move only on predefined roads. Therefore, the distance calculation should depend on the connectivity among these objects, and not on Euclidean distance.

As a result, the first group that considers Euclidean distance is impractical for spatial network databases, because they neglect network/road distance.

The second group of research focuses on solving the queries for spatial network databases, and multimedia databases. In these databases, the objects are indexed as points in nD space and the distance between two objects is the length of the shortest path connecting them. The approaches support the exact spatial queries on spatial network databases and multimedia databases [9,12,15,17]. And therefore, these works, which are based on network/road distance, are more realistic, compared with the works by the first group, which rely solely on Euclidean distance.

Authors in [15] introduces the *Incremental Network Expansion* (INE), which is an architecture that integrates network and Euclidean information. It is based on creating a search region for the query point that expands from the query that is similar to Dijkstra's algorithm [2]. The advantages of this approach are: (i) it offers a method for finding the exact distance in networks, and (b) the architecture can support other spatial queries (e.g., range search and closest pairs). However, this approach suffers from poor performance when the interest objects of the query are not densely distributed in the network, as this will lead to large portions of the database to be retrieved. This problem happens for large values of k.

The *Voronoi-based Network Nearest Neighbor* (VN^3) approach proposed by [9] is based on the properties of the NVD. It uses localized pre-computations of network distances for a very small percentage of neighboring nodes in the network to enhance query response time and reduce disk accesses. In addition, *Network Voronoi Polygons* (NVP) of a NVD can directly be used to find the first nearest neighbor. Subsequently, NVP's adjacency information provides a candidate set for other nearest neighbors. Finally the pre-computed distances are used to refine the set. The filter/refinement process in VN^3 is iterative: at each step, firstly a new set of candidates is generated from the VNPs, and then the pre-computed distances are used to select "only the next" nearest neighbor. The advantages of this approach are: (i) it offers a method that finds the exact distances in networks, (ii) fast query response time, and (iii) progressively returns the k nearest neighbors and their distances from the query point. The main disadvantage of this approach is its need for pre-computing and maintaining two different sets of data: (1) query-to-border computation: computing the network distances from the query point to the border points of its enclosing network Voronoi polygon, and (2) border-to-border computation: computing the network distances from the border points of NVP of the query point to the border points of any of the other NVPs. Furthermore, this approach suffers in performance with lower density data sets.

Author in [17] proposes a novel approach, termed *Progressive Incremental Network Expansion* (PINE) that efficiently addresses spatial queries in SNDB. The main idea behind this approach is to first partition a large network into smaller more manageable regions, then pre-compute distances across the regions. These two steps can be easily and efficiently implemented using a first order Voronoi diagram, then a computation similar to the INE can be used for the computation of intra-distances. The advantages of PINE are that it has less disk access time and less CPU time than VN^3. In addition, PINE's performance is independent of the density and distribution of the points of interest, and the location of the query object. By performing across-the-network computation for only the border points of the neighboring regions, global computations can be avoided.

We need to emphasize that the above-mentioned existing work mainly focuses on kNN (k Nearest Neighbor) queries, not RNN. Despite the importance of RNN in SNDB, the scarce studies in the literature are designed for Euclidean spaces, which are not applicable to SNDB. In this paper, however, we propose solutions for RNN queries based on network distance using SNDB as its underlying network. In particular, we will expand our previous work on PINE [15] to cater for RNN.

3 Background: Voronoi-based kNN query processing

Our proposed algorithms for RNN queries are based on (i) Voronoi diagram, (ii) Dijkstra's algorithms, and (iii) our previous kNN method, called PINE.

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 [14]. *Dijkstra's algorithm* is one of the most efficient algorithms that find shortest paths from the source node to all the other nodes [2]. In this section, we review the principles of Voronoi diagrams. We start with Voronoi diagram for two-dimensional Euclidean space and present only the properties that are used in our approach. We then discuss the network Voronoi diagram where the distance between two objects in space is their shortest path in the network rather than their Euclidean distance and hence can be used for spatial networks. Then, we briefly describe the PINE algorithm. A thorough discussion on Voronoi diagrams is presented in [14,17].

3.1 Voronoi diagram

Voronoi diagram (VD) has been used to solve spatial analysis problems [17]. The Voronoi diagram of a point set P, VD(P), is a unique diagram that consists of a set of collectively exhaustive and mutually exclusive Voronoi polygons (Voronoi cells) VPs. Each Voronoi polygon is associated with a point in P (called *generator point*) and contains all the locations in the Euclidean plane that are closer to the generator point of the Voronoi cell than any other generator point in P (refer to Fig. 2). The boundaries of the polygons, called Voronoi edges, are the set of locations that can be assigned to more than one generator. The Voronoi polygons that share the same edges are called adjacent polygons and their generators are called adjacent generators.

The following property holds for any Voronoi diagram and is used to answer kNN queries: "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)." (see [9,14] for further details). In general, a Voronoi diagram of a set of "sites" (points) is a collection of regions that divide up the plane. Each region corresponds to one of the sites, and all the points in one region are closer to the corresponding site than to any other site.

3.2 Network voronoi diagram

In many applications, there is an underlying spatial road network, in which movements of objects are based on. Therefore, the real distance between two objects in a spatial road network is the actual network distance, and not the Euclidean distance. Therefore, the methods using the Euclidean distance that give rough estimates might even be significantly wrong.

Several assumptions of the Voronoi diagram are violated in urban areas; distances between two addresses are not Euclidean; they have to be measured along the travel network(s). Thus, we use the NVD [14]. "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" [15,16]. Network Voronoi Diagram considers distances only in networks, not in the plane. It divides the network, not the space, into *Voronoi cells*. A Voronoi cell in a network is the set of nodes and edges that are closer to one *Voronoi generator* (*generator point*), than to any other.

Figure 3 gives an illustration of NVD. The underlying spatial road network is superimposed on the Voronoi diagram. The generator point is the object of interest, which is in this case shown as filled rectangular shape objects. Since there is an underlying road network in each Voronoi cell, the distance of all locations or points within a Voronoi cell are closer based on the real road distance to its generator point. Because road distance is used, instead of Euclidean distance, the shape of each Voronoi cell might be irregular, unlike the Voronoi cell in the Voronoi diagram (as in Fig. 2), which has a convex polygon shape.

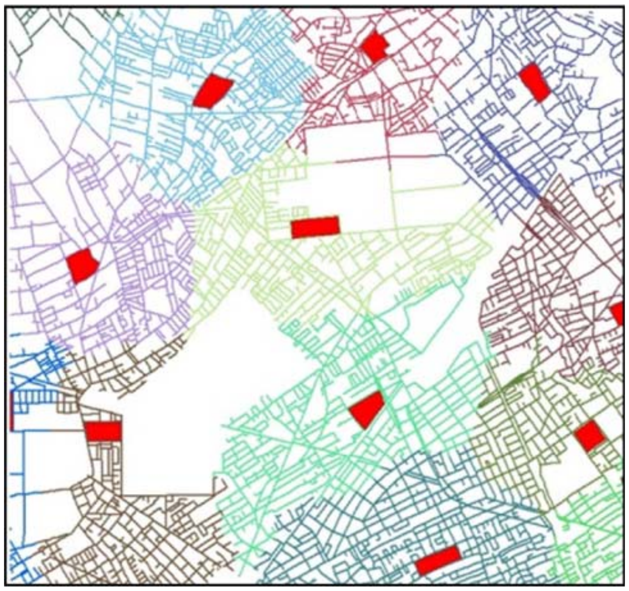


Fig. 3 Network Voronoi diagram

3.3 Progressive incremental network expansion

Our previous work on spatial kNN query processing proposed the *Progressive Incremental Network Expansion* (PINE), especially designed for mobile navigation [17,18]. PINE is a novel approach that reduces the problem of distance computation in a very large network into the problem of distance computation in a number of much smaller networks plus some online "local" network expansion. Therefore, PINE is very much suitable for mobile devices, which generally have limited on board memory resources and have lower computational power.

The main idea behind PINE is to first partition a large network into smaller/more manageable regions. We achieve this by generating a NVD over the points of interest. Each cell of this Voronoi diagram is centered by one object of interest, and contains the nodes that are closest to that object in network distance (and not Euclidian distance).

Next, we pre-compute the inter distances for each cell. That is, for each cell, we pre-compute the distances across the border points of the adjacent cells. This will reduce the pre-computation time and space by localizing the computation to cells and a handful of neighbor-cell node-pairs.

Now, to find the k nearest-neighbors of a query object q, we first find the first nearest neighbor by simply locating the Voronoi cell that contains q. This can be easily achieved by utilizing a spatial index (e.g., R-tree) that is generated for the Voronoi cells. Then, starting from the query point q, we perform network expansion on two different scales simultaneously to: (i) compute the distance from q to its first nearest neighbor (its Voronoi cell centre point), and (ii) explore the objects that are close to q (centres of surrounding Voronoi cells) and compute their distances to q during the expansion.

At the first scale, a network expansion similar to Incremental Network Expansion (INE) [15] is performed inside the Voronoi cell that contains q [i.e. VC(q)] starting from q. To this end, we utilize the actual network links (e.g., roads) and nodes (e.g., restaurants, hospitals) to compute the distance from the query point q to its first nearest neighbor [i.e., the generator point of VC(q)] and the border points of VC(q).

When we reach a border point of VC(q), we start a second network expansion at the Voronoi polygons scale. Unlike INE and but similar to Voronoi-based Network Nearest neighbor (VN^3) [9], the second expansion utilizes the inter-cell pre-computed distances to find the actual network distance from q to the objects in the other Voronoi cells surrounding VC(q). Note that both expansions are performed simultaneously. The first expansion continues until all border points of VC(q) are explored or all kNN are found.

We need to stress that PINE, which is based on Voronoi and Dijsktra's algorithm, is designed for kNN queries, not RNN. In this paper, we would like to apply PINE to RNN.

4 Reverse nearest neighbor queries

We have identified four different types of RNN queries that consider interest objects, query point, and other static (or quasi-static) non-query objects. Before we describe the four RNN query types, we need to formalize the definitions of some terminologies.

Definition 1 A *generator point* (*GP*) of a Voronoi polygon (VP) is the centre point of VP.

Definition 2 A *query point*, or sometime known as *query object* (*qObj*) is the point where the query is invoked. Hence, a query object is associated with a point, called query point. And hence the terms query point and query object will be used interchangeably.

Definition 3 An *interest object* (*iObj*) is a candidate object for the query result of RNN. An interest object is located at a point called interest point. Hence, the terms interest object and interest point will be used interchangeably.

Notation 1 RNN query can be written as: **RNN** $_{qObj}(iObj)$

where *qObj* is the query point, and *iObj* is the interest object.

Definition 4 An *RNN query* is a query to retrieve interest objects that consider the query point as their nearest neighbor.

In RNN queries, the query point (*qObj*) may be a generator point (*GP*) or a non-generator point (∼*GP*). Note that we use the ∼ symbol for negation. Likewise, the interest objects (*iObj*) may also be the generator points or non-generator points. Based on this, we classify four types of RNN queries. However, before the four RNN categories are presented, we need to define non-query objects (or rival objects *rObj*)

Definition 5 A *non-query object*, also known as a *rival object* (*rObj*) is an object of the same type of the query object (*qObj*), which is the rivalry of the query object whereby some interest objects might consider *rObj* (not *qObj*) as their nearest neighbors.

Using the above definitions, the four categories or RNN queries are as follows:

Type 1 $\text{RNN}_{GP}(GP)$, where both query point and interest objects are generator points.

Example Given an NVD whereby the generator points are restaurants, $\text{RNN}_{Restaurant}$ (*Restaurant*) is to find other restaurants that consider the query restaurant as their nearest neighbor (e.g. their nearest restaurant).

Type 2 $\text{RNN}_{GP}(\sim GP)$, where the query point is a generator point, but the interest objects are not.

Example Given an NVD containing restaurants as the generator points, and schools as the interest objects, $\text{RNN}_{Restaurant}$ (*School*) is to find schools that consider the query restaurant as their nearest neighbor restaurant.

Type 3 $\text{RNN}_{\sim GP}(\sim GP)$, where both query objects and interest objects are non-generator points.

Example Given a school as the query object (in this case, schools are not generator points), RNN_{School} (*School*) is to find other schools that consider the query school as their nearest neighbor (e.g. nearest school).

Type 4 $\text{RNN}_{\sim GP}(GP)$, where only the interest objects are generator points; but the query object is not.

Example Given an NVD whereby the generator points are restaurants, RNN_{School} (*Restaurant*) is to find restaurants that consider the query school as their nearest school. Note that there may be many schools in the system, and those schools which are not the query object are rival objects (*rObj*).

5 Proposed algorithms for RNN queries

Our developed algorithms to answer RNN queries depend on the existence of a 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 [17] creates a set of

NVDs, one for each different interest point (e.g., NVD for restaurants, schools, etc.). In this paper, we develop new algorithms to answer RNN queries that utilize the previously created NVDs, pre-computed distances, and PINE algorithm.

5.1 Type 1: $\mathrm{RNN}_{GP}(GP)$

This 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 NVD. 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 [9,17] for details).

The query first starts by using NVD to find the distances from the generator of the polygon (in this case it is also "Q" which is the generator point of $P1$) to all border points (i.e. $b1$, $b2$, $b3$ ..., etc.) and then the distances from those border points to adjacent generators. For example, in Fig. 4, we need to find dist(Q, $b1$) + dist($b1$, $P2$). For simplicity, unlike the NVD shown previously in Fig. 3, the underlying spatial network in Fig. 4 is not shown, and therefore, the polygons are shown to be convex. In actual NVD, the polygons are not convex polygons.

Once the neighboring generator points are reached, the algorithm starts a heap list with the dist(Q, *Adjacent Generator Points*) as the initial distance. The distances between all candidates are first measured [i.e. dist($P2$,$b14$) + dist($b14$, $P3$)] using NVD generator-to-border and vice versa, thus eliminating the repetition of the calculation among them ($P2$ to $P3 = P2$ to $P3$).

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

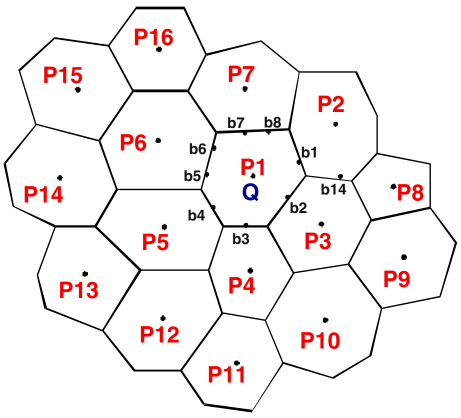


Fig. 4 RNN GP(GP)

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 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.

Let us take for example Fig. 4. The query point is $P1$ and thus the solution to the query belongs to the set of {$P2$, $P3$, $P4$, $P5$, $P6$, $P7$}.

First, we execute 1NN query using NVD to find the distance from $P1$ to $b1$, $b2$, $b3$, $b4$, $b5$, $b6$, $b7$ and $b8$, and the distances from these borders to the adjacent generator points. However, if there are two paths to one generator point, for example dist($P1$,$b7$) + dist($b7$,$P7$) and dist($P1$,$b8$) + dist($P8$,$P7$), then the longer distance should be eliminated. At this stage let's assume that $P3$ was found as the 1NN to Q and dist(Q,$P3$:10).

Next, the distances between the candidate neighbors should be calculated for example dist($P2$,$P7$:18), dist($P2$, $P3$:14), dist($P3$,$P4$:8), etc. Now, we check the found distances. If two polygons are closer to each other than the query to its 1NN, then we say that those two polygons do not belong to the set of candidates RNN. In this example, we found that $R3$ and $R4$ that are closer to each other than to the query point. Therefore the RNN candidate set is now {$P2$, $P5$, $P6$, $P7$}.

Each candidate interest object starts a heap list with the distance from the query point as initial value and executes 1NN query using the NVD generator-to-border distances. If a distance is found that is shorter than the initial value in its heap then the search stops for that polygon and announces that this candidate generator point is not the RNN to the query object. If however, the search ended with the distance from the query point less than the initial value then the candidate generator point is an RNN to Q.

$\mathrm{RNN}_{GP}(GP)$ algorithm is shown in Fig. 5. The function `contains( )` returns the Voronoi polygon containing the query point Q. Note that after 1NN of the query polygon is executed, it becomes the anchor, in which all 1NN of the adjacent polygons will be compared with. $\mathrm{RNN}_{GP}(GP)$ algorithm relies on 1NN of each of the adjacent polygon of the query polygon.

5.2 Type 2: $\mathrm{RNN}_{GP}(\sim GP)$

This query type can be solved using only the NVD properties. Since every object in the polygon considers the generator

```
Algorithm: RNN-1
Input: NVD($P_1$, $P_2$, … $P_n$)
Output: RNN $_{GP}(GP)$
1.  Let $Q$ be the query point
2.  Voronoi Polygon VP($Q$) = contains($Q$)
3.  Border points VP($Q$) = {$b_i$.. $b_j$}
4.  Candidate RNN is all adjacent polygons {$P_k$ .. $P_m$}
5.  //Compute the distances between all $P$in Candidate RNN
    For any $P_x$ and$P_y$in Candidate RNN
6.      If $P_x$ and$P_y$are adjacent then
7.          Find the common borders point between them CVP($P_x$,$P_y$) (i.e. Border
            Points VP($P_x$) intersection with VP($P_y$))
8.          Compute $dist(P_x, P_y) = dist(P_x, b_z) + dist(b_z, P_y)$  where $b_z$ belongs to
            CVP($P_x$,$P_y$)
9.      End If
10. End For
11. //Compute all distances between all $P$in Candidate RNN and $Q$
    For any $P_x$ in Candidate RNN
12.     Compute $dist(P_x, Q) = dist(P_x, b_z) + dist(b_z, Q)$  where $b_z$ belongs to VP($Q$)
13. End For
14. For any $P_x$ and$P_y$in Candidate RNN
15.     If $dist(P_x, P_y) < dist(Q$, 1NN($Q$)) then
16.         Remove $P_x$ and$P_y$from Candidate RNN
17.     End If
18. End For
19. For each candidate polygon $P_k$
20.     Execute 1NN($P_k$)
21.     If $dist(Q$, 1NN($Q$)) >$dist(P_k$, 1NN($P_k$)) Then
22.         Remove {$P_k$, 1NN($P_k$)} from Candidate RNN
23.     End If
24. End For
25. Return Candidate RNN
```

Fig. 5 RNN GP(GP) algorithm

```
Algorithm: RNN-2
Input: NVD($P_1$, $P_2$, … $P_n$), interest objects in each Voronoi Polygon $P_i$
Output: RNN $_{GP}(\sim GP)$
1. Let $Q$ be the query point
2. Voronoi Polygon VP($Q$) = contains($Q$)
3. Candidate RNN = objects(VP($Q$))
4. Return Candidate RNN
```

Fig. 6 RNN GP (∼GP) algorithm

point as the nearest neighbor, then the result of the query should be a set of interest objects that belong to the polygon.

Proposition 1 *If there are interest objects in the query polygon, then these interest objects are considered as the answer to the RNN query. If however there are no interest objects in the query polygon, then there is no result for the query* (*i.e. empty set solution*).

RNN$_{GP}(\sim GP)$ algorithm is shown in Fig. 6. The `contains( )` function as like in the first algorithm returns the Voronoi polygon containing the query point, and the new `objects( )` function returns interest objects in the given Voronoi polygon. Since all objects in each polygon, the `objects( )` function will simply search through the objects within the given polygon. Note that the function may return an empty set of interest objects, indicating that there is no interest object in that particular polygon.

5.3 Type 3: RNN$_{\sim GP}(\sim GP)$

One of the problems in answering this type of RNN query is that we have no pre-computed NVD that uses the interest objects. Hence, our algorithm will utilize any of the stored NVDs to solve the query. Using a pre-computed NVD helps reducing the network distance calculation time to answer such a query.

The idea is to use PINE expansion mechanism to explore all the candidate interest objects that are in the vicinity of the query object. Once a candidate interest object is reached, a 1NN query is issued using the original PINE algorithm to check if the query object is the 1NN of the candidate interest object or not. If yes, then the candidate interest object becomes a part of the solution. Note that we need to expand in all directions from the query point and the candidate interest points.

In other cases, polygons may contain an interest point but are neither adjacent to the polygon containing the query point, nor to polygons containing interest points. Those can be reached by crossing over what we call an "empty polygon" (a polygon containing no interest points), and they may contain a candidate solution. Here, we utilize the border-to-border distances provided by the NVD for the empty polygon from the adjacent edge of the generator polygon to the edge of the polygon that contains any interest objects. We will also use the results of Lemma 1 in [29].

Lemma 1 *"Let Q be a query point, n a graph node, and p a data point satisfying $dist(q, n) > dist(p, n)$. For any point $p' \neq p$ whose shortest path to Q passes through n, it holds that $dist(Q, p') > dist(p, p')$, i.e., p′ RNN(Q)"* [29].

From Lemma 1 we can conclude that if the expansion started from Q and reached an intersection point, a border point of NVD, or an interest point B passing through an interest point A, then point A is closer to Q than point B. Thus, for the proposed method we say that B is closer to A than it is to Q. This would limit our expansion in some directions. The expansion will continue in all directions and stop in a direction if a candidate interest point that satisfies the properties in Lemma 1 is reached.

For example, once a query point is initiated, the algorithm will check the current polygon for interest data points by utilizing the spatial index (e.g. R-tree). Then, the expansion mechanism of PINE is used for finding the interest points close to the query point. During this process, each found interest point will start a heap list with its distance to the query point as the initial entry, such as dist(Q, $iObj1$). Each found interest point starts another network expansion to fill up the rest of its heap list.

If a new interest data point is found then its distance, for example dist($iObj1$, $iObj3$), is compared with dist(Q, $iObj1$). The expansion stops in that direction only if dist ($iObj1$, $iObj3$) > dist(Q, $iObj1$). However, if dist($iObj1$, $iObj3$) < dist(Q, $iObj1$) then the entire expansion "all directions" from $iObj1$ stops, because $iObj3$ is considered

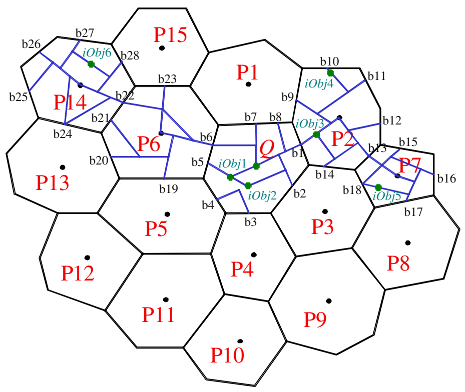


Fig. 7 RNN ∼GP(∼GP)

Algorithm: RNN-3

```
Input: NVD(P1, P2, … Pn), and interest objects {iObj1 .. iObjn}
Output: RNN ~GP(~GP)
1.  Let Q be the query point
2.  Use PINE algorithm to expand Q
3.  For each direction in the expansion of Q
        //Interest objects that can be reached by passing through an empty polygon,
        //use NVD border-to-border distance to the destination polygon, and then
        //start expanding from there for finding the inner distance to the interest object
4.      If an interest object iObj is found Then
5.          Stop expanding on the direction to iObj
6.          Store iObj and dist(Q,iObj) into Candidate RNN
7.      End If
8.  End For
9.  For each interest object iObj in Candidate RNN
10.     Execute 1NN using PINE (refer to Safar (2005))
11.     PINE will test every new distance once it reaches an intersection point,
        an interest object, or a polygon border point.
12.     Stop expansion, if this distance is longer than dist(Q,iObj)
13.     When 1NN of iObj is found, calculate dist(iObj,1NN(iObj))
14.     If dist(iObj,1NN(iObj)) <dist(Q,iObj) Then
15.         Remove iObj from Candidate RNN
16.     End If
17. End For
18. Return Candidate RNN
```

Fig. 8 RNN ∼GP(∼GP) algorithm

as 1NN of $iObj1$. Hence, $iObj1$ cannot be the reverse nearest neighbor of the query.

If however one of the neighboring polygons does not contain any interest data points (empty polygon), then the polygon is skipped to its adjacent polygons, by using the virtual border-to-border links. This can be illustrated by looking at $P6$ in Fig. 7. We notice that $P6$ is an adjacent polygon to the polygon of the query point and it contains no interest points, therefore, the distances from $b6$ to $b19$, $b20$, $b21$, $b22$ and $b23$ are utilized to pass this empty polygon to its neighboring polygons. Otherwise, a new network expansion should have started in $P6$ using the actual road network in that polygon.

As an example, consider the NVD of Fig. 7. From the query point we start an expansion and let us assume that we reach $iObj1$, $iObj3$, $b8$, $b7$, $b6$, $b2$, $b1$. Checking $P1$, $P3$, $P4$, $P6$, we find that the polygons contain no interest objects, so we directly get the border-to-border distance and check their adjacent polygons for interest objects.

Now, exploring through polygon $P6$ could lead us to $P14$, which contains $iObj6$. Once the border of $P14$ is reached the network expansion $b22$ is started to find the network distance to $iObj6$. On the other hand, instantaneously, another network expansion starts in $P2$ from $b1$ finding $iObj3$ and then stops.

Lets say for simplicity that dist$(Q, iObj1 : 12)$, dist$(Q, iObj3 : 20)$ and dist$(Q, iObj6 : 35)$ are the results of this candidate interest point locating stage with their distances from the query point. Although $iObj5$ can be reached from $b2$–$b18$ and then expanding in $P7$, we will not look into it because it is a similar case to $iObj3$.

Now, after all candidates are found, each interest object starts a heap list with the network distance to the query point as the initial entry and executes 1NN query. The search for 1NN performs the same steps again, starting with finding all the interest points and border points, however, this time the expansion distance is limited by the initial value of the heap. If the expansion goes a longer distance without finding any interest points, then it stops in that direction. Therefore, in this example $iObj5$ will never be reached because $b13$ distance from $iObj3$ is larger than 20. $iObj3$ continues the expansion to find $iObj4$: 22. Since there was neither an intersection point nor a border point after a distance of 20 that could be added to the heap, we were not able to tell earlier that $iObj4$ is further away from the query point. The result for $iObj3$ now clearly states that the query point is the nearest neighbor 1NN.

For $iObj1$, we immediately find that $iObj2$ is the nearest neighbor, thus the expansion stops in that direction and the result states that the query point is not the 1NN.

$iObj6$, however, starts to find the distances to the polygons border points $b21$, $b22$, $b23$, $b24$... etc. From there, using the border-to-border distance from NVD it searches for interest objects. The search in any direction stops when the distance to any border from $iObj6$ exceeds the initial heap value. For this example the search will stop with no interest objects found in a shorter distance stating that the query object is not the 1NN. The search is now finished with $iObj3$ as the reverse nearest neighbor of the query object.

RNN$_{\sim GP}(\sim GP)$ algorithm is shown in Fig. 8. Note that the first stage is the expansion from Q to find any interest object $iObj$, and the second stage is expanding each of the found interest object $iObj$ using PINE to find its 1NN. If the distance between interest object $iObj$ and its 1NN is shorter than the distance between interest object $iObj$ and Q, then obviously that $iObj$ is not RNN of Q. Note that the expansion technique uses PINE (refer to [17]) which uses NVD and its stored pre-computed values, especially border-to-border distances. In this way, expansion through the network of Voronoi Polygons can be performed efficiently.

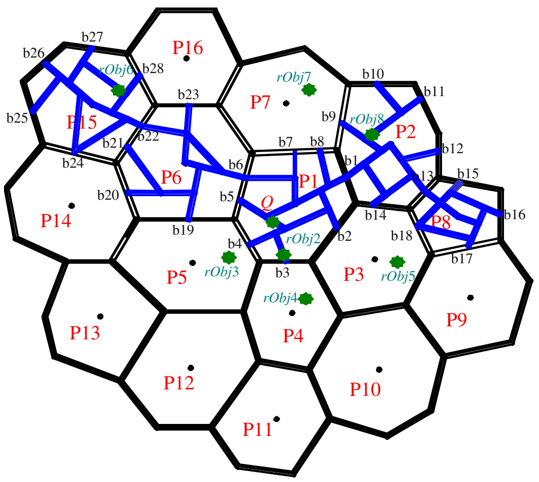


Fig. 9 RNN ∼GP(GP)

```
Algorithm: RNN-4
Input: NVD($P_1, P_2, \ldots P_n$), and rival objects {$rObj_1$ .. $rObj_n$}
Output: RNN $_{\sim GP}(GP)$
1.  Let $Q$ be the query point
2.  Use PINE algorithm to expand $Q$ to find generator points of other polygons
3.  The expansion in each path stops once a rival object $rObj$ or a border point $b$
    of a polygon that contains rival objects is reached
4.  The reached candidate polygons will be checked for any rival objects. If the
    polygon contains rival objects $rObj$ then it is not a candidate RNN. Otherwise
    generator point of the polygon is stored in Candidate RNN
    //use NVD border-to-generator point distance to reach the candidate generator
5.  For each generator points $GP_i$ in Candidate RNN
10.     Execute 1NN using PINE to find rival objects $rObj$
11.     PINE will test every new distance once it reaches an intersection point,
        a rival object, or a polygon border point.
12.     Stop expansion, if this distance is longer than $dist(Q,GP_i)$
13.     When 1NN of $GP_i$ is found, calculate $dist(GP_i,rObj)$
14.     If $dist(GP_i,rObj) < dist(Q,GP_i)$ Then
15.         Remove $GP_i$ from Candidate RNN
16.     End If
17. End For
18. Return Candidate RNN
```

Fig. 10 RNN ∼GP(GP) algorithm

5.4 Type 4: RNN$_{\sim GP}(GP)$

For this query type, the interest points are generator points which already have NVDs created for them that could be used to answer the RNN query. The steps for this algorithm are similar to that used for RNN$_{\sim GP}(\sim GP)$, with one exception that will be mentioned next.

The algorithm starts with network expansion to explore the possible generator points that can be reached by the query object, including the generator point of the polygon and the border points to neighboring polygons. However, if any of the adjacent polygons contains rival objects, then it is eliminated from the candidate list and no expansion is performed from that polygon. It is unnecessary to go into a polygon that contains rival objects, because the generator points of those polygons consider those rival objects as their nearest neighbors.

As an example, consider the NVD of Fig. 9. The network expansion will start at Q looking for border points, and interest points (which are generator points of other Voronoi polygon). In this example, it reaches $P1$ and stops in that direction. The expansion will also reach $b1$, $b2$, $b4$, $b5$, $b6$, $b7$, and $b8$. The polygons that we can reach from these border points are now checked for rival objects by utilizing the spatial index. Note that rival objects are objects that have the same type as the query object. In this example, only polygon $P6$ that does not contain any rival objects and hence can be considered as a candidate. The distance from Q to $P6$ can be found using NVD border-to-generator distance.

Once $P6$ is reached, a heap list is created with dist(Q, $P6$) as the initial entry. $P6$ will then start 1NN search to find rival objects, and since there are no rival objects in $P6$, we use NVD property to find the generator-to-border distance to all neighboring polygons. The reached polygons are checked for rival objects and the result found is that $P5$ and $P15$ contain rival objects.

Now, network expansion starts from $b19$ and $b22$ to find the distances to $rObj3$ and $rObj6$. The expansion finds that $dist(P6, rObj3)$ and $dist(P6, rObj6) > dist(Q, P6)$, thus the search stops and we can conclude that Q is the nearest neighbor for $P6$.

However, since $P14$ is an empty polygon and can be reached passing through another empty polygon, then it is considered as a candidate for RNN, and can be treated the same way as $P6$, but this time we need dist($b6$, $b20$), dist($b20$, $P14$) and to start expansion at $P14$. However, for simplicity we assume that dist($P14$, $rObj6$) < dist(Q, $P14$), that makes it not the RNN for Q.

Figure 10 shows the algorithm for RNN$_{\sim GP}(GP)$. The main similarity between this algorithm and the previous algorithm for RNN$_{\sim GP}(\sim GP)$ is that both algorithms have two stages: stage one is the expansion from Q to find GP or to find $iObj$, in case of RNN$_{\sim GP}(GP)$ and RNN$_{\sim GP}(\sim GP)$, respectively. Then stage two is to perform 1NN from each candidate RNN. Distance comparison is then carried out to determine whether the GP [in case of RNN$_{\sim GP}(GP)$] or $iObj$ (in case of RNN $_{\sim GP}(\sim GP)$) is RNN or not.

On the other hand, the main difference between RNN$_{\sim GP}$ (GP) algorithm (Fig. 10) and RNN$_{\sim GP}(\sim GP)$ algorithm (Fig. 8) lies in the first stage. In Fig. 10, the first stage is to expand Q to find GP, that is a generator point of another polygon, whereas in Fig. 8, the expansion from Q is to find interest object $iObj$ which is not a generator point of another polygon.

6 Performance evaluation

We conducted several experiments to evaluate the performance of our RNN algorithms. We used real-world data sets obtained from NavTech Inc., used for navigation and GPS devices installed in cars, and represent a network of approximately 110,000 links and 79,800 nodes of the road system in downtown Los Angeles. The experiments were using Oracle DBMS as the database server. We used different sets of points of interest (e.g., restaurants, shopping centers, etc.). We tested of 100 runs of RNN queries, each of which has a random query point. For each query, we calculated the number of RNNs together with their execution time.

6.1 $\mathrm{RNN}_{GP}(GP)$ query experimental results

For $\mathrm{RNN}_{GP}(GP)$ query, we experimented with a number of object types, like hospitals, shopping centers, schools, etc. In Table 1, we present the average query execution time results of 100 runs of the RNN queries of five different types of interest objects with various densities.

Density is defined as the ratio between the total numbers of interest objects over a specified range area. It is natural that the more dense the object population, the more objects being retrieved. For example, comparing hospitals (very low density) and restaurants (very high density), only 46 hospitals are retrieved compared to almost 3,000 restaurants.

Figure 11 depicts the graph for the average query response time. Note that there is only a slight increase (even looks constant) even in case of objects with high density. Refer to the various objects with different densities in Table 1. The slight increase of the query response time in this case is due to the fact that the query results depend on the number of neighboring interest points (or Voronoi cells) of the query point. And in our previous work [17], we have proven that on average the number of neighbors in the NVD is only six. Hence, the query execution time is primarily affected by the number of these small number neighboring Voronoi cells, and consequently, density is not much a factor in the query execution time.

Table 1 Performance of RNN GP(GP) queries

Average over 100 queries			Execution time (s)
Interest objects	Quantity	Density	
Hospital	46	0.0004	0.077
Shopping centres	173	0.0016	0.092
Schools	1,230	0.0150	0.095
Auto services	2,093	0.0326	0.113
Restaurants	2,944	0.0580	0.118

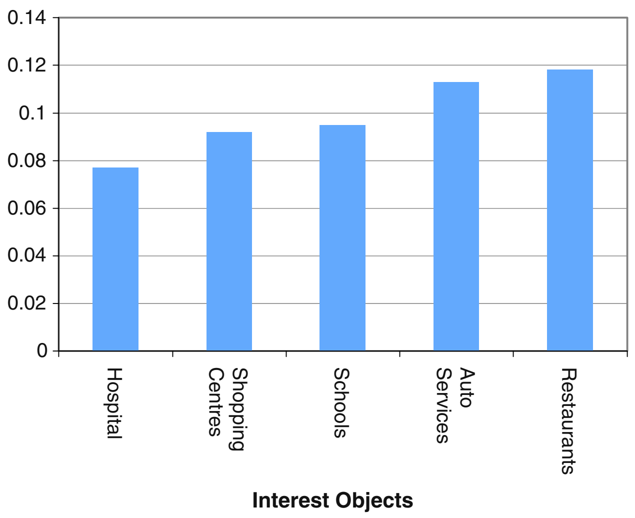


Fig. 11 Average execution time of RNN GP(GP) queries

6.2 $\mathrm{RNN}_{GP}(\sim GP)$ query experimental results

In the second set of experiments, we executed the query $\mathrm{RNN}_{GP}(\sim GP)$: for a given generator point as a query object, find all non-generator points from other Voronoi cells that consider the query object as their nearest neighbor. We experimented with different types of generator points, which are listed in Table 2 as the query points. The interest objects can be anything (e.g. other static objects), as long as they are not generator points.

In Table 2, we present the average results of the execution time of 100 runs of such queries. The interest objects, as well as their densities, are identical to those of Table 1. The main difference is in the execution time.

From the trend shown in Fig. 12, we conclude that on average the query response time of this RNN increases dramatically for objects with high density (e.g. restaurants). This is totally different from Fig. 11, where the execution time is rather constant. Note from Fig. 12 that for example, the double increase of density from 0.0326 (e.g. auto services) to 0.0580 (e.g. restaurants) results in more than triple jump in the execution time (from 0.56 to 1.49). The sharp increase of the query response time for $\mathrm{RNN}_{GP}(\sim GP)$ queries is due to the fact that as the density increases, the R-tree saving the polygons that represents the Voronoi cells increases in size as well. Consequently, it takes more time to check whether an interest point is inside the polygon (Voronoi cells) or not. In other words, density plays a major role in the execution time of $\mathrm{RNN}_{GP}(\sim GP)$.

Table 2 Performance of RNN GP(∼GP) queries

Average over 100 queries			Execution time (s)
Interest points	Quantity	Density	
Hospital	46	0.0004	0.18
Shopping centres	173	0.0016	0.19
Schools	1,230	0.0150	0.21
Auto services	2,093	0.0326	0.56
Restaurants	2,944	0.0580	1.49

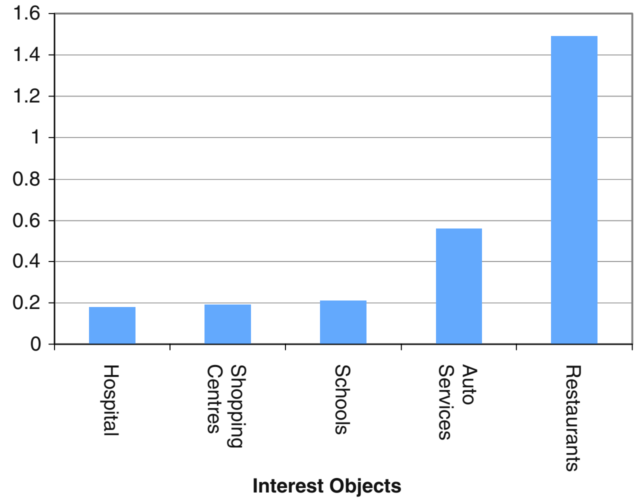


Fig. 12 Average execution time of RNN GP(∼GP) queries

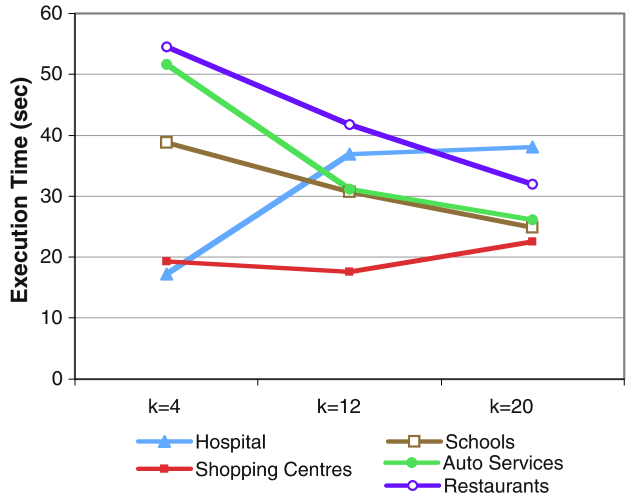


Fig. 13 Average execution time of RNN ∼GP(∼GP) queries

6.3 RNN$_{\sim GP}(\sim GP)$ query experimental results

For the query RNN$_{\sim GP}(\sim GP)$, for a non-generator point as a query object, we would like to find non-generator points from other Voronoi polygons that consider the query object as their nearest neighbor.

Since the query uses non-generator points for the query itself and the interest objects, there must be some reference point to the generator of the NVD. In this case, we use different generators as listed in Table 3.

In Table 3, we present the average results of 50 runs of the query. In this experiment, we show the number of RNNs that were found and the query execution time (in s) for various values of k (e.g. 4, 12, and 20).

From Fig. 13, we conclude that on average, the total query response time of RNN$_{\sim GP}(\sim GP)$ queries decreases as k increases for high density data and increases as k increases for low density data. For example, for the high density data (e.g. restaurants), the execution time decreases from 54 s ($k = 4$) to 32 s ($k = 20$). But for the low density data (e.g. hospitals), the execution time increases from 17 s ($k = 4$) to 38 s ($k = 20$).

For high density data, each candidate object have to explore a smaller area (i.e., Voronoi cell) to find if the query object is their nearest neighbor or not. Then, as the candidate object density increases, the probability that they would find other objects (rather than the query object) as their nearest neighbors also increases.

However, for low density data, each candidate object would have to explore a larger area to find if the query object is their nearest neighbor or not.

From Table 3, we also conclude that the average number of RNN is independent of the number of objects in the system and the density of the data. It only depends on the location of the objects in the system.

6.4 RNN$_{\sim GP}(GP)$ query experimental results

In the last set of experiments, we executed the query RNN$_{\sim GP}$ (GP): for a given query point, find interest objects that consider the query point as their nearest neighbor object. In Table 4, we present the average results of 100 runs of the query as k (number of query objects) increases for a fixed radius circular area of search around the query point. For

Table 3 Performance of RNN ∼GP(∼GP) queries

Average over 50 queries; k =number of query objects		$k = 4$		$k = 12$		$k = 20$	
Interest points	Qty (Density)	#RNN	Execution time	#RNN	Execution time	#RNN	Execution time
Hospital	46 (0.0004)	0.78	17.25	1.00	36.96	1.10	38.12
Shopping centres	173 (0.0016)	1.20	19.31	0.64	17.56	0.98	22.56
Schools	1,230 (0.0150)	0.98	38.87	1.02	30.73	1.06	24.89
Auto services	2,093 (0.0326)	1.04	51.66	1.06	31.13	0.98	26.09
Restaurants	2,944 (0.05800)	0.88	54.49	0.84	41.83	0.94	32.01

Table 4 Performance of RNN ~GP(GP) queries

Average over 100 queries; k =number of query objects; maximum distance = 1,000 m		$k = 4$		$k = 12$		$k = 20$	
Interest points	Qty (Density)	#RNN	Execution time	#RNN	Execution time	#RNN	Execution time
Hospital	46 (0.0004)	1.00	0.17	1.00	0.10	1.00	0.12
Shopping centres	173 (0.0016)	1.13	0.18	1.01	0.09	1.00	0.08
Schools	1,230 (0.0150)	1.56	0.22	1.77	0.17	1.59	0.19
Auto services	2,093 (0.0326)	4.11	0.53	3.41	0.48	3.24	0.42
Restaurants	2,944 (0.05800)	6.62	1.46	5.95	1.25	4.92	0.98

example, in the table, we show the query response time when the maximum radius of the circular search area from the query point is restricted to 1,000 m, and the value of k varies from 4, 12, and to 20 query objects.

The first and second columns specify the interest points and their density ratio (i.e., the number of interest points over the number of links in the network), respectively. From the third column and forward, each table entry has two values (averaged over 100 runs): (i) Number of reverse nearest neighbors that were found, and (ii) Execution time in seconds.

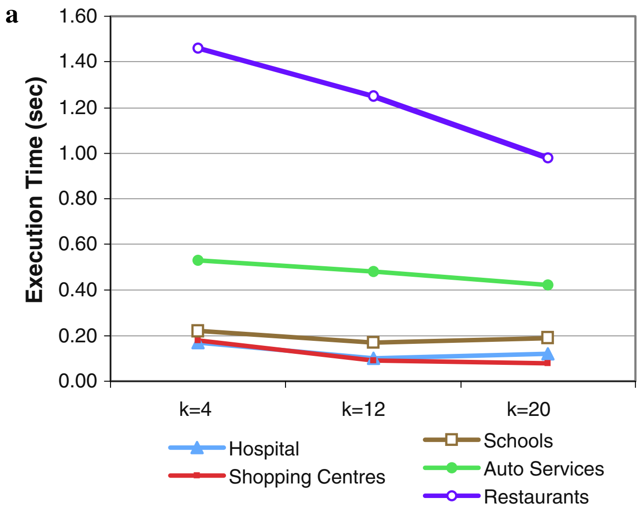
From Fig. 14a and b, we can see that on average, the total query response time of the queries decreases as the number of k increases. This is due to the fact that, as the number of query objects available in the system increases, the interest points would find other objects as their nearest neighbor rather than the query point and they would stop searching. Hence, the query would be executed faster.

In addition, as the density increases, the performance degrades. This is due to the fact that, as the density increases, more interest points would be available in the system. As a consequence, more interest points would issue nearest neighbor queries. Hence, the query execution time would increase.

In the next set of experiments, we executed the RNN query as the radius of the circular search area around the query point increases for a fixed k (number of objects) of a value equal to five. In Table 5, we present the average results of 100 runs, that represent: (i) Number of reverse nearest neighbors that were found, and (ii) Execution time in seconds.

From Fig. 15a and b, we can see that on average, the total query response time (or average number of RNNs) increases as the radius of the circular search area increases. This is simply because as the size of the search area increases, the number of objects and interest points available also increase. As a result, more time is required to process the query.

In addition, as the density increases, the performance also degrades. This is due to the fact that, as the density increases, more interest points would be available, and as a consequence, more interest points would issue nearest neighbor queries. Hence, the query execution time would increase.



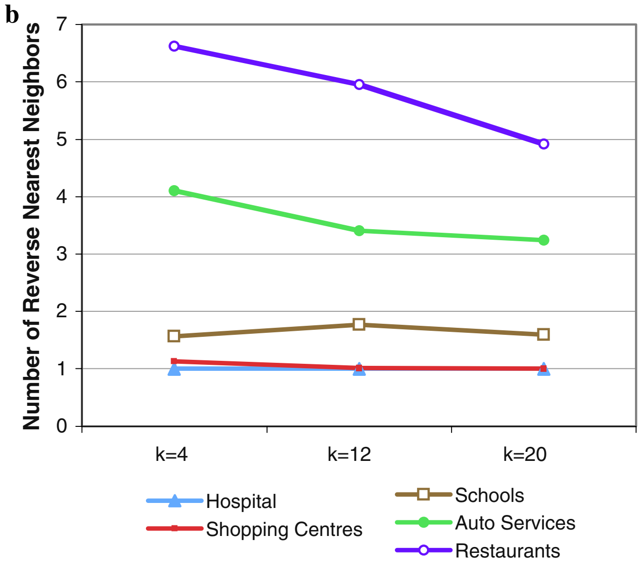


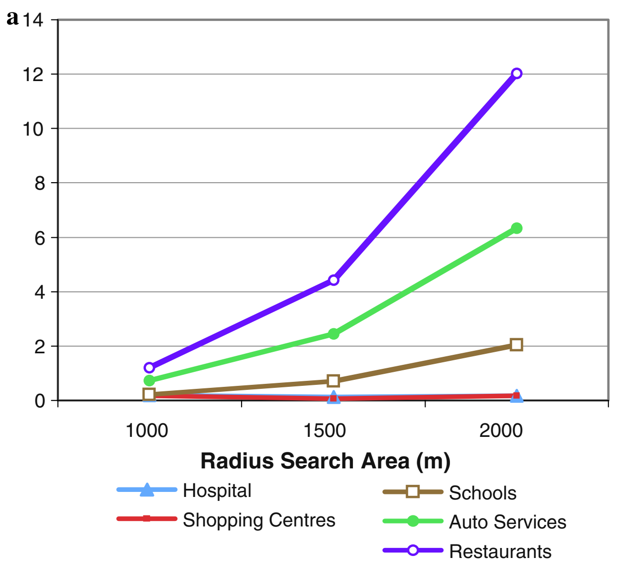
Fig. 14 **a** Average execution time of RNN ~GP(GP) queries. **b** Average number of reverse nearest neighbors for RNN ~GP(~GP) queries

7 Conclusion and future work

In this paper, we have demonstrated an emerging application of Voronoi network in spatial query processing, particularly

Table 5 Performance of RNN ∼GP(GP) queries as the radius of the circular search area increases

Average over 100 queries; $k = 5$		Maximum distance = 1,000 m		Maximum distance = 1,500 m		Maximum distance = 2,000 m	
Interest points	Qty (Density)	#RNN	Execution time	#RNN	Execution time	#RNN	Execution time
Hospital	46 (0.0004)	1.00	0.19	1.00	0.12	1.00	0.17
Shopping Centres	173 (0.0016)	1.09	0.17	1.04	0.06	1.12	0.17
Schools	1,230 (0.0150)	1.58	0.22	3.25	0.70	4.81	2.04
Auto Services	2,093 (0.0326)	4.28	0.74	7.96	2.45	12.27	6.34
Restaurants	2,944 (0.05800)	6.54	1.21	12.00	4.43	18.18	12.03



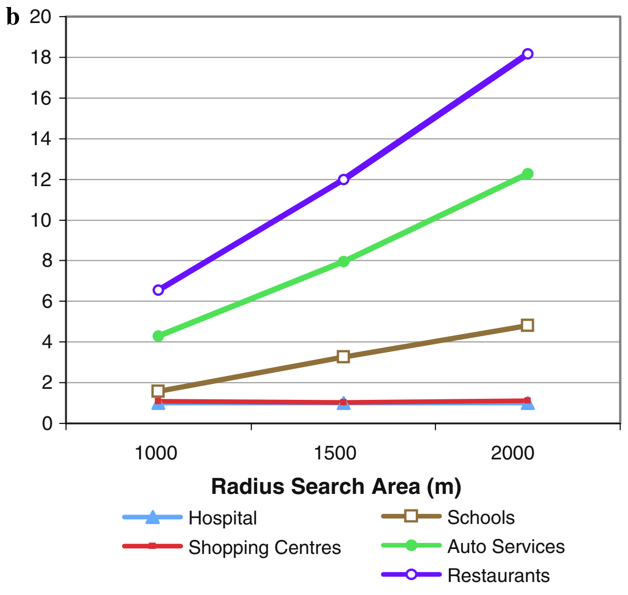


Fig. 15 **a** Average execution time of RNN ∼GP(GP) queries. **b** Average number of reverse nearest neighbors for RNN ∼GP(∼GP) queries

in RNN query processing. The approach presented in this paper is applied to road networks using real distances, whereas most of existing work focused on Euclidean distance.

This paper presented four different RNN query types, and proposed algorithms to efficiently process them. The proposed algorithms are novel and are not based on Euclidean distance. The proposed approaches that used network expansion mechanism like PINE together with NVD, offer many benefits, including reducing the number of computations required to answer an RNN query.

We performed several experiments to measure the performance of the four RNN queries. In general, our algorithms gave a good query response time. However, as expected, as the density of the data used increased, the performance of RNN queries slightly degraded. In addition, on average, the total query response time of RNN queries decreased (i.e. good performance) as the number of interest objects (k) used in the system increased.

This paper shows an interesting application of Voronoi network and opens up interesting and practical directions for future work on spatial and geographical query processing, which have direct impact to real commercial applications in a number of fields, like mobile multimedia, telecommunication and location-based services. Since online maps are now available on mobile devices, combined with the widely available of portable GPS receiver systems, investigating how moving objects are incorporated into RNN and other spatial and mobile queries (e.g. incremental KNN, continuous KNN, etc.) [26–28,30] would be beneficial to mobile users. Some of these applications should also be context sensitive [1,7].

Other important factors about mobile users and mobile navigation include traffic conditions, which play an important part [3]. In addition, it is also interesting to see how intelligence techniques may be in mobile navigation and mobile users [5,6], coupled with NVD to track moving users on the road network. It is expected that the efficient data broadcasting systems must be employed [22,23].

Acknowledgments This research has been funded by the Research Administration of Kuwait University (Project No: EO02/04), and the Australian Research Council (ARC) Discovery Project (Project No: DP0987687). We would also like to thank Ahmad Al-Saleh for his valuable help, feedback, and support.

References

1. Aleksy, M., Butter, T., Schader, M.: Architecture for the development of context-sensitive mobile applications. Mobile Inform. Syst. **4**(2), 105–117 (2008)
2. Dijkstra, E.W.: A note on two problems inconnection with graphs. Numer. Math. **1**(22), 269–271 (1959)
3. Doci, A., Xhafa, F.: WIT: a wireless integrated traffic model. Mobile Inform. Syst. **4**(3), 219–235 (2008)
4. Gadish, D.: Introducing the elasticity of spatial data. Int. J. Data Warehous. Min. IGI Global **4**(3), 54–70 (2008)
5. Goh, J.Y., Taniar, D.: Mobile data mining by location dependencies.The 5th International Conference Intelligent Data Engineering and Automated Learning (IDEAL). Lecture Notes in Computer Science, vol. 3177, pp. 225–231. Springer, Heidelberg (2004)
6. Goh, J.Y., Taniar, D.: Mining frequency pattern from mobile users.The 8th International Conference on Knowledge-Based Intelligent Information and Engineering Systems (KES), Part III. Lecture Notes in Computer Science, vol. 3215, pp. 795–801. Springer, Heidelberg (2004)
7. Gulliver, S.R., Ghinea, G., Patel, M., Serif, T.: A context-aware tour guide: user implications. Mobile Inform. Syst. **3**(2),71–88 (2007)
8. Jayaputera, J., Taniar, D.: Data retrieval for location-dependent queries in a multi-cell wireless environment. Mobile Inform. Syst. **1**(2), 91–108 (2005)
9. Kolahdouzan, M.R., Shahabi, C.: Voronoi-based *k* nearest neighbor search for spatial network databases. The 13th International Conference on Very Large Data Bases (VLDB), pp. 840–851. Morgan Kaufmann, San Francisco (2004)
10. Korn, F., Sidiropoulos, N., Faloutsos, C., Siegel, E., Protopapas, Z.: Fast nearest neighbor search in medical image databases. The 22nd International Conference on Very Large Data Bases (VLDB), pp. 215–226, Morgan Kaufmann, San Francisco (1996)
11. Lee, D.L., Zhu, M., Hu, H.: When location-based services meet databases. Mobile Inform. Syst. **1**(2), 81–90 (2005)
12. Liu, N.H., Wu, Y.H., Chen, R.: Efficient knn search in polyphonic music databases using a lower bounding mechanism. J. Multimedia Syst. **10**(6), 1432–1882 (2005)
13. Martinez, A., Martinez, J., Pérez-Rosés, H., Quirós, R.: Image processing using voronoi diagrams. The International Conference on Image Processing. Computer Vision, and Pattern Recognition, IPCV, pp. 485–491 (2007)
14. Okabe, A., Boots, B., Sugihara, K., Chiu, S.N.: Spatial Tessellations, Concepts and Applications of Voronoi Diagrams, 2nd edn. Wiley, New York (2000)
15. Papadias, D., Zhang, J., Mamoulis, N., Tao, Y.: Query processing in spatial network databases. The 29th International Conference on Very Large Data Bases (VLDB), pp. 802–813. Morgan Kaufmann, San Francisco (2003)
16. Roussopoulos, N., Kelley, S., Vincent, F.: Nearest neighbor queries. The ACM SIGMOD International Conference on Management of Data (ACM SIGMOD), pp. 71–79. ACM Press, New York (1995)
17. Safar, M.: K nearest neighbor search in navigation systems. Mobile Inform. Syst. **1**(3), 207–224 (2005)
18. Safar, M., Ebrahmi, D.: eDar algorithm for continuous knn queries based on PINE. Int. J. Inform. Technol. Web Eng. IGI Global **1**(4), 1–21 (2006)
19. Savary, L., Gardarin, G., Zeitouni, K.: GeoCache—a cache for GML geographical data. Int. J. Data Warehous. Min. IGI Global **3**(1), 67–88 (2007)
20. Seidl, T., Kriegel, H.P.: Optimal multi-step *k*-nearest neighbor search. The ACM SIGMOD International Conference on Management of Data (ACM SIGMOD), pp. 154–165 , ACM Press, New York (1998)
21. Tao, Y., Papadias, D., Shen, Q.: Continuous nearest neighbor search. The 28th International Conference on Very Large Data Bases (VLDB), pp. 287–298, Morgan Kaufmann, San Francisco (2002)
22. Waluyo, A., Srinivasan, B., Taniar, D.: Optimal broadcast channel for data dissemination in mobile database environment. The 5th International Workshop on Advanced Parallel Programming Technologies (APPT). Lecture Notes in Computer Science, vol. 2834, pp. 655–664. Springer, Heidelberg (2003)
23. Waluyo, A., Srinivasan, B., Taniar, D.: A taxonomy of broadcast indexing schemes for multi channel data dissemination in mobile database. The 18th International Conference on Advanced Information Networking and Applications (AINA), pp. 213–218. IEEE Computer Society, New York (2004)
24. Waluyo, A., Srinivasan, B., Taniar, D.: Research in mobile database query optimization and processing. Mobile Inform. Syst. 1(4):225–252 (2005)
25. Wang, L., Yang, C., Qi, M., Wang, R., Meng, X., Wang, X.: Design of a walkthrough system for virtual museum based on voronoi diagram. The 3rd International Symposium on Voronoi Diagrams ISVD, pp. 258–263. IEEE Computer Society, New York (2006)
26. Xuan, K., Zhao, G., Taniar, D., Srinivasan, B.: Continuous range search query processing in mobile navigation. The 14th IEEE International Conference on Parallel and Distributed Systems, ICPADS, pp. 361–368. IEEE Computer Society, New York (2008)
27. Xuan, K., Zhao, G., Taniar, D., Srinivasan, B., Safar, M., Gavrilova, M.: Continuous range search based on network voronoi diagram. Int. J. Grid Utility Comput. (in press) (2009)
28. Xuan, K., Zhao, G., Taniar, D., Srinivasan, B., Safar, M., Gavrilova, M.: Network voronoi diagram based range search. The 23rd IEEE International Conference on Advanced Information Networking and Applications, AINA (2009)
29. Yiu, M.L., Papadias, D., Mamoulis, N., Tao, Y.: Reverse nearest neighbors in large graphs. The 21st International Conference on Data Engineering (ICDE), pp. 186–187, IEEE Computer Society, New York (2005)
30. Zhao, G., Xuan, K., Taniar, D., Srinivasan, B.: Incremental k-nearest-neighbor search on road networks. J. Interconnect. Netw. **9**(4), 455–470 (2008)