

In-Route Nearest Neighbor Queries

Jin Soung Yoo Shashi Shekhar

*Department of Computer Science and Engineering,
University of Minnesota, 200 Union ST SE 4-192, Minneapolis MN 55414
E-mail: {jyoo, shekhar}@cs.umn.edu*

Abstract

Nearest neighbor query is one of the most important operations in spatial databases and their application domains, such as location-based services and advanced traveler information systems. This paper addresses the problem of finding the in-route nearest neighbor (IRNN) for a query object tuple which consists of a given route with a destination and a current location on it. The IRNN is a facility instance via which the detour from the original route on the way to the destination is smallest. This paper addresses four alternative solution methods. Comparisons among them are presented using an experimental framework. Extensive experiments using real road map datasets are conducted to examine the behaviors of the solutions in terms of five parameters affecting the performance. The overall experiments show that our strategy to reduce the expensive path computations to minimize the response time is reasonable. The spatial distance join-based method always shows better performance with fewer path computations compared to the recursive methods. The computation costs for all methods except the precomputed zone-based method increase with increases in the road map size and the query route length but decrease with increases in the facility density. The precomputed zone-based method shows the most efficiency when there are no updates on the road map.

Keywords: Nearest neighborhood query, route, road network, location-based services

1. Introduction

A very important query in spatial database systems and geographic information systems is the nearest neighbor (NN) search [16, 17]. The problem is: given a set of instances of a facility type, a distance metric, and a query object q , find a facility instance “closest” to q .

In the nearest neighbor literature, the *Minkowski* metrics, e.g., Euclidean distance [6, 14, 21, 26, 27] and graph path length, e.g., road distance [3, 4, 12] are common distance metrics. Query objects in the literature [1, 4, 7, 14, 21, 26, 27] can be of two types, namely, a point and line segments. A point-NN query is a conventional NN query [4, 14] (e.g., “find the nearest gas station to my hotel”). A variant to the point-NN query is a closest pair query between two point datasets [1, 7] (e.g., “find the pair of a gas station and a restaurant that has the smallest distance between them.”). Line segment-NN queries can be of two types. One finds the closest facility to all the line segments [21]. The other retrieves the nearest facilities of any point on the line segments [26, 27] (e.g., “find all nearest gas stations on my route from a starting position to a destination”).

In this paper, we present the problem of finding the in-route nearest neighbor (IRNN) for a query object tuple $q = \langle R, c \rangle$, where R is a given route with a destination and c is a current location on the route R [19]. The IRNN problem is searched with the detour distance via a facility instance from the query route R on the way to the destination. As the following example illustrates, the problem can arise naturally in a travel environment. It is an interesting issue in advanced traveler information systems [9, 10, 18, 20, 25] as well as location-based services [5, 16] for commuters with a strong preference for a specific route.

Consider Figure 1, where a set of instances of a facility type (e.g., gas station), $F = \{f_1, f_2, f_3, f_4, f_5\}$, is represented by stars on the road map. A solid line R shows the daily route of a commuter, say from work to home. We call this a query route R . Suppose that the commuter’s vehicle, which originated at position s , is currently at position c . Then, the driver asks where the nearest gas station is. The NN to the current location c is f_2 using the Euclidean distance metric. Next, consider the IRNN problem using a road-distance metric. There are two facility instances, f_4 and f_5 , near the route to destination d . Among new routes to d by way of a facility instance, let one via f_4 be $R1$ and another via f_5 be $R2$. The trip along $R1$ has the shortest distance to d via a facility instance. By contrast, $R2$ deviates less from the original route R . A tourist considering only total travel distance to his/her destination prefers f_4 on the shortest path route $R1$. However, a commuter prefers to continue along his/her preferred route R and chooses f_5 rather than f_4 since the deviation from R is smaller. In this paper, we focus on finding the IRNN such as f_5 .

Four alternative algorithms are presented for processing the IRNN query on road networks. Relevant NN query processing techniques in the literature [1, 7, 14, 22] are extended for solving this problem by incorporating a path computation algorithm [24] for a road-distance metric. We provide comparisons of the different solutions using an experimental framework. The behavior of the solution methods is examined with parameters affecting the performance, i.e., road map size, road map density, facility density, query route length and update rate on the road network.

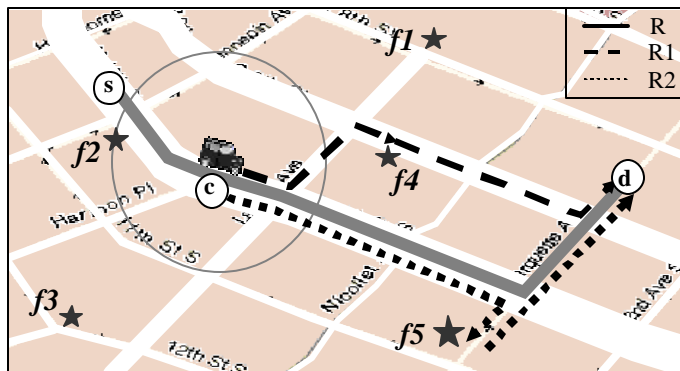


Figure 1. An example of nearest neighbor query

The spatial distance join-based method always shows better performance with fewer path computations compared to the recursive methods, the simple graph-based method and the recursive spatial range query-based method. The precomputed-zone based method outperforms the others under all setting when there is no update on the road map.

The rest of the paper is organized as follows. Section 2 outlines related NN query methods and presents our contributions. Section 3 describes related basic concepts and our problem definition. In Section 4, four alternative IRNN query methods are presented. In Section 5, we evaluate the experimental results of these methods. The conclusion is given in Section 6. This paper focuses on urban street maps where the instances of a facility type are located on streets. For simplicity, we focus on road maps which can be represented as bi-directional graphs.

2. Related work and our contributions

Relevant NN query processing techniques in the literature can be classified into two groups, namely, tree traversal [1, 4, 6, 14, 26, 27] and zone-precomputation [21, 23]. The tree traversal approach using the Euclidean distance metric employs a spatial database structure such as R-tree and searches it in a branch-and-bound manner. There are two types of traversals, a depth-first [14] and a best-first [6]. The zone-precomputation approach uses the pre-computed result of the NN obtained from partitioning a search data space. For instance, closest point problems in the computational geometry are usually solved using Voronoi diagram partition [13]. The Euclidean distance-based NN algorithms have been extended for a road-distance based NN. Recent work [4, 12] has proposed a method that utilizes a spatial tree traversal. In the graph theory and network analysis literature, multiple runs of a single-source all-destination shortest path algorithm [8] are used for a road-distance based zone-computation. Another approach [3] transforms a road network into a high dimensional space in order to utilize computationally simple metrics. However, the existing literature has not addressed the IRNN problem, which is of interest to commuters.

This paper makes three main contributions: First, we define the IRNN query problem. Second, we extend relevant NN query techniques for processing the IRNN query and incorporate a path search algorithm for a road-distance metric. Four alternative solution methods are presented. The first is the simple graph-based approach. It repeats a single-source shortest path finding algorithm by adjusting a path search area with the previous result.

The second and third methods employ a spatial index tree of facility instance points and adopt a tree traversal technique. The former uses recursive spatial range queries for filtering candidate IRNNs. The latter utilizes a spatial distance join technique between two datasets, a set of facility points, and a set of query route data. In the final method, precomputed zone-based, a road network is preprocessed into zones and the result of NN to every node obtained from the zone allocation is stored in the secondary memory. As the third contribution, we compare the four different solutions under an experimental framework. We examine the behavior of the algorithms as the facility density increases, the route length increases, the size and density of road map grows and the rate of update on the road map increases. The results show that the precomputed zone-based method outperforms the other algorithms under all parameter settings of facility density, route length and road map size. However, the method needs more storage space for storing the precomputed results and its performance degrades dramatically with updates on the road map. Our experiments show that our strategy to reduce the expensive path computations to minimize the response time is reasonable. The spatial distance join-based method always shows better performance with fewer path computations compared to the recursive methods, the simple graph-based method and the recursive spatial range query-based method. The overall response time for all methods except the precomputed zone-based method decreases with facility density and increases with road map size and route length.

3. Basic concepts and problem definition

3.1 Basic concepts

Road Networks: A road network $N=N(V, E)$ is modeled as a directed graph $G(V, E)$ which consists of a finite set of vertices V (or nodes) and a set of edges E (or links). In the graph, an intersection on a road map is represented as a vertex and a road segment between two intersections as an edge in the graph. A bi-directional road is represented using two edges. Each edge has its *weight* (or *cost*). We suppose the weight represents the road distance between two neighboring intersections. We also assume that a road network $N=N(V, E)$ is adjusted with a set of point instances of a facility type $F=\{f_1, f_2, \dots, f_m\}$. Let the original road network $N'=N'(V', E')$. The adjusted network with F is $N=N(V, E)$, where $V=V' \cup F$ and E' is subdivided to E if E' contains a facility point f_i

Shortest Path Finding Algorithms: The road distance between two points can be calculated using a shortest path finding algorithm, for which many well established graph-theoretical algorithms are available in the literature [18, 24]. We use Dijkstra's single-source shortest path finding algorithm [24]. Other shortest path finding algorithms [18], however, can be used, and we will explore those in future work.

Distance Functions: Consider a set of instances of a facility type $F= \{f_1, f_2, \dots, f_m\}$. A given route R can be represented by a starting point s and a destination d and a set of consecutive intersections between them, that is, $R=\{r_1, r_2, \dots, r_i\}$ where r_1 is s and r_i is d . We call them the branch points of the route. The current position c is a point on route R . We assume that R is the shortest path from s to d . Other symbols for our distance function are listed in Table1. All of the total travel distance functions from c to d via f_i can be defined as follows:

Symbol	Meaning
$D(x,y)$	The shortest distance from x to y
$D(x,y,z)$	The shortest distance from x to y via z
$D(x,y,z,R)$	The shortest distance from x to z and z to y via nodes on route R
$D_i(x,y,R)$	The shortest distance from x to y via node r_i on route R
$D(x,y,z,R^{max})$	The shortest distance from x to z and z to y via nodes on route R maximizing the use of R .
$D_i(x,y,R^{max})$	The distance from x to node r_i on route R and r_i to y where r_i is a branch point maximizing the use of R .

Table 1. Symbols for distance functions

General total travel distance via f_i :

$$D(c, d, f_i) = \{D(c, f_i) + D(f_i, d)\} \quad (1)$$

Route constrained total travel distance via f_i :

$$D(c, d, f_i, R) = D_j(c, f_i, R) + D_k(f_i, d, R) = \underset{r_j \in R}{\text{Min}} \{D(c, r_j) + D(r_j, f_i)\} + \underset{r_k \in R}{\text{Min}} \{D(f_i, r_k) + D(r_k, d)\} \quad (2)$$

To maximize the reuse of a route on a trip,

$$D(c, d, f_i, R^{max}) = D_j(c, f_i, R^{max}) + D_k(f_i, d, R^{max}) = \underset{r_j \in R}{\text{Min}} \{D(r_j, f_i)\} + D(c, r_j^{min}) + \underset{r_k \in R}{\text{Min}} \{D(f_i, r_k)\} + D(r_k^{min}, d) \quad (3)$$

For undirected network use,

$$D(c, d, f_i, R^{max}) = D_j(c, f_i, R^{max}) + D_j(f_i, d, R^{max}) = 2 * \underset{r_j \in R}{\text{Min}} \{D(r_j, f_i)\} + D(c, r_j^{min}) + D(r_j^{min}, d) \quad (4)$$

From the above (4), total travel distance function, the shortest detour distance via f_i from R is defined as follows:

Shortest detour distance from R via f_i :

$$D_{detour}(c, d, f_i, R^{max}) = D(c, d, f_i, R^{max}) - D(c, d) \quad (5)$$

3.2 Problem definition

The IRNN problem is defined as follows:

Given:

1. A set of instances of a facility type, $F = \{f_1, f_2, \dots, f_m\}$
2. A road network, $N = N(V, E)$
3. A query object $q = \langle R, c \rangle$, where R is a route with a start point s and a destination d , and c is a current location on R

Find: A facility instance f_i

Objective: The detour distance via a facility instance f_i from the route R is shortest such that

$$D_{detour}(c, d, f_i, R^{max}) \leq D_{detour}(c, d, f_j, R^{max}), \quad f_i \in F, f_j \in F - f_i$$

Constraints:

1. A graph path length is used as a distance metric.
2. A set of instances of a facility type F is located on road network N .
3. The cost to locate a query object q on network N is ignored.
4. Computation time and user reaction time is negligible with respect to travel time.

4. IRNN query processing methods

In this section, we discuss four solution methods for solving the IRNN problem: Simple Graph-Based (SGB), Recursive Spatial Range query-based (RSR), Spatial Distance Join-based (SDJ) and PreComputed Zone-based (PCZ).

4.1 Simple graph-based method

An acceptable solution of the IRNN query needs to find the NNs of each route branch point $\{r_1, r_2 \dots, r_t\}$ on a query route R . A naïve solution executes a shortest path search recursively at every branch point and compares their detour distances. When Dijkstra's single-source shortest path finding algorithm is used for road-distance, the facility node first permanently labeled during the path search becomes the NN to the branch point. SGB repeats the path computation at every route branch point by adjusting the path search upper bound with the result of the previous path computation. The computation complexity of SGB is $O(t(|e| + |v| \log |v|))$, where t is the number of branch points on the route, e is the number of edges of the network and v is the total number of nodes of the network, and $O(|e| + |v| \log |v|)$ is the computation complexity of Dijkstra's algorithm[21]. The disadvantage of this solution, however, is that as the number of branch points on the route increases, the more expensive the processing becomes. Figure 2 illustrates the pseudo code of SGB. Our additional codes to Dijkstra's are written in italics.

SGB(R, c)

/ R: query route, s: start node of R, d: destination node of R, c: current location */*

1. *minDetourDist = infinit // minDetourDist is minimum detour distance to R*
2. *for all nodes r of R from s to d*
3. *if r is a past node then pathDist(r) = pathDist(c,r) // pathDist(c,r):road distance from c to r*
4. *else pathDist(r)=0 // pathDist(x):road distance from a path computation source r to x*
5. *insert source r into Heap*
6. *for all nodes v except r*
7. *pathDist(v)= infinit*
8. *While (Heap is not empty)*
9. *get node u with smallest tentative distance in Heap*

10. if $minDetourDist < pathDist(u)$ or u is a facility node
11. if $minDetourDist > pathDist(u)$
12. $minDetourDist = pathDist(u)$
13. $NN = u$
14. continue at next node r
15. for all neighbor nodes v of u
16. if $pathDist(u) + weight(u,v) < pathDist(v)$
17. if $pathDist(v) == \infty$ insert neighbor node v in Heap with $pathDist(u) + weight(u,v)$
18. else set distance of neighbor node v in Heap to $pathDist(u) + weight(u,v)$

Figure 2. Algorithm of the simple graph-based method

4.2 Recursive spatial range query-based method

The second solution method is the recursive spatial range query-based. The motivation for this method is from the following property. Let p and q be two points on the road network, $EucDist(p,q)$ be the Euclidean distance and $pathDist(p,q)$ be the path distance between p and q .

Property 1. $EucDist(p,q) \leq pathDist(p,q)$

Although the Euclidean distance does not estimate the exact road-distance between two points, it can provide the lower bound of the road-distance. If there is no facility within the Euclidean distance search bound, it is clear that no road-distance based facility exists in the search area. In this case, a new path computation for finding IRNN is unnecessary. The cost of Euclidean distance-based query processing is relatively lower than that of a path finding computation and the approach is well studied in the spatial database literature [14, 15, 17]. In our work, it is reasonable that we consider a half of a detour distance via a facility point since the road map is supposed to use a bi-directional road network. We utilize a spatial range query to use Property 1. It is used to check if candidate facilities exist in the search bound. We assume that a set of facility points is stored in a spatial index tree.

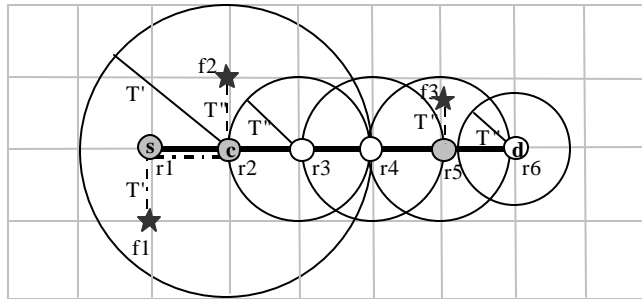


Figure 3. Recursive spatial range query-based method

The RSR method works as follows. At the first route point, the start position of a given route, launch a shortest path search algorithm and calculate the detour distance to its NN. The result is set to the minimum detour distance so far, T , which will be used as the search bound of the next route branch point. A spatial range query whose search radius is T is executed at the next branch point. If there exists a facility within the search area, calculate a path length to the candidate facility from the route node. If the new detour distance is less than T , T is updated with the new value. The procedure continues by adjusting the search bound of each branch point until the last route point, which is the destination.

Consider the example given in Figure 3. In the figure, let the grids be a road network and the grid size be 1. A query route $R=\{r_1, r_2, r_3, r_4, r_5, r_6\}$ is represented by a solid line. A current position c is located at r_2 . A set of facility points $F=\{f_1, f_2, f_3\}$ is represented by stars. The detour distance to each facility point is depicted by a dash line. The query processing step can be traced as follows. First, we execute a path search at the start position r_1 . We know that the NN to r_1 is f_1 and that the detour distance is 2 (T' is described in the figure) since the way to f_1 is opposite the current direction to the destination. The minimum detour distance T is set to T' . At the next route node r_2 , execute a spatial range query whose radius is T' and examine whether a facility exists within the Euclidean search area. If there is a facility in the area, e.g., f_2 , the detour length to the facility is calculated. In this example, the detour length is 1 (T'' in figure). T is updated with smaller T'' . The same procedure continues until destination r_6 . In this example, path computations at r_3 and r_4 can be skipped since there is no candidate facility within their search bound. The route nodes that need a new path search are colored gray in Figure 3. This method has fewer path computations than SGB. In this example, IRNN is f_3 because its detour distance from R is smallest. Figure 4 shows the pseudo code of RSR.

RSR(R, c)

/ R: query route, s: start node of R, d: destination node of R, c: current location */*

1. $(minDetourDist, NN) \leftarrow PathSearch(s)$ *//PathSearch(s): path computation from s to find a NN facility*
 2. *for all nodes r on R except s to d*
 3. $EucSearchDist = minDetourDist$;
 4. *if(ObjectExists(r, EucSearchDist))* *//Check if a facility exists within a EucSearchDist bound from r*
 5. $(detourDist, candiNN) \leftarrow PathSearch(r)$
 6. *if r is a past node then detourDist = detourDist + pathDist(c, r)*
 7. *if detourDist < minDetourDist*
 8. $(minDetourDist, NN) \leftarrow (detourDist, candiNN)$
-

Figure 4. Algorithm of the recursive spatial range query-based method

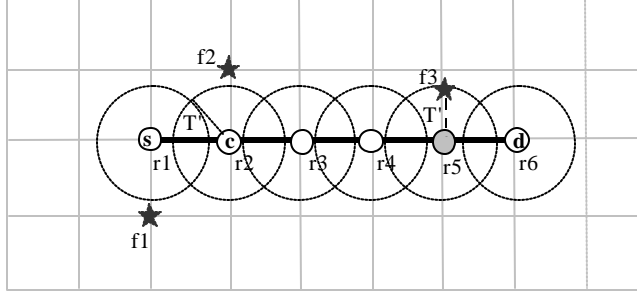


Figure 5. Spatial distance join-based method

4.3 Spatial distance join-based method

The previous RSR method launches a new spatial range search at every route node even though the number of path computations for processing the IRNN query decreases compared with SGB. One possibility for our progressive solution finds IRNN candidates located as close as possible. If we tighten the search bound at the beginning of query processing, we can reduce the number of path computations even more. To make the idea concrete, we adopt a spatial distance join operation whose output is ordered by the distance between two datasets [1, 7]. The distance join operation assumes that two data sets are indexed by each spatial index tree and the operation searches two trees in a branch-and-bound manner. In our case, the two data sets are a set of point instances of a facility type and a set of node points of a given query route. A spatial index tree for the latter is built during the IRNN query processing. For the spatial distance join operation, we adopt the functionality of the heap algorithm by [1].

The SDJ method works as follows. First, set the initial minimum detour distance T to infinite and start the distance join operation between the two data sets. According to the tree traversal method between two spatial trees, a heap is occupied with the following elements, i.e., $\langle r_i, f_j, dist \rangle$, $\langle r_i, MBR_{f_j}, dist \rangle$, $\langle MBR_{r_i}, f_j, dist \rangle$ or $\langle MBR_{r_i}, MBR_{f_j}, dist \rangle$ where r_i is a route node object, f_j is a facility point object, MBR_{r_i} is the minimum bounding rectangle (MBR) of route nodes and MBR_{f_j} is the MBR of facility points, and $dist$ is the Euclidean distance between two objects, one object and one MBR or two MBRs. The elements are ordered by $dist$ in the heap. If one element from the top of the heap consists of two objects $\langle r_i, f_j \rangle$ and its $dist$ is less than the current minimum detour distance T , then compute the detour distance from r_i to f_j . If the result is smaller than T , it is set to T and f_j becomes the IRNN so far. The procedure is repeated until the $dist$ of the next element from the top of the heap is greater than T . Figure 5, when compared with Figure 3, reveals the differences between the SDJ method and the RSR method. As can be seen in Figure 5, we can find the Euclidean distance closest pair (r_5, f_3) using the incremental distance join algorithm. From a path computation at r_5 , we get the tightening bound of the next search. This example shows only one path search at r_5 colored gray. Compared with the previous recursive solutions, SGB and RSR, SDJ is expected to be more efficient with fewer path computations. Figure 6 illustrates the pseudo code of SDJ.

SDJ(R, c)

/ R: query route, s: start node of R, d: destination node of R, c: current location */*

*/*R_f: spatial tree of facility points, R_r:spatial tree of route nodes */*

1. *minDetourDist = infinit*
2. *Q ← NewPriorityQueue()*
3. *EnQueue(Q, 0, <R_r.rootnode, R_r.rootnode>)*
4. *while not IsEmpty(Q)*
5. *Elem ← Dequeue(Q)*
6. *if both items <itemR, itemF> in Elem are objects<r,f>*
7. *(detourDist, candiNN) ← PathSearch(r)*
8. *if r is a past node then detourDist = detourDist +pathDist(c, r)*
9. *if IsEmpty(Q) or detourDist <= Front(Q).dist*
10. *(minDetourDist, NN) ← (detourDist, candiNN)*
11. *exit*
12. *else EnQueue(Q, detourDist, <r, f>)*
13. *else if itemR in Elem is MBR_r*
14. *if MBR_r is a leaf node*
15. *for all objects r of MBR_r*
16. *EnQueue(Q, dist(r, itemF), < r, itemF>)*
17. *else for all child nodes childMBR_r of MBR_r*
18. *EnQueue(Q, dist(childMBR_r, itemF), < childMBR_r, itemF >)*
19. *else //if itemF in Elem is MBR_f*
20. *if itemF in Elem is a leaf node*
21. *for all objects f of MBR_f*
22. *EnQueue(Q, dist(itemR, f), < itemR, f>)*
23. *else for all child nodes childMBR_f of MBR_f*
24. *EnQueue(Q, dist(itemR, childMBR_f), < itemR, childMBR_f >)*

Figure 6. Algorithm of the spatial distance join-based method

4.4 Precomputed zone-based method

Another important approach in the NN query literature uses pre-computed results. In this section, we present a method based on the pre-computed zones of a road network. The preprocessing step, given a set of instances of a facility type F located on the road network, computes a service zone s_i of each facility instance f_i . The service zone satisfies the following property.

Property 2. For every node v_j in a service zone s_i on the road network,

$$pathDist(v_j, f_i) \leq pathDist(v_j, f_k), \text{ where } f_i \text{ is the facility in service zone } s_i \text{ and } f_k \text{ not.}$$

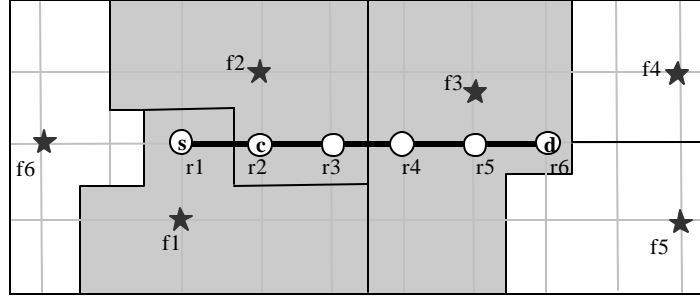


Figure 7. Precomputed zone-based method

The information of the service facility, $\langle v_j, f_i, D(v_j, f_i) \rangle$ for every node v_j , is stored in a traditional index structure, e.g., a B+tree, with a key v_j . For allocating the road-distance based service zones, we use the network partition algorithm by [22]. It is a slight variation of Dijkstra's algorithm, a multiple source shortest path finding algorithm. It has a running time $O(e|m+v|\log|v|)$, where m is the number of instances of a facility type, e is the number of edges of the network, and v is the number of nodes. Figure 7 shows allocated service zones on the road network. The service zones intersected by a given query route are gray colored and their facilities become candidate IRNNs. Figure 8 illustrates the pseudo code of the network allocation.

The precomputed service zones-based method (PCZ) works as follows. For every route branch point, find the service zones in which it is located and look up the precomputed distance from the route point to its service facility point. Its detour distance is adjusted with the current location on the route. The facility having the smallest detour distance becomes the IRNN. Figure 9 shows the pseudo code of PCZ.

```

Network_Partition(G,F)
/* G: Network Graph, F: facilities */
1. for facility nodes f in F
2.  pathDist(f)=0, serviceFacility(f)=f
3.  insert f into Heap
4. for all nodes v except F
5.  pathDist(v)=inifinit, serviceFacility(f)=Null
6. While (Heap is not empty)
7.  get node u with smallest tentative distance in Heap
8.  for all neighbor nodes v of u
9.    if pathDist(u) + weight(u,v) < pathDist(v)
10.     if pathDist(v) == inifinit
11.       insert neighbor node v in Heap with distance pathDist(u)+weight(u,v)
12.     else set distance of neighbor node v in Heap to pathDist(u)+weight(u,v)
13.     serviceFacility(v)=serviceFacility(u)

```

Figure 8. Algorithm of network allocation of service zones

PCZ(R, c)

/ R: query route, s: start node of R, d: destination node of R, c: current location */*

1. *minDetourDist=infin*
 2. *for all nodes r on R from s to d*
 3. *(detourDist, candiNN) ← findServiceFacility(r)*
 4. *if r is a past node then detourDist = detourDist + pathDist(c,r)*
 5. *if detourDist < minDetourDist*
 6. *(minDetourDist, NN) ← (detourDist, candiNN)*
-

Figure 9. Algorithm of the precomputed zone-based method

5. Experimental evaluation

We performed experimental evaluations to compare the four different solutions using four real road network datasets of different size and different density. Our purpose was to answer five individual questions and one overall question.

1. How does road map size affect the response time and the storage needs of all methods?
2. What is the effect of facility density on the performance of all methods?
3. What is the effect of route length on the performance of all methods?
4. How does road map density affect the response time?
5. How do updates on road network affect the performance of all methods?

For an overall question,

6. What is the choice of the different methods?

5.1 Experiment design

5.1.1 Data and query sets

We used a publicly available real digital road map. The dataset covers seven counties in Minnesota: Anoka, Carver, Dakota, Hennepin, Ramsey, Scott and Washington. Ramsey has dense urban and first ring suburbs. Hennepin is a mixed area of dense urban, first and second ring suburbs and large undeveloped areas. Other areas are second ring suburbs and large undeveloped area. We converted the map into four bi-directional road networks. The first network dataset, RAMSEY, contains 14,412(15K) nodes. The second dataset, HENNEPIN, contains 35,869(36K) nodes. The third, D&H, covers Dakota and Hennepin counties and contains 75,739(76K) nodes. The final dataset, ALL, covers all seven counties with a total of 190,354(191K) nodes. We used the sets of points of a facility type randomly generated from the above road network datasets whose densities were different, e.g., 0.001%, 0.5%, 1%, 5%, 10% etc. In this paper, facility density is defined as the number of facility points over the number of nodes of a road network. A query route was also randomly selected using the road network dataset.

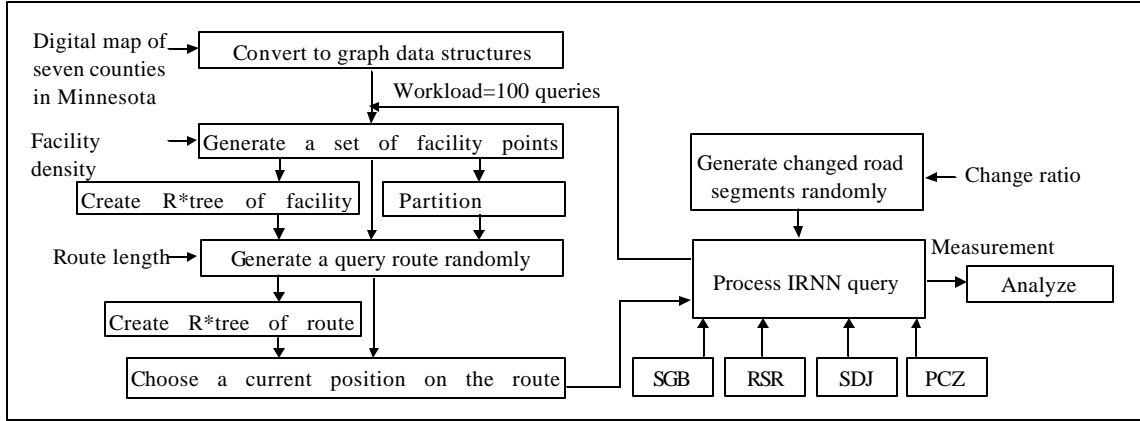


Figure 10. Experiment layout

In this paper, a route length is defined by the number of nodes that a route consists of. We used different route lengths, i.e., 30, 50, 100, 200, 500 and 1000. A current position was randomly selected on the route.

5.1.2 Experiment layout

Figure 10 shows the overall experiment layout. Performance is measured by executing workloads, each consisting of 100 queries generated as follows: i) A new set of facility points are randomly generated with a given density per each query. ii) A new query route per query is randomly generated. A node is randomly chosen as a starting position. Then consecutive nodes are selected randomly with the number of nodes equaling a given route length. iii) A current position is randomly selected on the generated route. iv) For RSR and SDJ, each R* tree for facility points and a query route is created. v) For PCZ, network allocation is computed and the precomputed NN result is saved in a B+tree. vi) To examine the effect of updates on the road network, changed road segments are randomly chosen with the given update ratio. The experiments were performed on a Sun Ultra SPARC Iie workstation, which has 512 M Bytes of main memory. The experiments were written in C/C++.

5.1.3 Experiment Configuration

Table 2 shows the overall configuration of the experiment. The candidates of the experiment are the proposed four methods.

Exp id	Variable parameters	Dependent variables	Fixed parameters
1	road map size	cpu time, storage size	number of facilities=200, route length=100
2	density, road map	cpu time, path search area	route length=200
3	route length, road map	cpu time, number of path computations	facility density=1%
4	road map, density	cpu time	route length=200
5	change rate of road map	cpu time	route length=200,
6	ratio of changed edges/route length	cpu time	facility density=1%

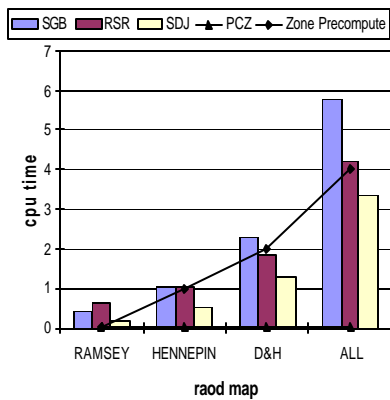
Table 2. Experiment configuration

5.2 Experiment results

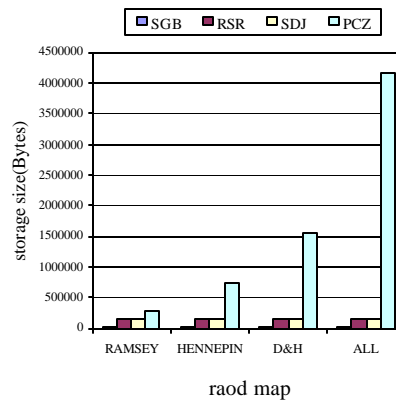
5.2.1 How does road map size affect the response time and the storage needs of all methods?

In the first experiment, we compared the effect of road map size on the performance and the storage needs of all four methods. Figure 11(a) illustrates the response time as a function of road map size where ALL(191K) > D&H(76K) > HENNEPINE(36K) > RAMSEY(15K). The query route length is fixed at 100 and the number of facility points at 200 in all road map datasets. Overall response time in all methods except PCZ increases with road map size. This happens because, as road map size becomes larger, the facility density becomes sparser with a fixed number of facilities and the path search area can grow, increasing the path computation time. By contrast, PCZ shows a constant time since it looks up the materialized result and has the simple calculation of detour distance from the current location. Among on-line computation methods, SDJ shows better performance than the recursive methods, SGB and RSR. Figure 11(a) also shows the precomputed cost of PCZ. The service zone computation cost depends on the size of road map.

Next, we compared the storage sizes of all methods. As shown in Figure 11(b), it is clear that the storage size of PCZ depends on the road map size since it stores the precomputed NN result of all nodes of the road network. By contrast, the current SGB method does not require storage. RSR and SDJ operate independently of the road map size. Their storage needs depend on the size of the spatial index of facility points and SDJ has an additional space for the spatial index of route nodes.



(a) Performance vs. road map



(b) Storage size vs. road map

Figure 11. Effect of different road map sizes (Number of facilities=200, route length =100)

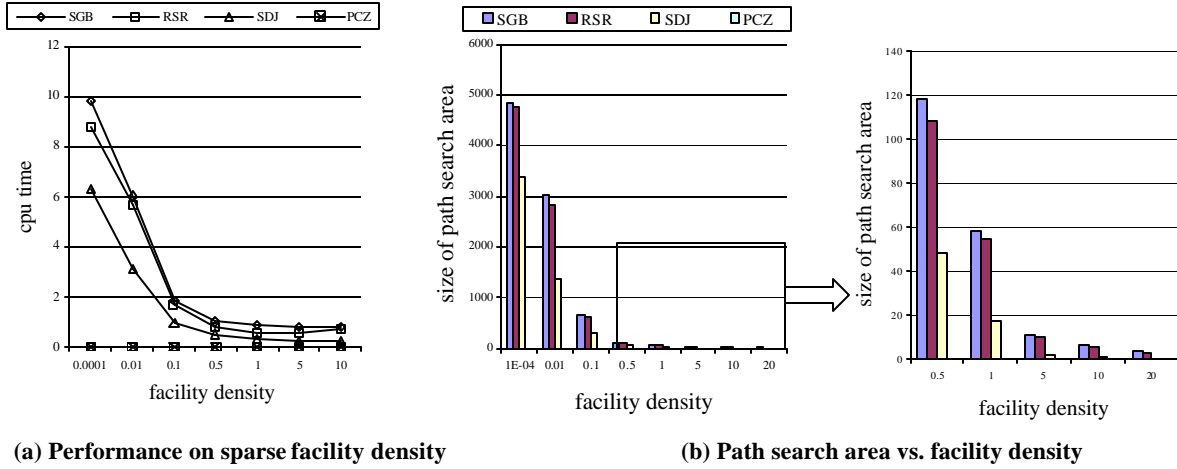


Figure 12. Effect of facility density (route length =200, data set=RAMSEY)

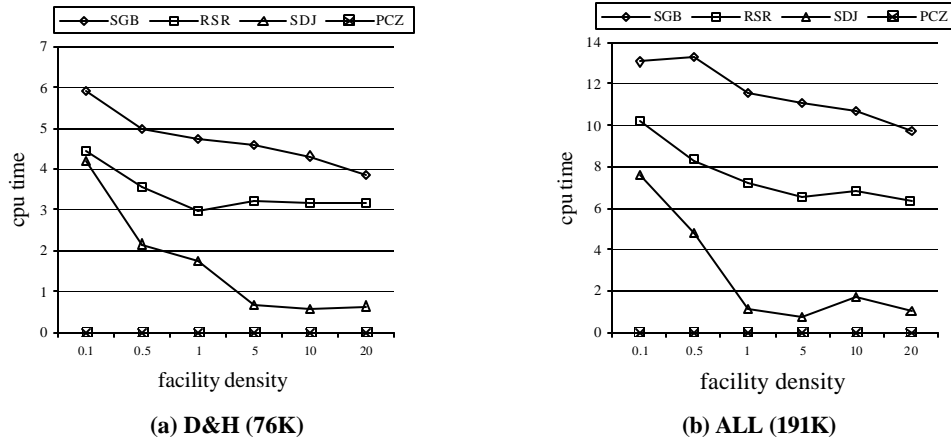


Figure 13. Performance vs. facility density in different road networks (route length =200)

5.2.2 What is the effect of facility density on the performance?

In the second experiment, we examined the impact on the performance of the methods as the facility density varies. First, we examined it with very sparse densities, e.g., 0.0001, 0.01 etc. Figure 12(a) shows that the response time is long when the density is very sparse, but it drops sharply and remains mostly constant after a threshold density, e.g., 0.5. The reason is the sparser the density, the bigger the expected search area becomes. Figure 12(b) illustrates this well. Figure 13 presents the response time as facility density on different size data sets, D&H and ALL respectively. The route length is fixed to 200. Figure 13 shows that the response time for all methods except PCZ decreases gradually with increase in the facility density. The curves for different size road networks (a) and (b) are similar. The reason is that the possibility that a facility exists near a query route is higher as a facility density is greater, producing a smaller search bound. The facility density has an overall positive impact on the performance. By contrast, PCZ receives little effect from changes in the facility density.

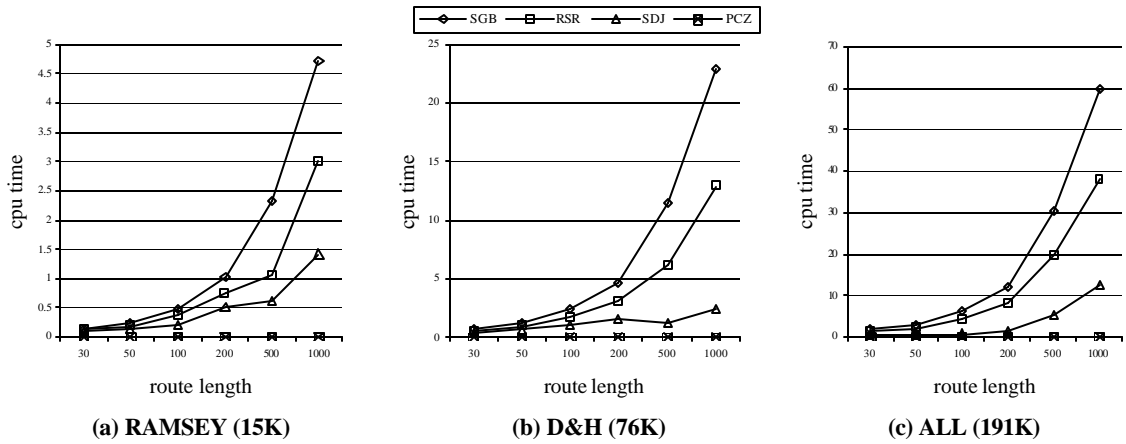


Figure 14. Performance vs. route length (facility density =1%)

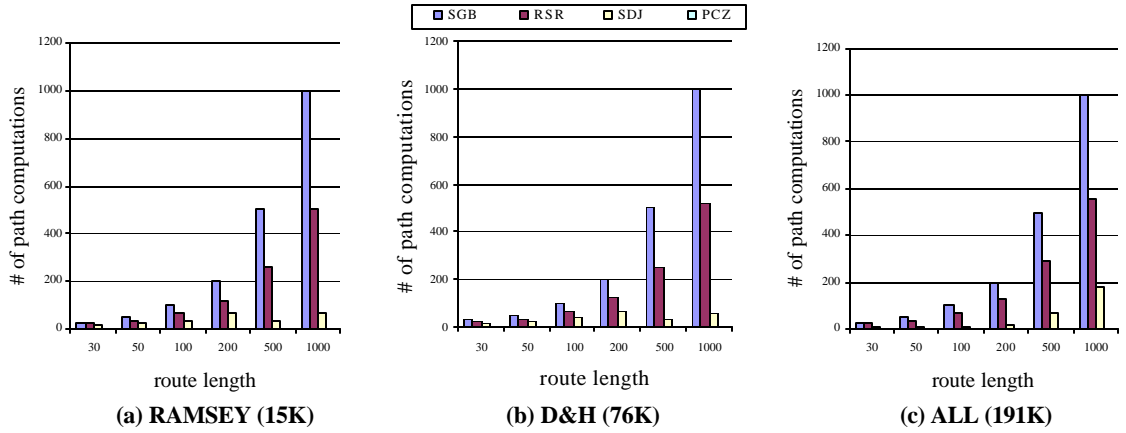


Figure 15. Number of path computations vs. route length (facility density =1 %)

5.2.3 What is the effect of route length on the performance?

In the third experiment, we examined the impact on the performance of all methods as the route length varies. Figure 14 illustrates the response time as a route length for different size data sets, RAMSEY, D&H and ALL respectively. The facility density was fixed at 1%. As shown in Figure 14, the overall response time of all methods except PCZ increases with increases in the route length. In the recursive methods, SGB and RSR, the longer the route length, the worse the performance becomes. SDJ clearly shows better performance than the recursive methods. PCZ shows very little effect from route length. Figures 14 (a), (b) and (c) have similar curves. We also examined the number of path computations each method executed. The number of path computations in SGB is the same as the route length. RSR shows roughly half the number of path computations as SGB. It can also be seen that SDJ, a non recursive method, uses fewer path computations and receives a very little effect of route length.

5.2.4 How does road map density affect the response time?

In the fourth experiment, we examined the effect of road map density on the performance. First, we examined it with data sets of different road density. RAMSEY has the greatest density among the experiment road data sets. HENNEPIN is the second most dense road data set. We set a high facility density value, 20%. In Figure 16(a), we can see that, unlike the result of the previous experiment, RSR does not perform better than SGB in RAMSEY and HENNEPIN road maps. The reason is that when the road map and the facility points become denser, the false hit ratio of the Euclidean candidates in RSR increases. Figure 16(b) shows that the higher the facility density becomes, the more the RSR response time increases.

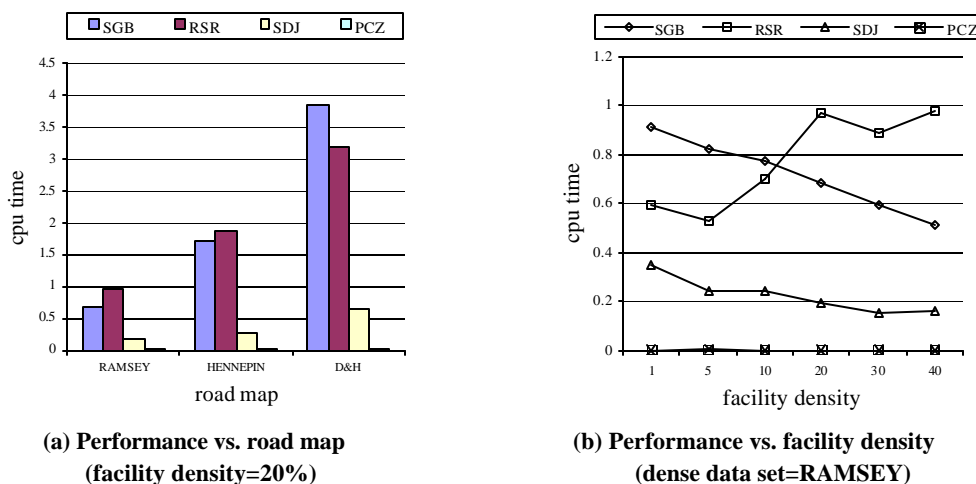
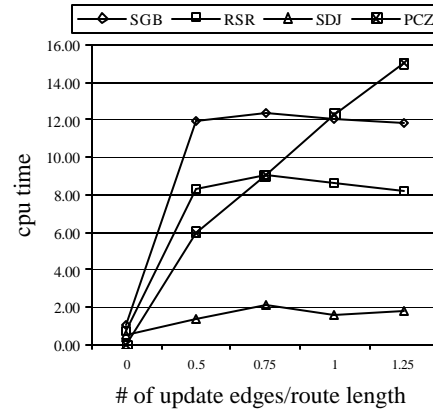
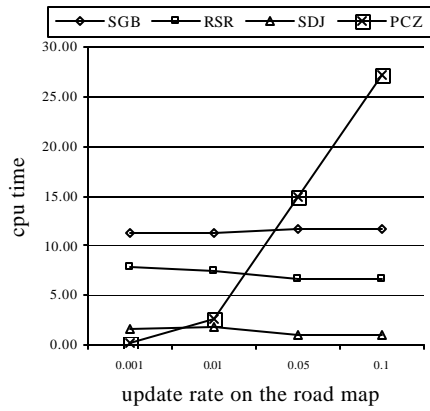


Figure 16. Effect of road map density (route length =200)

5.2.5 What is the effect of updates on the road map?

In the fifth experiment, we examined the impact of updates in the road network on the performance of all methods. Figure 17(a) shows the response time as the percentage of update edges over the total number of edges of the road network. The other parameters are fixed: facility density=1%, route length=200. As can be seen, the overall response time for all methods except PCZ remains constant with increases in update edges. This is likely due to the fact that the update load of the road network itself is negligible due to the relatively small size of the experiment data set and the resulting inexpensive memory operations. However, PCZ response time increases as the update ratio increases because of the recomputation of allocation.



(a) Performance vs. update rate on the road map (b) Performance vs. # of changes edges/route length

Figure 17. The effect of updates on road network
 (data set=ALL, density of facilities=1%, route length =200)

5.2.6 What is the choice of different methods?

In the final experiment, first, we examined the effect of the dominance zone on the performance of the algorithm in the updates of the road methods. We compared PCZ with SGB, which is expected to be most expensive in the total cost. Figure 17(b) shows that the response time of PCZ moves in the opposite direction with SGB when the ratio of the number of changed edges and route length shows almost 1. The breakpoint shows a case that the number of path computations in SGB is same with the number of path search launches for partially recomputing service zones for PCZ. Second, the choice among the on-line computation methods, PCZ, RSR and SDJ depends on the facility density and road map density as above experiments. SDJ shows better performance under all setting.

6. Conclusion

We studied the in-route nearest neighbor query for a given route and presented four different solution methods. We focused on comparisons among them using an experimental framework. The experimental results show that the precomputed zone-based method always outperforms the other methods when there are no updates on the road map. However, the response time of the precomputed zone-based method dramatically increases with increases in the update ratio. It also needs more storage space for storing the precomputed results. In the other methods, the overall response time decreases with increases in the facility density and increases with increases in the route length and the size of the road map. The spatial distance join-based method outperforms the recursive methods with fewer numbers of path computations. The experiment shows that our strategy to reduce the number of path computations to minimize the response time is reasonable. However, in the dense road map area, the performance of the recursive spatial range query-based method declines due to the increase of the false hit ratio of the Euclidean candidates and the simple graph-based method shows better response time than the spatial range query-based

method. In future work, we plan to extend our approach to continuous IRNN query problems and explore k-NN query on road networks for moving objects.

Acknowledgements

We thank Spatial database group members for their valuable feedback and we would also like to express our thanks to Kim Koffolt for her timely and detailed feedback to help improve the readability of this paper.

References

- [1] A. Corral, Y. Manolopoulos, Y. Theodoridis, M. Vassilakopoulos, Closes Pair Queries in Spatial Databases, Proc. ACM Conference on Management of Data (SIGMOD), 2000.
- [2] C.Jensen, J.Kolar, T.Pedersen, I.Timko, Nearest Neighbor Queries in Road Networks, Proc. ACM International Symposium on Advances in Geographic Information Systems(ACM GIS), 2003.
- [3] C. Shahabi, M. R. Kolahdouzan, M Sharifzadeh, A Road Network Embedding Technique for K-Nearest Neighbor Search in Moving Object Databases, Proc. International Symposium on Spatial and Temporal Databases(SSTD), 2001.
- [4] D. Papadias, J. Zhang, N.Mamoulis, Y.Tao, Query Processing in Spatial Network Databases, Proc. Very Large Data Bases Conference (VLDB), 2003
- [5] GITA and OGC's Emerging Technology Summit Series -Location-Based Services.
<http://www.openls.org/dvd1/ets1/index.htm>.
- [6] G. Hjaltason, H. Samet, Distance Browsing in Spatial Databases, Proc. ACM Transactions on Database Systems (TODS), 1999.
- [7] G. Hjaltason, H. Samet, Incremental Distance Join Algorithms for Spatial Databases, Proc. ACM Conference on Management of Data (SIGMOD), 1998.
- [8] H. Edelsbrunner, Algorithms in Computational Geometry, EATCS Monographs on Theoretical Computer Science, 1987.
- [9] J. H. Rillings and R. J. Betsold. Advanced Driver Information Systems. IEEE Trans. on Vehicular Technology, 1991.
- [10] J. L. Wright, R. Starr, S.Gargaro, GENESIS-Information on the Move, In Proc. of Annual IVHS American Conference, 1993.
- [11] J. Zhang, N. Mamoulis, D. Papadias, Y. Tao, All-Nearest-Neighbors Queries in Spatial Databases, Proc. IEEE Conf. on Scientific and Statistical Database Management (SSDBM), 2004.
- [12] J. Feng, T. Watanbe, Fast Search of Nearest Target Object in Urban District Road Networks, PYIWIT, 2002.

- [13] M. F. Worboys, GIS: A Computing Perspective, Taylor & Francis, 1995.
- [14] N. Roussopoulos, S. Kelleym, F. Vincent. Nearest Neighbor Queries, Proc. ACM Conference on Management of Data (SIGMOD), 1995.
- [15] P.Rigaux, M.School, A.Voisard, Spatial Databases with Application to GIS, Morgan Kaufmann, 2002.
- [16] S. Shekhar, R. R. Vatsavai, X. Ma, J. S.Yoo, Navigation Systems: A Spatial Database Perspective. as Chapter 3 in Location Based Services, Agnes Voisard and Jochen Schiller (Eds.), Morgan Kaufmann, New York, May 2004.
- [17] S. Shekhar, S.Chawla, Spatial Databases: A Tour, Prentice Hall, 2003.
- [18] S. Shekhar, M. Coyle, A. Kohli, Path Computation Algorithms for Advanced Traveller Information Systems, IEEE Computer Society, 1993.
- [19] S. Shekhar and J. S. Yoo, Processing In-Route Nearest Neighbor Queries: A Comparison of Alternative Approaches, ACM GIS, 2003
- [20] S. Shekhar, A. Fetterer, B. Goyal, Materialization Trade-Offs in Hierarchical Shortest Path Algorithms. Proc. Intl. Symp. on Large Spatial Databases, 1997
- [21] S. Bespamyatnikh, J. Snoeyink, Queries with Segments in Voronoi Diagrams. SODA, 1999.
- [22] S. Hakimi, M. Labbe, and E. Schmeichel, The Voronoi Partition of a Network and its Implications Location Theory ORSA, 1992.
- [23] S. Berchtold, B. Ertl, D. Keim, H.Kriegel, and T.Seidl. Fast nearest neighbor search in high-dimensional space. Proc. International Conference on Data Engineering (ICDE), 1998.
- [24] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, Second Edition, MIT Press, 2001.
- [25] W. C. Collier and R. J. Weiland. Smart Cars, Smart Highways, IEEE Spectrum, 1994.
- [26] Y. Tao, D. Papdias, Q. Shen, Continuous Nearest Neighbor Search, Proc. Very Large Data Bases Conference (VLDB), 2002.
- [27] Z. Song, N. Roussopoulos, K-Nearest Neighbor Search for Moving Query Point, Proc. International Symposium on Spatial and Temporal Databases(SSTD), 2001.