Published in final edited form as:

Telecommun Syst. 2014 March ; 55(3): 403–419. doi:10.1007/s11235-013-9795-x.

Continuous K -Nearest Neighbor Processing Based on Speed and Direction of Moving Objects in a Road Network

Guohui Li¹, Ping Fan^{1,2,*}, and Ling Yuan¹

¹School of Computer Science and Technology, Hua Zhong University of Science and Technology, Wuhan, China

²School of Computer Science, Xianning University, Hubei, China

Abstract

Recent research has focused on Continuous K -Nearest Neighbor (CKNN) query over moving objects in road networks. A CKNN query is to find among all moving objects the K -Nearest Neighbors (KNNs) of a moving query point within a given time interval. As the data objects move frequently and arbitrarily in road networks, the frequent updates of object locations make it complicated to process CKNN accurately and efficiently. In this paper, according to the relative moving situation between the moving objects and the query point, a Moving State of Object (MSO) model is presented to indicate the relative moving state of the object to the query point. With the help of this model, we propose a novel Object Candidate Processing (OCP) algorithm to highly reduce the repetitive query cost with pruning phase and refining phase. In the pruning phase, the data objects which cannot be the KNN query results are excluded within the given time interval. In the refining phase, the time subintervals of the given time interval are determined where the certain KNN query results are obtained. Comprehensive experiments are conducted and the results verify the effectiveness of the proposed methods.

Keywords

Continuous K -Nearest Neighbor query; Road network; Moving speed; Moving direction

1. Introduction

In recent years, with the rapid development of mobile communication[14], wireless communication[6,7] and positioning technology, mobile users can enjoy sorts of convenient services such as location information query [24], nearest neighbor query [1,2,5,11,12], range query, traffic conditions query etc. K Nearest Neighbor (KNN) query [9,18–20,22,23] is one important type of these services, which is to find the K Nearest Neighbors (KNN) of a moving user among all moving objects.

There are two different kinds of KNN query. One is to retrieve KNN results at a time instant [13, 17], namely *static queries*. Another is to continuously monitoring KNN query, which is

*Corresponding author: Ping Fan, School of Computer Science, Huazhong University of Science and Technology, Wuhan, China, fanping1028@gmail.com.

Continuous K Nearest Neighbor (CKNN) query. For a CKNN query, the KNN results need to be continuously returned to the user during a time interval. As the data objects move freely, the KNN query results change over time according to the change of object locations, which makes the query process more complicated than the process of *static queries*.

An example of CKNN query is shown in Fig. 1. The problem is to find two nearest neighbors to the query point q among five moving objects. At different time instants, 2NN query results could be different, as shown in Fig.1 (a) and Fig.1 (b). In Fig. 1(a), at time t_0 , two nearest neighbors to q are o_1 and o_4 . In Fig. 1(b), at time t_n , 2NN query results are changed to o_1 and o_3 . Hence, the frequent location updates of moving objects bring about the repetitive updates of query results.

Existing methods in CKNN query mostly aim at Euclidean spaces [10, 15, 23]. For the moving objects in road networks, the distance computation between data object and query point is quite different from that of Euclidean spaces. The distance in road network is defined as the length of the shortest path connecting data object and query point. Some papers proposed snapshot algorithms to process CKNN queries over static objects to a moving query point in road networks [4,12,16]. Huang [8] proposed a method based on periodical snapshot recalculation to process CKNN query where both data objects and query points move continuously in a road network. The main problem of this method lies in how to set the snapshot recalculation period appropriately. Long period setting will affect the CKNN quality and short period can bring too much communication cost.

In this paper, we focus on how to process CKNN queries over moving objects in road networks efficiently. For a moving object o in the KNN query candidate set to a query point q , an attribute *moving_state* is designed to show whether o is moving away from or getting closer to q . The value of $qo.moving_state$ can be set to *away* or *closer*, where *away* indicates that o is moving away from q , and *closer* indicates that o is getting closer to q .

The value of *moving_state* of each object near to q in a road network can be determined by considering relative moving speed and direction of the object to q . A Moving State of Object (MSO) model is proposed to examine the *moving_state* between object and query point. With the help of MSO model, an Object Candidates Processing (OCP) algorithm is proposed to efficiently distill the unqualified object candidates for CKNN query. There are two phases of OCP algorithm. One is the pruning phase, which can scale down the object candidates for the CKNN query within a given time interval. Since the CKNN query results would change during the given time interval due to the frequent updates of object locations, another refining phase is proposed to determine the time subintervals where the certain CKNN query results are obtained, which can highly reduce the distance computation cost.

The main contributions of the paper are summarized as follows:

1. A *Moving_State* of Object (MSO) model is proposed to determine the *moving_state* of object near to the query point q in a road network according to the relative moving speed and direction of the object to q .

2. A pruning algorithm is proposed to prune the objects which cannot be the CKNN query results within a given time interval with the help of MSO model.
3. A refining algorithm is proposed to determine the time subintervals where the certain CKNN query results are obtained by excluding the object candidates with the help of MSO model.

The rest of this paper is organized as follows. Section 2 reviews related work in snapshot NN query and continuous KNN query. The data structures and preliminary symbol explanations are presented in Section 3. Section 4 presents the proposed MSO model to determine the *moving_state* between object and query point. Section 5 presents the proposed OCP algorithm involving pruning phase and refining phase to obtain the CKNN query results efficiently. Section 6 evaluates the performance of our proposed methods with a set of simulation experiments in a real road network. Section 7 concludes the paper with a summary and directions for future work.

2. Related work

Processing KNN queries over moving objects in road networks is a hot research topic in recent years. In this section, we first review the existing snapshot NN methods for processing static KNN queries, and then discuss the related work that deals with continuous K nearest neighbor queries.

2.1 Snapshot NN Methods

Papadias et al. [17] proposed two methods to process static KNN queries in spatial network databases, which are Incremental Euclidean Restriction (IER) and Incremental Network Expansion (INE). The discussion about IER is omitted due to the fact that INE algorithm outperforms IER. INE expands from the query point q to search for KNNs and examines the objects in the order they are encountered. When the expansion radius is larger than the distance of the K^{th} -nearest neighbor object to the query point q , the expansion is terminated. The INE's efficiency depends on the density of objects. If the scope of the entire road network is large and the number of objects is relatively small, nearly the whole road network will be searched and the efficiency can be very low.

Kolahdouzan et al. [13] proposed the Voronoi Network Nearest Neighbor (namely VN³) method, which is based on the network Voronoi diagrams. In this method, the Network Voronoi Polygons (NVPs) and some network distances are pre-computed and the entire road network is divided into a number of small regions (namely cells). For a NN query to q , it first locates the cell it resides, and then according to the distance which is pre-calculated and stored in the adjacent table, the result of KNN query can be gained. However the performance of VN³ depends on the density and distribution of the dataset: as the dataset gets denser, the computational overheads of per-calculating NVPs become higher. As a result, VN³ is only suitable for sparse datasets. In summary, the above methods proposed for processing snapshot KNN queries in road networks are applicable to static objects. When objects move continuously, the KNN result would change over time and then the above methods are not suitable for continuous KNN queries, because of the high re-evaluation cost.

2.2 Continuous KNN Query Methods

For continuous nearest neighbor queries in a road network, Kolahdouzan et al. [12] propose a solution, called upper bound algorithm (UBA), which only performs snapshot KNN queries at the locations where they are required, and hence provides better performance by eliminating the computation of KNNs for the adjacent nodes that cannot have any split points between them. Cho et al. [4] develop a unique continuous search algorithm (UNICONS) to improve the search performance for CKNN queries. UNICONS first divides the path of the query object into subpaths by the intersections, and then the snapshot KNN queries are performed at two endpoints of each subpath. Finally, the KNNs for each subpath can be found from the union of the KNN sets at its two endpoints and the objects along it. The UBA and UNICONS methods are adequately designed to deal with CKNN queries over static objects (that is, only the query object moves continuously). When objects' locations change over time, the performance of these techniques would be significantly degraded.

Mouratidis et al. [16] proposed an Incremental Monitoring Algorithm (IMA), which can process K nearest neighbor monitoring over moving objects and query points and re-calculate query results whenever an update occurs. At each re-evaluation time, by processing the object updates, the query updates and edge updates, the query results may be incrementally obtained from the result at the previous timestamp. Thus 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. Y.-K.Huang et al. [8] proposed a continuous monitoring method over moving objects in a road network. The procedure of this monitoring method is divided into two phases, the pruning phase and the refinement phase. The main aim of 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. The inadequacy of this method is that the pruning distance is too large. Thus there are too many objects monitored and the cost for computing the distances between the query point and objects could be extremely high. Moreover, in the refinement phase, each candidate object should be examined whether it can replace the K^{th} NN or not. Thus the KNN result may be invalid due to the heavy overhead.

In this paper, we introduce a method based on the *moving_state* between object and query point to overcome the drawbacks of methods mentioned above. We aim to monitor fewer objects and reduce the distance computation cost as much as possible to efficiently monitor the KNN query results in a road network.

3. Data Structures and Preliminary Symbols

3.1 Data Structures

We assume that a road network is a graph consisting of nodes and edges, and each object moves in an edge with constant velocity. The road network is formally defined in definition 3.1.

Definition 3.1 A road network is modeled as a weighted undirected graph $G(V, E)$, in which V consists of all nodes of the network, E is a set of all edges, and $E \subseteq V \times V$.

The data structures of edge, moving object, moving situation and query are presented in the following tables respectively.

3.2 Preliminary Symbols

Considering the *moving_state*, which indicates whether the object is getting closer or moving away from q , can make CKNN query process more efficient. If an object o and the query q are in the same edge, the *moving_state* between object o and q can be determined according to the relative moving speed and direction of o to those of q . However, if o and q move on different edges, it is not so easy to examine the *moving_state* between object and query q . We define a *distance determinate* concept to handle that o and q reside in different edges, as shown below.

Definition 3.2 Suppose that there are two different edges e_i and e_j , for any two objects o_i and o_j which resides in edge e_i and e_j respectively, wherever o_i resides in e_i and wherever o_j resides in e_j , if the shortest distance from o_i to o_j passes through a unique path in the road network, then e_i and e_j are said to be *distance determinate*.

Conceptually, 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* with the following method. Suppose that e_i has two nodes $e_i.snode$ and $e_i.enode$, and e_j also has two nodes $e_j.snode$ and $e_j.enode$ as well. The distance between any pair of these four nodes can be calculated as $d(e_i.snode, e_j.snode)$, $d(e_i.snode, e_j.enode)$, $d(e_i.enode, e_j.snode)$ and $d(e_i.enode, e_j.enode)$, which are represented as d_1 , d_2 , d_3 and d_4 respectively. If and only if the following equations are satisfied:

$$d_2 = d_1 + e_j.w$$

$$d_3 = d_1 + e_i.w$$

$$d_4 = d_1 + e_i.w + e_j.w$$

Then e_i and e_j are *distance determinate*. A triangular matrix DD is constructed to record whether any pair of edges in the road network is *distance determinate* or not. For example, the value of $DD[i, j]$ is used to indicate whether e_i and e_j are *distance determinate* or not, where value 1 means *distance determinate*, and value 0 means *not*. A triangular matrix DN is used to record the shortest network distance between each pair of nodes. For example, the shortest distance between node n_i and n_j is recorded in $DN[i, j]$.

We use an example to illustrate the *distance determinate* concept, as shown in Fig. 2. Moving objects o_i are marked with black spots, the query point q is marked with a red spot, and the arrows indicate the directions of moving objects. In particular, if an object moves along the direction from the starting node to the ending node of the edge where it resides, its speed is a positive value. Otherwise its speed is a negative value. The moving velocity of object is enclosed in the round bracket followed with o_i . For each edge e_i , the length and the maximum speed limit are enclosed in the corresponding round bracket. Assume that the

query q is to process 2NN query, the information about edge and moving object according to their data structures are shown in Table 5, Table 6 and Table 7 respectively.

For the edges in Table 5, the starting nodes and ending nodes of edges are $n_1, n_2, n_3, \dots, n_7$. For each pair of these nodes, we can calculate the shortest distance according to the road network information. The calculation results are recorded in the matrix DN .

$$DN = \begin{bmatrix} 0 & & & & & & \\ 22 & 0 & & & & & \\ 36 & 14 & 0 & & & & \\ 11 & 33 & 47 & 0 & & & \\ 20 & 34 & 48 & 9 & 0 & & \\ 31 & 22 & 36 & 20 & 12 & 0 & \\ 47 & 38 & 24 & 36 & 28 & 16 & 0 \end{bmatrix}$$

For each pair of edge in Table 5, we can examine whether the pair of edge are *distance determinate* or not according to our proposed determination method. We take the edge e_1 and e_3 as an example to illustrate the determination process. For edge $e_1(n_1, n_2)$, $e_1.w=22$, and edge $e_3(n_3, n_7)$, $e_3.w=24$. The four distances between pairs of nodes n_1, n_2, n_3, n_7 are $d(n_1, n_3)$, $d(n_1, n_7)$, $d(n_2, n_3)$, and $d(n_2, n_7)$. Since

$$d(n_1, n_3) = d(n_2, n_3) + e_1.w$$

$$d(n_2, n_7) = d(n_2, n_3) + e_3.w$$

$$d(n_1, n_7) = d(n_2, n_3) + e_1.w + e_3.w$$

We conclude that e_1 and e_3 are *distance determinate*, then $DD[3, 1]=1$. The examination result for each pair of edge is recorded in the triangular matrix DD .

$$DD = \begin{bmatrix} 0 & & & & & & \\ 1 & 0 & & & & & \\ 1 & 1 & 0 & & & & \\ 1 & 1 & 1 & 0 & & & \\ 0 & 1 & 0 & 0 & 0 & & \\ 0 & 0 & 1 & 1 & 0 & 0 & \end{bmatrix}$$

For clarity, we present in Table 8 the primary symbols used in the paper, along with their interpretation.

4. Moving State of Object (MSO) Model

The determination method about *moving_state* of object regards whether the object o and query point q are on the same edge or not respectively.

4.1 Object o and Query point q on the Same Edge

As shown in Fig.3, q and o are on the same edge e which starts from n_s and ends at n_e . Assume that q is nearer to node n_s than o . When q and o move in negative direction, $|q.v|$ and $|o.v|$ denote the absolute velocity of q and o respectively.

According to the moving speeds and directions of q and o , the *moving_state* between object o and q can be set as follows:

1. *Closer_State I*

If $q.v > 0 \wedge o.v > 0 \wedge q.v > o.v$ or $q.v < 0 \wedge o.v < 0 \wedge |q.v| < |o.v|$ (Condition I)

then $qo.moving_state = closer$

2. *Closer_State II*

If $q.v > 0 \wedge o.v < 0$ (Condition II)

then $qo.moving_state = closer$

3. *Away_State I*

If $q.v > 0 \wedge o.v > 0 \wedge q.v < o.v$ and $q.v < 0 \wedge o.v < 0 \wedge |q.v| > |o.v|$ (Condition III)

then $qo.moving_state = away$

4. *Away_State II*

If $q.v < 0 \wedge o.v > 0$ (Condition IV)

then $qo.moving_state = away$

If o is nearer to node n_s than q , we can exchange o and q in the four conditions shown above. If one of the four conditions holds, then $qo.moving_state$ is set correspondingly.

For the network distance computation, if q and o are on the same edge, the network distance between q and o at time t , denoted as $D_{q,o}(t)$, can be calculated as:

$$D_{q,o}(t) = |q.dist - o.dist| + |q.v - o.v| \times (t - t_0)$$

Where $q.dist$ ($o.dist$) is the distances between q (o) and the start node of edge where q (o) resides, $q.v$ ($o.v$) is the moving speed, and t_0 is the start time.

4.2 Object o and Query point q on Two Different Edges

As shown in Fig.4, q moves on edge e_i and o moves on edge e_j . According to the *distance determinate* concept explained in section 3.2, firstly we examine whether e_i and e_j are *distance determinate* or not. If $DD[i, j] = 1$, then e_i and e_j are *distance determinate*. The shortest path connecting q and o should pass a unique path, and two endpoints of this path are denoted as $path(e_i, e_j).inode$ and $path(e_i, e_j).jnode$. The *moving_state* between object o and q can be set as follows:

1. If q is getting closer to $path(e_i, e_j).inode$ and o is getting closer to $path(e_i, e_j).jnode$

then $qo.moving_state = closer$

2. If q is getting closer to $path(e_i, e_j).inode$ and o is moving away from $path(e_i, e_j).jnode$

then if $|q.v| > |o.v|$

then $qo.moving_state = closer$

else $qo.moving_state = away$

3. If q is moving away from $path(e_i, e_j).inode$ and o is getting closer to $path(e_i, e_j).jnode$

then if $|q.v| > |o.v|$

then $qo.moving_state = away$

else $qo.moving_state = closer$

4. If q is moving away from $path(e_i, e_j).inode$ and o is moving away from $path(e_i, e_j).jnode$

then $qo.moving_state = away$

If $DD[i, j] = 0$, then e_i and e_j are not *distance determinate*. The *moving_state* between object o and q cannot be determined certainly, and then is set to be as *uncertain_away*. For the object with $qo.moving_state = uncertain_away$, we should consider it in the refining phase to check whether o is one of the KNN query results.

For the network distance computation, if q and o are on two different edges, q is moving on edge e_i which starts from n_i and ends at $n_{i'}$ and o is moving on edge e_j which starts from n_j and ends at $n_{j'}$.

If e_i and e_j are *distance determinate* (i.e. $DD[i, j] = 1$), the unique path e_i via e_j is certain, without loss of generality, assume that it is pass through nodes n_i and n_j . The network distance between q and o at time t , denoted as $D_{q,o}(t)$, can be calculated as:

$$D_{q,o}(t) = |q.dist + q.v \times (t - t_0)| + DN[i, j] + |o.dist + o.v \times (t - t_0)|$$

If e_i and e_j are not *distance determinate* (i.e. $DD[i, j] = 0$), The network distance between q and o , $D_{q,o}(t)$, consists of three parts: the distance between q and node n_i or $n_{i'}$, the shortest network distance between n_i (or $n_{i'}$) and n_j (or $n_{j'}$), and the distance between o and node n_j or $n_{j'}$. There are four different cases here: the first one considers nodes n_i and n_j , denoted as $D^1_{q,o}(t)$, the second one considers nodes n_i and $n_{j'}$, denoted as $D^2_{q,o}(t)$, the third one considers nodes $n_{i'}$ and n_j , denoted as $D^3_{q,o}(t)$, and the fourth one considers nodes $n_{i'}$ and $n_{j'}$, denoted as $D^4_{q,o}(t)$. The minimal one of these four distances is chosen as $D_{q,o}(t)$:

$$D^1_{q,o}(t) = |q.dist + q.v \times (t - t_0)| + DN[i, j] + |o.dist + o.v \times (t - t_0)|,$$

$$D^2_{q,o}(t) = |q.dist + q.v \times (t - t_0)| + DN[i, j'] + |o.e.w - o.dist + o.v \times (t - t_0)|,$$

$$D^3_{q,o}(t) = |q.e.w - q.dist + q.v \times (t - t_0)| + DN[i', j] + |o.dist + o.v \times (t - t_0)|,$$

$$D^4_{q,o}(t) = |q.e.w - q.dist + q.v \times (t - t_0)| + DN[i', j'] + |o.e.w - o.dist + o.v \times (t - t_0)|.$$

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

5. Object Candidates Processing (OCP) Algorithm

In order to make the CKNN query process over moving objects in road network more efficient, we propose an Object Candidates Processing (OCP) algorithm to scale down the object candidates with pruning phase and refining phase. With the help of MSO model, the

pruning method has two advantages compared with the method proposed by Huang *et.al* [8], as shown below:

1. Fewer computation cost for calculating the pruning distance;
2. Smaller pruning distance setting, which leads to fewer object candidates and better system performance.

The refining method is proposed to determine the time subintervals of the given time interval where the certain KNN query result can be obtained. Several observations are presented firstly in the refining phase to exclude the object candidates which cannot make KNN query results change, which can highly reduce the computation cost. In the following, the pruning phase and refining phase are described in detail.

5.1 Pruning Phase

As the objects are moving continuously in the road network within the time interval $[t_0, t_n]$, firstly 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 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 $[t_0, t_1], [t_1, t_2], \dots, [t_{i-1}, t_i], \dots, [t_{n-1}, t_n]$.

For each time subinterval $[t_{i-1}, t_i]$, when o and q moves on two different edges, if $qo.moving_state=closer$, $qo.moving_state$ remains constant. When o and q moves on the same edge, if $qo.moving_state=away$, $qo.moving_state$ remain constant, otherwise if $qo.moving_state=closer$, $qo.moving_state$ would change, as o and q will meet on the edge. In this paper, to simplify the discussion, assume that $qo.moving_state$ remains constant within each time subinterval. How to handle the change of $qo.moving_state$ will be discussed in the future work.

Huang *et.al* [8] proposes a method to prune object candidates within a time interval $[t_{i-1}, t_i]$ ($1 \leq i \leq n$), which assumes that the objects of KNN query results at time t_{i-1} all move away from q . For each o , the maximum of distance from q at time t_i plus the moving distances of q and o during $[t_{i-1}, t_i]$ is calculated as a monitoring distance. This method has two main drawbacks. The first one is that the monitoring distance is set too large which leads to high cost for distance computation in the CKNN query. The second one is that the computation cost for monitoring distance is high, since the distance between each object and q needs to be calculated.

In our paper, the proposed pruning method can overcome these drawbacks. The monitoring distance for the time interval $[t_{i-1}, t_i]$ is set as following two measures to actually improve the efficiency of CKNN query process. Firstly, the cost for monitoring distance computation is controlled strictly with the help of MSO model, where the time complexity of algorithm to

examine the *moving_state* between object and query point is $O(1)$. Secondly, the monitoring distance is set appropriately to exclude some object candidates.

For the time subinterval $[t_{i-1}, t_i]$, suppose that the KNN result of a query q at t_i is denoted as $o_1, o_2, \dots, o_K$ and these K objects are organized in the ascending order sequence with respect to their distances to q . By examining the *moving_state* between these objects and query q , we find the object with *moving_state* being *away* which ranks in the most back of the KNN result sequence according to other objects with *moving_state* being *away*, denoted as o_i . For the objects with *moving_state* being *closer* which rank before o_i in the sequence, it is impossible for them to go farther away to q than o_i . The set of these objects, denoted as OS_n , can be excluded from the monitoring distance computation process to improve the computation efficiency.

For each object o_j in $\{o_1, o_2, \dots, o_K\} - OS_n$ within $[t_{i-1}, t_i]$, according to different $qo.moving_state$, the distance $D_{q,o_j}(t_i)$ is calculated respectively.

$$(1) \text{If } qo.moving_state = closer \text{ then } D_{q,o}(t_i) = |D_{q,o}(t_{i-1}) - |q.v - o.v| \times (t_i - t_{i-1})| \quad (5.1)$$

$$(2) \text{If } qo.moving_state = away \text{ then } D_{q,o}(t_i) = D_{q,o}(t_{i-1}) + |q.v - o.v| \times (t_i - t_{i-1}) \quad (5.2)$$

After calculating the distances $D_{q,o_j}(t_i)$ within each subinterval of $[t_0, t_n]$, the maximal distance of these calculated distances is chosen as Monitoring Distance (MD), $MD = \max\{D_{q,o_j}(t_i) \mid 1 \leq j \leq K, 0 \leq i \leq n\}$.

An Additional Distance (AD) is considered to guarantee that all possible object candidates can be monitored during the CKNN query process within $[t_0, t_n]$. Taking q as a start point, if this point moves the distance MD 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 network distance between this object and query point at t_0 is larger than MD 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 MD 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 ($1 \leq x \leq m$), we can calculate the additional distance for each edge as $SL_x \times (t_n - t_0)$. The maximal one of these calculated additional distances is chosen to be AD , $AD = \max\{SL_x \times (t_n - t_0) \mid 1 \leq x \leq m\}$. The pruning distance is composed of MD and AD , $D_{pruning} = MD + AD$. Finally, whether an object o needs to be pruned or not is determined by comparing its network distance $D_{q,o}(t)$ within the time interval $[t_0, t_n]$ with the pruning distance $D_{pruning}$. If $D_{q,o}(t) \leq D_{pruning}$, the object can be a candidate for the CKNN query, otherwise, the object will be pruned.

Algorithm 1

Pruning Phase

Input: a set of moving objects O of previous KNN with $D_{o,q}(t)$, a number K , and a time interval $[t_0, t_n]$

Output: A set of objects O that are candidates for the CKNN query

/*Step1:determine the *moving_state* value of the moving objects O */

```

1.  $t_i = t_0$ ;
2. while  $t_i \leq t_n$ 
3.  $t_i = \min\{\text{the time } t \text{ when } o \in KNN \text{ and } q \text{ reach a node along their direction}\}$ ;
4. Time interval  $[t_0, t_n]$  is divided into  $[t_0, t_1], [t_1, t_2], \dots, [t_{i-1}, t_i], \dots, [t_{n-1}, t_n]$ ;
5. for each object  $o \in O$  and each time subintervals  $[t_{i-1}, t_i]$  do
6. if  $(o.e = q.e)$  //  $q$  and  $o$  are in same edge
    if  $(o.v > 0 \text{ and } q.v > 0) \text{ or } (o.v < 0 \text{ and } q.v < 0)$  //directions are same
        if  $(q.v > o.v \text{ and } |q.v| < |o.v|)$ 
             $qo.moving\_state = closer$ ; insert  $o$  to closer_object set;
        else if  $q.v < o.v \text{ and } |q.v| > |o.v|$ 
             $qo.moving\_state = away$ ; insert  $o$  to away_object set;
        end if
    end if
else //directions are different
    if  $(q.v > 0 \wedge o.v < 0)$ 
         $qo.moving\_state = closer$ ; insert  $o$  to closer_object set;
    end if
    if  $(q.v < 0 \wedge o.v > 0)$ 
         $qo.moving\_state = away$ ; insert  $o$  to away_object set;
    end if
end if
else //  $q$  and  $o$  are in two different edges
    according to DD, check whether  $q.e$  and  $o.e$  are distance determinate or not;
    if  $(DD_{q.e,o.e} == 0)$ 
         $qo.moving\_state = uncertain\_away$ ;
    else
         $o.e.spvianode = o.e.snode \text{ or } o.e.enedode$ ;
         $q.e.spvianode = q.e.snode \text{ or } q.e.enedode$ ;
        if  $(o \text{ is getting closer to } o.e.spvianode) \wedge (q \text{ is getting closer to } q.e.spvianode)$ 
             $qo.moving\_state = closer$ ; insert  $o$  to closer_object set;
        else if  $(o \text{ is getting closer to } o.e.spvianode) \wedge (q \text{ is moving away from } q.e.spvianode)$ 
            if  $(|q.v| > |o.v|)$ 
                 $qo.moving\_state = away$ ; insert  $o$  to away_object set;
            else
                 $qo.moving\_state = closer$ ; insert  $o$  to closer_object set;
            end if
        else if  $(o \text{ is moving away from } o.e.spvianode) \wedge (q \text{ is getting closer to } q.e.spvianode)$ 

```

```

    if ( $|q.v| > |o.v|$ )
         $qo.moving\_state = closer$ ; insert  $o$  to  $closer\_object$  set;
    else
         $qo.moving\_state = away$ ; insert  $o$  to  $away\_object$  set;
    end if
else
     $qo.moving\_state = away$ ; insert  $o$  to  $away\_object$  set;
end if
end if //  $DD_{q,e,o,e} = 1$  end.
7. end if // determining of  $moving\_state$  value is finished
8. end for
/*Step2: setting the pruning distance  $D_{pruning}$  */
9. find object  $o_i$  in  $away\_object$  which ranks in the back of sequence of KNN result;
10.  $monitoring\_objects = \{kNN\} - \{NN_{j < i} \in closer\}$  //  $j = 1..k$ ;
11.  $MD = 0$ ;
12. for each  $o \in monitoring\_objects$  do
13. calculate network distance  $D$  from  $o$  to  $q$  at  $t_i$  by using  $D_{q,o}(t_i)$  according to formula 4.1 or 4.2;
14. if ( $D > MD$ )  $MD = D$  end if
15. end for
16. calculate  $AD = \max\{SLx | 1 \leq x \leq m\}$ ;
17.  $D_{pruning} = MD + AD$ ;
18. /*Step3: pruning according to distance  $D_{pruning}$  */
19. according to  $D_{pruning}$ , begin to prune from  $q$  in the distance  $D_{pruning}$  range, if  $D_{q,o}(t_i) \leq D_{pruning}$  then insert  $o$  into the
    set of  $candidate\_object$ ;
20. return  $\{candidate\_object\}$ ;

```

As shown in Fig. 2, a query point q and seven objects are moving in a road network, where the length of edges, the speed and direction of all objects and q are marked. The task is to find 2NN within the time interval $[0, 4]$. At time 0, the query result is $\{o_1, o_2\}$, and their network distance to q is 8 and 12 respectively.

As the query point q moves with speed -2m/sec toward n_2 and its distance to node n_2 is 10, q takes 5 sec to reach node n_2 . Meanwhile, o_1 takes 4 sec to reach node n_6 , and o_2 takes 2 sec to reach node n_2 . Therefore, the object which firstly reaches the intersection node is o_2 , and the smallest arrival time is 2 sec (line 3, i.e. t_i mentioned in algorithm 1). By using the methods in section 4, we examine $qo_1.moving_state$ and $qo_2.moving_state$. As $o_1.e = q.e$, $o_1.v > 0$ and $q.v < 0$ (line 6), we can get that $qo_1.moving_state = away$. As $o_2.e \neq q.e$, $DD_{o_2.e, q.e} = DD[5, 2] = 1$ in matrix DD , the reference nodes of q and o_2 are n_2 , o_2 is getting closer to node n_2 , and q is also getting closer to node n_2 , we can get that $qo_2.moving_state = closer$. For the KNN query result sequence $\{o_1, o_2\}$, the object with $moving_state$ being *away* which ranks in the most back of sequence is o_1 . Then, the set of *monitoring object* is $\{o_1, o_2\}$ (line 9–10). Next, we calculate the monitoring distance MD according to formula 5.1 and 5.2. For object o_1 , $D_{q,o_1}(2) = 8 + (1+2) \times 2 = 14$, and for object o_2 , $D_{q,o_2}(2) = |12 - (1+2) \times 2| = 6$. Thus, we choose the maximal distance 14 as the monitoring

distance MD (line 12–15). After that, the additional distance AD is calculated. As shown in Fig.5, the boundary points P_{bound} , marked with short vertical bars, are on edges e_1 , e_2 , and e_5 . Then $AD = \max\{2 \times 2, 1 \times 2, 2 \times 2\} = 4$, so $D_{pruning} = 14 + 4 = 18$ (line 17). The pruning distance ranges of query q are marked with black thick cross lines. Since there are only three moving objects o_1 , o_2 and o_3 within the pruning range, the object candidate set $O_{candidate}$ is $\{o_1, o_2, o_3\}$ (line 19). The other objects outside the pruning range are excluded, as they cannot be 2NN query results within the time interval $[0, 2]$.

5.2 Refining Phase

With the pruning phase, we can prune the objects which cannot be the KNN query results, and obtain a object candidate set $O_{candidate} = \{o_1, o_2 \dots o_i \dots o_m\}$ ($m \leq K$) within the given time interval $[t_0, t_n]$. As the objects are moving continuously, the KNN query results may change during the given time interval. In order to reduce the repetitive query cost, a refining algorithm is proposed to determine the time subinterval where the certain KNN query results are obtained by excluding the objects which cannot be the KNN query results. Firstly, we should determine the time instants at which the KNN query results change. Referring to the time subinterval division in the pruning phase, the given time interval $[t_0, t_n]$ is divided into $[t_0, t_1], [t_1, t_2], \dots, [t_{j-1}, t_j], \dots, [t_{n-1}, t_n]$. For each subinterval, as the direction of moving object is certain, the linear equation of distance with time can be determined. For example, the 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 find the KNN query result changing time instants. Note that the first object in the KNN query result which may be replaced by other object is certainly the K^{th} NN, denoted as NN_K , where the distance of NN_K to q can be represented as $D_{q,NN_K}(t)$. For each object o in $O_{candidate}$, we can obtain the changing time instant by solving the equation $D_{q,o}(t) = D_{q,NN_K}(t)$ ($t_{j-1} \leq t \leq t_j$). The smallest one of these calculated time instants is chosen as the changing time instant, denoted as tc_1 , then the time interval $[t_{j-1}, t_j]$ is divided into $[t_{j-1}, tc_1]$ and $[tc_1, t_j]$. For time subinterval $[tc_1, t_j]$, the finding process is repeated until the found changing time instant is larger than t_j . Therefore, the time interval $[t_{j-1}, t_j]$ is divided into $[t_{j-1}, tc_1], [tc_1, tc_2], \dots, [tc_m, t_j]$. For each division time subinterval and each object of object candidates, if the network distance between the object and the query point is larger than the distance between NN_K to q , this object will be excluded.

The computation cost for finding KNN query result change time instants will increase significantly as the increasing number of object candidates. In order to strictly control the computation cost, we firstly exclude the object candidates which do not need to be considered in the finding process by the help of MSO model. For an object o' in $O_{candidate}$ within $[t_{j-1}, t_j]$, if the distance between o' and q is larger than that of object NN_K to q at t_{j-1} , only when the condition in one of the following observations is met, it is possible for o' to make the KNN query results change.

Observation 4.1 If the object o is getting closer to query q , and the object o' is getting closer to q , only if the relative moving speed of o' to q is faster than that of object o to q , the object o' could change the KNN query results by replacing o .

Observation 4.2 If the object o is moving away from query q , and the object o' is getting closer to q , the object o' could change the KNN query results by replacing o .

Observation 4.3 If the moving object o is moving away from query q , and the object o' is moving away from q , only if the relative speed of o to q is faster than that of object o' and q , the object o' could change the KNN query results by replacing o .

Proof of Observation 4.1:

Assume, on the contrary, that if the relative moving speed of o' to q is slower than that of object o to q , the KNN query results may change within $[t_{j-1}, t_j]$, namely, $|o'.v| < |o.v|$.

1. The distance from o to q is smaller than the distance from o' to q at t_{j-1} ,
 $D_{q,o}(t_{j-1}) < D_{q,o'}(t_{j-1})$.
2. As qo . *moving_state* = *closer*, we can get that $D_{q,o}(t_j) = |D_{q,o}(t_{j-1}) - |q.v - o.v|| \times (t_j - t_{j-1})$ (formula 5.1).
3. As qo' . *moving_state* = *closer*, we can get that $D_{q,o'}(t_j) = |D_{q,o'}(t_{j-1}) - |q.v - o'.v|| \times (t_j - t_{j-1})$ (formula 5.1).
4. As $D_{q,o}(t_{j-1}) < D_{q,o'}(t_{j-1})$, and $|q.v - o.v| > |q.v - o'.v|$, from (2)(3), we can definitely get that $D_{q,o}(t_j) < D_{q,o'}(t_j)$.

From the result of (4), we can conclude that the object o' cannot replace the object o to make the KNN query results change within $[t_{j-1}, t_j]$, which is contradiction with the assumption. Therefore, observation 4.1 is proved.

The proof of observation 4.2 and 4.3 can be proved in a similar way.

Algorithm 2

Refining Phase

Input: a set of moving objects O of previous KNN, $O_{candidate} = \{candidate_objects\}$ obtained from the pruning phase, and the time subinterval $[t_0, t_1], [t_1, t_2], \dots, [t_{j-1}, t_j], \dots, [t_{n-1}, t_n]$

Output: A set of tuple in form of $\langle \{o_1, o_2, \dots, o_k\}, [t_{j-1}, tc_1] \rangle, \langle \{o_1, o_2, \dots, o_k\}, [tc_1, tc_2] \rangle, \dots, \langle \{o_1, o_2, \dots, o_k\}, [tc_m, t_j] \rangle$, where $o_1, o_2, \dots, o_k$ are the KNN query results of q within each time subinterval

/*Step1: determining the result-changing time points */

1. set $i=1$;
2. For each subinterval $[t_{j-1}, t_j]$
3. set $tc_i = t_{j-1}$;
4. For each time instant $tc_i \ t_j$
5. for each object o in $O_{candidate}$
6. if $((NN_K$ in q .*closer_objects*) and $(qo$.*moving_state*==*closer* and $|o.v| > |NN_K.v|)$) or $((NN_K$ in q .*away_objects*) and $(qo$.*moving_state*==*closer* or $(qo$.*moving_state*==*away* and $|o.v| < |NN_K.v|))$
7. solve the equation $D_{q,o}(t) = D_{q,NN_K}(t)$ and get t ;
8. if $(tc_i \ t \ t_j)$ then
9. $t_i = t$;
10. insert the tuple $\langle \{o\}, t_i \rangle$ into Q_0 ;

```

11.     end if
12.     end if
13. end for
14. sort the result-changing time points in  $Q_0$  in ascending order;
15. set  $tc_i = \min\{t_i\}$ ;
16. if ( $o \notin \{NN_1, NN_2, \dots, NN_K\}$ )
17.    $O_{candidate} = O_{candidate} - NN_K + \{o\}$ ;
18.    $O_{KNN} = \{NN_1, NN_2, \dots, NN_{K-1}\} + \{o\}$ ;
19. else
20.    $O_{KNN} = \{NN_1, NN_2, \dots, NN_K\}$ ;
21. end if
22. insert the tuple  $\langle O_{KNN}, tc_i \rangle$  into  $Q$ ;
23. end for // time instant  $tc_i > t_p$ , finding all time instants
24. sort the result-changing time points in  $Q$  in ascending order;
25. divide the time interval  $[t_{j-1}, t_j]$  into  $n$  disjoint subintervals  $[tc_1, tc_2], [tc_2, tc_3], \dots, [tc_{n-1}, tc_n]$ 
26.  $t_0 = t_{j-1}$ ; /*Step2: determining the corresponding KNN result at the result-changing time instants */
27. while ( $Q$  is not empty) do
28.   de-queue  $\langle O_{KNN}, tc_i \rangle$ ;
29.   sort the distance between the object which is in  $O_{KNN}$  and query  $q$  and get a new  $O_{KNN}$ ;
30.   return the tuple  $\langle \{O_{KNN}\}, [t_0, tc_i] \rangle$ ; set  $t_0 = tc_i$ ;
31. end do
32. end for

```

We still use the example in Fig.2 to illustrate the refining phase. The task is to find 2NN within $[0, 4]$. By using the pruning algorithm, the time interval $[0, 4]$ is firstly divided into $[0, 2]$ and $[2, 4]$ where the moving direction of objects are certain. From the pruning phase in Fig.5, we can get that the object candidates are $\{o_1, o_2, o_3\}$ within the time interval $[0, 2]$. For the time interval $[0, 2]$, we find the 2NN query result changing time instants firstly. At time 0, the 2NN query result is $\{o_1, o_2\}$. According to $qo_3.moving_state = closer$ and $qo_2.moving_state = closer$, $|o_3.v| > |o_2.v|$, we know that it is possible for o_3 to replace o_2 by using the observation 4.1. By solving the equation $24 - 4t = 12 - 3t$, we can get that $t = 12$, which is larger than 2. Therefore, o_3 cannot replace o_2 within $[0, 2]$. According to $qo_1.moving_state = away$ and $qo_2.moving_state = closer$, we know that o_1 could replace o_2 by using the observation 4.2. By solving the equation $8 + 3t = 12 - 3t$, we can get that $t = 0.67$ (line 6–12). So the time interval $[0, 2]$ is divided into the subintervals $[0, 0.67]$ and $[0.67, 2]$. Meanwhile, tuple $\langle \{o_1, o_2\}, [0, 0.67] \rangle$ and tuple $\langle \{o_2, o_1\}, [0.67, 2] \rangle$ are obtained (line 26–30).

For the time interval $[2, 4]$, we firstly obtain the object candidates by using the pruning algorithm. At time 2, the 2NN query result is $\{o_2, o_1\}$. According to $qo_1.moving_state = away$ and $qo_2.moving_state = closer$, the object with *moving_state* being *away* which ranks in the back of query result sequence is o_1 . Then the set of *monitoring object* is $\{o_1\}$, and o_2 is excluded in the monitoring distance computation process. We can calculate the monitoring distance $MD = 20$ and $AD = 4$ respectively. Thus, the pruning distance

$D_{pruning}=MD+AD=24$. With this pruning distance, the object candidate set $\{o_2, o_1, o_3\}$ is obtained. According to $qo_2.moving_state=closer$ and $qo_1.moving_state=away$, o_2 can not make the KNN result change. According to $qo_3.moving_state=closer$ and $qo_1.moving_state=away$, with the observation 4.2, we can resolve the equation $14+3t=16-4t$ to get the time instant $t=0.29s$ (the actual time is 2.29s). Then, the time interval $[2, 4]$ is divided into subintervals $[2, 2.29]$ and $[2.29, 4]$ (line6–12). Finally, for each time subinterval within $[0, 4]$, we can obtain the KNN query results, represented as tuple $\langle \{o_1, o_2\}, [0, 0.67] \rangle$, tuple $\langle \{o_2, o_1\}, [0.67, 2] \rangle$, tuple $\langle \{o_2, o_3\}, [2, 2.29] \rangle$ and tuple $\langle \{o_2, o_3\}, [2.29, 4] \rangle$ (line 26–30).

6. Performance Evaluation

In this section, we evaluate the performance of our proposed method by experiments. Our experiments are implemented in visual C++, running on a PC with 1.86GHZ Pentium dual processor and 1GB memory. For our experiments, we consider a synthetic data set consisting of 1,000 moving objects whose start locations are uniformly spread over a road network of 150 nodes and 200 segments and an Oldenburg data set consisting of 7,035 nodes [21], as shown in Fig.6. In order to demonstrate the scalability of the proposed OCP algorithm, the experiments are also conducted on a larger real road network, which is the network of San Francisco Bay Area (BAY) in U. S. (<http://www.dis.uniroma1.it/~challenge9/download.shtml>), which consists of 321,270 nodes. The moving objects are generated using the generator proposed in [3], and each object starts at a randomly selected position in the range of interest. Subsequently, the object picks a random direction and moves at a speed randomly distributed between 0 mph and 70 mph. When an object o reaches node n along its moving direction in a road network, the next edge on which o moves is randomly chosen from the edges connecting n . In all the following figures, the running time (seconds) or precision is shown in a logarithmic scale.

Assume that there are 50 query objects in our system. The speeds of query objects are the same as that of moving objects mentioned above. Similarly, the next edge that the query object moves on is randomly selected once it reaches a network node. The default values of the number of nearest neighbors requested by each query object (i.e., K) is 10 and the length of query time interval is 100 time units. In our experiments, we compare the OCP algorithm with CKNN algorithm proposed by Y.K. Huang *et al.* [8] and the IMA algorithm proposed by Mouratidis *et al.* [16]. The IMA algorithm re-evaluates the snapshot KNN query when objects' location updates occur, whose update interval (UI) is set to 10 time units in this experiment.

At first, we evaluate the effect of query time interval on the CPU and the precision of OCP, CKNN and IMA algorithm. Here the precision refers to the percentage of time units at which the KNN result retrieved by executing the OCP, the CKNN and the IMA algorithms is correct. Fig.7 (a) (c) (e) shows that for these three algorithms in Synthetic, Oldenburg and BAY data set, the CPU cost increases with the increase of query interval length. For OCP and CKNN, because a larger query time length results in more objects reaching the network nodes, thus more subintervals need to be considered. For IMA, this is due to the fact that a larger query time length makes more location updates of objects so that more snapshot KNN

queries are required. Clearly, the OCP algorithm has a significantly better performance compared to the IMA algorithm and its performance is also better than that of CKNN, because there are fewer computations in OCP. Fig.7 (b) (d) (f) investigates the effect of the query time length on the precision of all algorithms in Synthetic, Oldenburg and BAY data set. As the KNNs at each timestamp can be completely determined by executing the OCP algorithm, the precision of the OCP algorithm is always equal to 100% under different query time lengths. However, if the IMA algorithm is adopted to answer a CKNN query, some of the KNN results are unknown due to the nature of discrete location updates. As we can see, the precision of IMA is low percentage for a larger query time length, which means that most KNN results are incorrect.

Fig. 8 illustrates the performance of the OCP, the CKNN and the IMA algorithms for Synthetic, Oldenburg and BAY data set as a function of K (ranging from 10 to 30). As shown in Fig. 8(a) (c) (e), when K increases, the CPU overhead for these three algorithms grows. The reason is that as K becomes greater, the number of qualifying objects increases so that more distance comparisons between these qualifying objects are required. However, OCP algorithm outperforms its competitors in all cases. Fig.8 (b) (d) (f) studies how the value of K affects the precision of the OCP, the CKNN and the IMA algorithms. Similar to Fig. 7(b) (d) (f), the precision of the OCP algorithm maintains 100% regardless of the value of K . However, the precisions of IMA are low percentage under various K .

In Fig.9 we can see the effect of the number of query points and K on the performance of three algorithms in Synthetic, Oldenburg and BAY data set. Observe in Fig. 9(a) (c) (e) that the running costs of three algorithms tend to higher as the number of query points increases. For OCP algorithm, the effect increases less due to the state between moving objects and query point and the stricter pruning condition. On the other hand, in the Fig. 9(b) (d) (f) the precision of the OCP algorithm is smoother and reaches above 95% while the number of query points rise, the reason is that as the number of query points increases, the moving objects processed in OCP algorithm increase slowly by their *moving_State* value. The experimental results demonstrate that OCP algorithm is superior to CKNN and it is optimized for IMA queries and the mechanism of *moving_State* value is reliable and helpful for monitoring CKNN queries.

7. Conclusions

In order to improve the efficiency of CKNN query over moving objects in road networks, an efficient Object Candidate Processing (OCP) algorithm is proposed to scale down the object candidates for CKNN query, meanwhile determine the time subintervals where the certain CKNN query results are obtained by using pruning phase and refining phase. In order to strictly control the computation cost of OCP algorithm, a Moving State of Object (MSO) model is designed to exclude the objects which do not need to be considered in the OCP algorithm. Comprehensive experiments demonstrate that our proposed method is feasible and efficient to improve the efficiency of CKNN query process.

One important future extension of this work is to find the update method of efficiently handling the speed changes of moving objects. Another extension is to utilize our proposed

approach to answer the issue of other variations of KNN query on moving objects with variable speed in road networks.

Acknowledgments

This work was supported by National Natural Science Fund of China under grants 60873030, 61173049 and 61100059, Doctoral Fund of Ministry of Education of China under grant 20090142110023, and National Natural Science Fund of Hubei province under grant 2010CDB02301.

References

1. Benetis, R.; Jensen, CS.; Karciauskas, G.; Saltenis, S. Nearest neighbor and reverse nearest neighbor queries for moving objects. Proceedings of the International Database Engineering and Applications Symposium; July 17–19, 2002; Canada.
2. Benetis R, Jensen CS, Karciauskas G, Saltenis S. Nearest neighbor and reverse nearest neighbor queries for moving objects. VLDB Journal. 2006; Vol. 15(3):229–249.
3. Brinkhoff T. A framework for generating network-based moving objects. GeoInformatica. 2002; 6(2):153–180.
4. Cho, H-J.; Chung, C-W. An efficient and scalable approach to cnn queries in a road network. Proceedings of the International Conference on Very Large Databases; 2005; Trondheim, Norway.
5. de Almedia, VT. towards optimal continuous nearest neighbor queries in spatial databases. Proceedings of ACM GIS; November 10–11, 2006;
6. Guan, Xin; Guan, L.; Wang, XG.; Ohtsuki, Tomoaki. A new load balancing and data collection algorithm for energy saving in wireless sensor networks. Telecommunication Systems. 2011; Vol 45(4):313–322.
7. Guerrero-Zapata, Manel; Zilan, Ruken; Barceló-Ordinas, José M.; Bicakci, Kemal; Tavli, Bulent. The future of security in Wireless Multimedia Sensor Networks. Telecommunication Systems. 2011; Vol 45(1):77–91.
8. Huang YK, Chen ZW, Lee C. Continuous K-Nearest Neighbor Query over Moving Objects in Road Network. APWeb-WAIM 2009, LNCS. 2009; 5446:27–38.
9. Hu, H.; Xu, J.; Lee, D. A Generic Framework for Monitoring Continuous Spatial Queries over Moving Objects. Proceedings of the SIGMOD Conference; 2005; Paris, France. p. 479–490.
10. Jensen, CS.; Kolar, J.; Pedersen, TB.; Timko, I. nearest neighbor queries in road networks. Proceedings of the ACM GIS; November 7–8, 2003; New Orleans, Louisiana, USA.
11. Kolahdouzan MR, Shahabi C. Alternative solutions for continuous K nearest neighbor in spatial network databases. GeoInformatica. 2004 Jun; Vol. 9(4):321–341.
12. Kolahdouzan, MR.; Shahabi, C. Continuous k nearest neighbor queries in spatial network databases. Proceedings of the Spatio-Temporal Databases Management (STDBM); August 30, 2004; Toronto, Canada.
13. Kolahdouzan MR, Shahabi C. Voronoi-based K nearest neighbor search for spatial network databases. VLDB. 2004:840–851.
14. Lee, Cheng-Chi; Liao, I-En; Hwang, Min-Shiang. An efficient authentication protocol for mobile communications. Telecommunication Systems. 2011; Vol. 46(1):31–41.
15. Mouratidis K, Hadjieleftheriou M, Papadias D. Conceptual partitioning: An efficient method for continuous nearest neighbor monitoring. SIGMOD. 2005
16. Mouratidis, K.; Yiu, ML.; Papadias, D.; Mamoulis, N. Continuous nearest neighbor monitoring in road networks. Proceedings of the International Conference on Very Large Data Bases; September 12–15, 2006; Seoul, Korea.
17. Papadias D, Zhang J, Mamoulis N, Tao Y. Query processing in spatial network databases. VLDB. 2003
18. Raptopoulou K, Papadopoulos AN, Manolopoulos Y. Fast nearest-neighbor query processing in moving-object databases. GeoInformatica. 2003 Jun; Vol. 7(2):113–137.

19. Tao, Y.; Papadias, D. Time parameterized queries in spatio-temporal databases. Proceedings of the ACM SIGMOD; 2002; Madison, Wisconsin.
20. Tao, Y.; Papadias, D.; Shen, Q. Continuous nearest neighbor search. International Conference on Very Large Data Bases; August 20–23, 2002; Hong Kong, China.
21. US Bureau of the Census. Technical Documentation. TIGER/Line Files. 1995
22. Xiong, X.; Mokbel, MF.; Aref, WG. SEA-CNN: Scalable Processing of Continuous K-Nearest Neighbor Queries in Spatio-temporal Databases. Proceedings of the 21st International Conference on Data Engineering (ICDE); 2005; Tokyo, Japan. p. 643-654.
23. Yu, X.; Pu, KQ.; Koudas, N. Monitoring k-Nearest Neighbor Queries over Moving Objects. Proceedings of the 21st International Conference on Data Engineering (ICDE); 2005; Tokyo, Japan. p. 631-642.
24. Zhang, J.; Zhu, M.; Papadias, D.; Tao, Y.; Lee, DL. Location-based spatial queries. Proceedings of the SIGMOD Conference; 2003; San Diego, CA. p. 443-454.

Biographies

Guohui Li was born in 1973. He got his PhD degree from Huazhong University of Science and Technology (HUST) in 1999. Now he is a full professor in HUST. His research interests mainly include advanced data engineering, mobile computing and real-time computing.

Ping Fan was born in 1974. He was PhD candidate in Huazhong University of Science and Technology (HUST). Now he is an associate professor in Xianning University. His current research interests include spatial databases, mobile computing.

Yuan Ling was born in 1975. She got her PhD degree from National University of Singapore. Now she is an assistant professor in Huazhong University of Science and Technology. Her research interests focus on database system implementation techniques, spatial databases.

Fig.1.
The example of a CKNN query

Fig.2.
Graph of a road network at time 0

Fig.3.
Example of q and o on the same edge

Closer_State I

Away_State I

Closer_State II

Away_State II

Fig.4.
Example of q and o move on two different edges

Fig.5.
Graph of a road network at time 2

Fig.6.
Road network of Oldenburg city

(a) Synthetic data set

(b) Synthetic data set

(c) Oldenburg data set

(d) Oldenburg data set

(e) BAY data set

(f) BAY data set

Fig.7.

The performance and precision on query time interval

(a) CPU time for Synthetic data set

(b) Precision for Synthetic data set

(c) CPU time for Oldenburg data set

(d) Precision for Oldenburg data set

(e) CPU time for BAY data set

(f) Precision for BAY data set

Fig.8.
The performance and precision on K

(a) CPU time for Synthetic data set

(b) Precision for Synthetic data set

(c) CPU time for Oldenburg data set

(d) Precision for Oldenburg data set

(e) CPU time for BAY data set

(f) Precision for BAY data set

Fig.9.

The performance and precision on the number of query points and K

Table 1Data structure of an edge e

Edge attributes	Descriptions
<i>.id</i>	The identity of edge
<i>.w</i>	The length of an edge
<i>.includeobj</i>	The set of objects on an edge
<i>.snode</i>	The starting node of an edge
<i>.enode</i>	The ending node of an edge
<i>.maxv</i>	The max speed limit of an edge

Table 2Data structure of an object o

Object attributes	Descriptions
$.id$	The identity of object
$.e$	The edge on which the object o moves
$.dist$	The distance between o and the starting node of the edge where it resides
$.v$	The moving speed of object o

Table 3

Data structure of moving situation

State attributes	Descriptions
$.q_i$	The identity of query object
$.o_i$	The identity of moving object
$.moving_state$	whether o_i is moving away from or getting closer to a query point q_i

Table 4Data structure of a query q

Query attributes	Descriptions
<i>.result</i>	The result set of <i>KNN</i> query
<i>.closer_objects</i>	The set of objects whose <i>moving_state</i> values are <i>closer</i>
<i>.away_objects</i>	The set of objects whose <i>moving_state</i> values are <i>away</i>
<i>.uptime</i>	The update time when <i>KNN</i> results change

Table 5

Information of edge

id	w	includeobj	snode	enode	maxv
e_1	22	$\{o_3\}$	n_1	n_2	2
e_2	14	$\{o_2\}$	n_2	n_3	1
e_3	24	$\{o_5\}$	n_3	n_7	2
e_4	11	$\{o_4\}$	n_1	n_4	1
e_5	22	$\{o_1,q\}$	n_2	n_6	2
e_6	20	$\{o_6\}$	n_4	n_6	2
...	...	...	...	...	...

Table 6

Information of moving object

id	e	Dist	v
o_1	e_5	18	1
o_2	e_2	2	-1
o_3	e_1	8	2

Table 7

Information of query

q_i	o_i	moving_state
q_1	o_1	away
q_1	o_2	closer

Table 8

Primary symbols

Symbol	Description
V	The set of all nodes
E	The set of all edges
$DN[i, j]$	The shortest network distance between n_i and n_j
$closer_object$	The object which is moving away from q
$away_object$	The object which is getting closer to q
$closer$	One value of $moving_state$ indicates o is getting closer to q
$away$	One value of $moving_state$ indicates which o is moving away from q
$uncertain_away$	One value of $moving_state$ indicates whether o is getting closer or moving away from q cannot be determined
$D_{pruning}$	The pruning network distance
$D_{q,o}(t)$	The network distance between q and o at time instant t