Noname manuscript No.
(will be inserted by the editor)

PNN Query Processing on Compressed Trajectories

Shuo Shang · Bo Yuan · Ke Deng ·
Kexin Xie · Kai Zheng · Xiaofang Zhou

Received: Mar 12, 2011 / Accepted: Sep 26, 2011

Abstract Trajectory compression is widely used in spatial-temporal databases as it can notably reduce (*i*) the computation/communication load of clients (GPS-enabled mobile devices) and (*ii*) the storage cost of servers. Compared with original trajectories, compressed trajectories have clear advantages in data processing, transmitting, storing, etc. In this paper, we investigate a novel problem of searching the Path Nearest Neighbor based on Compressed Trajectories (PNN-CT query). This type of query is conducted on compressed trajectories and the target is to retrieve the PNN with the highest probability (lossy compression leads to the uncertainty), which can bring significant benefits to users in many popular applications such as trip planning. To answer the PNN-CT query effectively and efficiently, a two-phase solution is proposed. First, we use the meta-data and sample points to specify a tight search range. The key of this phase is that the number of data objects/trajectory segments to be processed or decompressed should be kept as small as possible. Our efficiency study reveals that the candidate sets created are tight. Second, we propose a reconstruction algorithm based on probabilistic models to account for the uncertainty when decompressing the trajectory segments in the candidate set. Furthermore, an effective combination strategy is adopted to find the PNN with the highest probability. The complexity analysis shows that our solution has strong advantages over existing methods. The efficiency of the proposed PNN-CT query processing is verified by extensive experiments based on real and synthetic trajectory data in road networks.

Shuo Shang · Ke Deng · Kexin Xie · Kai Zheng · Xiaofang Zhou
School of Information Technology & Electrical Engineering, The University of Queensland
E-mail: shangs@itee.uq.edu.au

Bo Yuan
Division of Informatics, Graduate School at Shenzhen, Tsinghua University
E-mail: yuanb@sz.tsinghua.edu.cn

Keywords Compressed Trajectory · Path Nearest Neighbor · Road Networks · Spatial Databases

1 Introduction

Path Nearest Neighbor (PNN) Query [8,26,36] is a well defined problem in spatial databases, which is designed to find the closest data object to the specified path. Different from traditional Nearest Neighbor (NN) query [13, 24], PNN is a globally optimal choice for the nearest neighbor to a given path. We noticed that travelers in many situations cannot specify a preferred path, especially in an unfamiliar region, for example, when traveling overseas. In this situation, consulting the travel history (i.e., trajectories) of other people as reference is a sensible choice and is getting more and more popular along with the development of online LBS (Location Based Services). We assume that a traveler has already selected a trajectory shared by others in route-sharing web sites [1] as the preferred path and wants to know the nearest facility (e.g., supermarket, post office, etc.) relative to the route he/she will travel.

In practice, most of the existing trajectory data are uploaded and stored in compressed format. Compared with the original trajectories, compressed trajectories have clear advantages in data processing, transmitting, storing, etc. [20]. However, apart from the existing challenge created by the large number of data objects, compressed trajectory data present new challenges to the original PNN query [8]. First, due to the lossy compression, uncertainty must be taken into account in the trajectory decompression models and query processing algorithms. The impact of probabilistic and uncertain data, especially uncertain trajectories, has been extensively investigated in many recent studies [33,34,27,32,22,37]. Second, to reconstruct the original trajectory, the full decompression process may prevent the query from being answered promptly, due to its intolerable time and storage cost. In this work, we investigate the problem of how to find the PNN based on compressed trajectories (PNN-CT Query) effectively and efficiently. A series of novel algorithms are proposed and the complexity analysis shows that our solution has strong advantages over existing methods.

Ideally, a trajectory compression method should notably reduce (*i*) the computation/communication load of clients (GPS-enabled mobile devices), and (*ii*) the storage cost of servers. Despite the bulk of trajectory compression literature [10,19,4,3,16,7], no solution fulfills both requirements, except the linear prediction model based trajectory compression [30,21,14,25,31,17]. Linear prediction models assume that the moving object follows linear movements, and introduce little computation and storage overhead. On the client-side, it

[1] Bikely (http://www.bikely.com/)
GPS-Waypoints (http://www.gps-waypoints.net/)
Share My Routes (http://www.sharemyroutes.com/)
Microsoft GeoLife (http://research.microsoft.com/en-us/projects/geolife/)

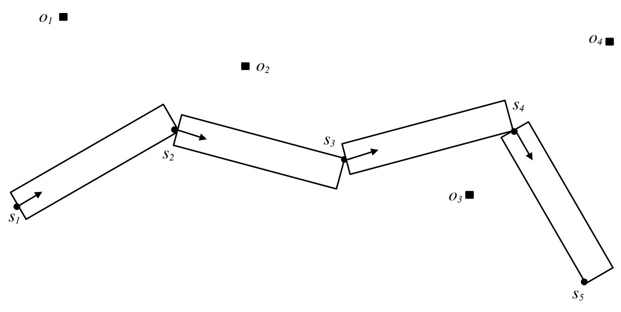


Fig. 1 Linear Prediction Model based Trajectory Compression

is not necessary to record and upload each point in the trajectory unless it breaches the constraint set by the prediction model. On the server-side, since the whole trajectory is compressed into a set of sample points, the pressure of storage is elevated notably.

Each trajectory compression algorithm requires a system specified error bound δ. For $\delta = 0$, the compression is lossless. Otherwise, it is a lossy compression process. In general, a larger error bound will lead to a better compression ratio, but more information will be lost in the compression process, which will result in a higher level of uncertainty when decompressing the trajectory. Consequently, the path connecting the two adjacent sample points is usually not unique. A straightforward method is to reconstruct a deterministic trajectory (e.g., via interpolation), and then solve the problem in a deterministic manner. However, this solution is insufficient, as it considers only a single (e.g., the most optimistic) scenario, which may be infeasible in practice. Motivated by these observations, in this paper, we explicitly incorporate uncertainty into trajectories on road networks and propose a novel probabilistic PNN queries.

An example of the linear prediction model based trajectory compression is demonstrated in Figure 1. Initially, a moving object is at the source s_1, and its current time-stamp, speed and heading are also recorded as the meta-data of s_1. Then, based on the heading vector and error bound, a rectangular moving range is defined. At point s_2, the moving object leaves the predefined moving range, and s_2 needs to be recorded with the updated meta-data. Similarly, points s_3, s_4 and s_5 are recorded and s_5 is the destination. Hence, a trajectory is compressed into a series of sample points $\{s_1, s_2, s_3, s_4, s_5\}$ with a set of meta-data. Consequently, given a compressed trajectory, our target is to reconstruct all possible trajectories connecting s_1 and s_5 and find the PNN with the highest probability among data objects o_1, o_2, o_3, o_4.

An intuitive approach to solving PNN-CT query is to fully reconstruct all possible trajectories connecting the source and destination, and perform the PNN query [8] on the decompressed trajectories. It is not practical for two reasons. First, due to the lossy compression, there may exist several possible sub-paths within the same trajectory segment. Suppose n is the number of vertices in each segment (rectangle), k is the number of trajectory segments, and m is the number of possible sub-paths in one segment. The complexity of the decompression process is $O(kn^2 + m^k)$, which can become very time consuming for moderate data sets. Second, there are m^k reconstructed trajectories and the original PNN query [8] needs to be performed m^k times. Obviously, the corresponding time and space cost may easily prevent the PNN-CT query from being answered promptly. Note that, in this situation, the original PNN query is inadequate to specify a tight candidate set for either data objects or trajectory segments, due to its loose upper/lower bound.

To process the PNN-CT query effectively and efficiently, a two-phase solution is proposed. First, we use the sample points and the corresponding meta-data to specify a tight searching range. Here, the network expansion method [9] and branch-and-bound strategy are adopted. In this phase, most of the data objects and trajectory segments can be pruned. The efficiency study reveals that the candidate sets created are tight. Second, we propose a reconstruction algorithm based on probabilistic models to account for the uncertainty when decompressing the remaining trajectory segments. The proposed probabilistic model is logically sound and all possible trajectories are taken into account with each of them assigned a corresponding probability. Furthermore, an effective combination strategy is adopted to find the PNN with the highest probability and the complexity analysis shows that our solution has strong advantages over existing methods. Note that the proposed scheme represents a general paradigm for handling compressed trajectory data and is independent of the specific underlying probabilistic model. To sum up, the major contributions of this paper are:

– A novel type of query to find the Path Nearest Neighbor based on Compressed Trajectories (PNN-CT). It provides new features for advanced spatial information systems, and benefits users in many popular applications such as trip planning.
– A probabilistic model to evaluate the uncertainty when decompressing the trajectory segments and an efficient method for trajectory decompression.
– A two-phase algorithm to answer the PNN-CT query effectively and efficiently. The efficiency study reveals that the candidate sets created are tight and the complexity analysis shows that our solution has strong advantages over existing methods.
– Extensive experiments on real and synthetic trajectory data to investigate the performance of the proposed approaches.

Note that in this work we are dealing with paths(routes) and spatial distance, and using the meta-data (e.g., time-stamp, maximum speed and heading vector) only in the path reconstruction and pruning procedures.

The rest of the paper is organized as follows. Section 2 introduces the trajectory compression and decompression models used in this paper as well as problem definitions. The PNN-CT 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 some discussions on related work in Section 5.

2 Preliminaries

2.1 Road Networks

In this work, road networks are modeled as connected and undirected 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. 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 objects are distributed along roads. If a data object is not located at a road intersection, we treat the data object as a vertex and further divide the edge it lies on into two edges. Thus, we assume that all data objects are in vertices for the sake of clarity. Note that in this work, the data objects are static.

2.2 Trajectory Compression

The linear prediction model [30,21,14,25,31] is adopted in this work, which can notably reduce (*i*) the computation/communication load of clients (GPS-enabled mobile devices) and (*ii*) the storage cost of servers. In practice, the compression error bound (deviation threshold) δ is specified by the system. A larger error bound often leads to a better compression ratio but more information will be lost in the compression process. If the moving object m deviates from the predicted path over the error bound, its current position needs to be recorded with some meta-data, such as its current time-stamp t, maximum speed v, heading vector h_v, etc. The vertices between two adjacent sample points are ignored. Hence, the trajectory is compressed into a series of sample points and a set of corresponding meta-data.

Definition: Deviation Distance
Given a moving object m, a sample point s and a heading vector $\overrightarrow{h_v}$, the deviation distance of m is

$$Deviation(m) = d_E(m, s) \times |sin(\theta)| \tag{1}$$

where θ is the intersection angle between $\overrightarrow{h_v}$ and $\overrightarrow{(s, m)}$, $d_E(m, s)$ is the Euclidean distance between m and s.

Definition: Moving Object Prediction Model PM
Given a moving object m, a source (or a sample point) s, a heading vector $\overrightarrow{h_v}$ and an error bound δ, we assume that the deviation distance of m is always less than δ. Otherwise, the current position of m needs to be recorded with its current time-stamp t, speed v and heading vector h_v.

An example is described in Figure 2. Suppose an object m is moving from source s with initial heading vector h_v. m_1, m_2 are the possible positions of

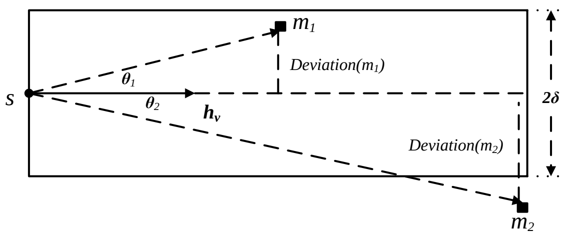


Fig. 2 Linear Prediction Model

data object m at time t_1, t_2 respectively. δ is the error bound specified by the system. The deviation distance of m_1 is $d_E(s, m_1) \times |sin(\theta_1)| < \delta$, thus it is not necessary to record m_1. By contrast, $Deviation(m_2) = d_E(s, m_2) \times |sin(\theta_2)| > \delta$, and the data point m_2 should be recorded with the corresponding time-stamp t_2, speed v_2 and heading vector h_{v2}.

Definition: Compressed Trajectory
A compressed trajectory CT in a road network G is a finite sequence of positions: $CT = s_1, s_2, ..., s_n$, where s_i is the sample point in G, for $i = 1, 2, ..., n$. Each sample point has a set of meta-data, including time-stamp t, speed v and heading vector h_v.

The sample points 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. In this work, we assume that all sample points have already been aligned to the vertices on the road network by some map-matching algorithms [2,6,12,35].

2.3 Trajectory Segment Decompression

Given a compressed trajectory segment $T_{seg}(s_i, s_j)$ connecting two adjacent sample points s_i, s_j, it is difficult to find the exact path $P(s_i, s_j)$ between s_i, s_j due to the compression loss. Here, we propose a probabilistic model to evaluate the uncertainty of the object's moving path. All possibilities have been considered and each of them is assigned a corresponding probability. We assume that the object m moves randomly between s_i and s_j, but its moving range is constraint by three conditions.

- First, $P(s_i, s_j)$ should not be over the constraint range of rectangle R, where R is the rectangle defined by sample points s_i, s_j, heading vector h_v and error bound δ. Otherwise, there must exist another sample point between s_i, s_j.
- Second, the minimum moving time between s_i, s_j is constrained by $(t_j - t_i)$, where t_i, t_j are time-stamps of s_i, s_j respectively.

– Third, there is no loop in $P(s_i, s_j)$, which means one point can not appear twice in one path.

The length of path $P = \{n_1, ..., n_k\}$ is defined as the sum of weights of all edges in P, that is

$$Length(P(n_1, ...n_k)) = \sum_{i=1}^{k-1}(weight(n_i, n_{i+1})) \tag{2}$$

The minimum moving time of path $P = \{n_1, ..., n_k\}$ is defined as follows:

$$Time(P(n_1, ...n_k)) = \sum_{i=1}^{k-1}(weight(n_i, n_{i+1})/v(n_i, n_{i_1})) \tag{3}$$

where $v(n_i, n_{i_1})$ is the maximum moving speed along the edge (n_i, n_{i+1}). Obviously, $Time(P(n_1, ...n_k))$ should be no grater than $(t_k - t_1)$.

And the probability of $P = \{n_1, ..., n_k\}$ is defined as the product of each edges' probability.

$$Prob(P(n_1, ...n_k)) = \prod_{i=1}^{k-1}(Prob(n_i, n_{i+1})) \tag{4}$$

Equation 2 & 4 are also applicable when combining sub-paths from different trajectory segments to create a full path $P(s, d) = \{P_1, P_2, ..., P_k\}$ connecting the source s and the destination d.

Algorithm 1: Trajectory Decompression

```
   Data: Adjacent sample points $s_i, s_j$
   Result: $Pathlist(s_i, s_j)$
1  $TD\_Function(s_i)$
2  for each vertex $n \in s_i.adjacentList$ do
3      if $n \notin R$ then
4          Continue;
5      if $n \in path(s_i).vertex$ then
6          Continue;
7      if $path(s_i).time + sd(s_i, n)/v(s_i, n) > t_j - t_i$ then
8          Continue;
9      $path(n).vertex \leftarrow path(s_i).vertex \cup n$;
10     $path(n).time \leftarrow path(s_i).time + sd(s_i, n)/v(s_i, n)$;
11     $path(n).prob \leftarrow path(s_i).prob/s_i.validNeighbors.size$;
12     if $n = s_j$ then
13         $Pathlist.add(path(n))$;
14         $path(n).clear()$;
15         Continue;
16     $TD\_Function(n)$;
17     $path(n).clear()$;
18 return;
```

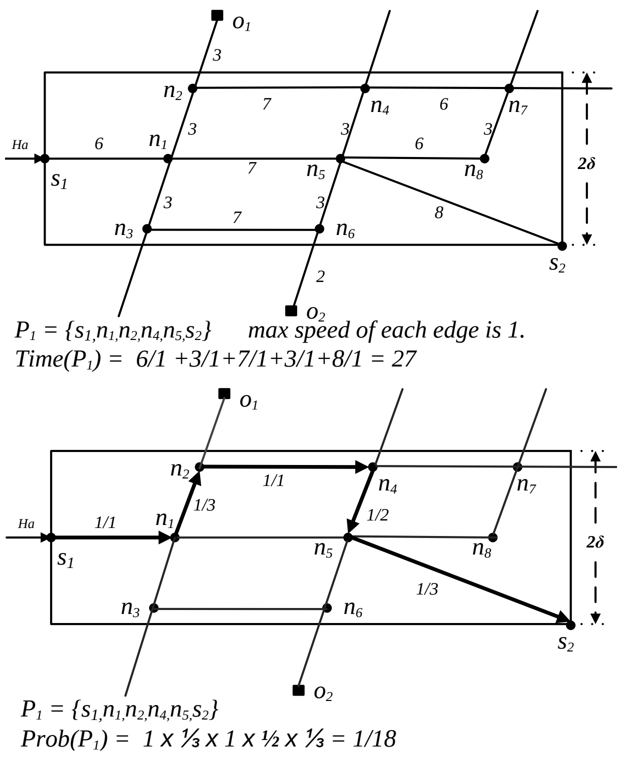


Fig. 3 Probabilistic Model

In general, all valid sub-paths in trajectory segment $T_{seg}(s_i, s_j)$ can be retrieved according to Algorithm 1. Suppose s_i, s_j are two adjacent sample points and $TD\_Function(s_i)$ is a recursive function, which conducts a depth-first transversal of all possible sub-paths. Initially, all adjacent vertices of s_i will be visited iteratively (*line* 2). For each adjacent vertex n, we check whether it conflicts three conditions: n should be contained by rectangle R (*line* 3-4), n should not appear twice in the same path (*line* 5-6) and the traveling time of the path should not be greater than $(t_j - t_i)$ (*line* 7-8). If n satisfies all three conditions, the information of $path(n)$ is recorded, including its vertices, distance and probability (*line* 9-11). Once the destination s_j is reached, one valid sub-path is found and stored in $Pathlist(s_i, s_j)$ (*line* 12-15). Otherwise, a new recursive function based on n will be conducted (*line* 16).

In Figure 3 we give an example of trajectory segment decompression. s_1, s_2 are the adjacent sample points, and $n_1, n_2, ..., n_8$ are vertices in the road network and o_1, o_2 are data objects. Each road segment is specified a weight to represent its network distance. The maximum moving speed along each edge is 1 and the moving time between s_1, s_2 is $(t_2 - t_1) = 30$, where t_1, t_2 are the time-stamps of s_1, s_2 respectively.

It is easy to find that there are three sub-paths P_1, P_2, P_3 which satisfy all three conditions stated above. For $P_1 = \{s_1, n_1, n_2, n_4, n_5, s_2\}$, according to Equation 3, it is easy to compute $Time(P_1) = 6/1 + 3/1 + 7/1 + 3/1 + 8/1 = 27$. Compared with the moving time computation, the computation of probability is more complex. The first edge of P_1 is (s_1, n_1) and its probability is $\frac{1}{1}$. Since we assume the moving object moves randomly, at the intersection n_1, the moving object m has four choices: $\{s_1, n_2, n_3, n_5\}$ for the next step. However, vertex s_1 is invalid in this case as a loop $< s_1, n_1, s_1 >$ will be formed. As a result, the rest three vertices share the same probability $\frac{1}{3}$. Next, at intersection n_2, n_4 holds $\frac{1}{1}$ probability to be the next point as o_1 is out of the range and n_1 will form a loop. Following the procedure, the probability of P_1 can be computed based on Equation 4. That is $Prob(P_1) = \frac{1}{1} \times \frac{1}{3} \times \frac{1}{1} \times \frac{1}{2} \times \frac{1}{3} = \frac{1}{18}$. Similarly, we can compute the moving time and probability of $P_2 = \{s_1, n_1, n_5, s_2\}$ and $P_3 = \{s_1, n_1, n_3, n_6, n_5, s_2\}$. P_2 has the moving time 21 and the probability $\frac{1}{1} \times \frac{1}{3} \times \frac{1}{4} = \frac{1}{12}$. P_3 has the moving time 27 and the probability $\frac{1}{1} \times \frac{1}{3} \times \frac{1}{1} \times \frac{1}{1} \times \frac{1}{3} = \frac{1}{9}$.

Since all other paths are invalid and pruned, the probabilities of P_1, P_2, P_3 need to be normalized according to Equation 5.

$$Prob_N(P_i) = \frac{Prob(P_i)}{\sum_{i=1}^{k}(Prob(P_i))} \tag{5}$$

where $Prob_N(P_i)$ is the normalized probability of P_i. Consequently, $Prob_N(P_1) = \frac{1/18}{1/4} = \frac{2}{9}$, $Prob_N(P_2) = \frac{1/12}{1/4} = \frac{1}{3}$ and $Prob_N(P_3) = \frac{1/9}{1/4} = \frac{4}{9}$. At this stage, the trajectory segment $T_{seg}(s_i, s_j)$ is decompressed into three possible sub-paths P_1, P_2, P_3 and the probability for each of them is also available.

2.4 Probabilistic Path Nearest Neighbor

Given any two locations a, b in road networks, 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 T_{raj} and a data object o in road networks, the minimum detour distance $d_D(o, T_{raj})$ of data object o from T_{raj} is defined as

$$d_D(o, T_{raj}) = \min_{n_i \in T_{raj}} \{sd(o, n_i)\}, \tag{6}$$

Definition: Path Nearest Neighbor
Given a trajectory T_{raj} and a set of data object O, the path nearest neighbor (PNN) of T_{raj} is the data object $o \in O$ with the minimum $d_D(o, T_{raj})$, such that $d_D(o, T_{raj}) \leq d_D(o', T_{raj}), \forall o' \in \{O - o\}$.

Definition: Probabilistic Path Nearest Neighbor
Given a path P and its PNN o, o holds the same probability as P, that is $Prob(o) = Prob_N(P)$. At the meantime, if o is the PNN of multiple paths

$P_1, P_2, ..., P_n$, the probability of o is

$$Prob(o) = \sum_{i=1}^{n} Prob_N(P_i) \tag{7}$$

As shown in Figure 3, it is easy to find that the PNN of P_1 is data object o_1, and $d_D(o_1, P_1) = 3$. For sub-path P_2, its PNN is data object o_2 and $d_D(o_2, P_2) = 5$. Sub-path P_3's PNN is also data object o_2 and $d_D(o_2, P_3) = 2$. As computed in Section 2.3, the normalized probability of sub-path P_1 is $\frac{2}{9}$, and o_1 takes probability $Prob(o_1) = \frac{2}{9}$ to be the PNN of trajectory segment $T_{seg}(s_1, s_2)$. In the meantime, data object o_2 is the PNN of both P_2 and P_3, thus o_2 holds the probability $Prob(o_2) = Prob_N(P_2) + Prob_N(P_3) = \frac{7}{9}$ to be the PNN of $T_{seg}(s_1, s_2)$. Therefore, data object o_2 is the PNN with the highest probability to the compressed trajectory segment $T_{seg}(s_1, s_2)$.

2.5 Problem Definition

Definition: Compressed Trajectory based Path Nearest Neighbor Query (PNN-CT)
Given a compressed trajectory CT, a set of data object O, a moving object prediction model PM and the meta-data (time-stamp t, maximum speed v and heading vector h_v) of each sample point $s \in CT$, Compressed Trajectory based Path Nearest Neighbor Query (PNN-CT) finds the data object o with the highest probability to be the Path Nearest Neighbor (PNN) of CT, such that $Prob(o) \geq Prob(o'), \forall o' \in \{O - o\}$.

3 PNN-CT Query Processing

Intuitively, the Compressed Trajectory based Path Nearest Neighbor (PNN-CT) query can be solved by decompressing all trajectory segments and preform a traditional PNN query processing [8] on all possible paths. However, in order to reconstruct the paths connecting the source and the destination, the valid sub-paths in each trajectory segment must be connected sequentially with those in the neighboring segments. The number of all possible paths may be extremely large, which prevents the query from being answered promptly. To overcome this weakness and solve the PNN-CT query effectively and efficiently, a two-phase solution is proposed. In the first phase, we use the meta-data and sample points to specify a tight search range (Section 3.1). The efficiency study reveals that the candidate sets for both data objects and trajectory segments are tight (Section 3.2). The second phase introduces the probabilistic PNN query processing based on an effective combination strategy (Section 3.3). The complexity analysis shows that our solution has strong advantages over existing methods (Section 3.4).

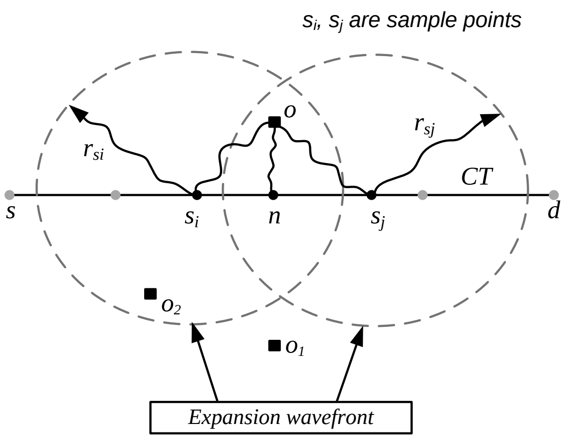


Fig. 4 Identifying Candidates

3.1 Identifying Candidates

Given a compressed trajectory CT, a set of data objects O, a moving object prediction model PM and the meta-data (time-stamp t, maximum speed v and heading vector h_v) for each sample point $s \in CT$, we try to specify a tight search range based on sample points and meta-data. In our approach, we estimate the minimum detour distance of a data object o from the compressed trajectory CT by a pair of lower bound $o.lb$ and upper bound $o.ub$.

Consider the schematic example demonstrated in Figure 4. There is a compressed trajectory CT connecting a source s and a destination d, while s_i, s_j are two adjacent sample points and o, o_1, o_2 are data objects. n is a vertex on the compressed trajectory segment $T_{seg}(s_i, s_j)$ (i.e., n is on a possible path connecting s_i and s_j). An upper bound of the minimum detour distance from o to CT is given by:

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

This upper bound can be further tightened and a global optimal upper bound is proposed, which will be discussed later in this section. Next, we estimate the lower bound of o based on the triangular inequality of shortest path. The triangular 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. In Figure 4, suppose (o, n) is the minimum detour from data object o to the trajectory segment $T_{seg}(s_i, s_j)$. According to the triangular inequality introduced above, we have:

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

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

Then, we adopt $d(s_i,n)$ to denote the network distance between s_i and n along the trajectory segment $T_{seg}(s_i,s_j)$ (not necessarily a network shortest path). Based on the definition of shortest path, it is easy to find that $d(s_i,n) \geq sd(s_i,n)$ and $d(n,s_j) \geq sd(n,s_j)$. Thus, $d(s_i,s_j) = d(s_i,n) + d(n,s_j) \geq sd(s_i,n) + sd(n,s_j)$ and we can use $d(s_i,s_j)$ to replace $sd(s_i,n) + sd(n,s_j)$ in the equation:

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

The length of $T_{seg}(s_i,s_j)$ is difficulty to retrieve since $T_{seg}(s_i,s_j)$ is uncertain. Here, we use the time-stamps t_i, t_j and the maximum speed v between s_i, s_j to estimate $d(s_i,s_j)$. These data are stored as meta-data with each sample point. Consequently, we can compute the maximum moving distance between s_i, s_j as $v \times (t_j - t_i)$. Intuitively, $v \times (t_j - t_i) \geq d(s_i,s_j)$. Thus, the lower bound of data object o is represented as

$$o.lb = \frac{sd(o,s_i) + sd(o,s_j) - v \times (t_j - t_i)}{2} \tag{9}$$

Now we expect to identify the candidates from data objects set O based on the upper/lower bound introduced above. Any data object o whose lower bound is greater than any other data object's upper bound will be pruned. For trajectory segment $T_{seg}(s_i,s_j)$, if the shortest path distances from o to s_i, s_j are known, the candidature of o can be determined. Here, Dijkstra's expansion [9] is adopted to select the data object candidates. From each sample point $s \in CT$, a browsing wavefront is expanded in Dijkstra's algorithm. Conceptually, the browsed region is restricted as a circle as shown in Figure 4, where the radius is the shortest network distance from the center s to the browsing wavefront, denoted as r_s.

Among all data objects scanned by the expansion wave, we define a global upper bound UB as

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

where O_s is the set of all scanned data objects, $O_s = \bigcup_{i=1}^{k+1} O_i$ and $(k+1)$ is the number of sample points. An example is demonstrated in Figure 4, where $o, o_2 \in O_s$ and $o_1 \notin O_s$. An efficiency study in Section 3.2 reveals that UB is tight. The expansion of browsing wavefront will be stopped once $\frac{r_{si} + r_{sj} - v \times (t_j - t_i)}{2}$ is greater than UB, where r_{si} (r_{sj}) is the radius of the corresponding browsed region. It can be proved that the data objects outside the browsed region cannot have shortest network distance to $T_{seg}(s_i,s_j)$ less than UB, and can be pruned safely (e.g., o_1).

Lemma 1: For any data object o outside the browsed region, we have $o.lb > UB$ and o can be pruned safely.

Proof: As shown in Figure 4, since the Dijkstra's algorithm always chooses the vertex with the smallest distance label for expansion, we have $sd(o_1, s_i) > r_{si}$ and $sd(o_1, s_j) > r_{sj}$. The browsing wavefront stops when $\frac{r_{si}+r_{sj}-v\times(t_j-t_i)}{2} > UB$, and we have:

$$o_1.lb = \frac{sd(o_1, s_i) + sd(o_1, s_j) - v \times (t_j - t_i)}{2} > UB$$

Therefore, the data objects outside the browsed region can be pruned safely.

The candidates to the i^{th} trajectory segment $T_{seg}(s_i, s_j)$ are:

$$C_i = \{o | \frac{sd(o, s_i) + sd(o, s_j) - v \times (t_j - t_i)}{2} \leq UB, o \in O\} \tag{10}$$

For some candidates in C_i, the network distances to both s_i and s_j are known and for other candidates only the network distance to either s_i or s_j is known when the browsing wavefronts stop expanding. For example, in Figure 4, only the network distance from o_2 to s_i is known. If network distances to s_i, s_j are known, o_2 may not be a candidate according to Equation 10. Therefore, we further continue the network expansion until all candidates in C_i have their lower bounds determined and invalid data objects will be pruned from C_i.

To further tighten the candidate set C_i, a new lower bound of the minimum detour distance based on Euclidean distance is proposed. In the prediction model introduced in Section 2.2, the compressed trajectory segment $T_{seg}(s_i, s_j)$ is represented by a rectangle R, and R is defined by adjacent sample points s_i, s_j, heading-vector h_v and error bound δ. Suppose that n is the closest vertex to data object o in $T_{seg}(s_i, s_j)$. The relationship between n and rectangle R can be expressed as follow:

$$\begin{cases} n \in T_{seg}(s_i, s_j) \\ T_{seg}(s_i, s_j) \subset R \end{cases} \Longrightarrow n \in R$$

Therefore, for any data object o, the lower bound of its minimum detour distance $sd(o, n)$ can be estimated by

$$o.lb = d_E(o, R) \tag{11}$$

where $d_E(o, R)$ is the Euclidean distance between data object o and rectangle R. If data object o is out of the rectangle R, as shown in Figure 5, the rectangle R is mapped into a two-dimensional coordinate system. The position of data object o is described by its coordinates $(o.x, o.y)$. $l(s_i, v_3)$ and $l(o, v_4)$ are two perpendicular straight lines and their intersection point is v_1. The Euclidean distance between o and R is $d_E(o, v_2) = d_E(o, v_1) - \delta = \sqrt{(o.x - v_1.x)^2 + (o.y - v_1.y)^2} - \delta$, when o is out of R. On the other hand, if $o \in R$, we define that $d_E(o, R) = 0$.

$$d_E(o, R) = \begin{cases} \sqrt{(o.x - v_1.x)^2 + (o.y - v_1.y)^2} - \delta \quad (o \notin R) \\ 0 \quad (o \in R) \end{cases}$$

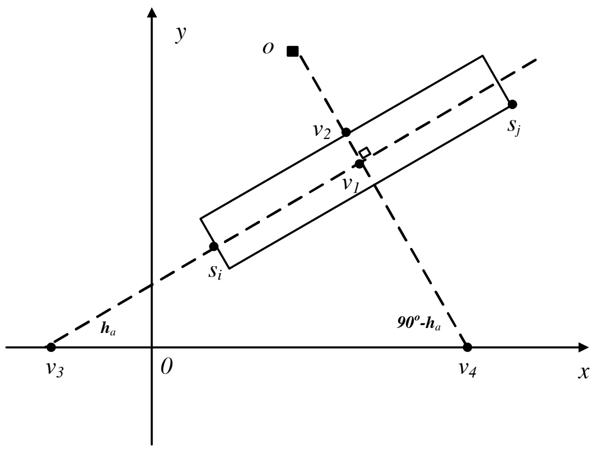


Fig. 5 Detour Distance Lower Bound

Different from Equation 9, $o.lb = d_E(o, R)$ is the lower bound estimated by Euclidean distance. To further tighten the candidate set C_i, the greater one between $\frac{sd(o,s_i)+sd(o,s_j)-v\times(t_j-t_i)}{2}$ and $d_E(o, R)$ is chosen as the lower bound of data object o. In Section 3.2, we prove that this lower bound is tight.

$$o.lb = max\{\frac{sd(o, s_i) + sd(o, s_j) - v \times (t_j - t_i)}{2}, d_E(o, R)\} \tag{12}$$

Then, based on Equation 12, every data object in C_i will be checked whether its lower bound $o.lb$ is greater than UB, as shown in Algorithm 2.

Algorithm 2: Tighten Lower Bound

```
1 for each data object $o \in C_i$ do
2     $o.lb = max\{\frac{sd(o,s_i)+sd(o,s_j)-v\times(t_j-t_i)}{2}, d_E(o, R)\}$;
3     if $o.lb > UB$ then
4         remove $o$ from $C_i$;
```

Sometimes we may have an empty C_i, which means there is no PNN candidate to the i^{th} compressed trajectory segment $T_{seg}(s_i, s_j)$. Therefore, in the following search, it is not necessary to decompress $T_{seg}(s_i, s_j)$ if $C_i = 0$. Here, we use T to denote the set of compressed trajectory segments whose data object candidate set is not empty.

To identify the complete data object candidate set to the whole compressed trajectory CT, the candidates to the individual trajectory segments are combined. Since one data object may be candidate to more than one trajectory segments, such duplicate candidates are merged and purged. The complete candidates to CT is

$$C = \{o|o \in \bigcup_{i=1}^{k} C_i\} \tag{13}$$

where k is the number of trajectory segments in CT.

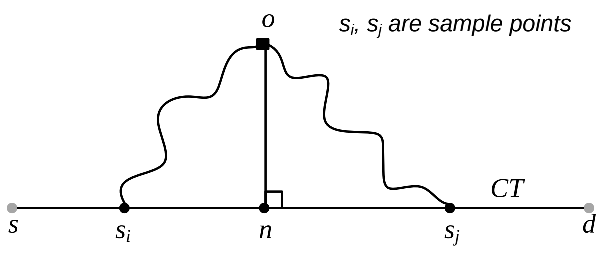


Fig. 6 Lower Bound Confliction

3.2 Efficiency Study

Theorem 1: $UB = min\{o.ub\}, o \in O_s$ is the tight upper bound among all data objects in O, e.g., any data object in $O - O_s$ must not have upper bound less than UB.

Proof : As shown in Figure 4, o is within the browsed region and o_1 is out of the browsed region. According to the definition, $UB = min_{\forall o_i \in O_s}\{o_i.ub\} \leq o.ub$. Since the search process based on Dijkstra's algorithm always chooses the vertex with the smallest distance label value for expansion, it is easy to find that:

$$\begin{cases} sd(o_1, s_i) > sd(o, s_i) \\ sd(o_1, s_j) > sd(o, s_j) \end{cases} \Longrightarrow o_1.ub > o.ub \geq UB$$

Therefore, UB is the tight upper bound among all data objects in O.

Theorem 2: For any data object $o \in O$, its lower bound of the minimum detour distance is tight.

Proof : The trajectory segment $T_{seg}(s_i, s_j)$ is represented by a rectangle R, which is defined by sample points s_i, s_j, heading vector h_v and error bound δ. The width of R is 2δ. If $\delta = 0$ (means the compression is lossless), the rectangle will degenerate to a straight line. It is just the exact path between two adjacent sample points. As shown in Figure 6, $T_{seg}(s_i, s_j)$ is the straight line connecting s_i and s_j. Straight line $P(o, n)$ is the minimum detour from $T_{seg}(s_i, s_j)$ to data object o, and $P(o, n)$ is perpendicular to $T_{seg}(s_i, s_j)$. In this context, the minimum detour distance of o is $d_D(o, T_{seg}(s_i, s_j)) = d_E(o, n) = d_E(o, R)$. From Equation 9, we can get

$$d_D(o, T_{seg}(s_i, s_j)) \geq \frac{sd(o, s_i) + sd(o, s_j) - v \times (t_j - t_i)}{2}$$

$$\Rightarrow d_E(o, R) \geq \frac{sd(o, s_i) + sd(o, s_j) - v \times (t_j - t_i)}{2}$$

$$\Rightarrow d_E(o, R) = o.lb \Rightarrow d_D(o, T_{seg}(s_i, s_j)) = o.lb$$

If there exists another lower bound greater than $o.lb$, in this case, it must be greater than $d_D(o, T_{seg}(s_i, s_j))$, which conflicts with the definition of the lower bound. Therefore, $max\ \{\frac{sd(o,s_i)+sd(o,s_j)-v\times(t_j-t_i)}{2}, d_E(o, R)\}$ is the tight lower bound for data object o.

The candidature of a data object is determined by whether its lower bound is greater than the global upper bound UB. In this section, we prove that the lower bound and upper bound we proposed are tight for a compressed trajectory. Therefore, the candidate sets specified by this upper/lower bound are tight.

3.3 Probabilistic PNN Query Processing

Now, we have two candidate sets: a data object set C and a trajectory segment set T. In this section, we present the algorithms based on an effective combination strategy (including step 2 & 3 as follows) to find the PNN with the highest probability. Given candidate sets C and T, the probabilistic PNN query processing takes three steps.

1. Each trajectory segment in T is decompressed into several sub-paths according to Algorithm 1;
2. For each sub-path, find its local PNN and compute the corresponding minimum detour distance;
3. Combine sub-paths and corresponding local PNNs to find the global PNN with the highest probability.

In the first step, we decompress the trajectory segments in T according to Algorithm 1. Then, each segment $T_{seg}(s_i, s_j) \in T$ is transformed into several sub-paths stored in $Pathlist(s_i, s_j)$ and each sub-path holds a corresponding probability. Here, we define a new relationship between trajectory segment and sub-paths:

$$P \in Pathlist(s_i, s_j) \Rightarrow P \in T_{seg}(s_i, s_j)$$

For each sub-path P, its minimum detour distance to the closest data object is denoted as $P.minDetour$. Suppose that path $P(s, d)$ is composed of a series of sub-paths $\{P_1, P_2, ..., P_k\}$, the PNN of $P(s, d)$ can be represented as:

$$P(s, t).minDetour = \min_{\forall P_i \in P}\{P_i.minDetour\} \tag{14}$$

In the second step, we compute the minimum detour distance of each data object $o \in C$, and the local PNN for each sub-path is found. $\forall o \in C$, its lower bound of the minimum detour distance is computed based on Equation 12 and the one with the minimum lower bound is selected in each time, since candidates with smaller lower bounds are more likely to be the solution, and should be processed first.

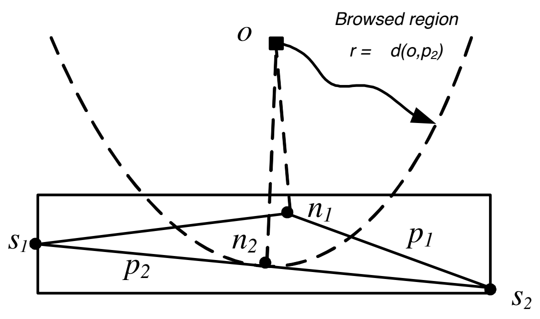


Fig. 7 Minimum Detour Distance

An example is demonstrated in Figure 7, where trajectory segment $T_{seg}(s_i, s_j)$ has been decompressed into two sub-paths P_1, P_2 connecting sample points s_1, s_2. By expanding a browsing wavefront from candidate o according to the Dijkstra's algorithm, n_1 is the first vertex in P_1 which has the minimum value among all vertices in the wavefront, i.e. wavefront will expand next from n_1 to the adjacent vertices of n_1. Similarly, n_2 is the closest vertex to o in P_2. The expansion of browsing wavefront stops when all sub-paths in any one trajectory segments are all reached by the browsing wavefront.

Lemma 2: Given a candidate o, if all sub-paths in a trajectory segment $T_{seg}(s_i, s_j)$ have been reached, the expansion process can be stopped.

Proof: As demonstrated in Figure 7, the expansion of browsing wavefront stops when $P_1, P_2 \in T_{seg}(s_i, s_j)$ are both reached. Consequently, we have the minimum detour distance from o to sub-paths $d_D(o, P_1) = sd(o, n_1)$, $d_D(o, P_2) = sd(o, n_2)$, and an expansion radius $o.r = max\{d_D(o, P_1), d_D(o, P_2)\}$. If the browsing wavefront continues to expand and reaches another sub-path $P_3 \notin T_{seg}(s_i, s_j)$, then we have $d_D(o, P_3) > o.r$. If P_3's PNN is o, $P_3.minDetour = d_D(o, P_3)$. According to Equation 14, $d_D(o, P_3)$ can not be better than $d_D(o, P_1)$ or $d_D(o, P_2)$ to be the global minimum detour. Therefore, the expansion process can be stopped when P_1, P_2 are both reached.

After that, for all unprocessed data objects in C, if its lower bound is greater than $o.r$, it will be pruned from C based on the similar reason stated in Lemma 2. Consequently, the local PNN for each sub-path $P \in T_{seg}(s_i, s_j)$ can be identified as Algorithm 3.

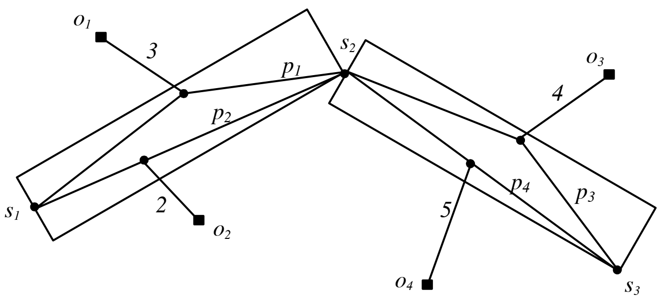


Fig. 8 Sub-Paths Combination

Algorithm 3: Local PNN Computation

```
1 while $C \neq null$ do
2     select $o$ with the minimum $o.lb$ from $C$;
3     calculate the minimum detour distance of $o$;
4     for each data object $o_i \in C$ do
5         if $o_i.lb \geq o.r$ then
6             remove $o_i$ from $C$;
7 Integrate computation results to find the local PNNs;
```

In the third step, we combine all the local PNNs to find the global PNN with the highest probability. As described in Figure 8, there are two trajectory segments $T_{seg}(s_1, s_2)$ and $T_{seg}(s_2, s_3)$. After decompression, we have sub-paths $P_1, P_2 \in T_{seg}(s_1, s_2)$ and each of them takes 50% probability. In the meantime, sub-paths $P_3, P_4 \in T_{seg}(s_2, s_3)$ and each of them takes 50% probability. Data objects o_1, o_2, o_3, o_4 are local PNNs to paths P_1, P_2, P_3, P_4 respectively. To find the global PNN with the highest probability, an intuitive method is to test all possible combinations of all sub-paths. For example, path $\{P_1, P_3\}$ holds $50\% \times 50\% = 25\%$ probability and its PNN is o_1. Path $\{P_1, P_4\}$ holds $50\% \times 50\% = 25\%$ probability and its PNN is o_1. Path $\{P_2, P_3\}$ holds $50\% \times 50\% = 25\%$ probability and its PNN is o_2. Path $\{P_2, P_4\}$ holds $50\% \times 50\% = 25\%$ probability and its PNN is o_2. Therefore, o_1 has 50% probability to be the global PNN, so as o_2.

Obviously, this approach can be very time consuming as we have to test all possible combinations. Suppose that there are k trajectory segments and each one is decompressed into m sub-paths. The complexity of the combining process is m^k, which prevents the query from being answered promptly. Here, we propose a novel method to optimize this process. For each trajectory segment $T_{seg}(s_i, s_j) \in T$, we define the upper/ lower bound for the minimum detour distance as follows.

$$T_{seg}(s_i, s_j).ub = \max_{\forall P \in T_{seg}(s_i, s_j)} \{P.minDetour\}$$

$$T_{seg}(s_i, s_j).lb = \min_{\forall P \in T_{seg}(s_i,s_j)} \{P.minDetour\}$$

Lemma 3: If there exists $A \subseteq T$, satisfying

$$\forall T_{seg}(s_i, s_j) \in A, \forall T_{seg}(s_e, s_h) \in T - A$$

$$T_{seg}(s_i, s_j).ub \leq T_{seg}(s_e, s_h).lb$$

there is no need to test combinations in $T - A$, i.e. we only need to test the combinations of trajectory segments in A.

Proof: Since $T_{seg}(s_i, s_j) \in A$ and $T_{seg}(s_g, s_h) \in T - A$, we have $T_{seg}(s_i, s_j).ub < T_{seg}(s_e, s_h).lb$. For any sub-path $P_x \in T_{seg}(s_i, s_j)$ and $P_y \in T_{seg}(s_g, s_h)$, we have $P_x.minDetour \leq P_y.minDetour$. Since the global PNN computation is based on Equation 14, the local PNN of P_y can not be better than the local PNN of P_x to be the global PNN. Therefore, we only need to consider combining trajectory segments in subset A.

An example is demonstrated in Figure 8. Since $T_{seg}(s_1, s_2).ub = 3 < T_{seg}(s_2, s_3).lb = 4$, there is no need to test sub-paths in $T_{seg}(s_2, s_3)$. For $T_{seg}(s_1, s_2)$, data object o_1 takes 50% probability to be the global PNN, so as o_2. This outcome is the same as the result by testing all possible combinations.

Algorithm 4: Sub-paths combination

```
1 while $T \neq null$ do
2     select $T_{seg}$ with the minimum $T_{seg}.ub$ from $T$ and put into $A$;
3     if $A.maxUB \leq T.minLB$ then
4         test all possible combinations in $A$;
5         break;
```

The combination procedure is introduced in Algorithm 4. We select the trajectory segment with the minimum upper bound from T and put it into A (*line* 2). If the maximum upper bound in A is no greater than the minimum lower bound in T, subset A is found and then we only need to test all possible combinations of sub-paths in A (*line* 3-5).

3.4 Complexity Analysis

Suppose data objects are uniformly distributed in road networks and the sample points in trajectories are uniformly distributed as well. We now analyze the complexity of PNN-CT query by estimating the cost in each searching phase. In the first phase, the network browsing is performed from each sample point to identify the candidate sets (data object candidate set C and trajectory segment set T). In a specified road network $G(V, E)$, the corresponding complexity is $O((k+1)(Vlg(V) + E))$ where $(k+1)$ is the number of sample points (while k is the number of trajectory segments). The second searching phase includes three steps: (1) we decompress each trajectory segment in T.

The complexity is $O(an^2)$ where a is the size of T and n is the number of vertices in each segment (rectangle); (2) we calculate the minimum detour distance of each data object in C, and the complexity is $O(b(Vlg(V)+E))$ where b is the size of data object candidate set C; (3) we select a sub-set A from T and test all possible combinations of A. The complexity is $O(m^c)$, where m is the number of possible paths in one trajectory segment and c is the size of A. Consequently, the complexity of PNN-CT query is

$$O((k+1+b)(Vlg(V)+E)+an^2+m^c) \tag{15}$$

An intuitive approach to solving PNN-CT query is to decompress all trajectory segments, and then perform the PNN query on all possible trajectories. It is a simple extension of PNN query and we use EPNN to denote this query process. The complexity of EPNN is

$$O((2+b_p)m^k(Vlg(V)+E)+kn^2+m^k) \tag{16}$$

where b_p is the data object candidates specified by the original PNN query. Through comparing Equation 15 and Equation 16, we have $((2+b_p)m^k) >> (k+b_1+1)$ and $k \geq a$ and $k \geq c$. Thus, the complexity of EPNN is much higher than the complexity of PNN-CT.

Now we analyze the optimization of each searching phase in PNN-CT. In the first phase, when the tight upper/lower bound was not in use, the bi-direction network expansion of PNN was applied instead. Here, we use $(v_1(t_2-t_1)+v_2(t_3-t_2)+...+v_k(t_{k+1}-t_k)) = 2r$ to estimate the distance of the whole trajectory. Consequently, the complexity is

$$O((2+b_p)(Vlg(V)+E)+kn^2+m^k) \tag{17}$$

Suppose the density of data object is ρ, and we have $b_2 = 2\rho\pi r^2$ and $b = (k+1)\rho\pi(r/k)^2$. Thus,

$$2+b_2 = 2+2\rho\pi r^2 = \rho\pi r^2(2+\frac{2}{\rho\pi r^2})$$

$$k+1+b = k+1+(k+1)\rho\pi(r/k)^2 = \rho\pi r^2(\frac{k+1}{k^2}+\frac{k+1}{\rho\pi r^2})$$

Compared with the number of data objects, k is a very small value and $k << \rho\pi r^2$. Thus, we have:

$$(2+\frac{2}{\rho\pi r^2}) \geq (\frac{k+1}{k^2}+\frac{k+1}{\rho\pi r^2}) \Rightarrow (2+b_2) \geq (k+1+b)$$

By comparing Equation 15 and Equation 17, it is clear that the upper/lower bound reduces the complexity notably.

In the second phase, when the effective combination strategy is not in use, all possible combinations in T are tested and the minimum detour distance for all possible paths should be computed. The complexity is

$$O((k+1+bm^a)(Vlg(V)+E)+an^2+m^c) \tag{18}$$

By comparing Equation 15 and Equation 18, we have $(k+1+bm^a) > (k+1+b)$ and the proposed combination strategy reduces the complexity notably.

Table 1 Parameter Setting

	BRN	**ORN**
Density of data objects ρ	5%-30% (default 10%)	5%-30% (default 10%)
The number of trajectory segments k	1-10 (default 6)	1-10 (default 6)
Compression error bound δ	50-300 (default 50)	100-350 (default 100)

4 Experiments

In this section, we conducted extensive experiments on real spatial data-sets to demonstrate the efficiency of PNN-CT query. The two data sets used in our experiments were Beijing Road Network (BRN) [2] and Oldenburg City Road Network (ORN)[3], 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 [18]. In ORN, the synthetic trajectory data were used. All algorithms were implemented in Java and tested on a Windows platform with Intel Core2 CPU (2.13GHz) and 1GB memory. Data objects were generated on the networks uniformly with densities from 5% to 30%,

$$density = \frac{the\ number\ of\ data\ objects}{the\ number\ of\ vertices\ in\ the\ network} \tag{19}$$

In our experiments, the networks resided in memory when running Dijkstra's algorithm 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 testes with different query inputs. The main performance metric was CPU time and the parameter settings are listed in table 1.

By default, the density of data objects was 10%, and the number of trajectory segments k was 6 in both BRN and ORN. In the meantime, the compression error bound was set to 50 ($\approx 0.35\ km$) for BRN, and 100 ($\approx 0.35\ km$) for ORN. To the best of our knowledge, no existing method in literature has been proposed to address the PNN-CT query investigated in this work. For the purpose of comparison, an algorithm based on the Path Nearest Neighbor (PNN) query [8] was also implemented (referred to as EPNN). In this algorithm, all trajectory segments were decompressed initially. Then, the PNN query was conducted on each possible path connecting the source and the destination.

4.1 Effect of Data Object Density

First of all, we investigated the effect of data object density on the performance of PNN-CT and EPNN with the default settings. Intuitively, the higher the

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

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

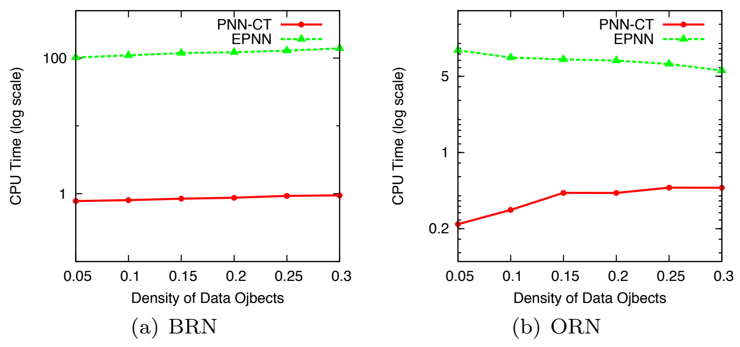


Fig. 9 Effect of Data Object Density

density of data objects, the smaller the required search range. In Figure 9(b), the CPU time of EPNN decreases while the density increases. However, in large data set, the higher data object densities may lead to more candidates to be processed, which may offset the time saved by the smaller search ranges. For example, in Figure 9(a), the CPU time of EPNN increases with the density. For PNN-CT, the higher data object density may result in more trajectory segments to be decompressed. Thus in both Figure 9(a) and Figure 9(b), the CPU times of PNN-CT increase slowly with the density. Nevertheless, the CPU time required by EPNN was two orders of magnitude higher than that of PNN-CT.

4.2 Effect of the Number of Compressed Trajectory Segments k

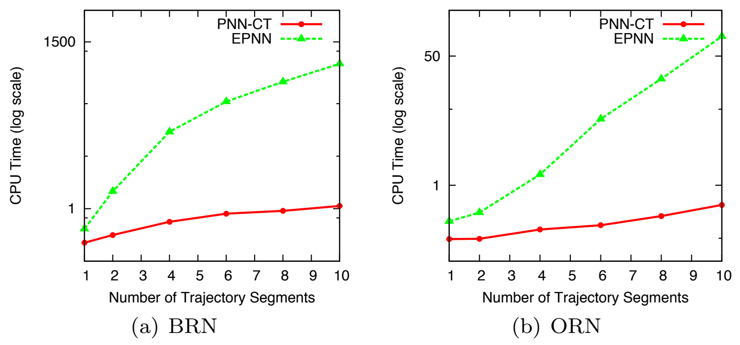


Fig. 10 Effect of Trajectory Segment Number k

Figure 10 shows the performance of PNN-CT and EPNN when the number of trajectory segments varies. Since longer trajectories cause more data objects to be processed, the CPU time is expected to be higher for both PNN-CT and EPNN. However, the CPU time of EPNN increases much faster than PNN-CT for two reasons. The first is due to the much larger number of candidates (data objects to be checked and trajectory segments to be decompressed) by using bi-direction network expansion method in EPNN, and the second is that EPNN has to process all possible combinations since no combination strategy is adopted. For instance, when k is equal to 10, PNN-CT outperforms EPNN by almost three orders of magnitude. Note that this result demonstrates the importance of smart selection of candidates and the necessity of an effective combination strategy.

4.3 Effect of Prediction Model Error Bound δ

Figure 11 shows the effect of error bound δ on the query time of PNN-CT and EPNN. In general, a larger error bound will lead to a greater level of uncertainty when decompressing, and more sub-paths to be considered. In both BRN and ORN, the CPU-time of EPNN increases exponentially, while the time cost of PNN-CT only rises slightly due to the proposed effective combination strategy.

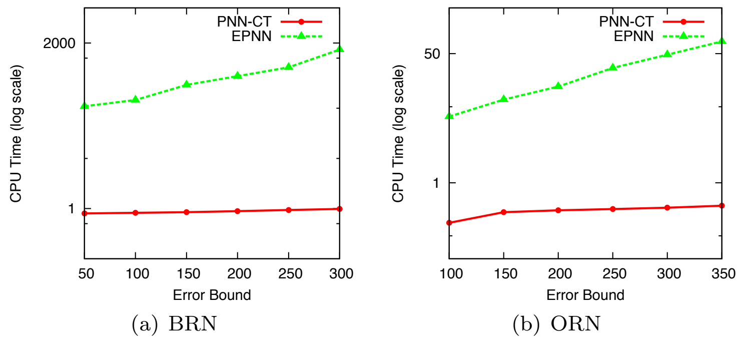
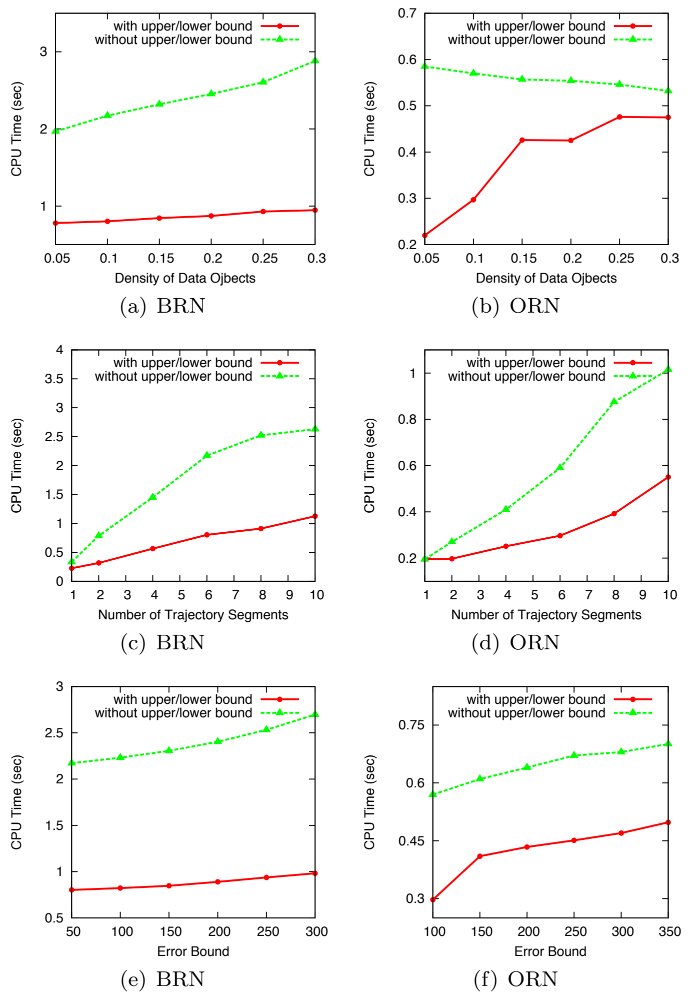


Fig. 11 Effect of Prediction Model Error Bound δ

4.4 Effect of the Upper/Lower Bound of the Minimum Detour Distance

This experiment tests the effect of the proposed tight upper/lower bound of the minimum detour distance (Section 3.1) on the performance of PNN-CT. These bounds are used to prune candidates (both data objects and trajectory segments) and to ensure that the data object candidates are processed in proper order. PNN-CT was run with and without the upper/lower bound. When the upper/lower bound was not in use, the bi-direction network expansion of PNN was applied instead. In Figure 12, it is clear that the performance is accelerated by 2-4 times with the help of our bounds, and the effect is more obvious in large data-set as shown in Figure 12(a) 12(c) 12(e).



(a) BRN (b) ORN

(c) BRN (d) ORN

(e) BRN (f) ORN

Fig. 12 Effect of Upper/Lower Bound

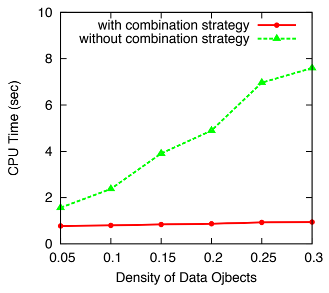
4.5 Effect of Combination Strategy

We also tested the effect of combination strategy (Section 3.3) on the performance of PNN-CT. A suitable combination strategy can avoid a huge amount of repeated computation and improve the performance notably. We run PNN-CT with and without the combination strategy. When the combination strategy was not in use, all possible sub-path combinations was tested to reconstruct possible paths, and PNNs of all possible paths were found. In Figure 12, among all six sub-figures, it is easy to find that without the sub-paths combination strategy, the corresponding CPU-times increase dramatically. By contrast, the CPU-time of PNN-CT only increases slightly.

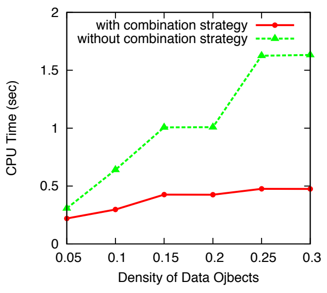
5 Related Work

Path Nearest Neighbor (PNN):
The concept of Path Nearest Neighbor is first proposed as In-route Nearest Neighbor (IRNN) [26,36], 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 to the nearest facility (e.g., petrol station) and will continue the journey along the previous route. Recently, k-PNN proposed by Chen et al. in [8] is an extension of the IRNN query. k-PNN can monitor the k nearest neighbors efficiently 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 prefer 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.

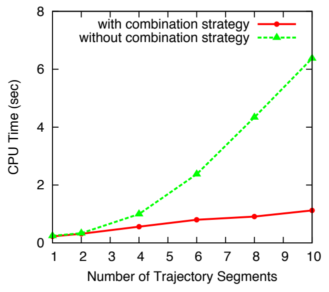
Moving Object Prediction Models:
Generally, moving object prediction models can be classified into two categories: linear prediction models [30,21,14,25,31,17] and non-linear prediction models [28,1,15,23,5,29,11]. Linear prediction models assume that the moving object follows linear movements, and introduce little computation and storage overhead. Given an object's location s_1 at time t_1 and its speed $\overrightarrow{v}$, the linear models estimate the object's future location at time t_2 by using the formula $s_2 = s_1 + \overrightarrow{v} \times (t_2 - t_1)$, where s_1, s_2 and $\overrightarrow{v}$ are $2-dimensional$ vectors. On the other hand, non-linear prediction models consider not only linear but also non-linear motions of moving objects. Compared with linear models, they can produce more accurate prediction results but incur additional computation over head and storage cost. Most of non-linear models are based on the historic position data of moving objects. For example, Recursive Motion Function [28] formulates an object's location based on its locations at the h most recent



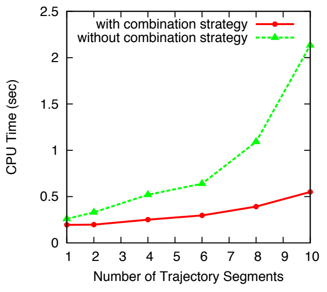
(a) BRN



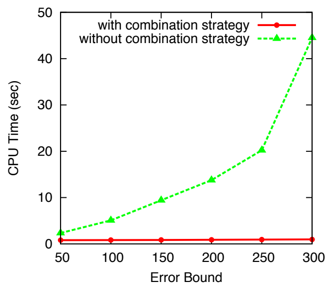
(b) ORN



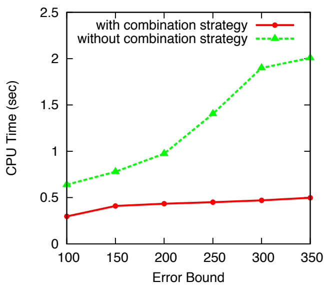
(c) BRN



(d) ORN



(e) BRN



(f) ORN

Fig. 13 Effect of Combination Strategy

time-stamps. In Hybrid Prediction Model [15], a pattern tree is maintained to describe the historic moving patterns of the corresponding moving object.

Trajectory Compression:
Trajectory compression is a widely used technology in spatial databases. Compared with original trajectory data, compressed trajectories have clear advantages in data processing, transmitting, storing, etc. [20]. Ideally, a trajectory compression approach can notably reduce (*i*) the computation/ communication load of clients (GPS-enabled mobile devices) and (*ii*) the storage cost of servers. Despite the bulk of trajectory compression literature [10,19,4,3,16,7],

no solution fulfills both requirements, except the linear prediction model based trajectory compression [30,21,14,25,31,17]. Traditional compression methods, including Douglas-Peucker Algorithm [10], Modified Douglas-Peucker [19] and Bellman's Algorithm [4,3,16], require the access to complete trajectory data. Although they can achieve high compression ratios, on the client-side, every point should be recorded and uploaded, which incurs intolerable computation/ communication load. On the other hand, in linear prediction model based trajectory compression, it is not necessary to record and upload each point in the trajectory unless it breaches the constraint set by the prediction model. On the server-side, since the whole trajectory is compressed into a set of sample points, the pressure of storage is reduced notably. Note that the non-linear prediction models [28,1,15,23,5,29,11] are not suitable for the trajectory compression. Compared with linear models, they can produce more accurate prediction results but incur additional computation load and storage cost. For example, Recursive Motion Function [28] and Hybrid Prediction Model [15] are two typical non-linear prediction models. To describe the movement of the moving object, a matrix and a pattern tree need to be maintained respectively. In most cases, the space required by the additional information offsets the space saved by compression. In addition, the extra information is problem specific and can hardly be shared among different trajectories. In the meantime, the non-linear prediction models may demand even higher computation effort than simply sampling every point in the trajectory.

6 Conclusion

Trajectory compression is widely used in spatial databases as it can notably reduce (*i*) the computation/communication load of clients (GPS-enabled mobile devices) and (*ii*) the storage cost of servers. Compared with original trajectories, compressed trajectories have strong advantages in data processing, transmitting, storing, etc. In this work, we investigated a novel PNN-CT query to find the path nearest neighbor based on compressed trajectories. This query can bring significant benefits to users in many popular applications such as trip planning. To address the PNN-CT query effectively and efficiently, we proposed a novel two-phase solution. The efficiency study revealed that the candidate sets created are tight and the complexity analysis showed that our solution has strong advantages over the existing methods. The efficiency of PNN-CT query processing was verified by extensive experiments based on real and synthetic trajectory data in road networks.

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).

References

1. C. C. Aggarwal and D. Agrawal. On nearest neighbor indexing of nonlinear trajectories. In *PODS*, pages 252–259, 2003.
2. H. Alt, A. Efrat, G. Rote, and C. Wenk. Matching planar maps. In *SODA*, pages 589–598, 2003.
3. R. Bellman and S. Dreyfus. Applied dynamic programming, princeton university press. 1962.
4. R. E. Bellman. On the approximation of curves by line segments using dynamic programming. *CACM*, 4(6):284, 1961.
5. A. Bhattacharya and S. K. Das. Lezi-update: an information-theoretic approach to track mobile users in pcs networks. In *MobiCom*, pages 1–12, 1999.
6. S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk. On map-matching vehicle tracking data. In *VLDB*, pages 853–864, 2005.
7. H. Cao, O. Wolfson, and G. Trajcevski. Spatio-temporal data reduction with deterministic error bounds. In *VLDB J.*, volume 15, pages 211–228, 2006.
8. Z. Chen, H. T. Shen, X. Zhou, and J. X. Yu. Monitoring path nearest neighbor in road networks. In *SIGMOD*, pages 591–602, 2009.
9. E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische Math*, 1:269–271, 1959.
10. D. Douglas and T. Peucker. Algorithms for the reduction of the number of points required to represent a line or its caricature. In *the Canadian Cartographer*, volume 10, pages 112–122, 1973.
11. F. Giannotti, M. Nanni, F. Pinelli, and D. Pedreschi. Trajectory pattern mining. In *SIGKDD*, pages 330–339, 2007.
12. J. Greenfeld. Matching gps observations to locations on a digital map. In *81th Annual Meeting of the Transportation Research Board*, 2002.
13. G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM TODS*, 24(2):265–318, 1999.
14. C. S. Jensen, D. Lin, and B. C. Ooi. Query and update efficient b+-tree based indexing of moving objects. In *VLDB*, pages 768–779, 2004.
15. H. Jeung, Q. Liu, H. T. Shen, and X. Zhou. A hybrid prediction model for moving objects. In *ICDE*, pages 70–79, 2008.
16. J. Kleinberg and E. Tardos. Algorithm design. In *Addison-Wesley, Reading, MA*, 2005.
17. R. Lange, T. Farrell, F. Drr, and K. Rothermel. Remote real-time trajectory simplification. In *PerCom*, pages 1–10, 2009.
18. 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.
19. N. Meratnia and R. A. d. By. Spatiotemporal compression techniques for moving point objects. In *EDBT*, pages 765–782, 2004.
20. J. Muckell, J.-H. Hwang, C. Lawson, and S. Ravi. Algorithms for compressing gps trajectory data: An empirical evaluation. In *ACM GIS*, 2010.
21. J. M. Patel, Y. Chen, and V. P. Chakka. Stripes: an efficient index for predicted trajectories. In *SIGMOD*, pages 635–646, 2004.
22. J. Pei, M. Hua, Y. Tao, and X. Lin. Query answering techniques on uncertain and probabilistic data: tutorial summary. In *SIGMOD*, 2008.
23. L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *IEEE Proceedings*, volume 77, pages 257–286, 1989.
24. N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *SIGMOD*, pages 71–79, 1995.
25. S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. In *SIGMOD*, pages 331–342, 2000.
26. S. Shekhar and J. S. Yoo. Processing in-route nearest neighbor queries: a comparison of alternative approaches. In *ACM GIS*, pages 9–16, 2003.
27. D. Suciu and N. Dalvi. Foundations of probabilistic answers to queries. In *SIGMOD Tutorial*, 2005.
28. Y. Tao, C. Faloutsos, D. Papadias, and B. Liu. Prediction and indexing of moving objects with unknown motion patterns. In *SIGMOD*, 2004.

29. Y. Tao, G. Kollios, J. Considine, F. Li, and D. Papadias. Spatio-temporal aggregation using sketches. In *ICDE*, page 214, 2004.
30. Y. Tao and D. Papadias. Spatial queries in dynamic environments. *ACM TODS*, pages 101–139, 2003.
31. Y. Tao, D. Papadias, and J. Sun. The tpr*-tree: An optimized spatiotemporal access method for predictive queries. In *VLDB*, pages 790–801, 2003.
32. Y. Tao, X. Xiao, and R. Cheng. Range search on multidimensional uncertain data. *ACM TODS*, 32(3), 2007.
33. G. Trajcevski, R. Tamassia, H. Ding, P. Scheuermann, and I. F. Cruz. Continuous probabilistic nearest-neighbor queries for uncertain trajectories. In *EDBT*, pages 874–885, 2009.
34. G. Trajcevski, O. Wolfson, K. Hinrichs, and S. Chamberlain. Managing uncertainty in moving objects databases. *ACM TODS*, 29(3), 2004.
35. C. Wenk, R. Salas, and D. Pfoser. Addressing the need for map-matching speed: Localizing globalb curve-matching algorithms. In *SSDBM*, 2006.
36. J. S. Yoo and S. Shekhar. In-route nearest neighbor queries. *GeoInformatica*, 9:117–137, 2005.
37. K. Zheng, G. Trajcevski, X. Zhou, and P. Scheuermann. Probabilistic range queries for uncertain trajectories on road networks. *EDBT*, 2011.