Processing In-Route Nearest Neighbor Queries: A Comparison of Alternative Approaches

Shashi Shekhar
Department of Computer Science and Engineering, University of Minnesota, 200 Union ST SE 4-192, Minneapolis MN 55414
(612) 624-8307
shekhar@cs.umn.edu

Jin Soung Yoo
Department of Computer Science and Engineering, University of Minnesota, 200 Union ST SE 4-192, Minneapolis MN 55414
(612) 626-7703
jyoo@cs.umn.edu

ABSTRACT
Nearest neighbor query is one of the most important operations in spatial databases and their application domains, e.g., location-based services, advanced traveler information systems, etc. 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. Several experiments using real road map datasets are conducted to examine the behavior of the solutions in terms of three parameters affecting the performance. Our experiments show that 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 decreases with increase in the facility density. The precomputed zone-based method shows the most efficiency when there are no updates on the road map.

Categories and Subject Descriptors
H.3 [**Information storage and retrieval**]: Information Search and Retrieval – *Search process.*

General Terms: Algorithms.

Keywords
Nearest neighborhood query, route, road network, location-based services, advanced traveler information systems.

1. INTRODUCTION
A very important query in spatial database systems and geographic information systems is the nearest neighbor (NN) search [11, 14]. 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 [14].

In the nearest neighbor literature, the *Minkowski* metrics, e.g., Euclidean distance [4, 12, 18, 23, 24] and graph path length, e.g., road distance [2, 10] are common distance metrics. Query objects in the literature [1, 4, 5, 12, 18, 23, 24] can be of two types, namely, a point and line segments. Point-NN query is a conventional NN query [4, 12] (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, 5] (e.g., "find the pair of a gas station and a restaurant that has the smallest distance between them."). Otherwise, line segments-NN query can be of two cases. One finds the closest facility to all the line segments [18]. The other retrieves the nearest facilities of every point on the line segments [23, 24] (e.g., "find all nearest gas stations during my route from a starting position to a destination").

GIS'03, November 7-8, 2003, New Orleans, Louisiana, USA.

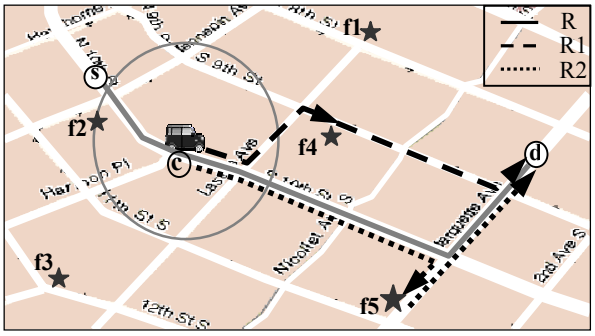


Figure 1. An example query

In this paper, we present the problem of finding the in-route nearest neighbor (IRNN) for a query object tuple q=<R, c>, where R is a given route with a destination d and c is a current location on the route R. The IRNN problem is searched with the detour distance via a facility instance from the query route R on the way to the destination d. As the following example illustrates, the problem can arise naturally in a travel environment. It is an interesting issue in advanced traveler information systems [7, 8, 15, 16, 17, 22] as well as location-based services [3, 13] for commuters with the 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 came from the starting 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 the 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. Otherwise, *R2* deviates less from the original route *R*. A tourist only considering total travel distance to its destination prefers f_4 on the shortest path route *R1*. However, a commuter wants to continue along his/her preferred route and chooses f_5 rather than f_4 since its deviation from *R* is smaller. In this paper, we focus on finding IRNN such as f_5.

This paper presents four alterative algorithms for processing the IRNN query on road networks. Relevant NN query processing techniques in the literature are extended for solving the problem by incorporating a path computation algorithm for a road-distance metric. We provide comparisons between the different solutions using an experimental framework. The behavior of the solution methods is examined with parameters affecting the performance, e.g., road map size, facility density and query route length. The experiment result shows the precomputed-zone based method always outperforms the others under all setting.

The rest of this paper is organized as follows. Section 2 outlines related NN query methods and presents our contributions. Section 3 describes related basic concept 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. Continuous NN queries and updates to road maps are beyond the scope of the paper.

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, 12, 23, 24] and zone-precomputation [18, 20].

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 traversal manner, a depth-first [12] and a best-first [4]. 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 diagrams partition [11].

The Euclidean distance based NN algorithms have been extended for a road-distance based NN. Feng et al. [10] propose a method that intermixes a tree traversal and a path search. In the graph theory and network analysis literature, multiple runs of a single-source all-destination shortest path algorithm [6] are used for a road-distance based zone-computation. Another approach [2] 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.

The contributions of this paper are: first, we define the IRNN query problem. Second, we extend relevant NN query methods 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 a 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 instances of a facility type and adopt a best-first 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 a query route data. In the final method, precomputed zone-based, a road network is preprocessed to zones and the result of NN to every node 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 and the size of road map grows. The experiment result shows that the precomputed zone-based method always outperforms the other algorithms under all parameter setting. However, it needs more storage space for storing the precomputed results than the others. The overall response time in all methods but the precomputed zone-based method decreases with a facility density and increases with a road map size and a route length. The spatial distance join based method always shows better performance compared to the recursive methods with less path computations.

3. BASIC CONCEPT AND PROBLEM DEFINITION

3.1 Basic Concept

***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*). 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. In this paper, we assume that a road network $N=N(V, E)$ is adjusted with a set of point instances of a facility type $F= \{f_1, f_2, \ldots, f_m\}$. 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-theoretic algorithms are available in the literature [11, 15]. We use Dijkstra's single-source shortest path finding algorithm [21]. Other shortest path finding algorithms [15], however, can be used. We will explore those in the future work.

***Distance Functions*:** Consider a set of instances of a facility type $F= \{f_1, f_2, \ldots, f_m\}$. A given route R can be represented by a starting point s and a destination d and a set of consecutive intersections between them, $R=\{r_1, r_2, \ldots, r_t\}$ where r_1 is s and r_t is d. We call them the branch points of the route. The current position c is a point on the route R.

The total travel distance functions from c to d via f_i can be defined as follows:

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:

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

If maximize the reuse of a route on a trip,

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

If use undirected network,

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

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, $F=\{f_1, f_2, \ldots, f_m\}$
2. A road network, $N=N(V, E)$
3. A query object $q=<R, c>$,
   where R is a route with a starting position s
   and a destination d, and c is a current location on R

***Find*:** A facility instance f_i

***Objectivity*:**

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

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

***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 the road network N.
3. The cost to locate a query object q on the 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 a correct NN of every branch position $\{r_1, r_2 \ldots, r_t\}$ of a query route R. The naïve solution is the method to execute a shortest path search recursively at every branch point and compare their detour distances. When Dijkstra's single-source shortest path finding algorithm is used for a path computation, the facility node first permanently labeled during the path search becomes the NN to the branch point. The computation complexity of the simple graph-based method 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 number of nodes of it since the computation complexity of Dijkstra's algorithm is $O(|e| + |v|log|v|)$ if F-heap is employed [21] and it is launched t times. One heuristic of this method repeats the path computation by adjusting the path search area using the previous result. The disadvantage of this solution, however, is that the more the number of branch points on the route increases, the more expensive the processing is.

4.2 Recursive Spatial Range Query-based Method

The motivation of the second solution method is from the following property.

Property 1. The Euclidean distance between two points p and q is less than equal to the road-distance between them.

Although the Euclidean distance does not estimate the exact road-distance between two points, the distance 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 cheaper than that of a path finding computation and the methods for it are well studied in the spatial database literature [14]. In our work, it is reasonable that we consider a half of detour distance via a facility point since it is supposed to use bi-directional road network. We utilize a spatial range query to use property 1. It is used to check if

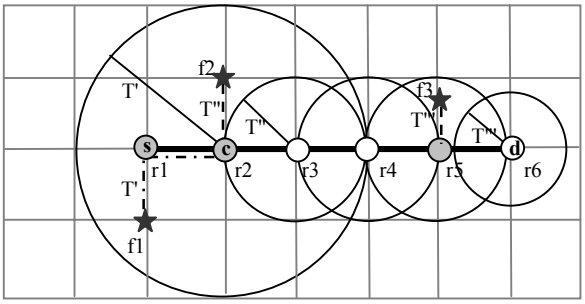


Figure 2. Recursive spatial range query-based method

candidate facilities exist within the search area. We assume that a set of facility points is stored in a spatial index tree.

Our method works as follows. At the first branch 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 branch point. At the next branch point, a spatial range query whose search radius is T is executed. If there exist a facility within the search area, calculate a path length to the candidate facility from the branch point. If the new detour distance is less than T, T is update with the new value. The procedure continues until the last branch point, the destination with adjusting its search bound.

Consider the example given in Figure 2. 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 dashed line. Now let's trace the IRNN query processing step. First, we execute a path search at the start position r_1. We know that NN to r_1 is f_1 and its detour distance is 2 since the way to f_1 is against the current direction to the destination. The minimum detour distance T is set to 2, T'. At the next branch point r_2, a spatial range query whose radius is T' is executed. There is a facility within the search area. Thus, the detour length to f_2 is calculated from a new path search at r_2. Let the result, 1 be T''. T is updated with smaller T''. The same procedure continues until the destination r_6. In this example, path computations at r_3 and r_4 are skipped since there is no candidate facility within their search bound. The branch points that need a new path search are colored gray in the Figure 2. This method shows less expensive path computations than the simple graph-based method. In this example, IRNN is f_3 because its detour distance from R is smallest.

4.3 Spatial Distance Join-based Method

In the recursive spatial range query-based method, a new spatial range search should be repeated at every branch point, even though the number of path computations for processing the IRNN query decreases. One possibility for our progressive solution is a method to find IRNN candidates as possible as quickly. If we tighten the search bound at the beginning, we can reduce more path computations. To make the idea concrete, we adopt a spatial distance join operation whose output is ordered by the distance between two data sets [1, 3]. The distance join operation assumes two data sets are indexed by each spatial index tree and 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 branch 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 spatial distance join-base 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 a tree traversal method between two spatial trees, a heap is occupied with the following pairs, i.e., $<r_i, f_j, dist>$, $<r_i, MBR_f, dist>$, $<MBR_r, f_j, dist>$ or $<MBR_r, MBR_f, dist>$ where r_i is a route branch point object, f_j is a facility point object, MBR_r is MBR of route branch points and MBR_f is MBR of facility points, and *dist* is the Euclidean distance between two objects or one object and MBR or two MBRs. They are ordered by *dist* in the heap. If one pair from the top of the heap consists of two objects $<r_i, f_j>$ 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 IRNN so far. The procedure is repeated until the *dist* of the next pair form the top of the heap is greater than T.

Figure 3 shows a difference between this method and the previous recursive spatial range query-based method. We can find the Euclidean distance closest pair (r_5, f_3) directly using the spatial distance join. 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 solution, this method expects efficiency with less path computations.

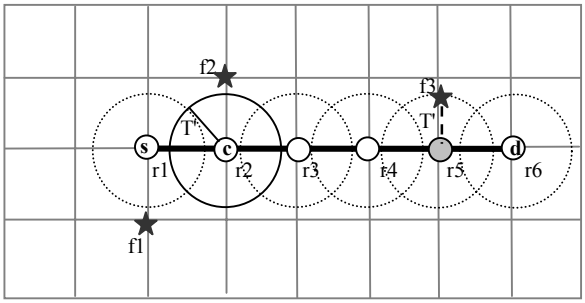


Figure 3. Spatial distance join-based method

4.4 Precomputed Zone-based Method

One important approach in NN queries uses the pre-computed result of NN. In this section, we present a solution based on the pre-computed zones of the road network.

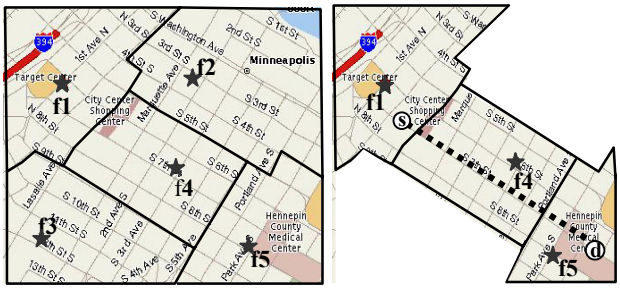
In the preprocessing step, given a set of instances of a facility type F located on the road network, we compute a service zone s_i of each facility instance f_i. The service zone satisfies the following property.

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

$$D(v_j, f_i) \leq D(v_j, f_k),$$

where f_i is the facility in service zone s_i and f_k not.

The information of the service facility, $<v_j, f_i, D(v_j, f_i)>$ for every node v_j, is stored in a traditional index structure, e.g., B-tree, with a key v_j.



(a) Service zones on the road network **(b) Filtered zones with a query route**

Figure 4. Precomputed zone-based method

For preprocessing the road-distance based service zones, we use the network partition algorithm by [19]. 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 of it. Figure 4a shows partitioned service zones on the road network and Figure 4b shows the service zones intersected by a given query route, which have candidate IRNNs.

The precomputed service zones-based method works as follows. First, it finds service zones that route branch points are included using a traditional index search. And then, calculate the detour distance to the facility f_i of a found service zone s_i using the precomputed distance $D(v_j, f_i)$. The IRNN becomes a facility having the smallest detour distance among the intersected service zones.

5. EXPERIMENTS

We performed experimental evaluations to compare the four different solutions using four real road network datasets. The following experiments to study the parameters affecting the performance of the algorithms are conducted:

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?

5.1 Data and Query Sets

We used a publicly available real road map. The dataset covers seven counties in Minnesota: Anoka, Carver, Dakota, Hennepin, Ramsey, Scott and Washington. We converted it 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. It contains 75,739(76K) nodes. The final, 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 above road network datasets whose densities were 0.001%, 0.01%, 0.1%, 0.5%, 1%, 5% and 10% respectively. 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 datasets. In this paper, a route length is defined by the number of branch points which a route consists of. We used different route lengths, e.g., 30, 50, 100, 200, 500, 1000 and 2000. A current position on the route was randomly selected from the route branch points.

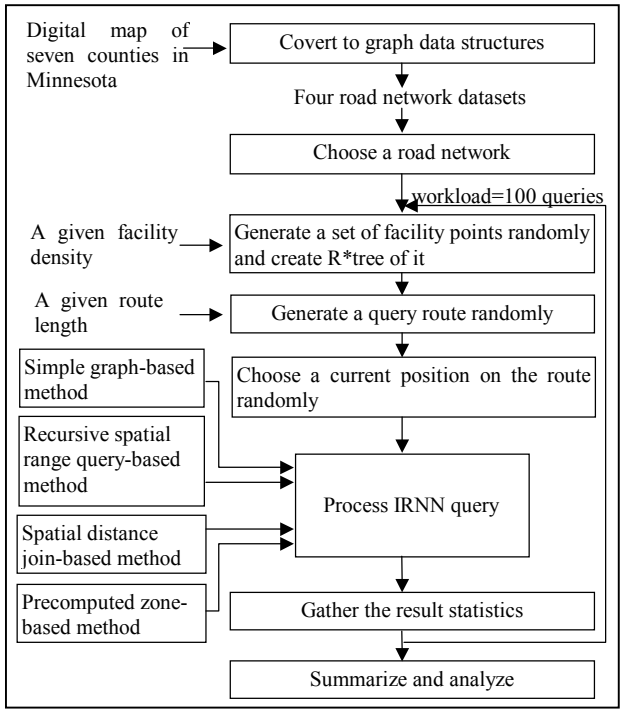


Figure 5. Experiment layout

5.2 Experiment Layout

Figure 5 shows the overall experiment layout. Performance is measured by executing workloads, each of them consisting of 100 queries generated as follows: i) A new set of facility points is randomly generated with a given density in every query. ii) A new query route is randomly generated. First a node is randomly chosen as a starting position. Then consecutive nodes are selected randomly as much as a given route length. iii) A current position is randomly selected from the route dataset. 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++.

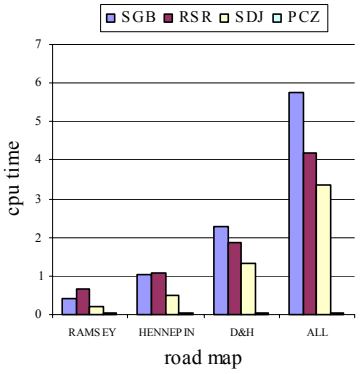


Figure 6. Performance vs. road map size

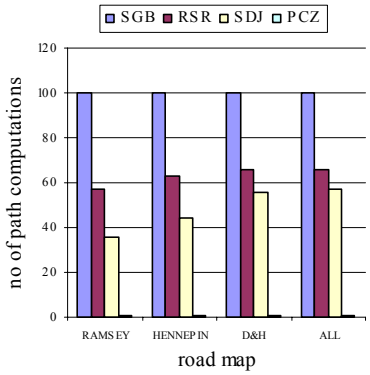


Figure 7. Number of path computations vs. road map size

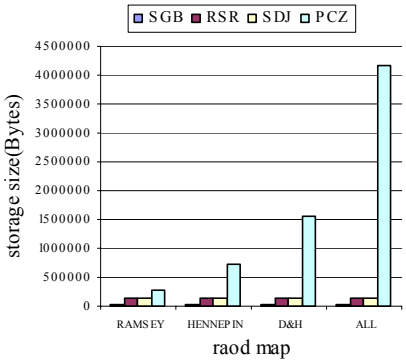
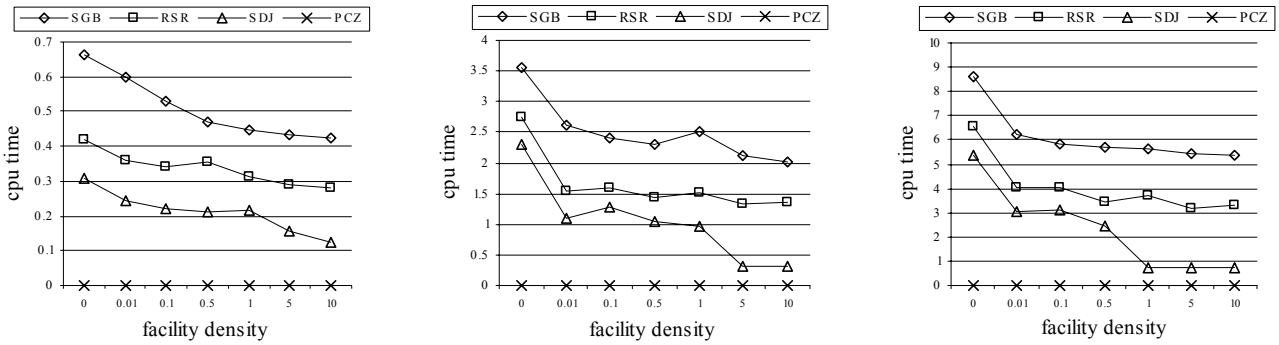


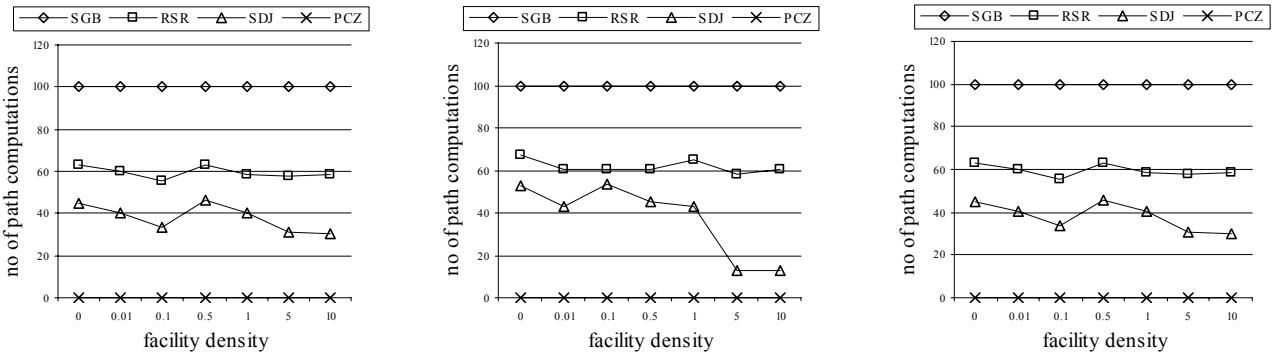
Figure 8. Storage size vs. road map size

(Number of facilities=200, route length =100)



(a) RAMSEY (15K) **(b) D&H (76K)** **(c) ALL (191K)**

Figure 9. Performance vs. facility density (route length =100)



(a) RAMSEY (15K) **(b) D&H (76K)** **(c) ALL (191K)**

Figure 10. Number of path computations vs. facility density (route length =100)

5.3 Experiment Results

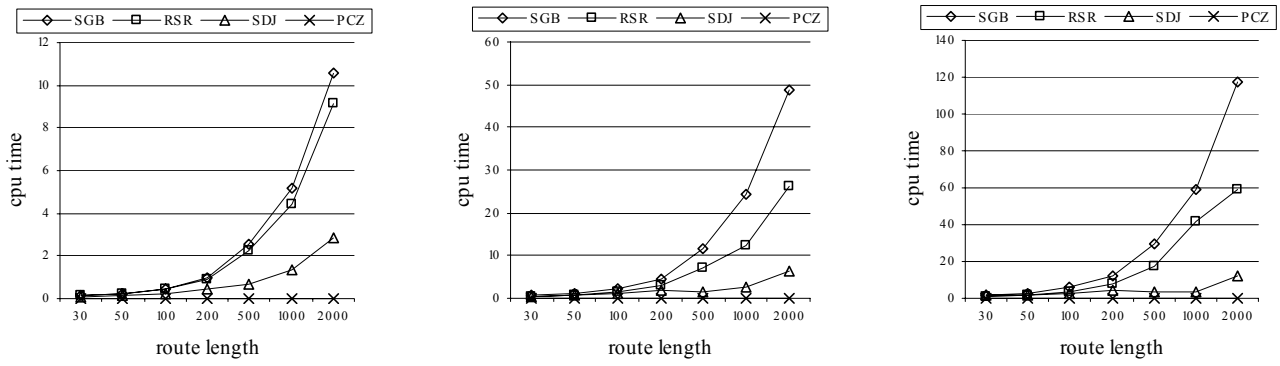
We evaluated the effect of road map size, facility density and route length on the performance of four different methods, SGB, RSR, SDJ and PCZ.

***Effect of road map size*:** In the first experiment, we compared the effect of road map size on the performance and the storage need of all methods. Figure 6 illustrates the response time as a function of road map size, e.g., ALL is the largest among the four road maps. The query route length is fixed to 100 and the number of facility points to 200 in all road map datasets. As shown in Figure 6, PCZ always outperforms the other methods. Its performance is not much affected by road map size. The computation complexity of PCZ is $O(|t|log|v|)$ where t is the number of branch points of a given route and v is the number of nodes of road map. As road map size, i.e., v, increases, the computation complexity increases. However, the cost is much cheaper compared with the other methods. Overall response time in all methods but PCZ increases with a road map size. This happens because, as road map size is larger, the facility density becomes sparser with a fixed number of facilities, increasing a path computation time. The performance of SGB becomes worse as a road map size since it launches more path finding algorithm than the others. SDJ always outperforms SGB and RSR.

Figure 7 shows the number of path computations as a function of road map size. PCZ has no online path search. SGB shows independently of road map size. The reason is the number of path computations depends on a route length not a road map size. In RSR and SDJ, the number of path computations increases slightly. We can infer that the number of path computations has an effect on the response time from Figures 6 and 7.

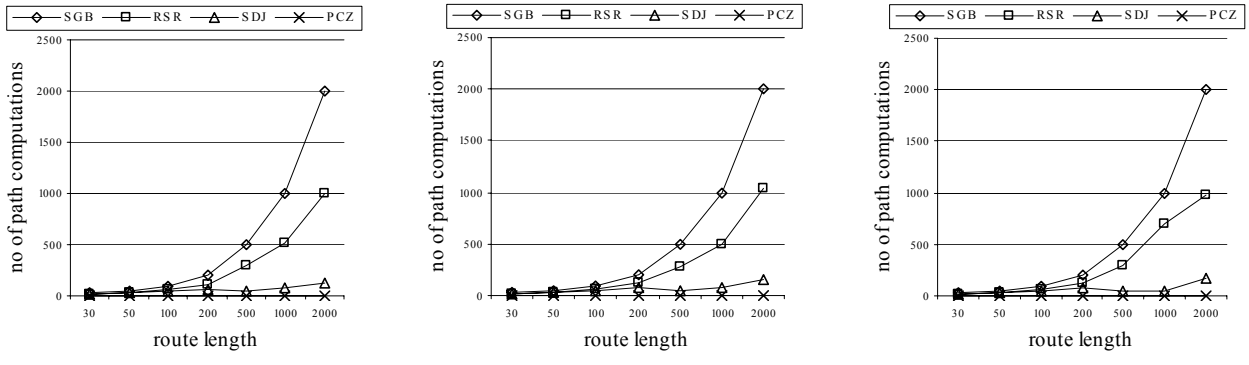
Next, we compare the storage sizes that the methods need. As shown in Figure 8, it is clear that the storage size that PCZ needs depends on the road map size since it stores the precomputed NN result of all nodes of the road map. Otherwise, SBD does not need storage for the query processing. RSR and SDJ are independently of the road map size. The reason is they need storage to store a spatial index tree of facility points and a spatial index tree of route branch points for SDJ.

***Effect of facility density*:** In the second experiment, we examined the impact of the performance of the methods as the facility density varies. Figure 9 shows the response time as facility density for RAMSEY, D&H and ALL respectively. The other parameter is fixed: route length=100. Figure 9 illustrates that the response time for all methods except PCZ decreases with increase in the facility density. It is the reason that the possibility that a facility exists near a query route is higher as a facility density is greater, triggering a smaller search bound. It is known that with a smaller search bound, the branch-and-bound algorithm which is adopted in RSR and SDJ become more efficient. A path computation also can be terminated earlier without the exhaustive scan of the road



(a) RAMSEY (15K) **(b) D&H (76K)** **(c) ALL (191K)**

Figure 11. Performance vs. route length (facility density =0.5%)



(a) RAMSEY (15K) **(b) D&H (76K)** **(c) ALL (191K)**

Figure 12. Number of path computations vs. route length (facility density =0.5%)

network. Figures 9a, 9b and 9c show similar curves. The facility density has the overall positive impact on the performance. Otherwise, PCZ receives little effect from changes in the facility density.

Figure 10 displays similar experiments except that the dependent variable is the number of path computations. In SGB, the number of path computations is independent of facility density since SGB repeats a new path search the route length times. RSR and SDJ show the number of their path computation drop with the larger density.

***Effect of route length*:** In the third experiment, we examined the impact on the performance of all methods as the route length varies. Figure 11 shows the response time as a route length for RAMSEY, D&H and ALL respectively. The facility density was fixed at 0.5%. As shown in Figure 11, the overall response time of all methods but PCZ increase with increase in the route length. In recursive methods, SGB and RSR, the longer the route length, the worse the performances become. SDJ clearly shows better performance than the recursive methods. PCZ shows a very little effect to route length. Figures 11a, 11b and 11c have similar curves.

Figure 12 shows similar experiments except that the dependent variable is the number of path computations. The number of path computations in SGB is the same as the route length. The experiment clearly shows that a non recursive method SDJ receives a little effect of route length.

To summarize, in the first experiment to examine the effect of the road map size on the performance and storage needs of the four different solutions, the precomputed based method, PCZ always outperforms the other methods. However, it needs more disk storage space than the others. Among non pre-processing methods, SBG, RSR and SDJ, SDJ always shows better performance with less number of path computations.

Second, in the experiment to examine the effect of facility density on the performance, all methods but PCZ shows the performance decreases as their facility density increases. PCZ has no impact of facility density. The number of path computations in RSR and SDJ decreases with increase in the facility density.

In the third experiment to examine the impact on the performance of the methods as the route length varies, the overall response time of all methods except PCZ increases with increase in route length. The recursive methods, SGB and RSR show that the longer route length, the worse the performances become. SDJ clearly shows the better performance than them. PCZ shows a very little effect to the route length. The number of path computations of non recursive method SDJ receives a little effect of route length.

6. CONCLUSION

We studied the in-route nearest neighbor query for a given route and presented the four different solution methods. The experimental results show that the precomputed zone-based method always outperforms the other methods but it needs more storage space for storing the precomputed results. In the other methods, the overall response time decreases with the facility density and increases with the route length and the size of road map. The spatial distance join-based method always outperforms the recursive methods with less number of path computations. The experiment shows our strategy to reduce the number of path computations to minimize the response time is reasonable.

One direction of future work is to concrete the current IRNN work. First, we study the effect of updates on dominance zones. Second, we examine IRNN query with other shortest path finding algorithms, e.g., A* etc. Another direction is to search a more general case of IRNN without a given route constraint and extend the IRNN problem with a continuous query problem.

7. REFERENCES

[1] A. Corral, Y. Manolopoulos, Y. Theodoridis, M. Vassilakopoulos, Closes Pair Queries in Spatial Databases, ACM SIGMOND, 2000.

[2] C. Shahabi, M. R. Kolahdouzan, M Sharifzadeh, A Road Network Embedding Technique for K-Nearest Neighbor Search in Moving Object Databases, SSTD, 2001.

[3] GITA and OGC's Emerging Technology Summit Series - Location-Based Services. http://www.openls.org/dvd1/ets1/index.htm.

[4] G. Hjaltason, H. Samet, Distance Browsing in Spatial Databases, ACM TODS, 1999.

[5] G. Hjaltason, H. Samet, Incremental Distance Join Algorithms for Spatial Databases, ACM SIGMOD, 1998.

[6] H. Edelsbrunner, Alogirthms in Computational Geometry, EATCS Monographs on Theoretical Computer Science, 1987.

[7] J. H. Rillings and R. J. Betsold. Advanced Driver Information Systems. IEEE Trans. on Vehicular Technology, 1991.

[8] J. L. Wright, R. Starr, S.Gargaro, GENESIS-Information on the Move, In Proc. of Annual IVHS American Conference, 1993.

[9] J. Zhang, N. Mamoulis, D. Papadias, Y. Tao, All-Nearest-Neighbors Queries in Spatial Databses, 2002.

[10] J. Feng, T. Watanbe, Fast Search of Nearest Target Object in Urban District Road Networks, PYIWIT, 2002.

[11] M. F. Worboys, GIS: A Computing Perspective, Taylor & Francis, 1995.

[12] N. Roussopoulos, S. Kelleym, F. Vincent. Nearest Neighbor Queries, In proceedings of the ACM SIGMOD, 1995.

[13] S. Shekhar, R. R. Vatsava, X. Ma, J. Yoo, Navigation Systems: A Spatial Database Perspective, In chapter 3 of the book, Location-Based Services, 2003.

[14] S. Shekhar, S.Chawla, Spatial Databases: A Tour, Prentice Hall, 2003.

[15] S. Shekhar, M. Coyle, A. Kohli, Path Computation Algorithms for Advanced Traveller Information Systems, IEEE Computer Society, 1993.

[16] S. Shekhar, A. Fetterer, D. Liu, Genesis: An Approach to Data Dissemination in in Advanced Travel Information Systems, Bulletin of the Technical Committee on Data Engineering: Special Issue on Data Dissemination, 1996.

[17] S. Shekhar, A. Fetterer, B. Goyal, Materialization Trade-Offs in Hierarchical Shortest Path Algorithms. Proc. Intl. Symp. on Large Spatial Databases, 1997

[18] S. Bespamyatnikh, J. Snoeyink, Queries with Segments in Voronoi Diagrams. SODA, 1999.

[19] S. Hakimi, M. Labbe, and E. Schmeichel, The Voronoi Partition of a Network and its Implications Location Theory ORSA, 1992.

[20] S. Berchtold, B. Ertl, D. Keim, H.Kriegel, and T.Seidl. Fast nearest neighbor search in high-dimensional space. In proceedings of International Conference on Data Engineering, 1998.

[21] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, Second Edition, MIT Press, 2001.

[22] W. C. Collier and R. J. Weiland. Smart Cars, Smart Highways, IEEE Spectrum, 1994.

[23] Y. Tao, D. Papdias, Q. Shen, Continuous Nearest Neighbor Search, VLDB, 2002.

[24] Z. Song, N. Roussopoulos, K-Nearest Neighbor Search for Moving Query Point, SSTD, 2001.