Information Systems 37 (2012) 13–32

Contents lists available at SciVerse ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

Vague continuous *K*-nearest neighbor queries over moving objects with uncertain velocity in road networks

Ping Fan [a,b], Guohui Li [a], Ling Yuan [a,*], Yanhong Li [a]

[a] School of Computer Science, Huazhong University of Science and Technology, Wuhan, China
[b] School of Computer Science, Xianning University, Hubei, China

ARTICLE INFO

Article history:
Received 26 November 2010
Received in revised form
25 July 2011
Accepted 24 August 2011
Recommended by: F. Korn
Available online 8 September 2011

Keywords:
Continuous *K* Nearest Neighbor queries
Uncertain velocity
Road network
Distance interval
Possibility

ABSTRACT

Recent research has focused on Continuous *K* Nearest Neighbor (*CKNN*) queries in road networks, where the queries and the data objects are moving. Most existing approaches assume the fixed velocity of moving objects. The release of fixed moving velocity makes the query process slowly due to the significant repetitive query cost. In this paper, we study *CKNN* queries over moving objects with uncertain velocity in road networks. A Distance Interval Model (DIM) is designed to calculate the minimal and maximal road network distances between moving objects and query point. Furthermore, we propose a novel Possibility-based Vague *KNN* (PV*KNN*) algorithm to process the query efficiently, which determines the *CKNN* query results with possibility within each division time subinterval of given time interval by applying the vague set theory. In the PV*KNN* algorithm, the query efficiency can be improved significantly with the pruning, distilling and possibility-ranking phases. With these phases, the objects candidates are scaled down and the given time interval is divided into subintervals to reduce the repetitive query cost. In addition, an index structure TPR^{uv}-Tree is designed to efficiently index moving objects with uncertain velocity in road network by involving edge connection and moving objects information. Experiments with simulation and comparison show that significant improvement in the performance of efficiency can be achieved with our proposed algorithms.

1. Introduction

With the increasing complicated traffic problems in road networks, which involve large spatio-temporal data sets, recent research has focused on processing queries in such environment. One of the important types of queries in spatio-temporal database is the Continuous *K* Nearest Neighbor (*CKNN*) queries [1–3]. A *CKNN* query is defined as retrieving the *K* nearest neighbors (*KNNs*) to a query point within a user-given time interval [4–8]. In road network scenarios, for example, a *CKNN* query could be finding two nearest police cars while one person is driving within next 1–5 min, which indicates that both the query object and the data objects are moving [9,10], different from the static query object or static data objects [11,12]. As the query and data objects are moving, the updating for the locations of moving objects frequently, especially in a larger user-given time interval, could make the query process slowly because of the significant repetitive query cost.

In order to improve the efficiency of *CKNN* query process for the moving objects, some previous work in this field assumed that the velocity of each moving object is fixed, since the motion of each object can be precisely determined under this assumption [13–15]. However, in road networks of real world, the objects move arbitrarily. The release of the fixed velocity makes it difficult to determine precisely the distance between moving objects and query object, which leads to a more complicated *CKNN* query process.

We take an example to illustrate the *CKNN* queries over moving objects with uncertain velocity in a road

* Corresponding author.
E-mail address: cherryyuanling@gmail.com (L. Yuan).

0306-4379/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.is.2011.08.002

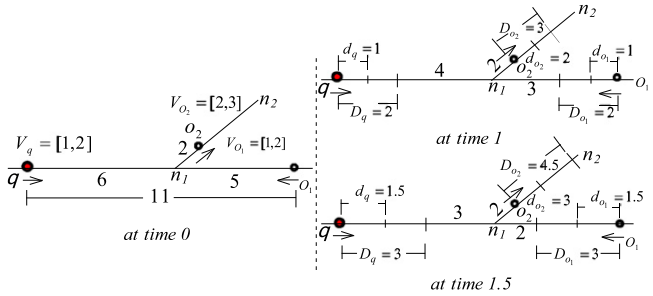

Fig. 1. Example of a C*K*NN query.

network, as shown in Fig. 1. A C*K*NN query is issued to find 1NN of query object q within a time interval [0, 1.5].

Assume that the nearer moving objects to the query point q are o_1 and o_2 at the start time, which can be determined by the mobile information system [16,17]. The start location of object o_1 is at the distance 5 from node n_1, with uncertain velocity within [1,2], and the moving direction of o_1 is toward node n_1. While the start location of object o_2 is at the distance 2 from node n_1, with uncertain velocity within [2,3], and the moving direction of o_2 is toward node n_2. The velocity of query point q is assumed within [1,2], and the moving direction of q is toward node n_1. The direction and velocity of the object is updated when the object reaches the road intersection. The moving distance of the objects can be calculated according to the moving direction and velocity. For example, at time 1, the minimal distance of the query point q can move is 1 ($d_q = 1$), and the maximal distance of q can move is 2 ($D_q = 2$). The minimal distance of the moving object o_1 can move is 1 ($d_{o_1} = 1$), and the maximal distance of o_1 can move is 2 ($D_{o_1} = 2$). The minimal distance of the moving object o_2 can move is 2 ($d_{o_2} = 2$), and the maximal distance of o_2 can move is 3 ($D_{o_2} = 3$). The minimal distance between q and o_1 at time 1 is denoted as $d_{q,o_1}(1) = 7$, and the maximal distance between q and o_1 at time 1 is denoted as $D_{q,o_1}(1) = 9$, then the distance between q and o_1 could be within a distance interval [7,9]. The minimal distance between q and o_2 at time 1 is denoted as $d_{q,o_2}(1) = 8$, and the maximal distance between q and o_2 at time 1 is denoted as $D_{q,o_2}(1) = 10$, then the distance between q and o_2 could be within a distance interval [8,10]. By comparing two distance interval [7,9] and [8,10], it is non-trivial to determine, which moving object is nearer to the query point q at time 1. At time 1.5, the minimal distance of the query point q can move is 1.5 ($d_q = 1.5$), and the maximal distance of q can move is 3 ($D_q = 3$). The minimal distance of the moving object o_1 can move is 1.5 ($d_{o_1} = 1.5$), and the maximal distance of o_1 can move is 3 ($D_{o_1} = 3$). The minimal distance of the moving object o_2 can move is 3 ($d_{o_2} = 3$), and the maximal distance of o_2 can move is 4.5 ($D_{o_2} = 4.5$). The minimal distance between q and o_1 at time 1.5 is denoted as $d_{q,o_1}(1.5) = 5$, and the maximal distance between q and o_1 at time 1.5 is denoted as $D_{q,o_1}(1.5) = 8$, then the distance between q and o_1 could be within a distance interval [5,8]. The minimal distance between q and o_2 at time 1.5 is denoted as $d_{q,o_2}(1.5) = 8$, and the maximal distance between q and o_2 at time 1.5 is denoted as $D_{q,o_2}(1.5) = 11$, then the distance between q and o_2 could be within a distance interval [8,11]. By comparing two distance interval [5,8] and [8,11], we can determine that o_1 could be nearer to q than o_2 at time 1.5. In order to tackle the comparing problem of intersect distance interval, such as [7,9] and [8,10], the vague set theory is applied to determine the possibility that which moving object is nearer to the query point. From the example of Fig. 1, we can conclude that the C*K*NN query result could be different within a query time interval, and the query result may not be acquired precisely. Furthermore, with the increased number of moving objects and larger time interval, it is more complicated to process C*K*NN queries. In order to tackle these issues, we propose methods in PV*K*NN algorithm to scale down the moving object candidates and divide the given time interval into time subintervals, and apply vague theory in the distance interval comparison to determine the C*K*NN query result.

The major contributions of this paper are summarized as follows:

- Our work remedies the major drawbacks of the past related work and provides a more practical and efficient solution for the C*K*NN problem in road networks.
- A Distance Interval Model (DIM) is designed to calculate the distance intervals for the moving objects to the query point in road network at a time instant.
- An index structure TPR^{uv}-Tree is proposed to efficiently index moving objects with uncertain velocity in road network by involving edge connection and moving objects information.
- A Possibility-based Vague *K*NN (PV*K*NN) algorithm is proposed to efficiently determine the *K*NN query result with the possibility within a given time interval.
- Simulation experiments are conducted to evaluate the performance of the proposed PV*K*NN algorithm using both synthetic and real data sets.

The rest of this paper is organized as follows. In Section 2, we discuss some related work about processing C*K*NN queries. In Section 3, we present the proposed Distance Interval Model (DIM). The index schema TPR^{uv}-Tree is presented in Section 4. In Section 5, we illustrate three phases of the PV*K*NN algorithm in detail. Section 6 shows extensive experiments on the performance of our approach. Section 7 concludes the paper with some future works.

2. Related work

The continuous K nearest neighbor query (CKNN) has been intensively studied in recent years due to the emergence of many e-commerce mobile services. Efficient processing of CKNN queries in road networks has also attracted a lot of attentions. Previous methods in this literature mainly focused on processing KNN queries for static objects or moving objects with fixed velocities [1–3,18,9,10].

2.1. Methods for CKNN queries over static objects

Kolahdouzan and Shahabi [18] propose a solution-based approach for KNN queries in spatial networks over static objects. Their approach, called VN^3, precalculates the *network Voronoi polygons* (NVPs) and some network distances, and then processes the KNN queries based on the properties of the Network Voronoi diagrams. Once objects move continuously, the KNN result would change over time. As a result, the VN^3 must be repeatedly evaluated and thus the performance significantly degrades because of the high re-evaluation cost. Furthermore, for continuous KNN queries, Kolahdouzan and Shahabi [8] propose a solution, namely *upper bound algorithm* (UBA), which performs snapshot KNN queries at the locations where they are required, and hence provides better performance by reducing the number of KNN evaluation. When the data objects moves, the re-evaluation cost would be significantly increased. Cho and Chung [19] develop a *unique continuous search algorithm* (UNICONS) for CKNN queries over the moving objects in road networks, which is restricted in processing the CKNN queries over static data objects (that is, only the query object moves continuously). UNICONS first divides the path of the query object into sub-paths by the intersections, and then the snapshot KNN queries are performed at two endpoints of each sub-path. Finally, the KNNs for each sub-path can be found from the union of the KNN sets at its two endpoints and the objects along it. When the data objects' locations change over time, the performance of this technique would be significantly degraded.

2.2. Methods for CKNN queries over moving objects

Mouratidis et al. [20] address the issue of continuous monitoring KNN on moving objects. They propose an *incremental monitoring algorithm* (IMA) to re-evaluate query at those time instants at which updates occur. At each evaluation time, the query result may be incrementally obtained from the result at the previous timestamp. Therefore, the overhead incurred by processing repeated queries can be reduced. However, due to the nature of discrete location updates, the KNNs of the query object within two successive updates are unknown. Thus, IMA would return invalid results between two successive update timestamps. As such, IMA would return incorrect results as long as there exist some time points within two consecutive updates at which two consecutive updates are unknown. Huang et al. [15] propose a continuous monitoring method over moving objects in road networks. The procedure of this monitoring method is divided into two phases, the pruning phase and the refining phase. The main aim of the pruning phase is to calculate the pruning distance. With the given pruning distance, unqualified objects are efficiently pruned. Then in the refinement phase, candidates are verified whether they belong to the KNNs of query q or not. Because the pruning distance setting of pruning strategy of this paper is a little large, several unqualified objects cannot be pruned, and then the performance of the refining phase cannot be improved enough. Li et al. [21] propose an efficient algorithm of CKNN query based on *moving_state* of the objects with fixed velocity in a road network. The proposed algorithm is composed of two phases: pruning phase and refining phase. In the pruning phase, a pruning distance is calculated efficiently to exclude the unqualified objects to scale down the object candidates with the help of *moving_state* of objects. The refining phase determines the time subintervals where the KNN query result is certain. These two phases can highly reduce the repetitive query cost incurred by the location update of moving objects. However, when the fixed velocity is released, in the pruning phase, the pruning distance calculation should consider the velocity interval of moving objects; in the refining phase, the obtained object candidate of each time subinterval is not the final KNN result, because the distance between each object in the object candidate and the query object cannot be compared directly. Therefore, when processing CKNN queries over the moving objects with uncertain velocity, these two phases should be modified according to the velocity intervals of moving objects.

The techniques mentioned above all focus on the CKNN queries over moving objects with fixed velocity. However, in road networks of real world, the velocity of each moving objects is uncertain, which leads to more repetitive re-evaluation queries as the updates of velocities. Once the assumption of fixed velocity of moving objects is released, it is more difficult to determine the trajectory of a moving object precisely. Existing research on the CKNN queries over moving objects with uncertain velocity focus exclusively on Euclidean spaces [22–24]. Yu et al. [24] propose a monitoring method to process CKNN queries over moving object. This method involves the Object-Indexing based on indexing the objects themselves and the Query-Indexing based on indexing the queries to determine the locations of moving objects. The grid index used by this method cannot capture the constraints imposed by a road network. Furthermore, the method processes objects and updates falling in circles and rectangles, while there is no trivial mapping or interpretation of these shapes into road networks. Huang et al. [25] investigate how to process a CKNN query over moving objects with velocity varying within a known interval. A cost-effective P^2KNN algorithm is proposed to find the objects that could be the KNNs at each time instant within the given query time interval, where a uncertain distance model is presented for representing the distance between a moving object and the query object at each time instant. Besides, a probability-based model is designed to quantify the possibility of each

object being one of the *K*NNs. However, the distance between the moving object and the query point is considered as the Euclidean distance rather than the network distance. Therefore, if processing the C*K*NN query over moving objects with uncertain velocity in road networks, the distance calculation method should be modified to accommodate to the road network distance.

When an object moves with uncertain velocity in the road network, it is not so easy to determine the accurate location of the object at a time instant. A Segment-Based Tracking method [26,27] can be used to track the continuously changing positions of the moving objects with a threshold C to minimize the updates. If the object's location is inside C, the object does not need to inform the server; otherwise, the object must update the circle C and inform the server. Our proposed algorithm just uses the minimal and maximal distance between the object and the query point to evaluate the possibility of candidate objects being C*K*NN, and it is not necessary to know the moving locations of the object.

When calculating the minimal distance and maximal distance between the object and the query point, if the moving object and the query point are on different edges, the shortest path distance between two nodes of the road network should be calculated. For the shortest path calculation in road networks, Hu et al. [28] propose an efficient index, namely distance signature, for distance computation over long distances. The distance signature of each object stores the shortest path between the object with other objects in a sequence in the form of a categorical value based on the exact distance, where each data object is distributed on the node. Since our paper focuses on the shortest path distance computation of edge nodes, the distance signature is not so suitable for our paper, because it mainly calculates the distance between the object and the node, although the objects is distributed on the nodes but not on all the nodes. Samet et al. [29] propose an algorithm to compute the shortest paths between the various vertices in the spatial network only once, which can decouples the process of computing shortest paths along the network from that of finding the neighbors, whereas in the methods that are based on Dijkstra's algorithm the shortest paths between some vertices are computed repeatedly as the query object and the number of sought neighbors change thereby causing the reapplication of the algorithm. Wei [30] proposes an indexing and query processing scheme for the shortest path query answering, namely TEDI, which is based on the tree decomposition methodology. The road network is firstly decomposed into a tree in which the node of the tree contains more than one node from the road network according to the MBR. A bottom-up operation can be executed to calculate the shortest path distance between any two nodes of the road network. TEDI is an efficient method to calculate the shortest path distance of two nodes in road networks, which would be used in our paper to calculate the shortest path distance of two nodes.

Li et al. [31] propose a CU*K*NN query monitoring method to continuously find the objects with uncertain speed that could potentially be the *k*-nearest neighbors (*K*NN) of the query. Based on the distance estimation between objects and query, an efficient method is designed to calculate the probability of each object being a *K*NN of a query. The distance calculation method of Li et al. [31] uses the line segments to calculate the distances between the objects and the query. When the moving object and query object are on the different edges without distance determinate, it is difficult to obtain the maximal distance between the moving object and query object. Then an approximate value is used to serve as the maximal distance. Furthermore, in the pruning phase and refining phase, Li et al. [31] just assumes the maximal and minimal distances between the moving object and the query object within the query time interval are monotonic with time. Actually, within the query time interval, the maximal and minimal distances are not monotonic with time, which would change when the objects reach the road intersection. Finally, Li et al. [31] uses a Possible-Line-Segment (PLS) to represent the possible distance between the object and the query within a time interval to calculate the probability of each object being a *K*NN of a query, where PLS is the distance interval at the middle time instant of the time interval. This method can simplify the probability calculation but decrease the precision. These points are all considered and improved in this paper.

In the following, the distance interval model and the proposed PV*K*NN algorithm are presented in detail to illustrate how to process C*K*NN queries efficiently under the condition that all objects (including the query object) move continuously with uncertain velocity in road networks.

3. Distance interval model

Before going into the details, let us first consider a relevant question: Given two objects o_1 and o_2, which is better for a C*K*NN query over moving object? This pairwise competition has a clear answer when o_1 and o_2 are precise points: the one nearer to the query point q wins the competition. How about o_1 and o_2 being uncertain? The answer is still clear: the one more likely to be nearer to q is better, but how to calculate the distances between o_1 and o_2 and query point with uncertain velocity? Motivated by this question, we propose a Distance Interval Model (DIM) to calculate the distances between moving objects and query point with uncertain velocity.

Note that the distance mentioned in this paper represents the road network distance. For clarity, we present in Table 1 the primary symbols used in the paper, along with their interpretation.

Firstly, we give a brief explanation to the distance interval concept in our proposed DIM. Assume an object o is moving away with a fixed velocity $o.v$ from a static query point q on an edge, the distance between o and q is denoted as $d(t_0)$ at start time t_0. At time t, the distance between o and q (denoted as $d(t)$) can be calculated as $d(t)=d(t_0)+o.v(t-t_0)$ (Eq. (3)). Assume that the velocity of object o becomes uncertain within $[o.v_m, o.v_M]$. By substituting $o.v_m$ and $o.v_M$ with $o.v$ in Eq. (3), respectively, we can obtain two distances: one is $d(t)$, indicating

Table 1
Primary symbols.

Symbol	Description
$\rightarrow$	Positive direction, indicating from starting node to ending node on an edge
$\leftarrow$	Negative direction, indicating from ending node to start node on an edge
$o\rightarrow \leftarrow q$	Object o and query point q are moving towards each other on an edge
$\leftarrow q$, $o\rightarrow$	Object o and query point q are moving with the opposite direction on an edge
o, $q\rightarrow \rightarrow$	Object o is moving towards query point q with the same positive direction on an edge
q, $o\rightarrow \rightarrow$	Query point q is moving towards object o with the same positive direction on an edge
$d_{q,o}(t)$	The minimal distance between query point q and object o at time t
$D_{q,o}(t)$	The maximal distance between query point q and object o at time t
SDN_{ij}	The shortest path distance between node n_i and n_j
DD_{ij}	The Boolean value indicating whether edge e_i and edge e_j are distance determinate or not
$d_{pruning}$	The pruning distance
$o_1 \prec o_2$	Object o_1 is nearer to the query point than object o_2

the minimal distance between o and q, and the other is $D(t)$ indicating the maximal distance between o and q. Thus, the possible distance between o and q at time t should be within the distance interval $[d(t), D(t)]$.

Since the data objects and query point move arbitrarily in road networks, the computation of distance between a moving object and the query point is more complicated than the distance calculation of Eq. (3). For the edge in road networks, we assume that there exist a starting node and an ending node. For the moving direction, we assume that if an object moves along the direction from the starting node to the ending node of the edge where it resides, its velocity is a positive value, denoted as $\rightarrow$, otherwise its velocity is a negative value, denoted as $\leftarrow$. In the following, we present the proposed DIM to illustrate how to calculate minimal distance and maximum distance between a moving object o and query point q with uncertain velocity in road networks. For simplicity, when considering the object o and query point q on the same edge, we just present the method about how to calculate the distance between the object o with positive moving direction and the query point q. Meanwhile, for the object with the negative moving direction, the distance calculation method could be similar.

3.1. Calculating the minimal distance $d_{q,o}(t)$

There are two different cases regarding the minimal distance calculation between a moving object o and the query point q. One case considers that o and q are on the same edge, and the other case is that o and q are on two different edges.

Case 1. Object o and query point q are on the same edge. The minimal distance between q and o at time t is denoted as $d_{q,o}(t)$, which can be calculated in the following four different cases, respectively, according to four different moving directions of o and q, as shown in Fig. 2(a). The $d_{q,o}(t_0)$ denotes the minimal distances between q and o at the start time t_0. The velocity of query point q is within the interval $[q.v_m, q.v_M]$, and the velocity of object o is within the interval $[o.v_m, o.v_M]$.

Case 1.1. Object o is moving toward query point q, denoted as $o\rightarrow \leftarrow q$. The moving situations of object o and the query point q at different times are presented in $o\rightarrow \leftarrow q$ of Fig. 2(b). When o and q move toward each other, the minimal distance between o and q can be calculated as d', where $d'>0$. When the line segments of o and q intersect with each other, the minimal distance of o and q would be 0, where $-d \le d' \le 0$ and d indicating the intersection distance. When $d'<-d$ indicating that the line segments of o and q have no intersection, object o and query point q move away with each other, denoted as $\leftarrow q$, $o\rightarrow$, where the minimal distance calculation of o and q can refer to case 1.2:

$$d' = d_{q,o}(t_0) - (o.v_M - q.v_M)(t - t_0)$$
$$d = ((o.v_M - o.v_m) + (q.v_m - q.v_M))(t - t_0)$$
$$d_{q,o}(t) = \begin{cases} d' & d' > 0 \\ 0 & -d \le d' \le 0 \end{cases}$$

Case 1.2. Object o and query point q are moving with the opposite direction, denoted as $\leftarrow q$, $o\rightarrow$, where it is impossible for object o and query point q to intersect with each other. The minimal distance between o and q can calculated as follows:

$$d_{q,o}(t) = d_{q,o}(t_0) + (o.v_m - q.v_m)(t - t_0)$$

Case 1.3. Object o is moving towards q with the same positive direction, denoted as o, $q\rightarrow \rightarrow$. The moving situations of object o and the query point q at different times are presented in o, $q\rightarrow \rightarrow$ of Fig. 2(b). When o move toward q, the minimal distance between o and q can be calculated as d', where $d'>0$. When o catch up with q, the line segments of o and q intersect with each other, the minimal distance of o and q would be 0, where $-d \le d' \le 0$ and d indicating the intersection distance. When $d'<-d$ indicating that the line segments of o and q have no intersection, the query point q moves toward object o, denoted as q, $o\rightarrow \rightarrow$, where the minimal distance calculation of o and q can refer to case 1.4:

$$d' = d_{q,o}(t_0) - (o.v_M - q.v_m)(t - t_0)$$
$$d = ((o.v_M - o.v_m) + (q.v_M - q.v_m))(t - t_0)$$
$$d_{q,o}(t) = \begin{cases} d' & d' > 0 \\ 0 & -d \le d' \le 0 \end{cases}$$

Case 1.4. Query point q is moving towards object o with the same negative direction, denoted as q, $o\rightarrow \rightarrow$. The moving situations of object o and the query point q at different times are presented in q, $o\rightarrow \rightarrow$ of Fig. 2(b). When q move toward o, the minimal distance between o

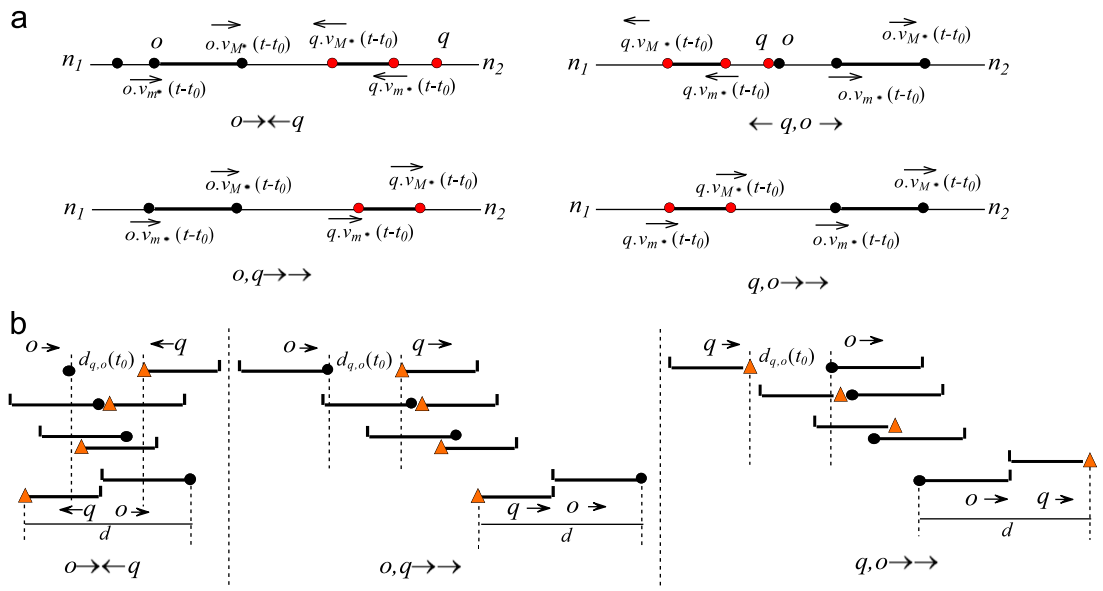

Fig. 2. Object o and query point q are on the same edge.

and q can be calculated as d', where $d' > 0$. When q catch up with o, the line segments of o and q intersect with each other, the minimal distance of o and q would be 0, where $-d \le d' \le 0$ and d indicating the intersection distance. When $d' < -d$ indicating that the line segments of o and q have no intersection, the query point o moves toward object q, denoted as $o, q \rightarrow \rightarrow$, where the minimal distance calculation of o and q can refer to case 1.3:

$$d' = d_{q,o}(t_0) - (q.v_M - o.v_m)(t - t_0)$$

$$d = ((o.v_M - o.v_m) + (q.v_M - q.v_m))(t - t_0)$$

$$d_{q,o}(t) = \begin{cases} d' & d' > 0 \\ 0 & -d \le d' \le 0 \end{cases}$$

Case 2. Object o and query point q are on two different edges. Assume that the query point q is moving on edge e_i, which starts from node n_i and ends at node n_i', and the object o is moving on edge e_j, which starts from n_j and ends at n_j'. Firstly, we give a definition about *distance determinate* to facilitate the distance calculation, as shown below.

Definition 3.1. Suppose that there are two different edges e_i and e_j in a road network, for any two objects o_i and o_j residing on edge e_i and e_j, respectively, no matter which object is on e_i or e_j, if each shortest distance path between e_i and e_j needs to pass through one same sub-path, we say that e_i and e_j are *distance determinate*.

For a pair of edges e_i and e_j in a road network, we can check whether e_i and e_j are *distance determinate* by calculating four distances between two nodes n_i and n_i' of e_i and two nodes n_j and n_j' of e_j. These four distances involves the shortest distance of (n_i, n_j) (denoted as SDN_{ij}), the shortest distance of (n_i, n_j') (denoted as $SDN_{ij'}$), the shortest distance of (n_i', n_j) (denoted as $SDN_{i'j}$), and the shortest distance of (n_i', n_j') (denoted as $SDN_{i'j'}$). If and only if these four distances satisfy Eq. (3.1), we conclude that e_i and e_j are *distance determinate*, because these four shortest network distances between e_i and e_j all pass through the sub-path from node n_i to n_j. Note that w in Eq. (3.1) indicates the weight of corresponding edge:

$$\begin{aligned} SDN'_{ij} &= SDN_{ij} + e_j.w \\ SDN'_{ij} &= SDN_{ij} + e_i.w \\ SDN'_{ij} &= SDN_{ij} + e_i.w + e_j.w \end{aligned} \tag{3.1}$$

The shortest path distance between two nodes, such as SDN_{ij}, $SDN_{ij'}$, $SDN_{i'j}$, and $SDN_{i'j'}$, can be calculated from the shortest path calculation tree using the TEDI method [30]. In the TEDI method, the road network is firstly decomposed into a tree in which the node of the tree contains more than one node from the road network according to the MBR (Minimal Boundary Rectangle). A bottom-up operation can be executed to calculate the shortest path distance for any two nodes of the road network. TEDI is an efficient calculation method, which offers orders-of-magnitude improvement over existing approaches. Using our proposed distance determinate method, two edges in road network, e.g. e_i and e_j, can be determined whether these two edges are distance determinate or not, denoted as a boolean variable DD_{ij}, if edge e_i and edge e_j are *distance determinate*, DD_{ij}=*True*, otherwise, DD_{ij}=*False*.

There are two different cases about calculating the minimal distance between q on edge e_i and o on edge e_j at time t (denoted as $d_{q,o}(t)$), one is that e_i and e_j are *distance determinate*, and the other considers that e_i and e_j are not *distance determinate*.

Case 2.1. Edge e_i and edge e_j are *distance determinate*. This case, DD_{ij}=*True*, means that the shortest network distance path of o and q should pass through the shortest distance sub-path of node n_i and node n_j. This sub-path distance is denoted as SDN_{ij}. Via this sub-path, object o and query point q can be on the same edge, where we can

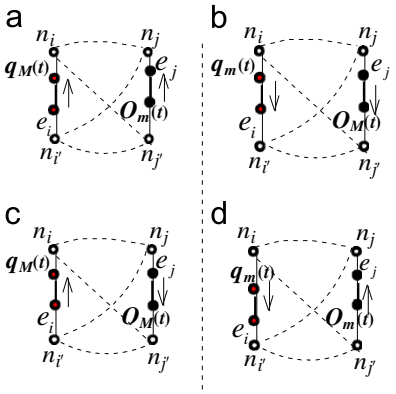

Fig. 3. Object o and query point q are on two different edges.

use the similar method described in case 1 to calculate the shortest distance between object o and query point q. Regarding different moving directions of query point q and object o, the calculation of $d_{q,o}(t)$ can be conducted as follows:

1. The moving directions of query point q and object o are described in Fig. 3(a):
$$d_{q,o}(t) = d_{q,o}(t_0) + SDN_{ij} - (q.v_M - o.v_M)(t - t_0)$$

2. The moving directions of query point q and object o are described in Fig. 3(b):
$$d_{q,o}(t) = d_{q,o}(t_0) + SDN_{ij} + (q.v_m + o.v_m)(t - t_0)$$

3. The moving directions of query point q and object o are described in Fig. 3(c):
$$d_{q,o}(t) = d_{q,o}(t_0) + SDN_{ij} + (q.v_m - o.v_M)(t - t_0)$$

4. The moving directions of query point q and object o are described in Fig. 3(d):
$$d_{q,o}(t) = d_{q,o}(t_0) + SDN_{ij} + (o.v_m - q.v_M)(t - t_0)$$

Case 2.2. Edge e_i and edge e_j are not *distance determinate*. The calculation of $d_{q,o}(t)$ is more complicated than the calculation described in case 2.1. We can use the pairwise distance computation method [32] to help to calculate $d_{q,o}(t)$.

We take the query point q on edge e_i (n_i, $n_{i'}$), object o on edge e_j (n_j, $n_{j'}$), and moving direction described in Fig. 3(a) as an example to illustrate the minimal distance calculations. The four possible shortest distance sub-paths for o and q are (n_i, n_j), (n_i, $n_{j'}$), ($n_{i'}$, n_j), and ($n_{i'}$, $n_{j'}$), while these four sub-path distances are denoted as SDN_{ij}, $SDN_{ij'}$, $SDN_{i'j}$, and $SDN_{i'j'}$, respectively. The minimal distance between q and o via these four sub-paths is calculated, respectively, denoted as $d^1_{q,o}(t)$, $d^2_{q,o}(t)$, $d^3_{q,o}(t)$, and $d^4_{q,o}(t)$, as shown below. The minimal distance $d_{q,o}(t)$ is equal to the minimum of these four distances. The minimal distance calculation for the query point q and object o with different moving directions, shown in Fig. 3(b)–(d), can be calculated using the same method, respectively:

$$d^1_{q,o}(t) = d_{q,o}(t_0) + SDN_{ij} - (q.v_M - o.v_M)(t - t_0)$$
$$d^2_{q,o}(t) = d_{q,o}(t_0) + SDN'_{ij} + (o.v_m - q.v_M)(t - t_0)$$
$$d^3_{q,o}(t) = d_{q,o}(t_0) + SDNi'j + (q.v_M - o.v_m)(t - t_0)$$
$$d^4_{q,o}(t) = d_{q,o}(t_0) + SDNi'j' + (q.v_m - o.v_m)(t - t_0)$$
$$d_{q,o}(t) = min(d^1_{q,o}(t), d^2_{q,o}(t), d^3_{q,o}(t), d^4_{q,o}(t)$$

3.2. Calculating the maximal distance $D_{q,o}(t)$

There are two different cases regarding the maximal distance calculation between a moving object o and the query point q with uncertain velocity. One case considers that o and q are on the same edge, and the other case is that o and q are on two different edges.

Case 1. Object o and query point q are on the same edge. The maximal network distance between q and o at time t is denoted as $D_{q,o}(t)$, which can be calculated in the following 1–4 equations, respectively, according to four different moving directions of o and q, shown in Fig. 2(a). $D_{q,o}(t_0)$ denotes the maximal distances between q and o at the start time t_0:

1. Object o is moving toward query point q, denoted as $o \rightarrow \leftarrow q$:
$$D_{q,o}(t) = D_{q,o}(t_0) - (o.v_m - q.v_m)(t - t_0)$$

2. Object o and query point q are moving with the opposite direction, denoted as $\leftarrow q$, $o \rightarrow$:
$$D_{q,o}(t) = D_{q,o}(t_0) + (o.v_M - q.v_M)(t - t_0)$$

3. Object o is moving towards q with the same positive direction, denoted as o, $q \rightarrow \rightarrow$:
$$D_{q,o}(t) = D_{q,o}(t_0) + (o.v_m - q.v_M)(t - t_0)$$

4. Query point q is moving towards object o with the same negative direction, denoted as q, $o \rightarrow \rightarrow$:
$$D_{q,o}(t) = D_{q,o}(t_0) + (o.v_M - q.v_m)(t - t_0)$$

Case 2. Object o and query point q are on two different edges. The *distance determinate* concept presented in Section 3.1 is also used here to help compute the maximal distance $D_{q,o}(t)$. The moving direction of q and o can be referred in Fig. 3. Two different cases about whether e_i and e_j are *distance determinate* or not are considered in the calculation of $D_{q,o}(t)$.

Case 2.1. Edge e_i and edge e_j are *distance determinate*. Regarding different moving directions of query point q and object o, the calculation of $D_{q,o}(t)$ can be conducted as follows:

1. The moving directions of query point q and object o are described in Fig. 3(a):
$$Dq,o(t) = D_{q,o}(t_0) + SDN_{ij} - (q.v_m - o.v_m)(t - t_0)$$

2. The moving directions of query point q and object o are described in Fig. 3(b):

$$D_{q,o}(t) = D_{q,o}(t_0) + SDN_{ij} + (q.v_M - o.v_M)(t - t_0)$$

3. The moving directions of query point q and object o are described in Fig. 3(c).

$$D_{q,o}(t) = D_{q,o}(t_0) + SDN_{ij} + (q.v_M - o.v_m)(t - t_0)$$

4. The moving directions of query point q and object o are described in Fig. 3(d):

$$D_{q,o}(t) = D_{q,o}(t_0) + SDN_{ij} + (o.v_M - q.v_m)(t - t_0)$$

Case 2.2. Edge e_i and edge e_j are not *distance determinate*. Similar to the calculation method used in the case 2.2 of Section 3.1, the maximal distance $D_{q,o}(t)$ for the o and q in Fig. 3(a) is equal to the maximum of four distances, denoted as $D^1_{q,o}(t)$, $D^2_{q,o}(t)$, $D^3_{q,o}(t)$,$D^4_{q,o}(t)$. These four distances can be calculated using the similar method to calculate $d^1_{q,o}(t)$, $d^2_{q,o}(t)$, $d^3_{q,o}(t)$, and $d^4_{q,o}(t)$:

$$D^1_{q,o}(t) = D_{q,o}(t_0) + SDN_{ij} - (q.v_m - o.v_m)(t - t_0)$$

$$D^2_{q,o}(t) = D_{q,o}(t_0) + SDN_{ij'} + (o.v_M - q.v_m)(t - t_0)$$

$$D^3_{q,o}(t) = D_{q,o}(t_0) + SDN_{i'j} + (q.v_M - o.v_m)(t - t_0)$$

$$D^4_{q,o}(t) = D_{q,o}(t_0) + SDN_{i'j'} + (q.v_M - o.v_M)(t - t_0)$$

$$D_{q,o}(t) = \max(D^1_{q,o}(t), D^2_{q,o}(t), D^3_{q,o}(t), D^4_{q,o}(t))$$

With the proposed DIM model, we can calculate the minimal and maximal distances at time t between moving object o and query point q with uncertain velocity to obtain the distance interval $[d_{q,o}(t), D_{q,o}(t)]$. The distance interval can be used to indicate the possible distance between o and q while processing the *CKNN* queries.

4. The TPR^{uv}-tree

In order to obtain the information of moving objects more efficiently, TPR-tree [33] like methods have been used to index moving object. Tao and Papadias [2] proposed a repetitive query processing approach with TPR-tree for processing C*K*NN queries, but the method is only applicable to static objects. Huang et al. [34] proposed a TPR$^{(s,d)}$-tree to efficiently index moving objects with uncertain speed and direction in Euclidean space. In TPR$^{(s,d)}$-tree, the objects are recursively grouped into a bottom-up manner according to their locations at the time when the index is built, while the speed interval and moving boundary information of each object are involved in each node. However, the moving boundary of each object is applicable for the Euclidean space, and not restricted by the road network. Chen and Meng [35] proposed an AU index schema to index the moving objects in road network. In order to decrease the overhead of keeping the index updated with the frequently changing object location data, AU (Adaptive

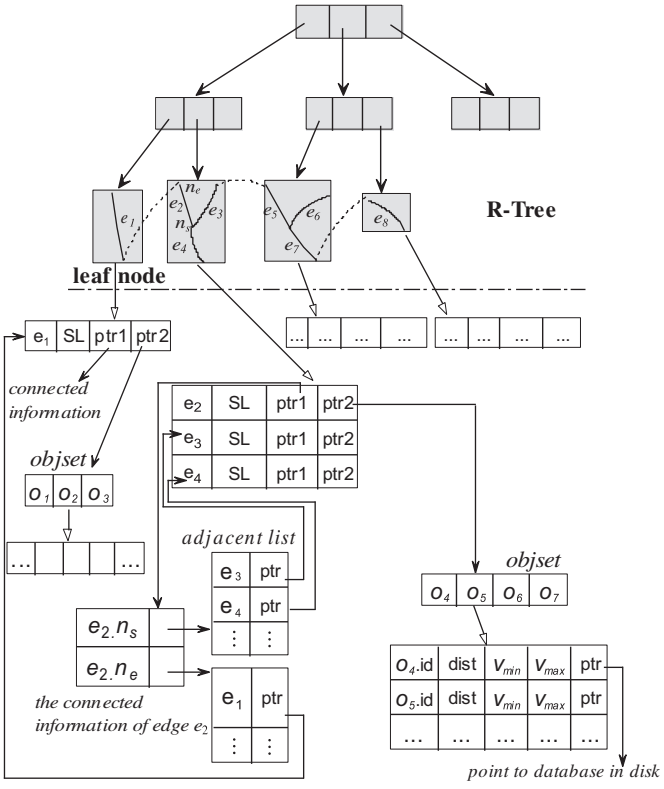

Fig. 4. The structure of TPR^{uv}-tree.

Unit) is used to group objects having similar moving patterns to dynamically adapting itself to cover the movement of the objects it contains. However, when an object moves from an edge in a leaf node of the AU index to an edge in another leaf node of the AU index, we should search from the root node of R-tree of AU index to update the corresponding object location information.

Our proposed query method calculates the pruning distance and divides the query time interval according to the time when the object moves across an intersection from one edge to another. Therefore, when an object moves from one edge to another, the index structure should be updated. In order to decrease the update cost of index structure for the objects moving from one edge to another edge, an index structure, namely TPR^{uv}- tree, is proposed to efficiently index moving objects with uncertain velocity in road network by involving edge connection and moving objects information, where *uv* indicates *u*ncertain *v*elocity.

As shown in Fig. 4, the top level of TPR^{uv}-tree indexes the spatial information of the road network, and each leaf node consists of edges included in the corresponding MBR. For example, in Fig. 4, the left most leaf node consists of edge e_1, the left second most leaf node consists of edges e_2, e_3, e_4, the left third most leaf node consists of edges e_5, e_6, e_7, and the left fourth most leaf node consists of edge e_8. For each leaf node, there is a direct access table involving the information of edges. Each entry of the direct access table consists of the edge ID, Speed limit (*SL*), pointer1 pointing to the edge connection information (*ptr*1), and pointer2 pointing to the object information (*ptr*2). For example of left second most leaf node, the direct access table of this leaf node is composed of three entries representing e_2, e_3, e_4 separately. For entry of e_2, *ptr*1 points to the edge connection information, where the starting node $e_2.n_s$ of e_2 connecting to the edge e_3, e_4, and the end node $e_2.n_e$ of e_2 connecting to the edge e_1. Two adjacent lists are connected with the starting and end node of the edge, respectively. Each entry of adjacent list is composed of the ID and the address of the connected edge. Still for the entry of e_2, *ptr*2 points to the moving object set *objset* on the edge e_2, where is $\{o_4, o_5, o_6, o_7\}$. There is also a direct access table to record the information of these moving objects. Each entry of this access table is composed of *o.id* representing the object identity, *o.dist* denoting the distance between object *o* and the starting node of the edge where it resides, $o.v_{min}$ and $o.v_{max}$ referring to the minimal and the maximal moving velocity, respectively, and *o.ptr* pointing to the object tuple in the database.

Based on the index structure TPR^{uv}-tree, it is more efficient to update the object information when an object moves from one edge to another edge. For example, when the object o_4 moves from the edge e_2 to the edge e_1, we can obtain the address of e_1 from the adjacent list connected to the end node $e_2.n_e$ of e_2. Then we can delete o_4 from the object set *objset* and corresponding object information in the access table, and inserting o_4 to the object set *objset* and corresponding object information in the access table of edge e_1. Therefore, we can avoid searching from the root node of TPR^{uv}-tree to find the information of connected edge to update corresponding object information.

5. PV*K*NN algorithms

The proposed DIM model can be used to calculate the distance intervals for the moving objects near to the query point. Based on the distance intervals of moving objects, we investigate to determine which *K* objects are the nearest ones to the query point. It is complicated to determine which objects are the *K*NN to the query point at each time instant, due to the high repetitive re-evaluation when the velocity of moving object updates. Since the real applications pay more attention on more appropriate query results rather than the most accurate results, we propose a Possibility-based Vague *K*NN (PV*K*NN) algorithm to determine the possibility of moving objects being the *K*NN to the query point within a given time interval.

The main process of PV*K*NN algorithm includes:

1. A *pruning phase*: efficiently prunes the objects that are impossible to be the *K*NN query results within the given time interval.
2. A *distilling phase*: obtains the division time subintervals of the given time interval where the object candidates can be distilled, meanwhile the minimal and maximal distances between moving objects and query point can be represented as a linear function of time.
3. A *possibility-ranking phase*: determines the moving objects with the possibility to be the *K*NN query results within each division time subintervals.

In the following, we present these three phases of PV*K*NN algorithm in detail.

5.1. Pruning phase

The main idea of pruning phase in the C*K*NN queries is to scale down the C*K*NN query region using a pruning distance, denoted as $d_{pruning}$. For any moving object, if the minimal distance between this object and query point is less than or equal to $d_{pruning}$ within a given time interval, this moving object can be considered in the process of C*K*NN queries, otherwise this object will be pruned. The calculation of the pruning distance is defined as $d_{pruning}=D_{o,q}^{max}+D_{add}^{max}$. The explanations about $D_{o,q}^{max}$ and D_{add}^{max} are presented in the following.

$D_{o,q}^{max}$ indicates the maximal distance among all the maximal distances between moving objects and query point within a given time interval (denoted as $[t_0, t_n]$). In order to continuously monitor the *K*NN result within the time interval $[t_0,t_n]$, we divide the time interval into several time subintervals within which the directions of moving objects are certain. At the start time t_0, for *K* moving objects nearest to the query point, we calculate the time instants for each object and query point reaching an intersection of the road network with their maximal moving velocity. The fastest one of these time instants,

denoted as t_1,is chosen to divide the time interval $[t_0,t_n]$ into two subintervals: $[t_0, t_1]$ and $[t_1, t_n]$. Next, the subinterval $[t_1, t_n]$ is divided into $[t_1, t_2]$ and $[t_2, t_n]$ with the similar process. This process is repeated until the chosen time instant is larger than t_n. Thus, the time interval $[t_0,t_n]$ is divided into $[t_0,t_1]$, $[t_1,t_2]$,…$[t_{j-1}, t_j]$,…$[t_{n-1},t_n]$. For each time subinterval, we just need to focus on the distance between moving object and query point at the time instant of boundary value. These chosen time instants are t_0, t_1, t_2, …, t_j…, t_{n-1}, and t_n. $D_{q,o_i}(t_j)$ is calculated to obtain the maximal distance between the i^{th} moving object and query point at time t_j using the method presented in the Section 3.2. After calculating all of the maximal distances between K objects nearest to the query point at t_0 and query point at all chosen time instants, the maximal one is chosen to be the $D_{q,o}^{max}$, shown in Eq. (5.1.1):

$$D_{q,o}^{max} = \max\{D_{q,o_i}(t_j)\} \quad (1 \le i \le K, 0 \le j \le n) \tag{5.1.1}$$

The D_{add}^{max} indicates an additional distance to the distance $D_{q,o}^{max}$ within the given time interval $[t_0,t_n]$ Taking the query point as a start point, if this point moves the distance $D_{q,o}^{max}$ via all the possible edges in the road network, it will terminate on a point of any of these possible edges. We take these terminated points on the possible edges as boundary points (denoted as P_{bound}). For an edge with P_{bound} and an object on this edge, if the minimal distance between this object and query point at t_0 is larger than $D_{q,o}^{max}$, and the object is moving with the maximal speed limit of this edge toward the P_{bound} within the time interval $[t_0,t_n]$, if this object can reach or pass through the P_{bound}, we need to consider this object in the CKNN query. Therefore, we need to add a distance to the $D_{q,o}^{max}$ to be the pruning distance, in case missing any possible object to be the KNN. If the number of P_{bound} is m, then the number of edges with boundary point is m, assume that the maximal speed limit of an edge with P_{bound} is denoted as SL_x $(1 \le x \le m)$, we can calculate the additional distance for each edge as $SL_x(t_n - t_0)$. The maximal one of these calculated additional distances is chosen to be D_{add}^{max}, shown in Eq. (5.1.2):

$$D_{add}^{max} = \max\{SL_x(t_n - t_0)\} (1 \le x \le m) \tag{5.1.2}$$

The pruning distance $d_{pruning}$ is composed of $D_{q,o}^{max}$ and D_{add}^{max}. Furthermore, we can determine whether an object is considered in the CKNN query process or not by calculating the minimal distance between this object and query point within the time interval $[t_0,t_n]$, denoted as $d_{q,o}^{min}$. With the chosen time instants of $[t_0,t_n]$, we can calculate the minimal distance between the object and query point at a chosen time instant t_j, denoted as $d_{q,o}(t_j)$, using the similar method presented in Section 3.1. By calculating the minimal distances at all chosen time instants, the minimal one is chosen to be the $d_{q,o}^{min}$, as shown in Eq. (5.1.3):

$$d_{q,o}^{min} = \min\{d_{q,o}(t_j)\} (0 \le j \le n) \tag{5.1.3}$$

Algorithm 1 presents the pruning phase on the whole. Firstly, for the K nearest objects to the query point at the start time t_0, the distance $D_{q,o}^{max}$ is calculated with the chosen time instants from the given time interval $[t_0,t_n]$. Secondly, an additional distance D_{add}^{max} is calculated based on the boundary point P_{bound}, which is determined by the distance $D_{q,o}^{max}$. Then, the pruning distance $d_{pruning}$ is calculated as $d_{pruning} = D_{q,o}^{max} + D_{add}^{max}$. Finally, whether an object o needs to be pruned or not is determined by comparing its minimal distance $d_{q,o}^{min}$ within the time interval $[t_0,t_n]$ with the pruning distance $d_{pruning}$. If $d_{q,o}^{min} \le d_{pruning}$, the object can be a candidate for the CKNN query, otherwise, the object will be pruned.

Algorithm 1. Pruning Phase

Input: M moving objects near to query point q within the time interval $[t_0, t_n]$, K moving objects (denoted as O_i) nearest to the q at the start time $t_0 (K \le M)$ in a road network
Output: A set $O_{candidate}$, including the moving objects, which can be the candidates for the CKNN query within $[t_0, t_n]$

1. For each object O_i and query point q within $[t_0,t_n]$
   calculate time instant reaching an road intersection with maximal velocity
2. Choose the minimal time instant as t_1, dividing $[t_0,t_n]$ to $[t_0,t_1]$, $[t_1,t_n]$
3. For $[t_1,t_n]$, repeat process 1, then choose time instant t_2, dividing $[t_1,t_n]$ to $[t_1,t_2]$ and $[t_2,t_n]$
4. Repeat process 1, 2 and 3, until the chosen time instant $> t_n$, the division time subintervals are: $[t_0,t_1]$, $[t_1,t_2]$,…$[t_{j-1},t_j]$,…$[t_{n-1},t_n]$. The boundary value of subinterval is denoted as t_j
5. For each object O_i
   calculate maximal distance $D_{q,o_i}(t_j)$ $(1 \le i \le K, 0 \le j \le n)$
6. Choose the maximal one to be the $D_{q,o}^{max}$, $D_{q,o}^{max} = \max\{D_{q,o_i}(t_j)\}$
7. For each possible moving direction of q
   q Moves the distance $D_{q,o}^{max}$; mark the terminate point on the edge as P_{bound};
8. The number of edges with P_{bound} is denoted as m;
9. The maximal speed limit of edge with P_{bound} is denoted as SLx $(1 \le x \le m)$;
10. For each edge with P_{bound}
    Calculate $SLx(t_n - t_0)$;
11. Choose the maximal one to be $D_{add}^{max} = \max\{SL_x(t_n - t_0)\}$;
12. Determine the pruning distance: $d_{pruning} = D_{o,q}^{max} + D_{add}^{max}$;
13. For an object o of M objects, within $[t_0, t_n]$:
    Calculate: $d_{q,o}(t_j)$ $(0 \le j \le n)$
    $d_{q,o}^{min} = \min\{d_{q,o}(t_j)\}$ $(0 \le j \le n)$
    If $(d_{q,o}^{min} \le d_{pruning})$ then insert o into the set $O_{candidate}$
    Else prune o

We take an example to illustrate the pruning phase. As illustrated in Fig. 5(a), there are four moving objects near query point q with uncertain velocity within the time interval $[t_0, t_3]$ in a road network, denoted as o_1, o_2, o_3, and o_4. Our task is to process continuously 2NN query over q within $[t_0, t_3]$. At the start time t_0, o_1 and o_3 are 2 nearest objects to q. In the pruning phase, we need to scale down the object candidates, which will be considered in the 2NN query process. Firstly, the time interval $[t_0, t_3]$ is divided into three time subintervals: $[t_0, t_1]$, $[t_1, t_2]$ and $[t_2, t_3]$ using our proposed time division method. Secondly, we can calculate the maximal distances between o_1, o_3 and q at time instants: t_0, t_1, t_2, and t_3, respectively. These maximal distances are represented as $D_{q,o_1}(t)$, $D_{q,o_3}(t)$ in Fig. 5(b). Then we can figure out that the maximal one of these maximal distances is $D_{q,o_1}(t_3)$, which can be the $D_{q,o}^{max}$. Thirdly, query point q moves distance $D_{q,o}^{max}$ via possible directions to obtain boundary points P_{bound}. As shown in Fig. 5(a), there are four boundary points on the edges. Each boundary point is marked as a short line on the edge, such as the short line

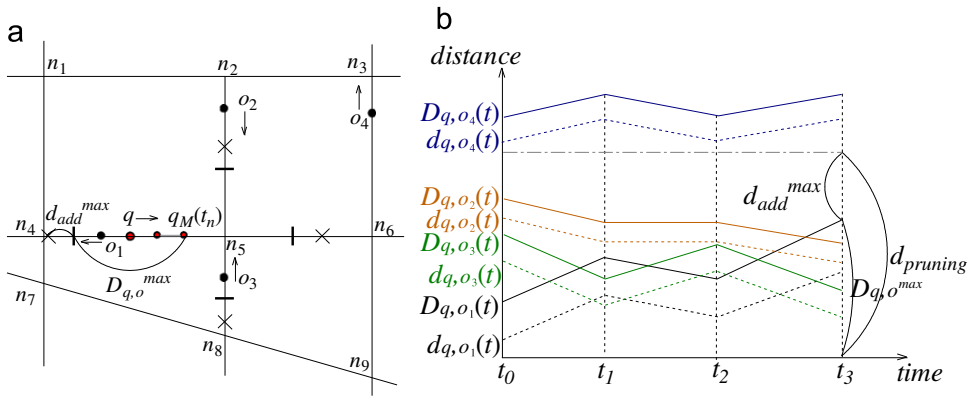

Fig. 5. An example about the pruning phase: (a) objects in a road network and (b) distance functions in time–distance space.

on the edge $[n_4, n_5]$. For each edge with the boundary point, we calculate the moving distance with the velocity of maximal speed limit of this edge within $[t_0, t_3]$. The maximal one of these calculated four distances is chosen to be D_{add}^{max}. The D_{add}^{max} is added to the $D_{q,o}^{max}$ to become the $d_{pruning}$. As shown in Fig. 5(b), the distance $d_{pruning}$ is presented as a gray dotted line. Finally, for each object of o_1, o_2, o_3 and o_4, we calculate the minimal distance between the object and q within $[t_0, t_3]$, denoted as $d_{qo_1}(t)$, $d_{q,o_2}(t)$, $d_{q,o_3}(t)$ and $d_{q,o_4}(t)$. As shown in Fig. 4(b), the lines of $d_{q,o_1}(t)$, $d_{q,o_2}(t)$, $d_{q,o_3}(t)$ are below the gray dotted line, and the line of $d_{q,o_4}(t)$ is above the gray dotted line. Therefore, we can conclude that the objects o_1, o_2, and o_3 will be considered in the following 2NN query process, and the object o_4 is pruned. The object candidates within $[t_0, t_3]$ can be denoted as $O_{candidate}=\{o_1, o_2, o_3\}$.

5.2. Distilling phase

With the pruning phase, we can prune the objects, which cannot be the *K*NN query results. The obtained object candidates are valid for the whole given time interval. However, during a time subinterval of the given time interval, some objects of the object candidates still can be excluded. In order to make the process of *K*NN query more efficient, we propose a candidate distilling method to distill the object candidates within a time subinterval where the objects, which cannot be the *K*NN are excluded. Since the *K*NN query is to find K nearest object to the query point, we mainly use the K^{th} smallest one of the maximal distances between the object candidates $O_{candidate}$ from the pruning phase and the query point as the excluding reference. During the given time interval $[t_0, t_n]$, the maximal distance between an object and query point is the K^{th} smallest one within a time subinterval, this object may be replaced by another object whose maximal distance to the query point is the K^{th} smallest one within another time subinterval. Therefore, we firstly determine the time subinterval where the maximal distance between one object and query point is the K^{th} smallest one (denoted as $D_{q,o}^{k}(t)$). Then, for each time subinterval, and each object of $O_{candidate}$ except the object whose maximal distance is $D_{q,o}^{k}(t)$, if the minimal distance between the object and query point is larger than $D_{q,o}^{k}(t)$, this object will be excluded from the $O_{candidate}$ for this time subinterval.

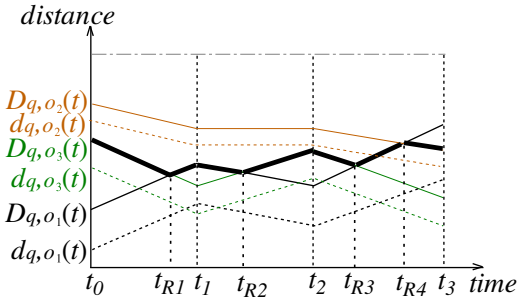

Fig. 6. Time subinterval division in distilling phase.

We still use the example shown in Fig. 5 to illustrate the basic idea of distilling phase. As our task is to process the 2NN query over query point q, we mainly find the 2nd maximal distance between the objects of $O_{candidate}$ (i.e.$\{o_1, o_2, o_3\}$) and q as the exclude reference within the time interval $[t_0, t_3]$. During the time subinterval $[t_0, t_1]$, the 2nd maximal distance between o_1, o_2, o_3 and query point q is changed from the maximal distance between o_3 and q (denoted as $D_{q,o_3}(t)$) to the maximal distance between o_1 and q (denoted as $D_{q,o_1}(t)$). The change time instant is denoted as t_{R1}, which indicates that the 2nd maximal distance is changed from $D_{q,o_3}(t)$ to $D_{q,o_1}(t)$. The t_{R1} can be obtained from the intersection point of line $D_{q,o_1}(t)$ with line $D_{q,o_3}(t)$ within $[t_0, t_1]$. The change time instant t_{R2}, t_{R3}, and t_{R4} can be obtained similarly, shown as in Fig. 6. The lines indicating the 2nd maximal distance are all marked with trick black polyline within $[t_0, t_3]$. During the time subinterval $[t_0, t_{R1}]$, as the line $d_{q,o_2}(t)$ is above line $D_{q,o_3}(t)$, which means that the minimal distance between o_2 and q is larger than $D_{q,o}^{k}(t)$, the object o_2 is excluded from $O_{candidate}$. The distilling result is recorded as a tuple $\langle\{o_1, o_3\}, [t_0, t_{R1}]\rangle$, where the maximal distance between o_3 and q is the 2nd maximal distance. However, o_1 or o_3 which is nearer to q has not been determined. During the time subinterval $[t_{R1}, t_1]$, the 2nd maximal distance is $D_{q,o_1}(t)$. As the line $d_{q,o_2}(t)$ is above line $D_{q,o_1}(t)$, o_2 is excluded from the $O_{candidate}$. Then the distilling result is recorded as a tuple $\langle\{o_3, o_1\}, [t_{R1}, t_1]\rangle$, where the maximal distance between o_1 and q is the 2nd maximal distance. For the time subinterval $[t_{R4}, t_3]$, the 2nd

maximal distance is $D_{q,o_2}(t)$, as line $d_{q,o_1}(t)$ and line $d_{q,o_3}(t)$ are all below the line $D_{q,o_2}(t)$, no object will be excluded. Then the distilling result is $\langle\{o_1, o_3, o_2\}, [t_{R4}, t_3]\rangle$. With the distilling phase, we can obtain a new object candidate set, $O_{candidate}=\{\langle\{o_1, o_3\}, [t_0,t_{R1}]\rangle, \langle\{o_3, o_1\}, [t_{R1}, t_1]\rangle, \langle\{o_3, o_1\}, [t_1, t_{R2}]\rangle, \langle\{o_1, o_3\}, [t_{R2}, t_2]\rangle, \langle\{o_1, o_3\}, [t_2, t_{R3}]\rangle, \langle\{o_2, o_3, o_1\}, [t_{R3}, t_{R4}]\rangle, \langle\{o_1, o_3, o_2\}, [t_{R4}, t_3]\rangle\}$.

In the following, we explain the distilling phase in detail. For the *C*KNN query within the given time interval $[t_0, t_n]$, using our proposed pruning method, we can scale down the object candidates to $O_{candidate}=\{o_1, o_2 \ldots o_i \ldots o_m\}$ $(m \geq K)$. Referring to the time subinterval division in the pruning phase, the given time interval $[t_0, t_n]$ is firstly divided into $[t_0,t_1]$, $[t_1,t_2]$,$\ldots[t_{j-1},t_j]$,$\ldots[t_{n-1},t_n]$. For each subinterval, as the direction of moving object is certain, the linear equation of distance with time can be determined. For example, the maximal distance between o_i and q, $D_{q,o_i}(t)(t_{j-1} \leq t \leq t_j)$, can be expressed as a linear equation with time. For each time subinterval $[t_{j-1},t_j]$, we investigate to find time instants to divide the subinterval further, so as to distill the object candidates. At the start time t_0, we calculate the maximal distances between all the objects of $O_{candidate}$ and q, if the K^{th} smallest one is the maximal distance between o_i and q, then we use $D_{q,o_i}(t)$ as the reference to find the division time instant within $[t_0,t_1]$, $D^k_{q,o}(t)=D_{q,o_i}(t)$. For each object (denoted as o_j) of $O_{candidate}$ except o_i, we calculate the time instant when the $D_{q,o_j}(t)=D_{q,o_i}(t)(t_0<t<t_1)$. If existing such a time instant t, this time instant is denoted as $T_j=t$. If we obtain more than one such time instant, the smallest one is chosen as the division time instant (denoted as T^0_{R1}). If the smallest time instant is T_x, then $T^0_{R1}=T_x$, and the maximal distance between o_x and q will be $D^k_{q,o}(t)$, $D^k_{q,o}(t)=D_{q,o_x}(t)$. For the time subinterval $[T^0_{R1}, t_1]$, the tuple $\langle o_x, T^0_{R1}\rangle$ is also recorded. For the time subinterval $[T^0_{R1}, t_1]$, we repeat such process to find the division time instants, which can be denoted as T^0_{R2}, $T^0_{R3}\ldots T^0_{Ri}$, until no such time distant can be found. For each division time instant, such as T^0_{Ri}, the corresponding object o_x whose maximal distance to q is $D^k_{q,o}(t)$ during the divided time subinterval $[T^0_{Ri}, t_1]$ is also recorded as $\langle o_x, T^0_{Ri}\rangle$. For the time subinterval $[t_{j-1}, t_j]$, we can find the division time instants using the same method, denoted as T^{j-1}_{R1}, $T^{j-1}_{R2}\ldots T^{j-1}_{Ri}$. After determining the division time instants, we can obtain time subintervals for the given time interval $[t_0, t_n]$, which could be $[t_0, T^0_{R1}]\ldots$ $[T^0_{Ri}, t_1]\ldots$ $[T^{j-1}_{Ri-1}, T^{j-1}_{Ri}]\ldots$, $[T^{n-1}_{Ri}, t_n]$. For each time subinterval, we compare the minimal distances between the objects of $O_{candidate}$ and q with the $D^k_{q,o}(t)$ to distill the object candidates, while the distilled objects are recorded in an object candidate set, denoted as O^j_i. Finally, we can obtain a new object candidate set, expressed as $O_{candidate}=\{\langle O^0_0, [t_0, T^0_{R1}]\rangle \ldots \langle O^0_i, [T^0_{Ri}, t_1]\rangle, \ldots \langle O^{j-1}_{i-1}, [T^{j-1}_{Ri-1}, T^{j-1}_{Ri}]\rangle, \ldots \langle O^{n-1}_i, [T^{n-1}_{Ri}, t_n]\rangle\}$.

Algorithm 2. Distilling Phase

Input: The output from pruning Phase: $O_{candidate}$, time subinterval division $[t_0,t_1]$, $[t_1,t_2]$,$\ldots[t_{j-1},t_j]$,$\ldots[t_{n-1},t_n]$

Output: A set $O_{candidate}$, including the moving objects, which can be the candidates for *C*KNN query within the time subinterval of $[t_0,t_n]$

1. For each object o in $O_{candidate}$
   Calculate the maximal distance at the start time t_0: $D_{q,o}(t_0)$
2. If the K^{th} smallest one is the maximal distance between o_i and q by comparing the calculated maximal distances
   $D^k_{q,o}(t)=D_{q,o_i}(t)$
3. For the time subinterval $[t_0,t_1]$, for each object o_j in $O_{candidate}-\{o_i\}$
   Find the time instant t, $D_{q,o_j}(t)=D^k_{q,o}(t)$
   If t exists, $T_j=t$;
4. Comparing all the $T_j(0<j\leq m-1)$
   If smallest one $T_x=\min(T_j)$
   then is the division time instant, $T^0_{R1}=T_x$,$\langle o_x, T^0_{R1}\rangle$ is also recorded
5. For the time subinterval $[T^0_{R1}, t_1]$, $D^k_{q,o}(t)=D_{q,o_x}(t)$
   Repeat 3, 4 to obtain the division time instants T^0_{R2}, T^0_{R3},$\ldots T^0_{Ri}$
6. For the time subinterval $[t_{j-1},t_j]$
   Repeat 3,4,5 to obtain the division time instants T^{j-1}_{R1}, T^{j-1}_{R2} ,$\ldots T^{j-1}_{Ri}$;
7. Time subintervals can be obtained: $[t_0, T^0_{R1}]\ldots[T^0_{Ri}, t_1]$, $[t_1, T^1_{R1}]\ldots[T^{j-1}_{Ri-1}, T^{j-1}_{Ri}]\ldots$, $[T^{n-1}_{Ri}, t_n]$
8. For each time subinterval $[T^{j-1}_{Ri-1}, T^{j-1}_{Ri}]$ and $\langle o_x, T^{j-1}_{Ri-1}\rangle$, then $D^k_{q,o}(t)=D_{q,o_x}(t)$
   For each object o_j in $O_{candidate}-\{o_x\}$
   if $d_{q,o_j}(t)<D^k_{q,o}(t)$
   then insert o_j in O^{j-1}_{i-1}
9. An object candidate set is obtained, expressed as $O_{candidate}=\{\langle O^0_0, [t_0, T^0_{R1}]\rangle \ldots \langle O^0_i, [T^0_{Ri}, t_1]\rangle, \ldots \langle O^{j-1}_{i-1}, [T^{j-1}_{Ri-1}, T^{j-1}_{Ri}]\rangle, \ldots \langle O^{n-1}_i, [T^{n-1}_{Ri}, t_n]\rangle\}$

5.3. Possibility-ranking phase

With the distilling phase, we determine the time subintervals of given time interval to distill the object candidates for the *C*KNN query process. For each time subinterval, though the object candidates are determined, the K objects nearest to q has not been obtained, and which object in these K objects is nearer to q has not been determined as well. In the possibility-ranking phase, we propose methods to tackle these two issues to obtain the *C*KNN query results.

As the velocity of moving object is uncertain, it is difficult to calculate the distance between the object and query point at each time instant. For each object o_i in the object candidate set within the division time subinterval, we can calculate the minimal distance (denoted as d_i) and maximal distance (denoted as D_i) between o_i and q. The distance between o_i and q at any time instant within the time subinterval should be within the distance interval $[d_i, D_i]$. Since the users pay more attentions on which objects are the *K*NN within a time interval, the possibility of o_i, which could be one nearest object to q within a division time subinterval, can be calculated by comparing the distance interval $[d_i, D_i]$ with the distance intervals of other objects. Firstly, we compare the distance intervals of two moving objects, denoted as o_1 and o_2. The distance interval of o_1 is denoted as $[d_1, D_1]$, and the distance interval of o_2 is denoted as $[d_2, D_2]$. The relative positions of $[d_1, D_1]$ and $[d_2, D_2]$ can be different, which may result in that o_1 or o_2 is nearer to the query point.

As shown in Fig. 7(a), $D_1<d_2$, it is clear that the distance between o_1 to q is definitely less than the distance between o_2 to q. We can conclude that o_1 is nearer to q than o_2. Similarly, as shown in Fig. 7(e), since $D_2<d_1$, which indicates that the distance between o_2 to q is definitely less than the distance between o_1 to q, we can conclude that o_2 is nearer to q than o_1. For these two cases, it is explicit to determine o_1 or o_2 is nearer to q. However, as shown in

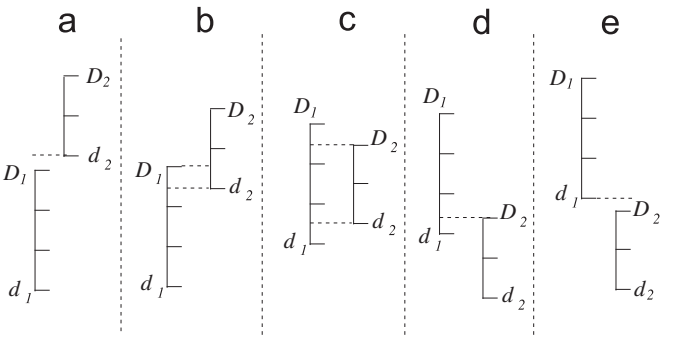

Fig. 7. Different relative positions of distance intervals of two moving objects.

Fig. 7(b)–(d), since two distance intervals $[d_1,D_1]$ and $[d_2,D_2]$ intersect, it is difficult to determine which object is nearer to the query point precisely. According to the relative position of $[d_1,D_1]$ and $[d_2,D_2]$, we can calculate the possibility about o_1 is nearer to q than o_2, and the possibility about o_2 is nearer to q than o_1 as well, by using the vague set theory. The symbol $o_1 \prec o_2$ denotes that o_1 is nearer to q than o_2, and $p(o_1 \prec o_2)$ denotes the possibility about o_1 is nearer to q than o_2. Definition 5.3.1 demonstrates how to determine which object is nearer to q according to the possibility value.

Definition 5.3.1. Given two objects o_1 and o_2, which are in the object candidate set within the division time subinterval, if $p(o_1 \prec o_2) > 0.5$, o_1 is nearer to the query point q than o_2. If $p(o_1 \prec o_2)=0.5$ that o_1 or o_2 is nearer to q is randomly determined.

According to the vague set theory, the possibility about o_1 or o_2 is nearer to q should satisfy the following properties.

Property 5.3. *Given two objects o_1 and o_2 and their corresponding distance intervals $[d_1, D_1]$ and $[d_2, D_2]$:*

(1) $0 \le p(o_1 \prec o_2) \le 1$;

(2) $p(o_1 \prec o_2)+p(o_2 \prec o_1)=1$, $p(o_1 \prec o_1)=0.5$;

(3) $p(o_1 \prec o_2) \ge 0.5$ if and only if $D_2-d_1 \ge D_1-d_2$ and

(4) If $p(o_1 \prec o_2) \ge 0.5$ and $p(o_2 \prec o_3) \ge 0.5$, then $p(o_1 \prec o_3) \ge 0.5$.

The possibility $p(o_1 \prec o_2)$ can be calculated based on the proportion of intersection segment between two distance intervals $[d_1, D_1]$ and $[d_2, D_2]$ in the whole distance intervals, as shown in Definition 5.3.2.

Definition 5.3.2. Given two objects o_1 and o_2 and their corresponding distance interval $[d_1, D_1]$ and $[d_2, D_2]$, the possibility about o_1 is nearer to the query point q than o_2 can be calculated as Eq. (4.3.1):

$$p(o_1 \prec o_2) = \frac{\mathrm{Max}(0,(D_1-d_1+D_2-d_2)-\mathrm{Max}(0,D_1-d_2))}{D_1-d_1+D_2-d_2} \tag{5.3.1}$$

For Eq. (5.3.1), we formally verify that all the properties in Property 5.3 can be satisfied. The proof for each property is presented as below:

Proof of Property 5.3.

(1) As $0 \le Max\,(0, (D_1-d_1+D_2-d_2)\text{-}Max\,(0, D_1-d_2)) \le D_1-d_1+D_2-d_2$,
Then $0 \le p(o_1 \prec o_2) \le 1$;

(2) As $p(o_1 \prec o_2)+p(o_2 \prec o_1)=(Max(0,(D_1-d_1+D_2-d_2)-Max(0,D_1-d_2)+Max(0,(D_1-d_1+D_2-d_2)-Max(0,D_2-d_1))/(D_1-d_1+D_2-d_2)$

a. if $D_1-d_2 \ge 0$ and $D_2-d_1 \ge 0$
Then $p(o_1 \prec o_2)+p(o_2 \prec o_1)=(D_2-d_1+D_1-d_2)/(D_1-d_1+D_2-d_2)=1$;

b. if $D_1-d_2 < 0$ and $D_2-d_1 > 0$
Then $p(o_1 \prec o_2)+p(o_2 \prec o_1)=(D_2-d_1+D_1-d_2+Max(0, D_1-d_2))/(D_1-d_1+D_2-d_2)=1$;

c. if $D_1-d_2 > 0$ and $D_2-d_1 < 0$
Then $p(o_1 \prec o_2)+p(o_2 \prec o_1)=(Max(0, D_2-d_1)+D_2-d_1+D_1-d_2)/(D_1-d_1+D_2-d_2)=1$;
Especially, if o_2 is o_1, then $p(o_1 \prec o_1) = \frac{D_1-d_1}{2(D_1-d_1)} = 0.5$;

(3) Sufficiency: if $D_2-d_1 \ge D_1-d_2$ then $p(o_1 \prec o_2) \ge 0.5$

a. if $D_1-d_2 \ge 0$, $p(o_1 \prec o_2) = \frac{D_2-d_1}{D_1-d_2+D_2-d_1} \ge \frac{D_2-d_1}{2(D_2-d)} \ge 0.5$;

b. if $D_1-d_2 < 0$, $p(o_1 \prec o_2) = \frac{D_1-d_1+D_2-d_2}{D_1-d_1+D_2-d_2} = 1 \ge 0.5$;

Necessity: if $p(o_1 \prec o_2) \ge 0.5$ then $D_2-d_1 \ge D_1-d_2$

a. if $D_1-d_2 \ge 0$, $p(o_1 \prec o_2) = \frac{D_2-d_1}{D_1-d_2+D_2-d_1} \ge 0.5$
then $D_2-d_1 \ge (D_2-d_1+D_1-d_2)/2$
so $D_2-d_1 \ge D_1-d_2$;

b. if $D_1-d_2 < 0$, $p(o_1 \prec o_2) = \frac{D_2-d_1}{D_1-d_2+D_2-d_1} \ge 0.5$
then $D_2-d_1 \ge 0$,
so $D_2-d_1 \ge D_1-d_2$.

(4) if $P(o_1 \prec o_2) \ge 0.5$, then $D_2-d_1 \ge D_1-d_2$ according to property (3)
if $P(o_2 \prec o_3) \ge 0.5$, then $D_3-d_2 \ge D_2-d_3$ according to property (3)
thus $D_2-d_1+D_3-d_2 \ge D_1-d_2+D_2-d_3$
then $D_3-d_1 \ge D_1-d_3$
therefore, $P(o_1 \prec o_3) \ge 0.5$, according to property (3).

For five different cases shown in Fig. 6, we can calculate $p(o_1 \prec o_2)$ using Eq. (5.3.1). For the case in Fig. 7(a), $p(o_1 \prec o_2)=1$, the case in Fig. 7(e), $p(o_1 \prec o_2)=0$, no matter the values of two distance intervals. For the case in Fig. 7(b), assume that $[d_1, D_1]=[3.7]$ and $[d_2, D_2]=[4.5, 9.5]$, we can obtain $p(o_1 \prec o_2)=0.72$, which means that o_1 is nearer to query q than o_2. For the case in Fig. 7(c), assume that $[d_1, D_1]=[4, 15]$ and $[d_2, D_2]=[4.5, 9.5]$, we can obtain $p(o_1 \prec o_2)=0.34$, which indicates that o_2 is nearer to query q than o_1. For the case in Fig. 7(d), assume that $[d_1, D_1]=[8, 11]$ and $[d_2, D_2]=[4.5, 9.5]$, we can obtain $p(o_1 \prec o_2)=0.79$, which means that o_1 is nearer to query q than o_2.

With Eq. (5.3.1), for each object (denoted as o_i) in the object candidate set within a division time subinterval, we can calculate the possibility that o_i is nearer to the query point than another object (denoted as o_j) in the object candidate set, denoted as $p_{ij}=p(o_i \prec o_j)$. If there are m objects in the object candidate set, we can determine the possibility about o_i being one nearest neighbor to the query point (denoted as P_i) by calculating the portion of the sum of possibility o_i to any other object in the sum of all of the possibility about any two objects in the object candidate set, as shown in Definition 5.3.3. Since $p_{ji}=1-p_{ij}$, then $p_{ij}+p_{ji}=1$, the sum of all of the possibility about any two objects in m objects is $\sum_{i=1}^{m}\sum_{j=1}^{m} p_{ij} = m^2/2$.

Definition 5.3.3. For the object o_i in the object candidate set within a division time subinterval, the possibility about o_i is one nearest neighbor to the query point q can be calculated as Eq. (5.3.2):

$$P_i = \frac{\sum_{j=1}^{m} p_{ij}}{\sum_{i=1}^{m}\sum_{j=1}^{m} p_{ij}} = \frac{2\sum_{j=1}^{m} p_{ij}}{m^2} \tag{5.3.2}$$

We still use the example shown in Fig. 6 to illustrate how to calculate the possibility for the object, which can be one nearest neighbor to the query point. Referring to the distilling phase, we can obtain the object candidate set as $\{o_2, o_3, o_1\}$ within the division time subinterval $[t_{R3}, t_{R4}]$. Using Eq. (5.3.1), we calculate the possibility about one object of $\{o_2, o_3, o_1\}$ is nearer to q than another object in $\{o_2, o_3, o_1\}$, denoted as $p_{11}=0.5$, $p_{12}=0.37$, $p_{13}=0.67$, $p_{21}=0.63$, $p_{22}=0.5$, $p_{23}=0.95$, $p_{31}=0.33$, $p_{32}=0.05$, $p_{33}=0.5$. According to Eq. (5.3.2), we calculate the possibility for o_1, o_2 and o_3, respectively. While $P_1=2(0.5+0.37+0.67)/9=0.34$, similarly, $P_2=0.46$ and $P_3=0.2$. By sorting these possibilities in descending order, it is (0.46, 0.34, 0.2). We can conclude that the 2NN query result within $[t_{R3},t_{R4}]$ is $\langle[t_{R3},t_{R4}],\{(o_2,0.46),(o_1,0.34)\}\rangle$, which indicates that the possibilities of o_2 and o_1, which are two nearest neighbors are higher than the possibility of o_3, which can be the nearest neighbor to the query point.

The possibility-ranking phase algorithm is presented as below:

Algorithm 3. Possibility-ranking Phase

Input: The output from distilling Phase:
$O_{candidate}=\{\langle O_0^0, [t_0, T_{R1}^0]\rangle,\ldots\langle O_i^0, [T_{Ri}^0, t_1]\rangle,\ldots\langle O_{i-1}^{j-1}, [T_{Ri-1}^{j-1}, T_{Ri}^{j-1}]\rangle,\ldots\langle O_i^{n-1}, [T_{Ri}^{n-1}, t_n]\rangle\}$
Output: KNN query results, including the K objects with possibility for time subinterval
1. For each object o_i in O_{i-1}^{j-1} within $[T_{Ri-1}^{j-1}, T_{Ri}^{j-1}]$
   Calculate the distance interval $[d_i, D_i]$
2. For any two object o_i and o_j in O_{i-1}^{j-1} within $[T_{Ri-1}^{j-1}, T_{Ri}^{j-1}]$
   Calculate $p(o_i \prec o_j)$ using Eq. (5.3.1)
   $p_{ij}=p(o_i \prec o_j)$
3. For each object o_i in O_{i-1}^{j-1} within $[T_{Ri-1}^{j-1}, T_{Ri}^{j-1}]$
   Calculate P_i using Eq. (5.3.2)
   P_i is recorded in (o_i, P_i)
4. Sorting the calculated P_i in descending order
   The first K objects are the query result for $[T_{Ri-1}^{j-1}, T_{Ri}^{j-1}]$

6. Performance evaluations

In this section, we present the results of several experiments to analyze the performance of the proposed PVKNN algorithm. Firstly, the performance of PVKNN algorithm is analyzed by measuring the CPU time of processing a CKNN query. Several important effective aspects of CKNN queries on performance of PVKNN algorithm are evaluated, which are the density of moving objects, the value of K, and the velocity interval of each moving object. Secondly, the precision and false negative ratio of PVKNN algorithm are evaluated by measuring the percentage of obtaining real KNNs and missing KNNs, respectively. Thirdly, the performance of PVKNN algorithm is compared with another CKNN query approach mainly on the factor of precision. Especially, the TPR^{uv}-tree is used as the underlying index in these two approaches to improve the search performance. Finally, the update cost of TPR^{uv}-tree is estimated with different numbers of moving objects and data updates.

6.1. Experimental data sets

All the experiments are conducted on an Intel Pentium(R) Dual CPU 1.86 GHz with 2 GB RAM running Windows *XP* Professional. The page size is set to 4KB. The algorithm is implemented using Microsoft Visual Studio C++ and the STL library. One synthetic data set is used in our simulation, which consists of 20,000 nodes. A real network road network of Oldenburg in Germany is used in the experiments, as shown in Fig. 8 (http://www.census.gov/geo/www/tiger/), which consists of 6,105 nodes. In order to demonstrate the scalability of the proposed PVKNN algorithm, the experiments are also conducted on a larger real road network, which is the network of San Francisco Bay Area (BAY) in US (http://www.dis.uniroma1.it/~challenge9/download.shtml), which consists of 321,270 nodes. A network-based Generator of Moving Objects [36] is used to

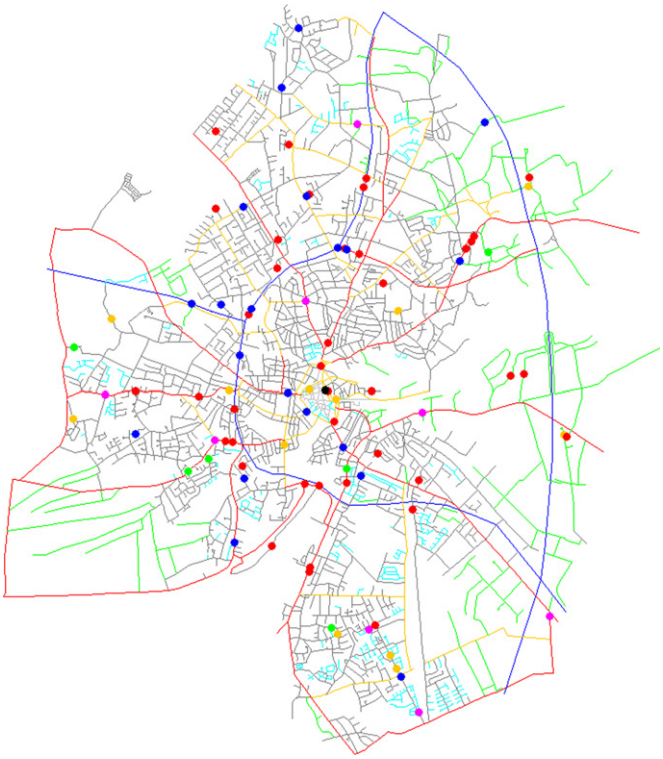

Fig. 8. Road network of Oldenburg city.

Table 2
Parameter setting in the experiments.

Parameters	Value
Number of query objects	1k
Number of moving objects	**10k**, 20k,…,100k
Density of moving objects	**10%**, 15%, 20%, 25%
Number of data updates	**100k**,…,500k,…,1000k
Number of *K*	1, 5, **10**, 20
Multiple of minimal velocity of moving objects	1, **2**, 5, 10
Dataset	Synthetic dataset, Oldenburg dataset, San Francisco Bay Area (BAY) dataset

generate the moving objects and query objects, whose start locations are uniformly spread over a road network with different densities from 10% to 25% (density will be explained in Section 6.2).

The minimal velocity of a moving object can be randomly set between 0 and 4 m per time unit. The maximal velocity is designed to be a multiple of the minimal velocity, and the value of multiple is randomly chosen from 1 to 10. The moving velocity interval and direction of each object would change when it reaches a road intersection. For the query objects, the setting method of moving velocity interval is similar to that of the moving objects. The C*K*NN query may vary in the value of *K*, such as 1.5,10 and 20, and we set the default value of *K* as 10 and set the default length of query time interval as 100 time units. Table 2 presents the setting of parameters used in the experiments. The values in bold face are used as the default values of corresponding parameters in our experiments.

6.2. Performance of the PV*K*NN algorithm

First of all, we analyze the performance of PV*K*NN algorithm by measuring the running time (CPU time) of processing a C*K*NN query, where three important effective factors are chosen to demonstrate the performance of PV*K*NN algorithm. These important factors are the density of moving objects, the value of *K*, and the velocity interval of each moving object. Here, we define the density of moving objects as follows:

$$\text{Density} = \frac{\text{the number of moving objects}}{\text{the number ofedges in the road network}}$$

Fig. 9(a)–(c) gives the running time of PV*K*NN algorithm in processing a C*K*NN query for the synthetic dataset, Oldenburg dataset and BAY dataset, respectively, under various densities of moving objects varying from 10% to 25%. The experimental result shows that running time would increase when the density of moving objects increases. Especially, when the density of moving objects changes from 20% to 25%, the running time would increase more severely but still smaller than 3.5 s. Next, we analyze the performance of the PV*K*NN algorithm with different *K* varying from 1 to 20 for the synthetic, the Oldenburg and BAY data sets, respectively. As shown in Fig. 9(d)–(f), the running time would increase when *K* is equal to 20, because it is more costly to determine which object among all the object candidates is *K*NN as the number of object candidates becomes larger with larger *K*.

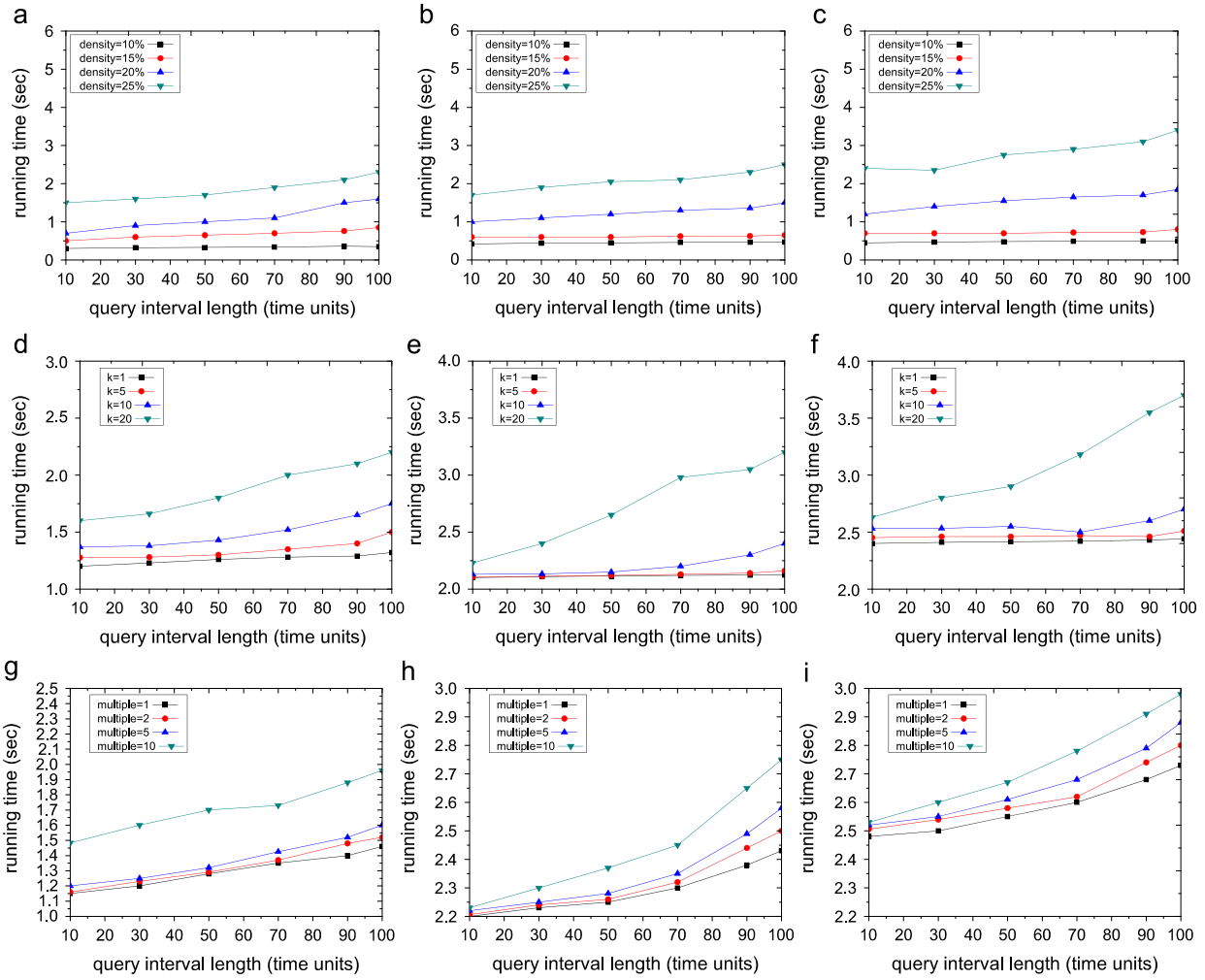

Fig. 9. Performance of the PV*K*NN algorithm: (a) different density (Synthetic data), (b) different density (Oldenburg data), (c) different density (BAY), (d) different *K* (synthetic data), (e) different *K* (Oldenburg data), (f) different *K* (BAY), (g) different velocity interval (synthetic data), (h) different velocity interval (Oldenburg) and (i) different velocity interval (BAY).

Fig. 9(g)–(i) illustrates the running time of PV*K*NN algorithm for the synthetic, the Oldenburg and BAY data sets, respectively, with different velocity intervals of moving objects. The maximal velocity of the velocity interval is set as the multiple of minimal velocity, and the value of multiple is set as 1, 2, 5 and 10 accordingly. When the value of multiple is 1, we can use the PV*K*NN algorithm to efficiently processing C*K*NN queries over moving objects with fixed velocity in road network. When the value of multiple increases from 1 to 2 or from 2 to 5, the increase of running time is not so significant. However, when the value of multiple increases from 5 to 10, the running time increases relative significantly, because the 10 multiple would make the velocity interval quite larger to bring about larger pruning distance, which means that quite more object candidates should be monitored in the distilling phase and possibility-ranking phase of our PV*K*NN algorithm.

In summary, the running time for the PV*K*NN algorithm processing a C*K*NN query is short. With the increase of the density of moving objects, the value of *K*, and the value of multiple about the velocity interval, the running time of the PV*K*NN approach increases slightly within an acceptable range.

6.3. *Precision of the PVKNN algorithm*

In this subsection, the precision of the PV*K*NN algorithm is measured by the percentage of real *K*NNs are obtained, represented as $\#(V\mathit{K}NN_{obtain} \cap \mathit{K}NN_{real})/\#(V\mathit{K}NN_{obtain}) \times 100\%$. The set of real *K*NNs, denoted as $\mathit{K}NN_{real}$, can be found by calculating the exact distance between each object and the query point. The set of obtained *K*NNs, denoted as $V\mathit{K}NN_{obtain}$, can be obtained by the possibility-ranking phase of PV*K*NN algorithm (note that only the *K* higher-possibility objects are selected finally). The precision analysis can demonstrate what percentage of real *K*NNs is obtained, meanwhile what percentage of obtained *K*NNs is mistaken as the real *K*NNs, which can be presented as false positive ratio and calculated as (1—precision). However, the precision

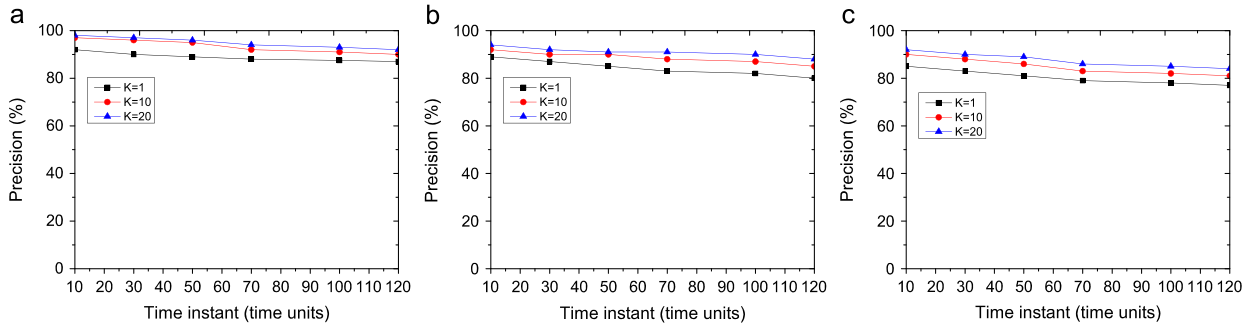

Fig. 10. Precision of the PV*K*NN algorithm. (a) Synthetic data, (b) Oldenburg data and (c) BAY data.

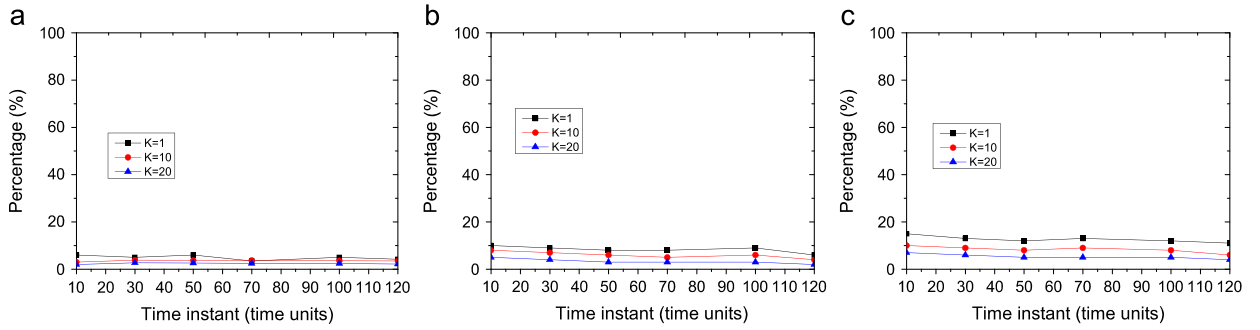

Fig. 11. False negative ratio of PV*K*NN algorithm. (a) Synthetic data, (b) Oldenburg data and (c) BAY data.

cannot measure what percentage of real *K*NNs is missed, which can be presented as false negative ratio. The false negative ratio is represented as $(\#(K\text{NN}_{real})—\#(V K\text{NN}_{obtain} \cap K\text{NN}_{real}))/\#(V K\text{NN}_{obtain}) \times 100\%$, where $K\text{NN}_{real}$ and $V K\text{NN}_{obtain}$ can be acquired from the precision analysis process as well.

Fig. 10(a)–(c) analyzes the precision of PV*K*NN algorithm at different time instants for the synthetic, Oldenburg and BAY data sets, respectively. The experimental results show that when *K* increases, the precision of the PV*K*NN algorithm increases slightly, because the possibility of obtaining the real *K*NNs in the set of $V K\text{NN}_{obtain}$ increases with the increase of *K*. When time passes, because there are more object candidates to be determined whether they are obtained as *K*NNs, then the set of obtained object candidates would be larger, the precision would decreases slightly. However, the precision could be always about 85% in all cases, which demonstrate the effectiveness of our proposed PV*K*NN algorithm. Besides precision, as shown in Fig. 11(a)–(c), we evaluate the false negative ratio of PV*K*NN algorithm for the synthetic, Oldenburg and BAY data sets, respectively. When *K* increases, the false negative ratio decrease slightly, because the possibility of missing the real *K*NNs in the set of $V K\text{NN}_{obtain}$ decreases with the increase of *K*. When time passes, the false negative ratio decreases slightly, because the set of obtained object candidates becomes larger as the time passes. From the experimental results of precision and false negative ratio, the high precision and low false negative ratio make the PV*K*NN algorithm be a valid method to process the C*K*NN query over moving objects with uncertain velocity in the road network.

6.4. Comparison

In this subsection, we compare the PV*K*NN algorithm with a C*K*NN approach [15], which can efficiently process C*K*NN query over moving objects with fixed velocity in terms of the precision. The precision of the C*K*NN algorithm is represented as the proportion X/*K*, where X is the number of the real *K*NNs retrieved by the refining phase and *K* is the number of objects required to retrieve by issuer. In the C*K*NN approach, when processing the moving objects with uncertain velocity, the average value of maximal and minimal velocity of each moving object is used as the fixed velocity of the moving object. Just for fairness, we also apply our proposed index structure, TPR^{uv}-tree presented in Section 4, to the referred C*K*NN approach. The TPR^{uv}-tree can efficiently index moving objects with uncertain velocity in road network by involving edge connection and moving objects information. In the TPR^{uv}-tree, for the C*K*NN approach, the maximal and minimal velocities of each object are set as their average value.

Fig. 12(a)–(c) compares the precision of PV*K*NN and C*K*NN according to different query time interval lengths (varying from 10 to 120 time units) with the synthetic, Oldenburg and BAY data sets, respectively. From the experimental results, we can see that the precision of C*K*NN approach is about 80%, and the precision of the PV*K*NN algorithm can be about 90% in all different query interval lengths. The reason why the precision of the PV*K*NN algorithm is higher than that of the C*K*NN approach is that the velocity interval of each object in PV*K*NN algorithm can describe the motion of moving

object better, which can make the possibility of obtaining the real *K*NNs higher. In addition, when time passes, the precision of PV*K*NN algorithm would decreases slightly, as discussed in Section 6.3. For the C*K*NN approach, when time passes, the precision would also decrease, because the pruning distance would become larger with time passes, which brings about more object candidates to be determined as real *K*NNs, and then the possibility of retrieving real *K*NNs from larger object candidate set decreases.

The number of *K* is another critical parameter affecting the precision of processing C*K*NN queries. As shown in Fig. 12(d)–(f), when *K* increases from 1 to 20, the precision of the PV*K*NN algorithm increases slightly, as discussed in Section 6.3. For the C*K*NN approach, the change of precision is not stable according to the increase of *K*, since the possibility of retrieving the real *K*NNs is not so related with the value of *K*. No matter how *K* changes, the precision of the PV*K*NN algorithm is higher than that of the C*K*NN approach, which indicates that the PV*K*NN algorithm is quite suitable for the highly dynamic environments in which objects change their velocities frequently.

6.5. Update cost

We estimate the update performance of the index update TPR^{uv}-Tree in terms of index update I/Os with different number of moving objects and data updates using the synthetic, Oldenburg, and BAY data sets, respectively.

Fig. 13(a)–(c) analyzes the index update I/Os with the number of moving objects varying from 10 K to 100 K. When the number of moving objects increases, the index update I/Os would increases slightly. In the BAY dataset, the index update I/Os is more than that of the Synthetic and Oldenburg dataset, since the road network of BAY is more complicated than Synthetic and Oldenburg. If the road network is more complicated, when a moving object reaches a road intersection, the TPR^{uv}-Tree needs more I/Os cost to delete the object information from an *objset* and

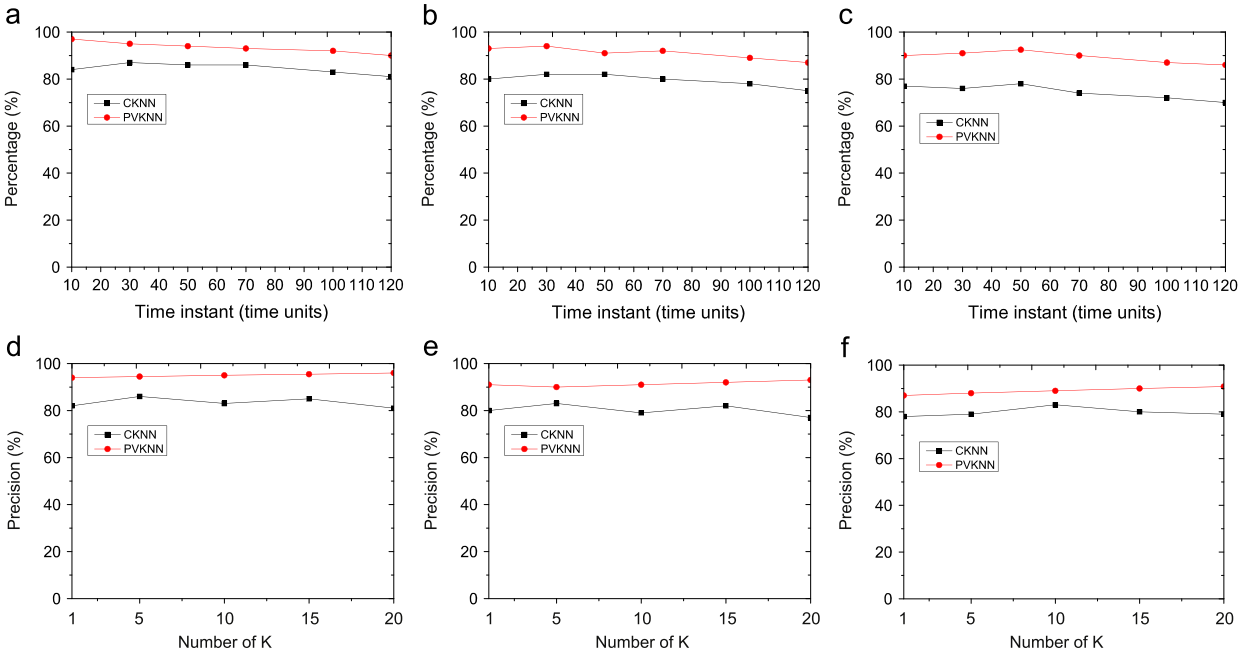

Fig. 12. PV*K*NN algorithm versus C*K*NN algorithm (a) Query time (Synthetic data), (b) Query time (Oldenburg data), (c) Query time (BAY), (d) Number of K (Synthetic data) (e) Number of K (Oldenburg data) and (f) Number of K (BAY).

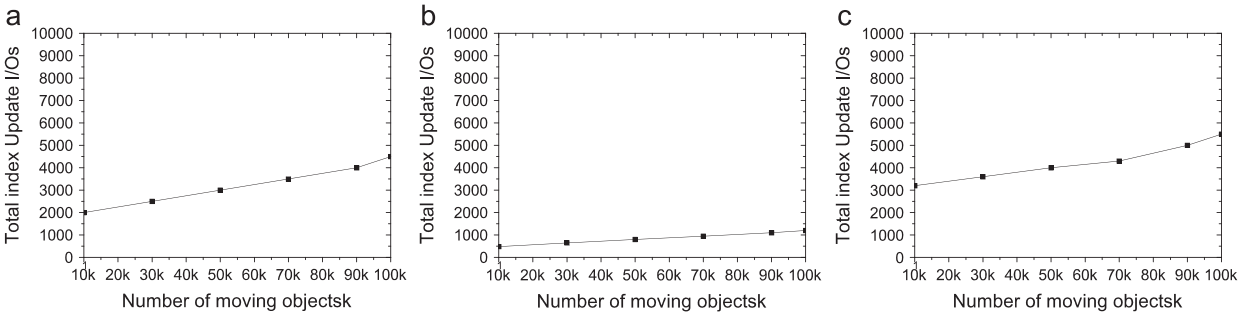

Fig. 13. Update cost over different number of moving objects. (a) Synthetic data, (b) Oldenburg data and (c) BAY data.

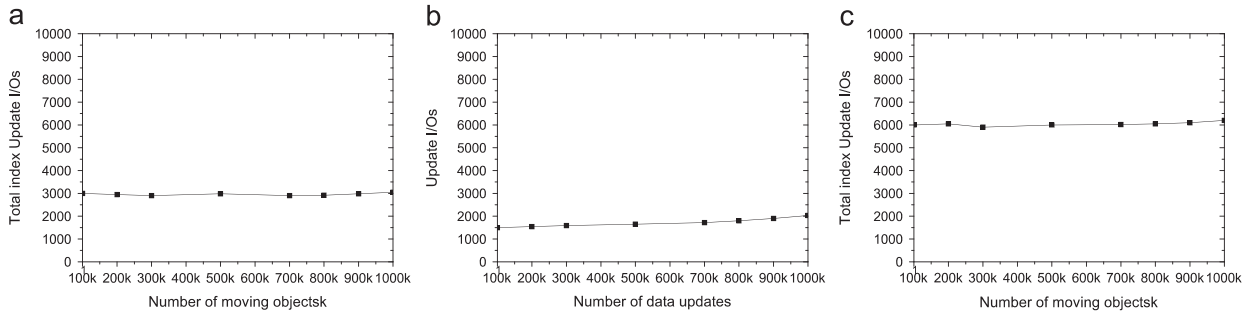

Fig. 14. Update cost over query time. (a) Synthetic data, (b) Oldenburg data and (c) BAY data.

insert it to another *objset*. As shown in Fig. 14(a)–(c), when the number of data updates increases from 100 K to 1000 K, the changes of the index update I/Os are slim, because the data updates only incurs the update of the *objset* connecting to the access table of the leaf node of the TPR^{uv}-Tree without cost of node splitting and propagation of MBR updates. Getting from the experiment results, we can conclude that the direct access table of TPR^{uv}-Tree involving the connected information of edges and objects has a great contribution to improve the update performance.

7. Conclusions

In this paper, we studied the problem of processing C*K*NN query over moving objects with uncertain velocity in road networks within a given time interval. We proposed a Distance Interval Model (DIM) to calculate the distance interval for a moving object with uncertain velocity and direction at a time instant, which involves the minimal and maximal distances between a moving object and the query point. With the calculation methods of DIM, we developed the PV*K*NN algorithm to efficiently process a C*K*NN query. The pruning phase of PV*K*NN algorithm is used to pruning the objects, which are impossible to be the query result within the given time interval. Then, the distilling phase is designed to determine the division time subintervals of the given time interval where the qualified object candidates are distilled, meanwhile the minimal and maximal distances between the moving objects and the query point can be represented as a linear function of time. Finally, the possibility-ranking phase is presented to determine the *K*NN query results with possibility for each division time subinterval. Simulation experiments have demonstrated the efficiency and effectiveness of the proposed approach.

There are several interesting avenues for future work. One extension of this work is to process continuous range monitoring of mobile objects in road networks and reverse nearest-neighbor queries with the proposed approach. Another extension is to further enhance the precision of C*K*NN query results.

Acknowledgements

This work was supported by National Natural Science Fund of China under grants 61173049 and 61100059, Doctoral Fund of Ministry of Education of China under grant 20090142110023, and the Science and Technology Research Project of Department of Education of Hubei Province under grant Q20102807.

References

[1] Y. Tao, D. Papadias, Q. Shen, Continuous nearest neighbor search, in: International Conference on Very Large Data Bases, Hong Kong, China, August 20–23, 2002.
[2] Y. Tao, D. Papadias, Time parameterized queries in spatio-temporal databases, in: Proceedings of the ACM SIGMOD, Madison, Wisconsin, 2002.
[3] K. Raptopoulou, A.N. Papadopoulos, Y. Manolopoulos, Fast nearest-neighbor query processing in moving-object databases, GeoInformatica 7 (2) (2003) 113–137.
[4] V.T. de Almedia, Towards optimal continuous nearest neighbor queries in spatial databases, in: Proceedings of the ACM GIS, November 10–11, 2006.
[5] R. Benetis, C.S. Jensen, G. Karciauskas, S. Saltenis, Nearest neighbor and reverse nearest neighbor queries for moving objects, in: Proceedings of the International Database Engineering and Applications Symposium, Canada, July 17–19, 2002.
[6] R. Benetis, C.S. Jensen, G. Karciauskas, S. Saltenis, Nearest neighbor and reverse nearest neighbor queries for moving objects, VLDB Journal 15 (3) (2006) 229–249.
[7] M.R. Kolahdouzan, C. Shahabi, Alternative solutions for continuous K nearest neighbor in spatial network databases, GeoInformatica 9 (4) (2004) 321–341.
[8] M.R. Kolahdouzan, C. Shahabi, Continuous *K* nearest neighbor queries in spatial network databases, in: Proceedings of the Spatio-temporal Databases Management, 2004.
[9] Y.F. Li, J. Yang, J.W. Han, Continuous *K*-nearest Neighbour Search for Moving Objects, SSDBM, 2004.
[10] G.S. Iwerks, H. Samet, K. Smith, Continuous *K*-nearest Neighbor Queries for Continuously Moving Points with Updates, VLDB, 2003.
[11] Y. Tao, D. Papadias, Q.M. Shen, Continuous Nearest Neighbor Search, VLDB, 2002.
[12] M.R. Kolahdouzan, C. Shahabi, Continuous *K*-nearest Neighbor Queries in Spatial Network Databases, STDBM, 2004.
[13] G. Iwerks, H. Samet, K. Smith, Continuous *K*-nearest neighbor queries for continuously moving points with updates, in: Proceedings of the International Conference on Very Large Databases, Berlin, Germany, 2003.
[14] K.C.K. Lee, H.V. Leong, J. Zhou, A. Si, An efficient algorithm for predictive continuous nearest neighbor query processing and result maintenance, in: Proceedings of the International Conference on Mobile Data Management, 2005.
[15] Y.-K. Huang, Z.W. Chen, C. Lee, Continuous *K*-nearest neigh bor query over moving objects in road networks, APWeb/WAIM2009, Lecture Notes in Computer Science, vol. 5446, 2009, pp. 28–38.
[16] R. Cheng, D.V. Kalashnikov, S. Prabhakar, Querying imprecise data in moving object environments, IEEE Transactions on Knowledge and Data Engineering 16 (9) (2004) 1112–1127.
[17] Y. Tao, C. Faloutsos, D. Papadias, B. Liu, Prediction and indexing of moving objects with unknown motion patterns, in: Proceedings of the ACM SIGNOD, Maison de la Chimie, Paris, France, June 13–18, 2004.

[18] M.R. Kolahdouzan, C. Shahabi, Voronoi-based *K* nearest neighbor search for spatial network databases, in: Proceedings of the VLDB, 2004, pp. 840–851.
[19] H.J. Cho, C.W. Chung, An efficient and scalable approach to CNN queries in a road network, in: Proceedings of the International Conference on Very Large Data Bases, Trondheim, Norway, 2005.
[20] K. Mouratidis, M.L. Yiu, D. Papadias, N. Mamoulis, Continuous nearest neighbor monitoring in road networks, in: Proceedings of the International Conference on Very Large Data Bases, Seoul, Korea, 2006.
[21] G.H. Li, P. Fan, Y.H. Li, J.Q. Du, An Efficient Technique for Continuous *K*-nearest Neighbor Query Processing on Moving Objects in a Road Network, CIT, 2010.
[22] K. Mouratidis, M. Hadjieleftheriou, D. Papadias, Conceptual partitioning: an efficient method for continuous nearest neighbor monitoring, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2005.
[23] X. Xiong, M.F. Mokbel, W.G. Aref, SEA-CNN: scalable processing of continuous *K* nearest neighbor queries in spatio-temporal databases, in: Proceedings of the International Conference on Data Engineering, 2005.
[24] X. Yu, K.Q. Pu, N. Koudas, Monitoring *K*-nearest neighbor queries over moving objects, in: Proceedings of the International Conference on Data Engineering, 2005.
[25] Y.-K. Huang, C.-C. Chen, C. Lee, Continuous *K*-nearest neighbor query for moving objects with uncertain velocity, GeoInformatica 13 (2009) 1–25.
[26] A. Civilis, C.S. Jensen, S. Pakalnis, Techniques for efficient road-network-based tracking of moving objects, IEEE TKDE 17 (5) (2005) 698–712.
[27] O. Wolfson, H. Yin, Accuracy and resource consumption in tracking and location prediction, in: Proceedings of the Symposium of Spatial and Temporal Databases, 2003, pp. 325–343.
[28] H. b. Hu, D.L. Lee, Victor C.S. Lee, Distance Indexing on Road Networks, VLDB, 2006.
[29] H. Samet, J. Sankaranarayanan, H. Alborzi, Scalable Network Distance Browsing in Spatial Databases, SIGMOD, 2008.
[30] F. Wei, TEDI: Efficient Shortest Path Query Answering on Graphs, SIGMOD, 2010.
[31] G.H. Li, Y.H. Li, L.C. Shu, Ping Fan, CkNN Query Processing over Moving Objects with Uncertain Speeds in Road Networks, ApWeb, 2011.
[32] Z.D. Xu, H.-A. Jacobsen, Processing Proximity Relations in Road Networks, SIGMOD, 2010.
[33] S. Saltenis, C.S. Jensen, S.T. Leutenegger, M.A. Lopez, Indexing the positions of continuously moving objects, in: Proceedings of the ACM SIGMOD, 2000.
[34] Y.-K. Huang, S.-J. Liao, C. Lee, Evaluating continuous *K*-nearest neighbour query on moving with uncertainty, Information System 34 (2009) 415–437.
[35] J.D. Chen, X.F. Meng, Update-efficient indexing of moving objects in road networks, Geoinformatica 13 (4) (2008) 397–424.
[36] T. Brinkhoff, A framework for generating network-based moving objects, GeoInformatica 6 (2) (2002) 153–180.