Finding the Most Accessible Locations - Reverse Path Nearest Neighbor Query in Road Networks

Shuo Shang [†] Bo Yuan [§] Ke Deng [†] Kexin Xie [†] Xiaofang Zhou [†]

[†] School of Information Technology and Electrical Engineering, The University of Queensland
[§] Division of Informatics, Graduate School at Shenzhen, Tsinghua University
[†] {shangs, dengke, kexin, zxf}@itee.uq.edu.au [§] yuanb@sz.tsinghua.edu.cn

ABSTRACT

In this paper, we propose and investigate a novel spatial query called Reverse Path Nearest Neighbor (R-PNN) search to find the most accessible locations in road networks. Given a trajectory data-set and a list of location candidates specified by users, if a location o is the Path Nearest Neighbor (PNN) of k trajectories, the influence-factor of o is defined as k and the R-PNN query returns the location with the highest influence-factor. The R-PNN query is an extension of the conventional Reverse Nearest Neighbor (RNN) search. It can be found in many important applications such as urban planning, facility allocation, traffic monitoring, etc. To answer the R-PNN query efficiently, an effective trajectory data pre-processing technique is conducted in the first place. We cluster the trajectories into several groups according to their distribution. Based on the grouped trajectory data, a two-phase solution is applied. First, we specify a tight search range over the trajectory and location data-sets. The efficiency study reveals that our approach defines the minimum search area. Second, a series of optimization techniques are adopted to search the exact PNN for trajectories in the candidate set. By combining the PNN query results, we can retrieve the most accessible locations. The complexity analysis shows that our solution is optimal in terms of time cost. The performance of the proposed R-PNN query processing is verified by extensive experiments based on real and synthetic trajectory data in road networks.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Spatial databases and GIS

General Terms

Algorithms, Performance

Keywords

Reverse Path Nearest Neighbor Search, Trajectories, Locations, Road Networks, Efficiency

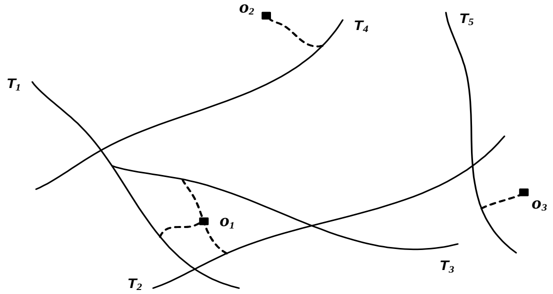


Figure 1: An example of R-PNN query

1. INTRODUCTION

The continuous proliferation of mobile devices and the rapid development of Global Positioning Systems (GPS) enable people to log their current geographic locations and share their trajectories to the web-sites such as Bikely[1], GPS-Waypoints[2], Share-My-Routes[3], Microsoft GeoLife[4], Foursquare[5], etc. The availability of the massive trajectory data creates various novel applications. An emerging one is the Reverse Path Nearest Neighbor (R-PNN) query, which is designed to retrieve the most accessible locations among a set of location candidates. Given a trajectory set T, the query input is a set of location candidates O. If a location $o \in O$ is the Path Nearest Neighbor (PNN) (i.e., the closest data point to a specified path, according to a distance metric) [14, 24, 5, 13] of k trajectories, we define the influence-factor of o as k ($o.if = k$). The R-PNN query asks for the location with the highest influence-factor among all location candidates (i.e., the location with the highest accessibility to the trajectories). A similar work is studied in [20] to find the most influential sites by considering the relationship in Euclidean space between data points (i.e., points to points in Euclidean space vs. trajectories to points in road networks in our work).

An example of R-PNN query is demonstrated in Figure 1, $\tau_1, \tau_2, ..., \tau_5$ are trajectories and o_1, o_2, o_3 are query locations given by users. According to the definition of Path Nearest Neighbor, it is easy to find that o_1 is the PNN of trajectories

[1] http://www.bikely.com/
[2] http://www.gps-waypoints.net/
[3] http://www.sharemyroutes.com/
[4] http://research.microsoft.com/en-us/projects/geolife/
[5] http://foursquare.com/

τ_1, τ_2 and τ_3, which means that the influence-factor of o_1 is 3 ($o_1.if = 3$). In the meantime, o_2 is the PNN of τ_4 ($o_2.if = 1$) and o_3 is the PNN of τ_5 ($o_3.if = 1$). Therefore, o_1 is the data point with the highest influence-factor and the R-PNN query will return data point o_1 to users.

This type of queries is useful in many important applications, such as urban planning, facility allocation, traffic monitoring, location based services, etc. For example, suppose people want to set up a new facility (e.g., post office, bank, petrol station, etc.) in the city and the trajectory data of potential customers are available. The R-PNN query can be used to find the most accessible location among all given location candidates, to maximize the commercial value of the new facility. In traffic monitoring, the R-PNN query can be used to find the optimal location to monitor the maximum number of vehicles.

In this work, the proposed R-PNN query is applied in road networks, since in a large number of practical scenarios objects move in a constraint environment (e.g., roads, railways, rivers, etc.) rather than a free space. A trajectory is a sequence of sample points of a moving object. We assume that all sample points have already been aligned to the vertices on the road network according to some map-matching methods [8, 1, 3, 17] and between two adjacent sample points a and b, the moving objects always follow the shortest path connecting a and b. A straightforward idea to solve the R-PNN problem is to search the exact PNNs for all trajectories in the data-set, and then combine the results to get the location with the highest influence-factor. However, it is not practical since every trajectory in the trajectory data-set, which is supposed to be large in R-PNN query, has to be processed during the query processing. The extremely high computation cost will prevent the query from being answered in real time.

The R-PNN query is challenging due to three reasons. First, we consider the relationship between trajectories and data points (i.e., the minimum distance between a trajectory and a data point) in this query. To acquire the minimum distance between trajectory τ and data point o, every sample point in τ should be considered. This is more complex than the relationship considered in the Reverse Nearest Neighbor (RNN) search (i.e., the minimum distance between two different data points), which is the main reason why the existing techniques of RNN search [10, 15, 21, 11, 16, 20, 23, 19] are not suitable for our work. Second, in R-PNN, the query input is a list of location candidates, not a single point (e.g., [10, 21, 11, 15, 16, 19]). Conventional single point query models lack effective pruning techniques to address our problem efficiently (e.g., given m query locations, the query should be performed m times to find the most accessible locations). Third, in this work, the objects' movement is constrained in road networks, rather than a free space (e.g., [10, 21, 11, 16, 19, 4]). The optimization techniques in the free space may fail to solve the problem in spatial networks since the bounds proposed in the free space is not always valid in spatial networks.

To overcome the challenges and process the R-PNN query efficiently, an effective trajectory data pre-processing technique is conducted in the first place. By extending the conventional k-medoids object clustering method in spatial networks [22], we can cluster the trajectory data into several groups based on their distribution. This technique has two notable benefits to our work. First, it can help us tighten the search range during the query processing. Second, it allows the use of a divide-and-conquer strategy to further enhance the performance of R-PNN query. The effect of the proposed trajectory clustering method is demonstrated by extensive experiments on real data-sets. Based on the grouped trajectory data, a novel two-phase algorithm is proposed. First, we specify a tight search range over the trajectory and location data-sets, since refining every trajectory in the data-set will lead to intolerable computational cost. Here, the network expansion method [6] and branch-and-bound strategy are adopted. In this phase, most of the trajectories and query locations can be pruned safely. The efficiency study reveals that our approach defines the minimum search area. Second, we propose a series of optimization techniques to support the searching of exact PNNs for trajectories in the candidate set. By combining the PNN query results, we can retrieve the location with the highest influence-factor. The complexity analysis shows that our solution is optimal in terms of time cost. To sum up, the major contributions of this paper are:

- We define a novel type of query to find the Reverse Path Nearest Neighbor (R-PNN) in road networks. It provides new features for advanced spatial-temporal information systems, and benefits users in many popular applications such as urban planning.
- We propose an effective trajectory clustering technique that can further enhance the R-PNN query efficiency.
- We devise a two-phase algorithm to answer the R-PNN query efficiently. The efficiency study reveals that our approach defines the minimum searching area and the complexity analysis shows that our solution is optimal in terms of time cost.
- We conduct extensive experiments on real and synthetic trajectory data to investigate the performance of the proposed approaches.

The rest of the paper is organized as follows. Section 2 introduces the road networks and trajectories used in this paper as well as problem definitions. The R-PNN query processing is described in Section 3, which is followed by the experimental results in Section 4. This paper is concluded in Section 6 after discussions on related work in Section 5.

2. PRELIMINARIES

2.1 Road Networks

In this work, road networks are modeled as connected and undirected planar graphs $G(V, E)$, where V is the set of vertices and E is the set of edges. A weight can be assigned to each edge to represent its length or application specific factors such as traveling time obtained from historical traffic data [7]. Given two locations a and b in road networks, the network distance between them is the length of their shortest network path (i.e., a sequence of edges linking a and b where the accumulated weight is minimal). The data points are distributed along roads and if a data point is not located at a road intersection, we treat the data point as a vertex and further divide the edge that it lies on into two edges. Thus, we assume that all data points are in vertices for the sake of clarity.

2.2 Trajectory

The raw trajectory samples obtained from GPS devices are typically of the form of $(longitude, latitude, time-stamp)$. How to map the $(longitude, latitude)$ pair onto the digital map of a given road network is an interesting research problem itself but outside the scope of this paper. We assume that all sample points have already been aligned to the vertices on the road network by some map-matching algorithms [1, 3, 8, 17] and between two adjacent sample points a and b, the moving objects always follow the shortest path connecting a and b. As the trajectory's time-stamp attribute is not related to our work, we define the trajectory in the following format.

Definition: Trajectory
A trajectory of a moving object τ in road network G is a finite sequence of positions: $\tau = \{p_1, p_2, ..., p_n\}$, where p_i is the sample point in G, for $i = 1, 2, .., n$. □

2.3 Problem Definition

Given any two locations a and b in a road network, the shortest network path between them is denoted as $SP(a, b)$ and the length of $SP(a, b)$ is denoted as $sd(a, b)$. Given a trajectory τ and a data point o in a road network, the minimum distance $d_M(o, \tau)$ between data point o and trajectory τ is defined as

$$d_M(o, \tau) = \min_{v_i \in \tau}\{sd(o, v_i)\}, \tag{1}$$

where v_i is the vertex belonging to τ.

Definition: Path Nearest Neighbor (PNN)
Given a trajectory τ and a set of data points O, the Path Nearest Neighbor (PNN) of τ is the data point $o \in O$ with the minimum $d_M(o, \tau)$. That is, $d_M(o, \tau) \leq d_M(o', \tau), \forall o' \in \{O - o\}$. □

Definition: Reverse Path Nearest Neighbor (R-PNN) Query
Given a trajectory set T and a data point set O, if $o \in O$ is the Path Nearest Neighbor of k trajectories $\tau_1, \tau_2, ..., \tau_k \in T$, the influence-factor of o is k ($o.if = k$). Reverse Path Nearest Neighbor Query finds the data point $o \in O$ with the highest influence-factor. That is, $o.if \geq o'.if, \forall o' \in \{O - o\}$. □

3. QUERY PROCESSING

Intuitively, the Reverse Path Nearest Neighbor (R-PNN) query can be solved by searching the exact PNNs for all trajectories in the data-set and combining the search results to find the location with the highest influence-factor (i.e., the most accessible location). It is not practical since every trajectory in the data-set should be processed during the query processing, resulting in intolerable computation cost. To overcome this challenge and solve the R-PNN query efficiently, we conduct an effective trajectory data preprocessing technique (k-medoids trajectory cluster) in the first place (Section 3.1). Then, based on the grouped trajectory data, a two-phase solution is proposed. In the first phase, we specify a tight searching range over trajectory and location data-sets (Section 3.2). The efficiency study reveals that our approach employs the minimum search area (Section 3.3). The second phase introduces the searching process for the most accessible location, with the support of a series of optimization techniques (Section 3.4). The complexity analysis shows that our solution is optimal in terms of time cost (Section 3.5).

3.1 Trajectory Data Pre-Processing

To enhance the performance of R-PNN query processing, an effective trajectory data pre-processing technique is conducted in the first place. Given a trajectory data-set T, the proposed data pre-processing procedure takes two steps. First, according to the k-medoids clustering method, the trajectory data-set is divided into k clusters. For each trajectory cluster C_i, $i \in [1, k]$, the corresponding medoid m_i is recorded. Second, we compute and record the minimum network distance between m_i and every trajectory $\tau \in C_i$ and other related information. This technique has two major contributions: (i) it can help us tighten the searching range during the query processing; (ii) it allows the use of a divide-and-conquer strategy to further enhance the query efficiency.

k-medoids trajectory clustering method is adopted in the first step. It is an extension of the conventional k-medoids clustering algorithm in spatial networks [22]. Initially, we randomly select k vertices from the road network as the medoids. How to select a suitable value of k will be discussed in Section 3.3. Then, each trajectory is assigned to the cluster corresponding to the nearest medoid reachable from it. If trajectories in the same cluster are close to the medoid, the clustering is effective. **Remark.** In the following search (Section 3.2), we use $d_M(m_i, \tau)$ and $sd(m_i, o)$ to estimate the minimum distance between data point $o \in O$ and τ. Based on our observation, small values of $d_M(m_i, \tau)$ will lead to small gaps between the upper and lower bounds of $d_M(o, \tau)$, and enhance the pruning effectiveness. For this reason, an evaluation function (Equation 2) is applied to evaluate the quality of the partition.

$$R((C_i, m_i) : i \in [1, k]) = \sum_{i=1}^{k} \sum_{\tau \in C_i} d_M(m_i, \tau) \tag{2}$$

Here, m_i is the medoid of cluster C_i for $i \in [1, k]$. The lower the value of R, the better the partition. After evaluation, a medoid will be replaced by a new vertex from the road network and trajectories will be assigned to the clusters based on new medoids. If the R value of the new partition is less than the old one, the change will be committed. Otherwise, the change will be rolled back and a new medoid replacement will be attempted. This process terminates when no replacements can lead to a better result and the corresponding clusters and medoids are recorded.

In the second step, for each cluster C_i, we compute and record the values of $d_M(m_i, \tau)$, the minimum network distance between m_i and each trajectory $\tau \in C_i$, and other useful information to enhance the performance of the following R-PNN query processing. $d_M(m_i, \tau)$ can be computed by

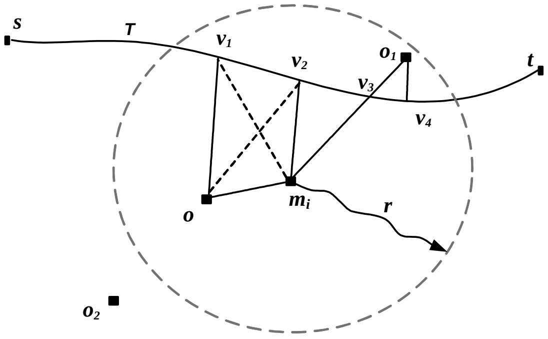


Figure 2: Identifying Candidates

applying Dijkstra's expansion [6]. By expanding a browsing wavefront from m_i according to the Dijkstra's algorithm, the first vertex $v \in \tau$ touched by the wavefront is just the closest vertex to m_i in τ. That is, $sd(m_i, v) = d_M(m_i, \tau)$, where $v \in \tau$. v is also recorded as the closest vertex to m_i in τ. Since this searching process has already been conducted in the trajectory clustering phase, the values of $d_M(m_i, \tau)$ can be easily obtained from the searching results of the first step. In addition, we compute and record the value of $\max\{d(\tau.s, v), d(v, \tau.t)\}$ for vertex v and the corresponding trajectory τ, where $\tau.s$ and $\tau.t$ are the source and destination of trajectory τ respectively and $d(\tau.s, v)$ is the network distance between $\tau.s$ and v along the trajectory τ (not shortest path distance). The value of $\max\{d(\tau.s, v), d(v, \tau.t)\}$ will be used to define the lower bound of $d_M(o, \tau)$, $o \in O$, in the following search.

3.2 Identifying Candidates

Given a set of trajectory clusters $C_i, i \in [1, k]$ and a set of data points (location candidates) O, we try to specify a tight searching range based on a series of filter and refinement techniques. In our approach, we estimate the minimum network distance $d_M(o, \tau)$ between data point o and trajectory τ by a pair of lower bound $d_M(o, \tau).lb$ and upper bound $d_M(o, \tau).ub$ based on the triangle inequality of shortest paths. The triangle inequality in road networks can be represented as:

$$\begin{cases} sd(v_1, v_2) + sd(v_1, v_3) \geq sd(v_2, v_3) \\ sd(v_1, v_2) - sd(v_1, v_3) \leq sd(v_2, v_3) \end{cases}$$

where $sd(v_1, v_2)$ indicates the shortest path distance between two vertices v_1 and v_2.

Consider the schematic example demonstrated in Figure 2. Vertex m_i is a medoid of trajectory cluster C_i and τ is a trajectory in the same cluster. v_1, v_2, v_3, v_4 are vertices belonging to trajectory τ, and o, o_1, o_2 are data points. Suppose v_1, v_2 and v_4 are the closest vertex in τ to o, m_i and o_1 respectively. According to the triangle inequality introduced above, we have

$$\begin{cases} sd(o, m_i) + sd(m_i, v_2) \geq sd(o, v_2) \\ sd(o, v_2) \geq d_M(o, \tau) = sd(o, v_1) \end{cases}$$

$$\Rightarrow d_M(m_i, \tau) + sd(o, m_i) \geq d_M(o, \tau)$$

Consequently, an upper bound of the minimum distance $d_M(o, \tau)$ between data point o and trajectory τ is given by

$$d_M(o, \tau).ub = d_M(m_i, \tau) + sd(o, m_i) \quad (3)$$

In the following paragraphs, we introduce our method of estimating $d_M(o, \tau).lb$. Obviously, $P(o, v_1, v_2, m_i)$ is a path connecting data point o and medoid m_i. According to the definition of the shortest path, it is easy to find that

$$d(o, v_1, v_2, m_i) = sd(o, v_1) + d(v_1, v_2) + sd(v_2, m_i) \geq sd(o, m_i)$$

$$\Rightarrow d_M(o, \tau) \geq sd(o, m_i) - d_M(m_i, \tau) - d(v_1, v_2)$$

Then, we can use the value of $\max\{d(s, v_2), d(v_2, t)\}$ (precomputed in Section 3.1) to replace $d(v_1, v_2)$ and get

$$\begin{cases} d_M(o, \tau) \geq sd(o, m_i) - d_M(m_i, \tau) - d(v_1, v_2) \\ max\{d(s, v_2), d(v_2, t)\} \geq d(v_1, v_2) \end{cases}$$

$$\Rightarrow d_M(o, \tau) \geq sd(o, m_i) - d_M(m_i, \tau) - \max\{d(s, v_2), d(v_2, t)\}$$

Therefore, the lower bound of $d_M(o, \tau)$ is described by the following equation.

$$d_M(o, \tau).lb = sd(o, m_i) - d_M(m_i, \tau) - \max\{d(s, v_2), d(v_2, t)\} \quad (4)$$

By comparing $d_M(o, \tau).ub$ and $d_M(o, \tau).lb$, we can find that the gap between them are mainly affected by the value of $d_M(m_i, \tau)$.

$$d_M(o, \tau).ub - d_M(o, \tau).lb = 2d_M(m_i, \tau) + \max\{d(s, v_2), d(v_2, t)\}$$

This observation shows that lower values of $d_M(m_i, \tau)$ may lead to tighter lower/bound gaps, and enhance the pruning effectiveness and justifies conducting trajectory data clustering in the first place.

Remark. The value of $d_M(m_i, \tau) - sd(o, m_i)$ is not suitable to be used as the lower bound of $d_M(o, \tau)$. According to the triangle inequality, it is easy to find that $sd(o, v_1) \geq sd(v_1, m_i) - sd(o, m_i)$. Since v_2 is the closest vertex to m_i in τ, we have $sd(v_1, m_i) \geq sd(v_2, m_i)$. Thus, we can use $sd(v_2, m_i)$ to replace $sd(v_1, m_i)$ and get $sd(o, v_1) \geq sd(v_2, m_i) - sd(o, m_i)$. That is, $d_M(o, \tau) \geq d_M(m_i, \tau) - sd(o, m_i)$. However, for any data points $o_1, o_2 \in O$, the value of $d_M(m_i, \tau) - sd(o_1, m_i)$ is always less than $d_M(o_2, \tau).ub = d_M(m_i, \tau) + sd(o_2, m_i)$. Thus, $d_M(m_i, \tau) - sd(o, m_i)$ cannot be used as the lower bound of $d_M(o, \tau)$.

Now we need to identify a data point candidate set $\tau._{CS}$ for each trajectory $\tau \in T$ based on the upper and lower bounds introduced above. Only data points in the candidate set may turn out to be the PNN of the corresponding trajectory. For the same trajectory, any data point whose lower bound is greater than any other data point's upper bound will be pruned. For trajectory $\tau \in C_i$, if the shortest path distance from medoid m_i to o is known, the candidature of o can be determined (since the values of $d_M(m_i, \tau)$ and $max\{d(s, v_2), d(v_2, t)\}$ are acquired in the trajectory data pre-processing phase). Here, Dijkstra's expansion [6] is adopted to select the data point candidates. From each medoid m_i, $i \in [1, k]$, a browsing wavefront is expanded in Dijkstra's algorithm. Conceptually, the browsed region is restricted within a circle as shown in Figure 2 where the radius is the shortest network distance from the medoid to the browsing wavefront, denoted as r. Among all data points scanned by the expansion wave, we define a global upper bound $\tau.ub$ for trajectory τ as

$$\tau.ub = \min_{\forall o \in O_s(i)} \{d_M(o, \tau).ub\} \tag{5}$$

where $O_s(i)$ is the set of scanned data point (scanned by the expansion wave from m_i). An example is demonstrated in Figure 2, where $o, o_1 \in O_s(i)$ and $o_2 \notin O_s(i)$. The expansion of browsing wavefront stops once $r - d_M(m_i, \tau) - \max\{d(s, v_2), d(v_2, t)\}$ is greater than $\tau.ub$, for every $\tau \in C_i$. It can be proved that all data points outside the browsed region cannot have shortest network distances to τ less than $\tau.ub$, and can be pruned safely (e.g., o_2).

Lemma 1: For any data point o outside the browsed region, we have $d_M(o, \tau) > \tau.ub$ and o can be pruned from the data point candidate set $\tau_{\cdot CS}$ safely.

Proof: As shown in Figure 2, since the Dijkstra's algorithm always chooses the vertex with the smallest distance label for expansion, we have $sd(o_2, m_i) > r$. The browsing wavefront stops when $r - d_M(m_i, \tau) - \max\{d(s, v_2), d(v_2, t)\} > \tau.ub$. We can use $sd(o_2, m_i)$ to replace r and get $d_M(o_2, \tau).lb = sd(o_2, m_i) - d_M(m_i, \tau) - \max\{d(s, v_2), d(v_2, t)\} > \tau.ub$. Hence, the data points outside the browsed region can be pruned from τ_{CS} safely. □

The data point candidates for trajectory $\tau \in C_i$ can be described by the following equation.

$$\tau_{\cdot CS} = \{o | sd(o, m_i) - d_M(m_i, \tau) - \max\{d(s, v_2), d(v_2, t)\} \leq \tau.ub\} \tag{6}$$

Consequently, following the approach introduced above, we can specify a data point candidate set for every trajectory $\tau \in T$. Next, a backward process is conducted to find the trajectory candidates for every data point $o \in O$ (i.e., a data point o may only turn out to be the PNN of the trajectories in the candidate set $o_{\cdot CS}$). If $o \in \tau_{\cdot CS}$, we define that $\tau \in o_{\cdot CS}$. An example of the backward process is demonstrated by the following equations.

$$\begin{cases} \tau_1{}_{\cdot CS} = \{o_1, o_2\} \\ \tau_2{}_{\cdot CS} = \{o_1, o_2\} \\ \tau_3{}_{\cdot CS} = \{o_1, o_2\} \\ \tau_4{}_{\cdot CS} = \{o_3\} \\ \tau_5{}_{\cdot CS} = \{o_1, o_2\} \\ \tau_6{}_{\cdot CS} = \{o_1\} \\ \tau_7{}_{\cdot CS} = \{o_3\} \end{cases} \Longrightarrow \begin{cases} o_1{}_{\cdot CS} = \{\tau_1, \tau_2, \tau_3, \tau_5, \tau_6\} \\ o_2{}_{\cdot CS} = \{\tau_1, \tau_2, \tau_3, \tau_5\} \\ o_3{}_{\cdot CS} = \{\tau_4, \tau_7\} \end{cases}$$

Assumption 1: The number of trajectories in T is much greater than the number of data points in O.

In real scenarios, it is impractical for a user to input a large number (e.g., hundreds) of query locations (data points) before successfully making a query. Therefore, it is reasonable to assume that the size of trajectory database is much greater than the number of query locations.

Pigeonhole Principle: If n items are put into m pigeonholes with $n > m$, then at least one pigeonhole must contain more than $\lfloor \frac{n}{m} \rfloor$ items.

Obviously, the size of $o_{\cdot CS}$ is greater than $o.if$, and we can use $o_{\cdot CS}\cdot_{size}$ to estimate the upper bound of $o.if$. Suppose $T_{\cdot num}$ is the size of trajectory data-set T and $O_{\cdot num}$ is the size of data point data-set O. For every data point o whose candidate-set size is less than $\lfloor \frac{T_{\cdot num}}{O_{\cdot num}} \rfloor$, we have

$$\begin{cases} o_{\cdot CS}\cdot_{size} \leq \lfloor \frac{T_{\cdot num}}{O_{\cdot num}} \rfloor \\ o.if \leq o_{\cdot CS}\cdot_{size} \end{cases} \Rightarrow o.if \leq \lfloor \frac{T_{\cdot num}}{O_{\cdot num}} \rfloor$$

According to the Pigeonhole Principle and Assumption 1 introduced above, o must not be the data point with the highest influence-factor. And we remove o from $\tau_{\cdot CS}$, $\tau \in T$ temporally. Let us review the last example. There are 7 trajectories in T and 3 data points in O, thus $\lfloor \frac{T_{\cdot num}}{O_{\cdot num}} \rfloor = 2$. Since $o_3{}_{\cdot CS}\cdot_{size} = 2$ is no greater than $\lfloor \frac{T_{\cdot num}}{O_{\cdot num}} \rfloor$, o_3 must not be the most accessible location and o_3 should be removed from $\tau_i{}_{\cdot CS}$, $i \in [1, 7]$. Then, we find that the candidate sets for τ_4 and τ_7 are empty. That means it is not necessary to search the exact PNNs for τ_4 and τ_7. Therefore, the trajectory whose data point candidate set is empty ($\tau_{\cdot CS} = \varnothing$) can be pruned from the trajectory data-set T. The remaining trajectories make up the trajectory candidate set $T_{\cdot CS}$, and the temporally removed data points are restored.

In the first searching phase, we specify a tight trajectory candidate set $T_{\cdot CS}$ from trajectory data-set T. For each trajectory $\tau \in T_{\cdot CS}$, we carefully maintain a data point candidate set $\tau_{\cdot CS}$, which contains the data points that may turn out to be the PNN of τ. In the next phase, we will search the exact PNNs for trajectories in $T_{\cdot CS}$ and retrieve the data point with the highest influence-factor.

3.3 Efficiency Study

In this section, we present our approach to selecting a suitable value of k as the number of trajectory clusters (Section 3.1). Our target is to specify the minimum search range and get the minimum amount of data point candidates, which can help us further enhance the performance of R-PNN query processing. Suppose trajectories are uniformly distributed in road networks and data points are uniformly distributed as well. We now analyze the search range (finding candidates, Section 3.2) by estimating the scanned area in the first searching phase. According to the approaches introduced in Section 3.2, the total scanned area A_S can be described as

$$A_s = k\pi(sd(m_i, o) + 2d_M(m_i, \tau) + \max\{d(\tau.s, v), d(v, \tau.t)\})^2 \tag{7}$$

where m_i is the medoid for cluster $C_i, i \in [1, k]$, o is the nearest data point to m_i (due to the definition of $\tau.ub$ and the expansion nature of the Dijkstra's algorithm) and τ is the farthest trajectory in C_i to m_i (due to the pruning strategy introduced in Lemma 1). Then, we use R to denote the value of $d_M(m, \tau)$, r to denote the value of $sd(m, o)$ and δ to denote the value of $\max\{d(\tau.s, v), d(v, \tau.t)\}$. A_s can be expressed as $A_s = k\pi(r + 2R + \delta)^2$. Suppose $Area(G)$ is the area of the network $G(V, E)$, the value of R and r can be calculated as follows.

$$\pi R^2 = \frac{Area(G)}{k} \Rightarrow R = \sqrt{\frac{Area(G)}{\pi k}}$$

$$r = \sqrt{\frac{Area(G)}{\pi k}} - \sqrt{\frac{Area(G)}{\pi O_{\cdot num}}}$$

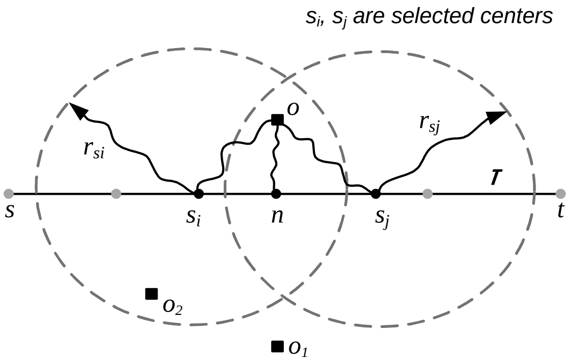


Figure 3: Lower bound and Upper bound

Although the exact amount of query points $O._{num}$ is unavailable to us before the query is performed, we can use the average value of $O._{num}$ in the similar queries (or the average value of the historical queries) to estimate it. Thus, through substituting R and r into Equation 7, A_s can be expressed as $A_s = ak + b\sqrt{k} + c$, where

$$\begin{cases} a = \pi\delta^2 + \frac{Area(G)}{O._{num}} - 2\delta\sqrt{\frac{\pi Area(G)}{O._{num}}} \\ b = 6\ (\delta\sqrt{\pi Area(G)} - \frac{Area(G)}{\sqrt{O._{num}}}) \\ c = 9 \times Area(G) \end{cases}$$

When $k = (\frac{-b}{2a})^2$, we can get the minimum value of A_s and the minimum amout of data point candidates.

3.4 Searching the Most Accessible Locations

Now, we have a trajectory candidate set $T._{CS}$ and a data point candidate set $\tau._{CS}$ for each trajectory $\tau \in T._{CS}$. In this section, we present the algorithms to find the exact Path Nearest Neighbor to the trajectory $\tau \in T._{CS}$. By combining the PNN query results, we can retrieve the data point with the highest influence-factor.

Given a trajectory $\tau \in T._{CS}$ and a data point set $\tau._{CS}$, the Path Nearest Neighbor query processing takes two steps.

1. Further tighten the data point candidate set $\tau._{CS}$ according to the proposed lower/upper bound.
2. Compute the minimum network distance between every candidate $o \in \tau._{CS}$ and τ, and then combine the results to find the exact PNN to τ.

Initially, a set of vertices in τ are selected as the expansion centers such that τ is divided into several path segments. If the shortest path distances from o to the two ends s_i and s_j of a path segment $P(s_i, s_j)$ are known, the candidature of o can be determined. As the schematic example shown in Figure 3, the path segment $P(s_i, s_j)$ is illustrated and n is a vertex on $P(s_i, s_j)$. o, o_1 and o_2 are data points in $\tau._{CS}$. An upper bound of the minimum distance from data point o to $P(s_i, s_j)$ is given by $o.ub = \min\{sd(s_i, o), sd(o, s_j)\}$. In the first searching phase (Section 3.2), the value of $d_M(o, \tau)$ is estimated by an upper bound $d_M(o, \tau).ub$ (Equation 3). To further tighten the candidate set $\tau._{CS}$, the minimum one between $\min\{sd(s_i, o), sd(o, s_j)\}$ and $d_M(o, \tau).ub$ is selected as the upper bound of data point o.

$$o.ub = \min\{\min\{sd(s_i, o), sd(o, s_j)\}, d_M(o, \tau).ub\} \qquad (8)$$

In Figure 3, suppose n is the closest vertex to o in (s_i, s_j). We estimate the lower bound of o according to the triangle inequality stated before.

$$\begin{cases} sd(o, n) \geq sd(s_i, o) - sd(s_i, n) \\ sd(o, n) \geq sd(o, s_j) - sd(n, s_j) \end{cases}$$

$$\Rightarrow sd(o, n) \geq \frac{sd(s_i, o) + sd(o, s_j) - sd(s_i, n) - sd(n, s_j)}{2}$$

Then, we use $d(s_i, s_j)$ to denote the path between s_i and s_j along τ (not necessary a shortest path). Based on the definition of shortest path, it is easy to find that $d(s_i, s_j) = d(s_i, n) + d(n, s_j)$ is greater than $d(s_i, n) + d(n, s_j)$. So, we can use $d(s_i, s_j)$ to replace $d(s_i, n) + d(n, s_j)$ in the last equation.

$$sd(o, n) \geq \frac{sd(s_i, o) + sd(o, s_j) - d(s_i, s_j)}{2}$$

Next, we adopt the maximum one between $\frac{sd(s_i,o)+sd(o,s_j)-d(s_i,s_j)}{2}$ and $d_M(o, \tau).lb$ (Equation 4) as the lower bound of o.

$$o.lb = \max\{\frac{sd(s_i, o) + sd(o, s_j) - d(s_i, s_j)}{2}, d_M(o, \tau).lb\} \qquad (9)$$

Our objective is to find all data points which have lower bound not more than any other point's upper bound. From s_i, a browsing wavefront is expanded in road networks as Dijkstra's algorithm [6] and so does s_j. In concept, the browsed region is round as shown in Figure 3 where the radius is the network distance from the center to the browsing wavefront, denoted as r. Among all data objects scanned by the expansion wave, we define a global upper bound UB as

$$UB = \min_{o \in O_s}\{o.ub\}$$

where O_s is the set of all scanned data objects, $O_s = \bigcup_{i=1}^{\lambda} O_i$ and λ is the number of expansion centers. An example is demonstrated in Figure 3, where $o, o_2 \in O_s$ and $o_1 \notin O_s$. The expansion of browsing wavefront will be stopped once $\frac{r_{si}+r_{sj}-d(s_i,s_j)}{2}$ is greater than UB. Given a data point o_1 outside the browsed region, since the Dijkstra's algorithm [6] always chooses the vertex with the smallest distance label for expansion, we have $sd(o_1, s_i) > r_{si}$ and $sd(o_1, s_j) > r_{sj}$. Thus, $\frac{sd(o_1,s_i)+sd(o_1,s_j)-d(s_i,s_j)}{2} > UB$ and $o_1.lb > UB$, which means that the shortest network distance from o_1 to $P(s_i, s_j)$ cannot be less than UB, and can be pruned safely. Consequently, a tighter candidate set $\tau._{CS}$ is created and the first step of the PNN query processing terminated.

In the second step, we compute the minimum distance $d_M(o, \tau)$ from each data point $o \in \tau._{CS}$ to τ by applying Dijkstra's expansion [6]. By expanding a browsing wavefront from o according to the Dijkstra's algorithm, the first vertex v touched by the wavefront is just the closest vertex to o in τ. That is, $sd(o, v) = d_M(o, \tau)$. The data point with the minimum value of $d_M(o, \tau)$ is the PNN to trajectory τ.

Finally, we combine the PNN query results for trajectories in $T._{CS}$ to retrieve the data point with the highest influence-factor. It is returned as the most accessible location to users.

3.5 Complexity Analysis

Suppose data points are uniformly distributed in spatial networks and the vertices in τ are uniformly distributed as well. We now analyze the complexity of PNN query processing by estimating the cost in each step. In the first step, the Dijkstra's expansion is performed from centers to identify candidates. In a given network $G(V, E)$, the complexity is $O(\lambda(Vlg(V) + E))$ where λ is the number of centers. The second step computes the minimum distance between candidates and trajectory τ. The cost is $O(\mu(Vlg(V) + E))$, where μ is the number of candidates. By combining the two steps, the time complexity of PNN query processing is $O((\lambda+\mu)(Vlg(V)+E))$. Clearly, the number of centers and the number of candidates determine the overall time cost of PNN query processing.

In an extreme case where every vertex in τ is a center, the candidate set is minimized but the number of centers is maximized. Since the expansions are based on both candidates and centers, the overall performance may suffer, especially in case of sparse data points. In another extreme case, only the two ends of P (i.e., source and destination) are selected as centers (as in [5]). While the number of centers is minimized, the candidate set may be very large. The optimal selection of centers can be estimated using linear programming. Suppose $\{s_1 = s, s_2.., s_{n-1}, s_n = t\}$ are vertices in τ in the order from s to t. Let A be a $n \times n$ matrix where $a_{ij} = 1$ if the i^{th} and the j^{th} exits are adjacent centers (i.e., there are no exits in between them that are centers) and $a_{ij} = 0$ otherwise. Our goal is to minimize the objective function

$$\omega = \sum(a_{ij}\frac{1}{4}\pi(d_{ij} + r)^2\rho) + \sum a_{ij}. \tag{10}$$

subject to $i < j$, $\sum_{j=1}^{n} a_{0j} = 1$, $\sum_{i=1}^{n} a_{i0} = 1$, $\sum_{j=1}^{n} a_{ij} \leq 1$, $\sum_{i=1}^{n} a_{ij} \leq 1$, and $\sum_{j=1}^{n} a_{ij} = \sum_{k=1}^{n} a_{ki}$. ρ is the density of data points and d_{ij} is the network distance between the i^{th} and j^{th} centers along the trajectory (not necessarily a shortest path). We use $\sum(a_{ij}\frac{1}{4}\pi(d_{ij}+r)^2\rho)$ to estimate the number of candidates and $\sum a_{ij}$ to estimate the number of centers, where

$$r = \min\{\sqrt{\frac{Area(G)}{\pi(\tau._{CS\cdot size})}}, d_M(o, \tau).ub\}$$

For online processing, the time cost of finding the optimal selection of centers by solving the above objective function may not be practical. Thus, we simplify the objective function (10) by assuming that the gaps between adjacent centers are equal and the vertices in τ are uniformly distributed. Our aim is changed to find the optimal number of centers. Then, we have

$$\omega(\lambda) = \lambda(\frac{1}{4}\pi(\frac{\tau.l}{\lambda} + r)^2\rho) + \lambda. \tag{11}$$

where $\tau.l$ is the length of τ. The λ resulting in the minimum ω can be retrieved using the derivative of the Function 11

$$\omega(\lambda)' = \frac{\partial\omega}{\partial\lambda} = 0. \Rightarrow \lambda = \sqrt{\frac{\pi\rho(\tau.l)^2}{4 + \pi\rho r^2}}. \tag{12}$$

Note that if the value of $\omega(\lambda)$ is greater than $\tau._{CS\cdot size}$ ($\tau._{CS}$ is achieved in Section 3.2), it is not necessary to conduct the PNN optimization techniques introduced in Section 3.4, since the time-cost required by the network expansion offsets the time saved by tightening the candidate set.

Table 1: Parameter setting

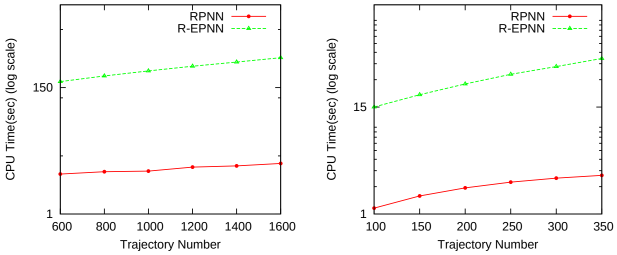
	BRN	ORN
Trajectory Number	600-1600 (default 1000)	100-350 (default 200)
Location Number	40-120 (default 80)	20-60 (default 40)

4. EXPERIMENTS

In this section, we conducted extensive experiments on real spatial data-sets to demonstrate the performance of the proposed R-PNN search. The two data-sets used in our experiments were Beijing Road Network (BRN) [6] and Oldenburg City Road Network (ORN)[7], which contain 28,342 vertices and 6,105 vertices respectively, stored as adjacency lists. In BRN, we adopted the real trajectory data collected by the MOIR project [12]. In ORN, the synthetic trajectory data were used. All algorithms were implemented in Java and tested on a Windows platform with Intel Core i5-2410M Processor (2.30GHz, 3MB L3) and 4GB memory. In our experiments, the networks resided in memory when running the Dijkstra's algorithm [6], as the storage memory occupied by BRN/ORN was less than 1MB, which is trivial for most hand-held devices in nowadays. All experiment results were averaged over 20 independent tests with different query inputs. The main performance metric was CPU time and the parameter settings are listed in table 1.

By default, the number of trajectories were 1600 and 200 in BRN and ORN respectively. In the meantime, the number of query locations (data points) was set to 80 for BRN, and 40 for ORN. The query locations were randomly selected from road networks. For the purpose of comparison, an algorithm based on Path Nearest Neighbor (PNN) query [5] was also implemented (referred to as R-EPNN). In this algorithm, the PNN query was conducted to find the exact PNN for every trajectory in the data-set and the most accessible location was found by combining all the PNN query results.

4.1 Effect of the Trajectory Number $T._{num}$



(a) BRN

(b) ORN

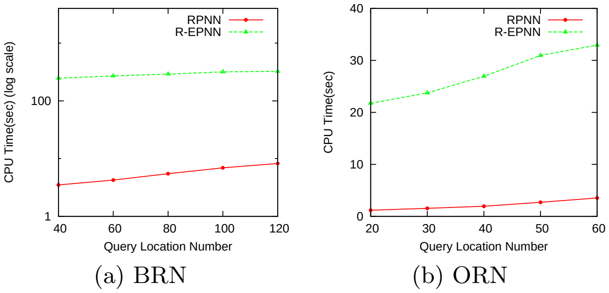
Figure 4: Effect of the trajectory number

[6] http://www.iscas.ac.cn/

[7] www.cs.fsu.edu/ lifeifei/SpatialDataset.htm

First of all, we investigated the effect of the trajectory number on the performance of R-PNN and R-EPNN with the default settings. Figure 4 shows the performance of R-PNN and R-EPNN when the number of trajectories varies. Intuitively, a larger trajectory data-set causes more trajectories to be processed during the query processing, and the CPU time is expected to be higher for both R-PNN and R-EPNN. However, the CPU time of R-EPNN increased much faster than R-PNN for two reasons. Firstly, every trajectory in the data-set should be addressed by an individual PNN query in R-EPNN and its query time increases linearly with the number of trajectories. In the meantime, in R-PNN, a tight trajectory candidate-set is specified, and larger values of $\lfloor \frac{T_{\cdot num}}{O_{\cdot num}} \rfloor$ can further enhance the pruning effect when identifying candidates. The second reason is due to the much larger number of candidates (data objects to be checked) when using bi-direction network expansion method in the original PNN query [5]. For instance, when $T_{\cdot num}$ is equal to 1600 and 350 in Figure 4(a) and Figure 4(b) respectively, R-PNN outperforms R-EPNN by almost three orders of magnitude. Note that this result demonstrates the importance of smart selection of trajectory candidates in the first search phase and the necessity of PNN query optimization in the second phase.

4.2 Effect of the Query Location Number $O_{\cdot num}$



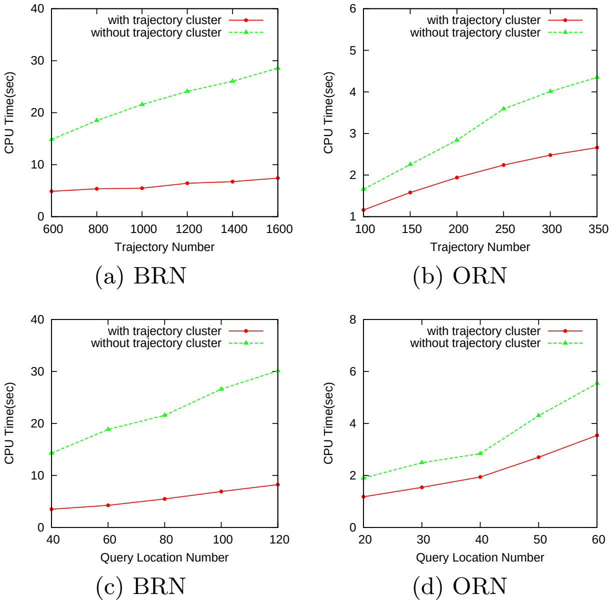
(a) BRN (b) ORN

Figure 5: Effect of the query location number

In Figure 5, the experiment results demonstrate the effect of the query location number $O_{\cdot num}$ on the performance of R-PNN and R-EPNN. Intuitively, a larger number of query locations may lead to higher time cost in the query processing for both R-PNN and R-EPNN. In Figure 5(a) 5(b), the CPU time of R-EPNN increased linearly with the number of query locations. For R-PNN, a larger $O_{\cdot num}$ value will result in the decrease of $\lfloor \frac{T_{\cdot num}}{O_{\cdot num}} \rfloor$ ($T_{\cdot num}$ is fixed), and thus weakens the pruning effect in the first search phase. Consequently, in both Figure 5(a) and Figure 5(b), the CPU times of R-PNN increased slowly with the query location number. Nevertheless, the CPU time required by EPNN was two orders of magnitude higher than that of R-PNN.

4.3 Effect of Trajectory Clustering

This experiment investigated the effect of trajectory clustering (Section 3.1) on the performance of R-PNN. This preprocessing technique is used to further enhance the R-PNN query efficiency: (1) it can tighten the search range during the query processing; (2) it allows the use of a divide-and-conquer strategy. R-PNN was run with and without the trajectory clustering technique respectively. When the technique was not in use, the centroid of the whole trajectory data-set was used in the network expansion. In Figure 6, it is clear that the performance was accelerated by 2-4 times with the help of trajectory cluster.



(a) BRN (b) ORN

(c) BRN (d) ORN

Figure 6: Effect of trajectory clustering

4.4 Effect of PNN Optimization Techniques

Figure 7(a) and Figure 7(b) demonstrate the effect of PNN optimization techniques (Section 3.4) on the candidate ratio with different levels of data density. In our experiments, data points were uniformly generated on the networks with densities from 5% to 25%.

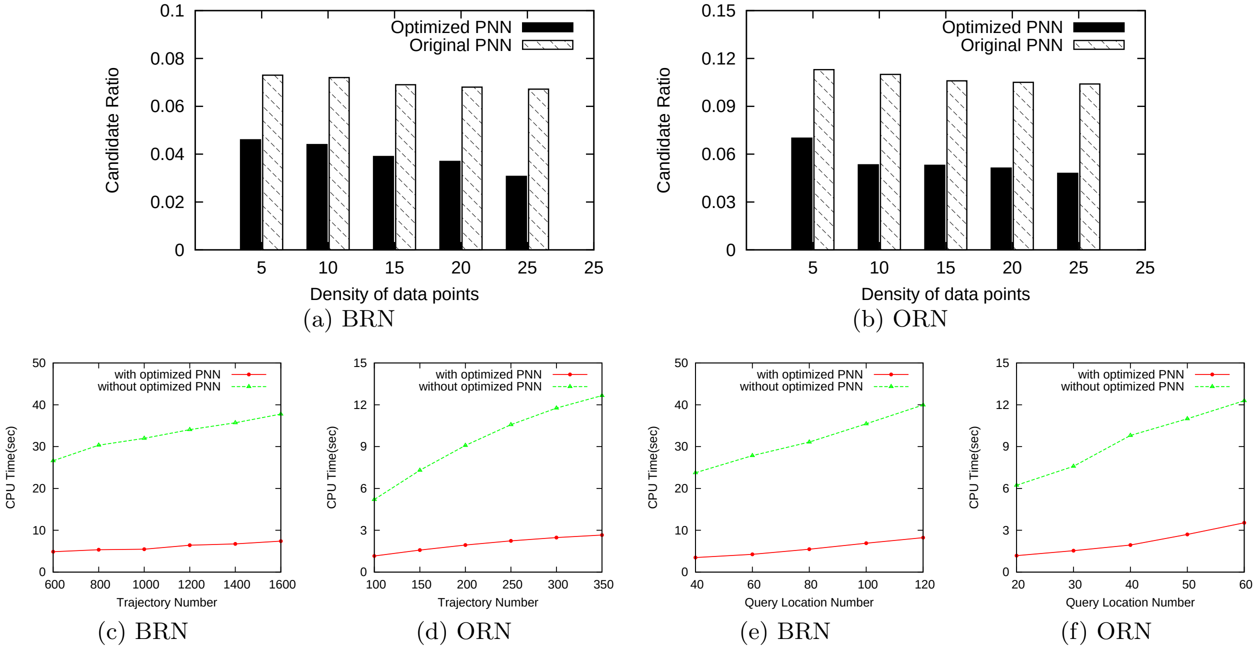
$$Density\ of\ data\ points = \frac{O_{\cdot num}}{V_{\cdot num}} \tag{13}$$

$$Candidate\ Ratio = \frac{CS_{\cdot size}}{O_{\cdot num}} \tag{14}$$

where O is the data point set, V is the vertex set in the network and CS is the data point candidate set specified by the PNN query. A lower candidate ratio means a higher pruning effect.

Intuitively, the higher the density of data points, the smaller the required search range. In Figure 7(a) and Figure 7(b), the candidate ratio decreased while the density increased for both optimized PNN and original PNN [5]. Obviously, in both BRN and ORN, our optimized PNN employed a much higher pruning effect and the corresponding candidate ratio was only 1/2 to 1/3 of the original PNN.

We also tested the effect of PNN optimization techniques on the performance of R-PNN. The optimized PNN can avoid processing a large number of data point candidates and improve the performance notably. We run R-PNN with and without the PNN optimization techniques respectively. When the optimized PNN was not in use, the original PNN



(a) BRN

(b) ORN

(c) BRN

(d) ORN

(e) BRN

(f) ORN

Figure 7: Effect of PNN optimization techniques

[5] was conducted instead. In Figures 7(c) 7(d) 7(e) 7(f), among all four sub-figures, it is easy to find that without the optimized PNN, the corresponding CPU times increased dramatically. By contrast, the CPU time of R-PNN with the optimized PNN only increased slightly.

5. RELATED WORK

Reverse Nearest Neighbor (RNN) problems have been extensively studied in the last two decades [21, 11, 16, 19, 15, 9, 10, 20, 18, 2, 4, 23]. Let O and Q be two data-sets in the same space. Given a query point $q \in Q$, a RNN query finds all the data points $o \in O$ whose nearest neighbors (NN) in Q are q. This means that there does not exist any other point $q' \in Q$ such that $dist(o, q') < dist(o, q)$.

The concept of RNN query was first proposed in [10] by Korn et al. Their idea is to pre-compute the distance between each data point o and its nearest site p. Thus, each $o \in O$ corresponds to a circle, whose center is the data point o and whose radius is $dist(o, p)$. Given a query point q, its RNN is just the points that contains q in its circle. Similar approaches were adopted in [21] and [11]. In some of the following studies [15, 16, 19], pre-computation is not a necessary part in the RNN query processing and a similar mechanism is shared by the proposed techniques. First, the space that cannot contain any RNN is pruned. Second, the candidates are identified as the data points lie within the remaining space. Third, a range query is conducted for each candidate to check whether the query point is its NN or not.

However, all of these techniques fail to address the R-PNN problem due to three reasons. First, in R-PNN query, we consider the relationship between trajectories and locations (data points), rather than points and points (e.g., [23]). Second, in R-PNN, the query input is a set of location candidates, not a single point. Third, the movement of moving objects is constrained in spatial networks, rather than the free space (e.g., Euclidean space).

Path Nearest Neighbor (PNN) is first proposed as In-route Nearest Neighbor (IRNN) [14, 24], which is designed for users that drive along a fixed path routinely. As this kind of drivers would like to follow their preferred routes, IRNN queries are proposed for finding the nearest neighbor with the minimum detour distance from the fixed route, based on the assumption that a commuter will return to the route after visiting the nearest facility (e.g., petrol station, bank, post office, etc.) and will continue the journey along the previous route. Recently, k-PNN proposed by Chen et al. in [5] is an extension of the IRNN query. k-PNN can efficiently monitor the k nearest neighbors to a continuously changing shortest path, and the user only needs to input the destination rather than exactly the whole query path. The models of IRNN and k-PNN are both based on the same assumption that the user prefers to follow their previous route with the minimal detour distance. In IRNN query and PNN query, solutions based on R-tree and network expansion are adopted respectively. As an intuitive extension of the conventional PNN query, Shang et al. [13] proposed the best point detour (BPD) query, which aims to identify the point detour with the minimum detour cost. Furthermore, they investigated the continuous-BPD query in the scenario where the path continuously changes when a user is moving to the destination.

6. CONCLUSION

In this paper, we proposed and investigated a novel Reverse Path Nearest Neighbor (R-PNN) query to find the most accessible location in road networks. This type of queries can be found in many important applications such as urban planning, facility allocation, traffic monitoring, location based services, etc. To address the R-PNN query efficiently, a trajectory clustering technique is employed initially. Based on the grouped trajectory data, a two-phase algorithm is proposed. In the first phase, our approach can specify the minimum search range when identifying the trajectory/data-point candidates. In the second phase, our PNN optimization technique is optimal in time cost as shown by the complexity analysis. Extensive experiments were also conducted to demonstrate the near optimal query performance of the proposed R-PNN query.

7. ACKNOWLEDGEMENT

We wish to thank the anonymous reviewers for several comments and suggestions that have improved the paper. This work was supported by the ARC grant DP110103423 and the National Natural Science Foundation of China (No.60905030).

8. REFERENCES

[1] H. Alt, A. Efrat, G. Rote, and C. Wenk. Matching planar maps. In *SODA*, pages 589–598, 2003.

[2] R. Benetis, C. S. Jensen, G. Karciauskas, and S. Saltenis. Nearest and reverse nearest neighbor queries for moving objects. In *VLDB J*, pages 229–249, 2006.

[3] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk. On map-matching vehicle tracking data. In *VLDB*, pages 853–864, 2005.

[4] M. A. Cheema, X. LIN, W. Zhang, and Y. Zhang. Influence zone: Efficiently processing reverse k nearest neighbors queries. In *ICDE*, 2011.

[5] Z. Chen, H. T. Shen, X. Zhou, and J. X. Yu. Monitoring path nearest neighbor in road networks. In *SIGMOD*, pages 591–602, 2009.

[6] E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische Math*, 1:269–271, 1959.

[7] H. Gonzalez, J. Han, X. Li, M. Myslinska, and J. Sondag. Adaptive fastest path computation on a road network: A traffic mining approach. In *VLDB*, pages 794–805, 2007.

[8] J. Greenfeld. Matching gps observations to locations on a digital map. In *81th Annual Meeting of the Transportation Research Board*, 2002.

[9] J. M. Kang, M. F. Mokbel, S. Shekhar, T. Xia, and D. Zhang. Continuous evaluation of monochromatic and bichromatic reverse nearest neighbors. In *ICDE*, pages 806–815, 2007.

[10] F. Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *SIGMOD*, pages 201–212, 2000.

[11] K.-I. Lin, M. Nolen, and C. Yang. Applying bulk insertion techniques for dynamic reverse nearest neighbor problems. In *IDEAS*, 2003.

[12] K. Liu, K. Deng, Z. Ding, M. Li, and X. Zhou. Moir/mt: Monitoring large-scale road network traffic in real-time. In *VLDB*, pages 1538–1541, 2009.

[13] S. Shang, K. Deng, and K. Xie. Best point detour query in road networks. In *ACM GIS*, pages 71–80, 2010.

[14] S. Shekhar and J. S. Yoo. Processing in-route nearest neighbor queries: a comparison of alternative approaches. In *ACM GIS*, pages 9–16, 2003.

[15] I. Stanoi, M. Riedewald, D. Agrawal, and A. E. Abbadi. Finch: Evaluating reverse k-nearest-neighbor queries on location data. In *VLDB*, 2001.

[16] Y. Tao, D. Papadias, and X. Lian. Reverse knn search in arbitrary dimensionality. In *VLDB*, 2004.

[17] C. Wenk, R. Salas, and D. Pfoser. Addressing the need for map-matching speed: Localizing globalb curve-matching algorithms. In *SSDBM*, 2006.

[18] R. C.-W. Wong, M. T. Ozsu, P. S. Yu, A. W.-C. Fu, and L. Liu. Efficient method for maximizing bichromatic reverse nearest neighbor. In *VLDB*, pages 1126–1137, 2009.

[19] W. Wu, F. Yang, C. Y. Chan, and K.-L. Tan. Finch: Evaluating reverse k-nearest-neighbor queries on location data. In *VLDB*, 2008.

[20] T. Xia, D. Zhang, E. Kanoulas, and Y. Du. On computing top-t most influential spatial sites. In *VLDB*, pages 946–957, 2005.

[21] C. Yang and K.-I. Lin. An index structure for efficient reverse nearest neighbor queries. In *ICDE*, 2001.

[22] M. L. Yiu and N. Mamoulis. Clustering objects on a spatial network. In *SIGMOD*, pages 443–454, 2004.

[23] M. L. Yiu, D. Papadias, N. Mamoulis, and Y. Tao. Reverse nearest neighbors in large graphs. *IEEE TKDE*, 18(4):540–553, 2006.

[24] J. S. Yoo and S. Shekhar. In-route nearest neighbor queries. *GeoInformatica*, 9:117–137, 2005.