Received October 15, 2018, accepted November 8, 2018, date of publication November 14, 2018,
date of current version December 19, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2881414

ExTCKNN: Expanding Tree-Based Continuous K Nearest Neighbor Query in Road Networks With Traffic Rules

HONGJUN LI[1], (Member, IEEE), BIAO CAI[1], SHAOJIE QIAO[2], QING WANG[3], AND YAN WANG[4]
[1]School of Information Science and Technology, Chengdu University of Technology, Chengdu 610059, China
[2]School of Cybersecurity, Chengdu University of Information Technology, Chengdu 610225, China
[3]School of Computer Sciences, Florida International University, Miami, FL 33199, USA
[4]School of Software, East China Jiaotong University, Nanchang 330013, China

Corresponding authors: Biao Cai (caibiao@cdut.edu.cn) and Shaojie Qiao (sjqiao@cuit.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61701049, Grant 61802035, and Grant 61772091, in part by the Sichuan Provincial Education Department Project under Grant 15ZB0070, in part by the Digital Media Science Innovation Team of CDUT under Grant 10912-kytd201510, in part by the Sichuan Science and Technology Program under Grant 2018JY0448, in part by the Innovative Research Team Construction Plan in the Universities of Sichuan Province under Grant 18TD0027, and in part by the National Natural Science Foundation of Guangxi under Grant 2018GXNSFDA138005.

ABSTRACT The existing continuous nearest neighbor query algorithms of moving objects in road networks do not consider any traffic rule and assume that the speed of moving objects is constant and the topology of road networks never change. However, in real road networks, the object's speed and the road network's structure change frequently Hence, these would make the existing methods ineffective when applying to the real-world road network environment To overcome the aforementioned disadvantages, we propose a Data Modeling approach of Road Networks with traffic rules (called DMRN^R) and design a novel Expanding Tree-based Continuous k Nearest Neighbors algorithm (abbreviate for ExTCKNN) that can be well adopted to the actual road network environment. The algorithm consists of three steps: 1) it obtains the query results to store using DMRN^R in the initial phase; 2) it maintains the data model of road networks by monitoring the real-time change information; and 3) the results are generated according to the submitted query with the updated data model and the latest state of moving objects The merit of the proposed algorithm lies in that it queries the nearest neighbors by taking the movements of the moving object and the variety of the road networks into consideration Extensive experiments are conducted and the experimental results demonstrate a significant improvement of the proposed method when compared with conventional solutions.

INDEX TERMS Road networks, k nearest neighbor, continuous query, expanding tree data model of road network

I. INTRODUCTION

With the rapid development of mobile computing, a large number of applications based on moving objects have emerged, such as spatial databases, geographic information, intelligent transportation system [1], [30] and etc. These applications require highly efficient spatio-temporal query [1]–[5], [15], [16], [24], [25]. Among them, k-Nearest Neighbor (*kNN*) query is a basic problem in the research field of moving object database, and it has received extensive attention from researchers [5], [8]. For example, the users in the intelligent transportation system want to get the query results: (1) The taxi driver wants to get the 5-nearest passengers to take a taxi. (2) The marching driver needs the 3-nearest ambulances to get help.

Previous literature of *kNN* query focuses on the static data. As the relevant applications proliferate, many extended studies have emerged. **C**ontinuous ***k*** **N**earest **N**eighbor query (***CkNN***) [9]–[14], [19], [20], [30], [31] is one of the most important extensions. It refers to the query that can continuously return the latest k nearest neighbors of the given query object in real time.

Some algorithms have been proposed for *CkNN* query in road networks [1], [6]–[9], [17], but these methods are designed to deal with continuous nearest neighbor queries of static objects, that is, the query object remains stationary.

Mouratidis *et al.* [12] propose the Incremental Monitoring Algorithm (IMA) to process *CkNN* query in mobile environments with moving states; Huang *et al.* [13] propose a Continuous *k*NN algorithm, which can determine the kNN set of the query object at each time point. Fan *et al.* [22] propose an Object Candidate Processing (OCP) algorithm based on the moving state of the object model. Shen *et al.* [30] design a novel kNN search algorithm using V-Tree which can support dynamical updates of moving objects by pruning large numbers of irrelevant vertices in the road network. Bok *et al.* [31] propose a method to process the continuous k-NN query in the grid environments efficiently.

Although some works focus on the C*k*NN in the road networks, it is hard to apply these approaches in the real-world transportation networks due to the following three reasons.

(1) The traffic rules of road networks will greatly affect the distance between objects, so it is very important to consider traffic rules for *k*NN query in road network. However, unfortunately, the existing approaches do not take the traffic rules into consideration, so they cannot obtain correct query results in the real-world road networks. Consider the following example.

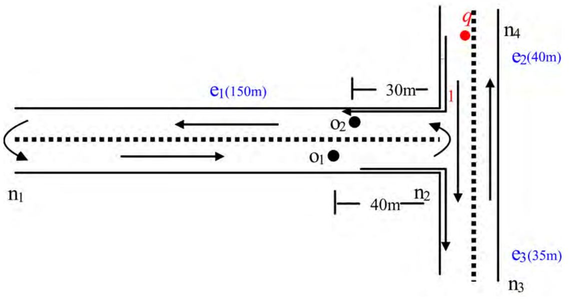


FIGURE 1. **Road networks with traffic rules.**

Example 1: As shown in Fig. 1, moving objects o_i are marked by black spots, the query point q is marked by a red spot, and the arrows indicate the road directions. For each road e_i, the length is enclosed in the corresponding round bracket.

According to the previous algorithms, for example, IMA algorithm [12], the network distance is 10m from o_1 to o_2. However, the network distance should be 70m according to the traffic rules.

The proposed algorithm in this paper takes traffic rules into account, including the directions of moving objects at road intersections, the speed limitations of the roads, road rules (i.e., one-way or two-way) and so on.

(2) The current approaches can only deal with the situation when the moving objects keep unmoved or move at a uniform speed, and the network topology of the transportation networks does not change as well. Unfortunately, in the real-world road networks, the rate of the moving objects usually varies continuously, the topology of the road network changes dynamically due to traffic management and other factors. For example, the traffic management departments will change some two-way to one-way roads during rush hours.

(3) The natural discreteness of position updating makes the kNN of query object unknown between two consecutive update timestamps. It will make the existing algorithms unable to return valid results between two consecutive update timestamps [32].

To overcome the limitations of the aforementioned methods, we design a new **D**ata **M**odel of **R**oad **N**etwork with traffic **R**ules (DMRN^R) for the real-world mobile environment. By utilizing DMRN^R model, an Expanding Tree-based Continuous *k* Nearest Neighbor monitoring algorithm (ExTCKNN) in the road network is proposed. A comparison between the ExTCKNN and existing studies is shown in Table 1.

TABLE 1. **Comparison with existing CkNN algorithms.**

Studies	Space	Traffic Rule	Dynamical change of objects and network	Discreteness of position updating
[12]	Network	No	Yes	No
[13]	Network	No	No	No
[22]	Network	No	Yes	No
[31]	Euclidean	No	No	No
ExTC KNN	Network	Yes	Yes	Yes

There are three essential phases in the proposed ExTCKNN algorithm. First, ExTCKNN gets the initial nearest neighbor set of the query object by expanding from the query object in the initialization phase. Second, the monitoring phase is proposed to maintain the expanding tree by updating the corresponding moving objects, query objects, and road. Finally, when a new query is submitted, the algorithm returns the latest *k*NN objects of the query object from the expanding tree in the query phase.

The main contributions of the paper are concluded as follows:

(1) A new Data Model of Road Network with traffic Rules (DMRN^R) is proposed to store road network information and query results. The rules of the road network and the updated status of moving objects are considered in the model, which achieves better performance in practice.

(2) A continuous *k* nearest neighbor monitoring algorithm based on DMRN^R model is proposed. It can process the changes of mobile objects and road networks efficiently.

(3) The proposed algorithm is evaluated by an extensive simulation. We also compare our results with those using the existing solutions.

The rest of this paper is organized as follows. Section II reviews related work in continuous *k*NN query. The descriptions of the data model and preliminary symbols are presented in Section III. Section IV shows the initial phase of our proposed ExTCKNN algorithm. Section V introduces the monitoring phase of the ExTCKNN algorithm. Section VI evaluates the performance of our proposed methods with a set of simulation experiments in a real road network. Section VII concludes the paper with a summary and directions for future work.

II. RELATED WORK

Continuous kNN queries over moving objects in road networks have been widely studied in the last few years [23], [26], [27]. Chen [18] models the road network by a Probabilistic Time-dependent Graph (PT-Graph), whose roads are associated with uncertain delay functions. In addition, a query algorithm in the PT-Graph, called a Trip Planner Query (TPQ), is proposed. The query retrieves trip plans that traverse a set of query points in PT-Graph, having the minimum traveling time with high confidence. Zhao *et al.* [11] propose an algorithm, called Voronoi CKNN (VCKNN), by using the Voronoi diagram for CKNN. There is no need to segment the query path in this approach. Hence, it improves the overall performance. Liao *et al.* [29] tackle two problems in the query by using overlay graph-based indices: (1) redundant calculations are produced because of the uncertainty of the objects' locations during the query process; (2) the distribution of the moving objects has been neglected. Two novel algorithms, Object-Last (OL) and Guide-Forest (GF), are proposed to boost the performance of kNN queries by revisiting the structure of the overlay graphs. The OL algorithm reduces half of the searching space through optimizing the overlay graph structure based on the objects' distribution. The GF algorithm adds the guidance information into the overlay graph structure and searches on a heterogeneous graph composed of both the original graph and the overlay graph.

Many works focus on the continuous nearest neighbor query of moving objects when the query object keeps moving while the moving objects remain stationary. Delling *et al.* [17] propose partition-based shortest-path algorithms to answer kNN queries such as which one is the closest restaurant or the best post office to stop on the way home. They also provide various trade-offs between indexing effort and query time. The most flexible variant of this algorithm allows the road network to change frequently. Cho *et al.* [9] propose a unique continuous searching algorithm (UNICONS) to improve the performance of CkNN queries. The algorithm consists of the following steps: (1) the path of the query object is divided into sub-paths according to the intersection of the road network; (2) the kNN set of two endpoints of each sub-path is calculated; (3) finally, kNN set of the query object is obtained from the union of the kNN sets of two endpoints on each sub-path. These algorithms assume that only the query objects move continuously in the road network. When all objects change their location over time, the performance will decrease significantly.

Some works focus on studying the CkNN query when the query objects and moving objects keep moving. Huang *et al.* [13] propose a CkNN algorithm, to present the distance between the query object and the moving object as an equation. It divides the time interval into subintervals which determine the changes of the kNN set of the query object. Fan *et al.* [22] propose an Object Candidate Processing (OCP) algorithm based on the moving state of the object model. In the pruning phase, the algorithm prunes the objects which cannot be the kNN query result within a given time interval. In the refining phase, the subintervals of the given time interval are determined where the certain kNN query results are obtained. Shen *et al.* [30] propose a new index, V-Tree, which has two salient features. V-Tree is a balanced search tree and can support efficient kNN search. Additionally, it can support dynamical updates of moving objects. They also design a novel kNN search algorithm using the borders to efficiently compute k nearest objects, which can deal with kNN search on a large number of online queries in road networks. Kyoungsoo *et al.* [31] propose a new continuous k-NN query processing method called Pattern Based k-NN (PB-kNN) extending the basic kNN processing concept proposed in [19] for efficiently sharing and transmitting multimedia data in LBS. The proposed method utilizes the Distance Relation Patterns (DRP) for processing a continuous kNN query efficiently. DRP is a list of relative coordinates sorted by the distance between a point in a cell and other cells in ascending order so that they process a kNN query by visiting the cells sequentially.

However, these existing CkNN algorithms in road networks have the following disadvantages. First, traffic rules are not be considered. Second, the algorithms construct the data model based on the assumption that the moving speed and direction of objects and the topology of road networks will not change during the movement. However, it is not always the case in reality. Finally, the existing algorithms are unable to return valid results between two consecutive update timestamps because of the natural discreteness of position updating.

To overcome the issues of the methods mentioned above, an effective CkNN query algorithm is proposed in this study. In the mobile environment all the objects (including query objects) are continuously moving in the road network with traffic rules and the algorithm performs well on CkNN query processing

III. DATA MODEL OF ROAD NETWORKS WITH TRAFFIC RULES (DMRN^R)

A. kNN QUERY IN ROAD NETWORKS WITH TRAFFIC RULES

Definition 1 (k Nearest Neighbor query, ***kNN****) [25]:* Given a set of moving objects O and a query object q, a kNN query returns a set O' with k objects($O' \subseteq O$), so for any object $r \in O', p \in (O\text{-}O')$, O' satisfies:

$$k\text{NN}(q) = \{r \in O' | \forall r \in O', \forall p \in (O - O'),\ \text{dist}(q, r) \leq \text{dist}(q, p)\} \quad (1)$$

where dist(q, p) represents the distance between two moving objects p and q. In the road network, dist(q, p) is the sum of the road weights of the shortest path between p and q.

In existing kNN query methods, the data model of road networks is viewed as an undirected graph where dist(p, q) = dist(q, p), which means that the distances from p to q and q to p are identical. However, this is not applicable for moving objects in the road networks due to the restrictions of traffic rules, such as one-way or no sign of the left turn. Let us consider another example.

Example 2: As observed in Fig. 1, the minimum distance is 70m from o_1 to o_2 and 230m from o_2 to o_1 based the actual traffic rules because the object o_2 has to turn around at n_1 to reach o_1. In other words, dist(o_1, o_2) $\neq$ dist(o_2, o_1) may be observed in road networks.

Therefore, the distances discussed in this study are those from query object q to target objects o, which means only dist(q,o) has been considered. Nevertheless, the proposed method can be readily applied in cases of dist(o, q).

k NN query in road networks with traffic rules are defined as follows:

Definition 2 (kNN Query in Road Networks With Traffic Rules): Given road networks with traffic rules, a moving object set O, and query object q, kNN query returns an object set $O' \subseteq O$ with k objects under traffic rules, so for any object $r \in O'$ and $p \in (O\text{-}O')$, O' satisfies Formula 2:

$$k\text{NN}(q) = \{r \in O' | \forall r \in O', \forall p \in (O - O'), \text{dist}(q, r) \leq \text{dist}(q, p)\} \quad (2)$$

where dist (q, p) refers to the network distance from query object q to target object p based on traffic rules. Note that dist (q, p) $\neq$ dist (p, q).

B. MODEL OF ROAD NETWORKS

The proposed model of road networks could describe the following characteristics:

- Moving pattern of the moving object: all objects in road networks move freely based on traffic rules.
- Updating information regarding the moving object: when there are some changes of objects (eg., the speed or the road it is in), updated information that contains object ID, previous location, current location, and speed will be sent to the server.
- Status of road networks: dynamically updating topological structure and weights of roads.

In this paper, roads, road intersections, moving objects, and query information are modeled using four different models, respectively.

1) ROAD MODEL

The model of road (T_{road}) is a six-element group (rid, $<n_s, n_e>$, $<w_p, w_n>$, $<s_{pmax}, s_{nmax}>$, S_{obj}, $e.IL$). Herein, rid refers to the ID of the road, n_s and n_e refer to the starting and end points of the road respectively, w_p and w_n refer to road weights in the positive and negative directions, respectively, s_{pmax} and s_{nmax} refer to the upper limits of speed in positive and negative directions, respectively, S_{obj} refers to the current moving object set on this road, and $e.IL$ refers to the list of query objects q affected by this road ($\{<q, n, d>\}$), n is either n_s or n_e, d is a number describing the affecting range of road on query object q. For instance, d = 100 indicates that all objects within 100m from n are included in the k nearest neighbor set of q.

Example 3: If the length of the road is represented by its weight, the model of road $n_1 n_2$ in Fig. 1 is described as follows:

$$(e_1, <n_1, n_2>, <150\text{m}, 150\text{m}>, <15\text{m/s}, 12\text{m/s}>, \{o_1, o_2\}, \{<q, n_2, 100\text{m}>\})$$

2) ROAD INTERSECTION MODEL

In real road networks, the road intersection can reflect the traffic rules for moving objects at this node. Hence, the model of road intersection is defined as follows:

The model of road intersection (T_{node}) is a two-tuples ($conid$, $rules$). Herein, $conid$ refers to the id of this intersection, $rules$ refer to traffic rules set at this intersection, which can be described by the four-tuples $<RoadID_{from}, END_{from}, RoadID_{to}, END_{to}>$. In other words, rules can be defined by:

$$rules = \{<RoadID_{from}, END_{from}, RoadID_{to}, END_{to}>, \ldots\}$$

$RoadID_{from}$ and $RoadID_{to}$ represent the id of roads that the moving object is leaving and entering, respectively. END_{from} reflects whether the moving object left $RoadID_{from}$ from the endpoint n_e: if yes, $END_{from} = 1$; if no, $END_{from} = 0$. END_{to} reflects whether the moving object entered $RoadID_{to}$ from the endpoint n_e: if yes, $END_{to} = 1$; if no, $END_{to} = 0$.

Example 4: The intersection n_2 in Fig. 1 can be described by:

$$\text{s} = (n_2, \{<e_1, 1, e_3, 0>, <e_1, 1, e_1, 1>, <e_2, 0, e_1, 1>, <e_2, 0, e_3, 0>, <e_3, 0, e_2, 0>\})$$

where the rule $<e_2, 0, e_1, 1>$ reflects that objects can enter the end point of *e1* from the start point of *e2*, that is, the right turn rule labeled 1 in Fig. 1.

3) MOVING OBJECTS MODEL

The model of moving object (T_{obj}) is a five-tuples (oid, e_j, t_u, $dist$, v_i). Herein, oid refers to the id of this moving object, e_j refers to the road where the object moves in, t_u refers to the time of the last update for this object, $dist$ refers to the distance from this object to e_j. n_s (the starting point of the road it is located in at moment t_u), v_i refers to the speed of this object at time t_u. A positive speed means it is moving from the starting point to the endpoint and a negative speed means it is moving from the endpoint to the starting point.

Example 5: o_1 and o_2 in Fig. 1 can be described by the following moving object model:

$$(o_1, e_1, 0, 110\text{m}, 12\text{m/s}), \quad (o_2, e_1, 0, 120\text{m}, -10\text{m/s})$$

4) MODEL OF QUERY OBJECT q

A query model (T_{query}) is used for store the information of query objects. Meanwhile, query objects also are moving objects. Therefore, besides storage of information

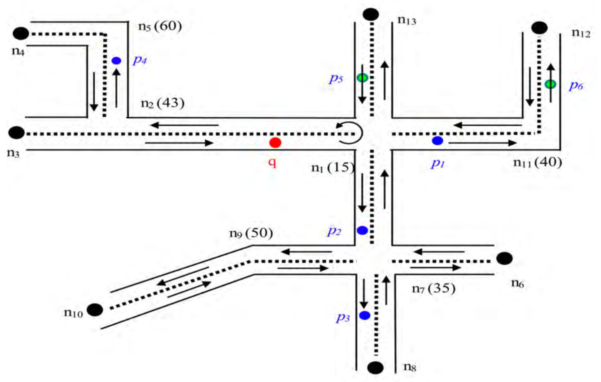


FIGURE 2. Road networks and kNN query.

about moving object for each q, it is also used for store the current kNN set of q (represented by $q.kNN$), the distance to the $k$ith nearest neighbor of q (represented by $q.kNN\_dist$) and the query expanding tree (T_q). For example, in the road network shown in Fig.2, the 4NN set of the query object q is $\{p1, p2, p3, p4\}$, and the distances from q to them are 23, 32, 42, 50, respectively. The objects $p5$ and $p6$ do not belong to the 4NN set of q.

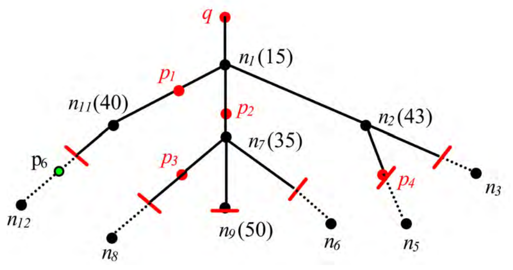


FIGURE 3. Expanding tree.

Fig.3 shows the expanding tree corresponding to the Fig.2 obtained in the initialization phase of the proposed CkNN query algorithm. The expanding tree can be involved in the monitoring phase of the query process to reduce the cost of re-calculating k nearest neighbor. The establishment and maintenance algorithms will be discussed in the next section.

Table 2 summarizes symbols and definitions involved in this study.

IV. INITIALIZATION PHASE OF EXTCKNN QUERY ALGORITHM

A. EXPANDING TREE

Based on the above models, we propose a novel Expanding Tree-based CkNN query algorithm for the moving object in road networks. The expanding tree of the query object is defined as follows:

Definition 3: Expanding Tree (T_q) of query object q is a tree structure that satisfies the following requirements:

- q is the root of this tree;

TABLE 2. Primary symbols.

Symbol	Description
T_{road}	Data structure of road in road network
T_{node}	Data structure of node in road network
T_{obj}	Data structure of moving object in road network
T_{query}	Data structure of query object in road network
$dist(q,p)$	The shortest network distance from q to p
$weight(n_1,n_2)$	The weight of the road (n_1,n_2)
$q.kNN$	The set of the k nearest neighbor of q
q.kNN_dist	The network distance from q to the k-th nearest neighbor
T_q	The expanding tree of q
$L(n_i)$	The network distance from q to node n_i

- All inner nodes involved are network nodes that can be reached from q;
- All leaf nodes involved are network nodes or the location of the road that can be reached from q;
- The shortest network distance from q to any node is no longer than the distance to the kth nearest neighbor of q ($q.kNN\_dist$);
- All successive nodes of any node involved can be reached directly from this node, except for the leaf nodes.

Fig. 3 shows the expanding tree of query object q obtained based on road networks and 4NN query in Fig. 2. Herein, the network distance from q to object p_4 ($q.kNN\_dist$) is 50. As we can see, the red line is the leaf node of this tree and the dashed line and nodes below this line (n_3, n_5, n_6, n_8, n_{12}) do not belong to this tree. The dashed line and those nodes are presented to clarify the road where leaf nodes are. Meanwhile, the red dots other than q (p_1, p_2, p_3, p_4) and p_6 do not belong to this tree. They are presented to clarify the location of the k nearest neighbors corresponding to q. The number in the bracket close to the nodes refers to the shortest network distance from q to this node. As shown in Fig. 3, leaf nodes can either be intersections in road networks (e.g., n_9 in Fig. 3) or a random location on roads (e.g., other leaf nodes in Fig. 3 except n_9). The leaf nodes below n_{11} refer to the divided locations on Road $n_{11}n_{12}$ that are 10m away from n_{11}.

We can obtain the following lemma based on the definition of the expanding tree:

Lemma 1: There are k and only k moving objects in the expanding tree of the query object q and all moving objects belong to the k nearest neighbor set of q ($q.kNN$).

Proof: Assume there are k' moving objects in the expanding tree:

If $k' > k$, then there should be at least one moving object o that satisfies dist(q, o) $> q.kNN\_dist$. According to the definition of the expanding tree, o is not supposed to be in the expanding tree. Therefore, $k' > k$ is invalid.

If $k' < k$, then there must be at least one moving object o in the kNN set of q that satisfies dist$(q, o) \leq q.kNN_dist$ and o is outside the expanding tree. According to the definition of the expanding tree, distance from q to leaf nodes in the expanding tree is $q.kNN\_dist$ and distance from q to o outside the expanding tree will be longer than $q.kNN\_dist$. In other words, dist$(q, o) > q.kNN_dist$. Hence, $k' < k$ is not valid.

In summary, $k' = k$ is the only possible case. Hence, there are k and only k moving objects in the expanding tree of query object q.

If o in the expanding tree does not belong to $q.kNN$, then dist$(q, o) > q.kNN_dist$. According to the definition of the expanding tree, this object should be outside the expanding tree. Hence, all objects in the expanding tree are in the k nearest neighbor set of q.

Additionally, the object moves continuously, and objects in the expanding tree are most possibly viewed to be the k nearest neighbor of q for the next query. This provides knowledge for continuous monitoring of nearest neighbors. Therefore, maintenance of expanding tree is required during the monitoring phase so that the k nearest neighbor set of the query object can be obtained by the expanding tree.

B. INITIALIZED *k* NEAREST NEIGHBOR RESULT SET AND EXPANDING TREE

Upon the submission of a new kNN query request, the initialized results will be obtained by the proposed expanding tree-based CkNN (ExTCKNN) algorithm. In the initialization phase, the ExTCKNN involves the Dijkstra-based algorithm [28]. Starting from q, the road network is expanded along moving directions of q and the expanding process terminates when the distance from the network node to q exceeds $q.kNN\_dist$. Meanwhile, an expanding tree of q is established, and the list of the affected roads in T_{road} is updated.

Since part of the roads in the road network affects the results of q, the proposed algorithm only stores part of the road information that affects the query using *partial_roads*. Additionally, during the expanding process, the current shortest network distance from q to node n_i is recorded as $L(n_i)$. Meanwhile, the current shortest network distance from q to the moving object o_i encountered during the expanding process is recorded using min heap $O_{encounter}$. It is worthwhile to notice that, o_i may not be in the expanding tree of q. For instance, it may be located on the dashed line shown in Fig. 3.

The initialization phase of the ExTCKNN algorithm is shown in Algorithm 1.

In this algorithm, the proposed road network model and query object q are set as inputs. First, in lines 1-3, min heap H_{node} used for storage of road intersections and set of moving objects ($O_{encounter}$) encountered during the expanding process are initialized to be null and the root node of expanding tree is defined as q. The shortest distances from q to all nodes in the road network are set to ∞. Then, line 4 expands q to the end point of the current road by calling Function1-1. In lines 5-6, $q.kNN\_dist$ is updated based on the current k nearest neighbor set of q. If k objects have been identified, the k nearest neighbor set is confirmed, and the algorithm terminates.

Algorithm 1 ExTCKNN_InitkNN(q)

```
Input: $T_{road}$, $T_{node}$, $T_{obj}$, query object $q$
Output: query model $T_{query}$, $O_{encounter}$
Steps:
1) $partial\_roads$, $H_{node}$, $O_{encounter} \leftarrow$null;
2) $T_q$.root$\leftarrow$q; $q.kNN\_dist \leftarrow \infty$; $q.kNN \leftarrow$null;
3) for each $n_i \in T_{node}$ do   {L($n_i$)$\leftarrow\infty$; V($n_i$)$\leftarrow$false; }
4) ExtendQToNode($q$);          //Function1-1, expand $q$ to
   end point of the current road
5) if $q.kNN$.length= $k$ then
6)   { Update($q.kNN$, $q.kNN\_dist$); return;}  //Update the
   distance to the $k$th nearest neighbor
7) ExtendNodes($H_{node}$);       // Functions 1-2 will be clarified
   below
8) for each node $n \in T_q$  do
9)    if V($n$) = false then
10)      DeleteNodeFromT($T_q$);
11) for each road $n_in_j$ $\in partial\_roads$ do
12)   {  Node splitnode = New Node($q.kNN\_dist$–L($n_i$));
13)      Insert($T_q$, $n_i$, splitnode);
14)      AddIL($n_in_j$, $q$, $n_i$, splitnode); }
```

Otherwise, line 7 expands nodes in H_{node} by applying Function1-2. Fig. 3 shows the final expanding tree, which contains n_3, n_5, n_6, n_8, n_{12}, and these nodes should be deleted as they do not belong to the expanding tree (line 10). In Lines 11-14, we obtain the split location of each road in *partial_roads* and insert it into T_q as the leaf node, that is the red-labeled roads in Fig. 3.

Function 1-1 ExtendQToNode(q) (Line 4 in Algorithm 1)

```
Input: query object $q$
Output: $T_{query}$, $O_{encounter}$
Steps:
1) $e \leftarrow q.e_j$;
2) if $q.v_i$ >0 then    $v \leftarrow e.v_e$;
3) else  $v \leftarrow e.v_s$;   AddIL($e$, $q$, $q$, $v$);
4) Insert($partial\_roads$, $q$, $v$);
5) Insert($T_q$, $q$, $v$); Insert($H_{node}$, $v$); L($v$) $\leftarrow$d($q$, $v$);
6) UpdateKNN($q.kNN$,ObjectsInRoad($q$, $v$), $k$);
   Insert($O_{encounter}$, ObjectsInRoad($q$, $v$));
```

Function1-1 expands q to the end point of the road. Lines 1-4 obtain node v along the moving direction of q according to the speed of q and store the road qv into *partial_roads*. Line 5 inserts v, whose parent node is q, into the expanding tree and heap H_{node} and record the shortest network distance from q to v. Line 6 adds moving objects on Road qv into the k nearest neighbor set of q until k objects are in the set. Objects o on road qv are inserted to $O_{encounter}$ with the distance from q to o is as key.

Function1-2 is used for expanding of nodes in H_{node}. Line 1 gets node n with the minimum key in H_{node}. If the key of node n is below $q.kNN\_dist$, Lines 2-17 expand the tree from n. Herein, Lines 4-5 identifies the road from n to its father node in T_q and delete it from *partial_roads* because all sections of this road affect query q. Lines 6-16 investigate all adjacent nodes n_{adj} of n(except its father node). Line 8 updates the query results ($q.kNN$ and $q.kNN\_dist$, an update of $q.kNN\_dist$ is only triggered in the presence of k objects in $q.kNN$) and $O_{encounter}$ based on information about moving objects on Road nn_{adj}.

Function 1-2 ExtendNodes(H_{node}) (Line 7 in Algorithm 1)

```
Input: Expanded node set $H_{node}$
Output: $T_{query}$, $O_{encounter}$ ($O_{encounter}$ is min heap)
Steps:
1) $n \leftarrow$ExtractMin($H_{node}$);
2) while $n$.key$<$ $q.kNN\_dist$ do {
3)   $V(n) \leftarrow$true;
4)   $e \leftarrow$road(FindFather($T_q$, $n$), $n$); // Assume $e$ is the road from the father node of $n$ in $T_q$ to $n$.
5)   AddIL($e$,$q$,$e.n_s$, $e.n_e$); Delete($partial\_roads$, $e$);
6)   for each adjacent node $n_{adj}$ of $n$ do{
7)     Insert($partial\_roads$, $n$, $n_{adj}$);
8)     UpdateKNN($q.kNN$,ObjectsInRoad($n$, $n_{adj}$), $k$);
       Insert($O_{encounter}$, ObjectsInRoad($n$, $n_{adj}$));
       Update($q.kNN$, $q.kNN\_dist$);
9)     if $V(n_{adj})$=false then { // if $n_{adj}$ has not been detected
10)      $dist$=$L(n) + nn_{adj}.w$;
11)      if FindNode($H_{node}$, $n_{adj}$) = $null$ || (FindNode($H_{node}$, $n_{adj}$)!=$null$ && IsTurnAround(FindNode($H_{node}$, $n_{adj}$), $n_{adj}$)) then { // Determine whether $n_{adj}$ should be inserted into $H_{node}$
12)        Insert($T_q$, $n$, $n_{adj}$);
13)        Insert($H_{node}$, $n_{adj}$);$L(n_{adj})$ =$dist$; }
14)      else if $dist$ < $L(n_{adj})$ then {
15)        UpdateFather($T_q$, $n$, $n_{adj}$); // Update father node of $n_{adj}$in expanding tree $T_q$ to be $n$
16)        $L(n_{adj})$ =$dist$; UpdateKey($H_{node}$, $n_{adj}$,$dist$);} // Update key of $n_{adj}$in $H_{node}$ to be $dist$ } }
17)  $n \leftarrow$ExtractMin($H_{node}$); }
```

Lines 9-16 insert unexpanded node n_{adj} into H_{node} and T_q and the key is the shortest network distance from q to this node via n. As turning around is allowed on roads, $n_2 \rightarrow n_1 \rightarrow n_2$ could be observed in the expanding tree. Hence, it should be determined whether it is a turning around case in presence of n_{adj} in the expanding tree. Specifically, it should be determined whether nodes between the added n_{adj} and the to-be-added n_{adj} are symmetric: if yes, n_{adj} is added; if no, n_{adj} is abandoned. Line 11 is responsible for this process. If n_{adj} has been inserted into H_{node}, its key is updated to the current shortest network distance. The expanding process terminates when the key (i.e., $dist(q, n)$) of node n extracted by Line 17 from H_{node} is no smaller than $q.kNN\_dist$.

Function 1-2 is the maintenance phase of expanding tree corresponding to query object q. Its time complexity is $O(n \times m)$ by analysis of this function.

Lemma 2: Let n denote the number of nodes extended in the expanding tree during the maintenance phase, and m mean the average number of roads that are connected to each node. Then the computational complexity to maintain the expanding tree is $O(n \times m)$.

Proof: Lines 2-17 in Function1-2 is used to extend the expanding tree, which consists of two layers of loops, because the number of extended nodes is n, then the outer loop executes n times, and the inner loop (lines 6-16) examines the adjacent nodes of each extended node, so the inner loop runs for m times, and each operation in the inner loop can be done in constant time. Therefore, the time complexity of the Function1-2 is $O(n \times m)$, that is, the time complexity to maintain the expanding tree is $O(n \times m)$.

By analyzing Algorithm 1, we can get the time complexity of the initialization phase of ExTCKNN as follows:

Lemma 3: The time complexity of the initialization phase of ExTCKNN is the same to the time complexity to maintain the expanding tree, and it is $O(n \times m)$;

Proof: the Function1-1(line 4) in Algorithm 1 can be completed in a constant time, that is, the time complexity of Function1-1 is $O(1)$. The Function1-2(line 7) is used to maintain the expanding tree, and the time complexity is $O(n\times m)$. Assume that there are n' nodes in the expanding tree, then the time complexity of the lines 8-10 is $O(n')$. If there are m' road in *partial_roads*, the time complexity of lines 11-14 is $O(m')$. Therefore, the computational complexity of the algorithm1 is $O(1) + O(n \times m) + O(n') + O(m')$, that is $O(n \times m)$.

As shown in Fig. 2, the nodes n_1, n_2, n_7, n_{11} are expanded and $V(n_1)$, $V(n_2)$, $V(n_7)$, $V(n_{11})$ are true after executing Algorithm 1. T_q is established with a breadth priority and for every node n in H_{node}, $L(n)$ is equivalent to $n.key$. The key value of object o stored in $O_{encounter}$ is defined as the shortest network distance from q to o. Although some objects in $O_{encounter}$ are not currently considered as k nearest neighbor objects, it is possible to be included in the k nearest neighbor set during the next update period through the continuous monitoring process. They can accelerate the query process; thus, $O_{encounter}$ should be modified during the continuous monitoring process.

V. k NEAREST NEIGHBOR MONITORING ALGORITHM FOR MOVING OBJECT IN ROAD NETWORKS

After the initialization phase, k nearest neighbor set of query object q is obtained, and continuous monitoring is applied in this set during the query process. In the real-world cases, both moving objects and roads may dynamically change, and the processing method of updating of moving objects (except for query object) is proposed as follows.

A. UPDATING OF MOVING OBJECT

Assume that the update information of a moving object o includes $oid(o.oid)$ and its latest location information (including the new road e_{new} and the distance from starting point of this road). The model of road networks should be updated according to moving object updates.

1) UPDATE MISSION OF MOVING OBJECT

For moving object updates, the proposed data model of road networks should be updated:

- Update road model (T_{road}): identify the road e_j containing o according to $o.oid$ and T_{obj}, delete o from the object list of e_j, and add it into that of e_{new};
- Update object model (T_{obj}): update the information of o in T_{obj} according to its current location on e_{new}.
- Update $O_{encounter}$ and query model (T_{query}): the update of the query model is the key task and will be described below.

2) PATTERNS OF MOVING OBJECT UPDATE

The updating phase of moving objects can be classified into several patterns as given in Example 6:

Example 6: In Fig. 2, the 4NN of q that has been initialized by Algorithm 1 is completely included in q's expanding tree. At the next updating operation, locations of p_4, p_5, p_6 are updated in three patterns.

- Pattern 1 (moving in and leaving the expanding tree): p_6 moves towards n_{12} but is still outside the expanding tree;
- Pattern 2 (leaving the expanding tree): p_4 moves towards n_5 and leaves the expanding tree;
- Pattern 3 (entering the expanding tree): p_5 moves from $n_{13}n_1$ to n_1n_2.

Pattern 1 includes the case that the moving object is in the expanding tree before and after updates.

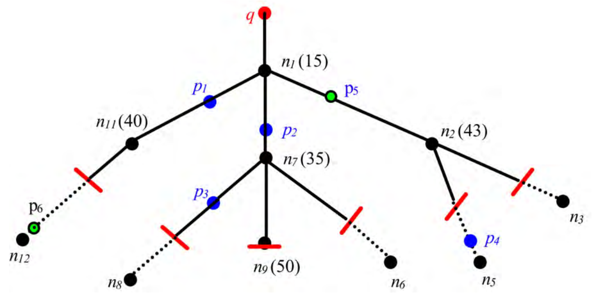


FIGURE 4. The situation of objects after 3 objects update.

Fig. 4 shows the object location after the update operation. Figures 4 and 5 indicate the changes of the q's k nearest neighbors when objects enter or exit. In other words, Pattern 2 and 3 will induce changes to k nearest neighbor results of q, while Pattern 1 will not.

3) PROCESSING OF MOVING OBJECT UPDATE

An object updates at a specific moment may involve all three patterns. In these cases, the initial output $O_{encounter}$ of k nearest neighbor of query q should be maintained, besides updating k nearest neighbor results of q.

a: $O_{encounter}$ MAINTENANCE

$O_{encounter}$ is responsible for storing moving objects encountered during the establishment of the expanding tree. The roads that contain the objects of $O_{encounter}$ affect the query q (i.e., roads in e.IL that contains q). Therefore, we only need to determine whether the updated objects are in these road sets, which implies the maintenance is determined by the fact whether e.IL of the road which the update objects are in contains q. Two situations may be observed:

(1) The objects are in the affecting road set before and after an update, o has been in $O_{encounter}$, and the expanding tree contains partially, if not completely, the road where updated o is in. Herein, the latest distance of the object is determined based on the distance from this object to its father node in the expanding tree and the key value of its father node and heap $O_{encounter}$ is updated. An example is the update of p_6 in the above case.

(2) The objects enter or leave the affecting road set after update (Pattern 2 and 3 in the above case). Herein, the object is either inserted to or removed from $O_{encounter}$.

b: T_{query} MAINTENANCE

T_{query} maintenance depends on the relation between the updated objects number of Pattern 2 (Num_{exit}) and Pattern 3 (Num_{enter}). Let O_{exited} and $O_{entered}$ are the objects sets of Pattern 2 and Pattern 3, respectively.

(1) When $\boldsymbol{Num_{exit}} \leq \boldsymbol{Num_{enter}}$, there are at least k moving objects in the current expanding tree of q and, the distance from q to these objects is no longer than $q.kNN\_dist$. So the latest k nearest neighbor of q are still in the expanding tree. Herein, k nearest neighbor candidate set of q is $q.kNN + O_{entered} - O_{exited}$ and all objects in this set are currently in $O_{encounter}$. Therefore, the following two steps should be executed:

- Obtaining a kNN set: as $O_{encounter}$ is a minimum heap, the first k objects in $O_{encounter}$ are the latest k nearest neighbor of query object q;
- Expanding tree compression: the expanding tree should be updated because the distance from q to its latest kth neighbor is no larger than $q.kNN\_dist$, for instance, the case mentioned above. The red labeled in the expanding tree shown in Fig. 4 will move towards the root to obtain the expanding tree in Fig. 5.

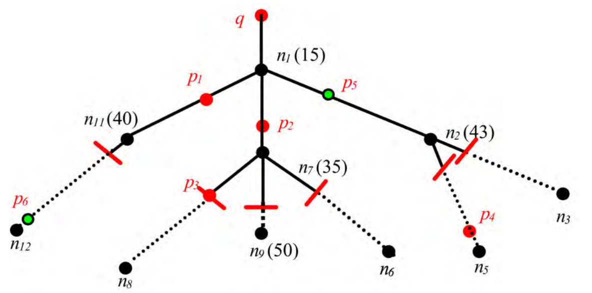


FIGURE 5. The expanding tree after 3 objects update.

(2) When $\boldsymbol{Num_{exit}} > \boldsymbol{Num_{enter}}$, the number of the objects in the expanding tree with distances no larger than $q.kNN\_dist$ are less than k and all objects in $q.kNN + O_{entered} - O_{exited}$ are supposed to be in the expanding tree. Also, these objects are supposed to be the k nearest neighbors of q and in $O_{encounter}$. Therefore, two steps should be executed as follows:

- Obtaining a kNN set: the first k objects extracted from $O_{encounter}$ are the latest k nearest neighbor of q.
- The expanding tree is further expanded: if the number of objects in $O_{encounter}$ is below k, the expanding tree is expanded by Function 1-2 to expand the searching area. Herein, H_{node} is initialized using the red labeled nodes (i.e., leaf nodes in T_q) (see Fig. 3) and obtain the latest expanding tree and $O_{encounter}$ accordingly. The cost for k nearest neighbor calculation is reduced as information of previous expanding tree has been used. Notably, leaf nodes may move toward the lower level of the expanding tree in this case.

B. UPDATING OF THE QUERY OBJECT

1) UPDATING PATTERNS OF QUERY OBJECT

According to the new location of query object q (q_{new}), updates can be classified as:

- Pattern 1: query object exits the expanding tree. It means q_{new} is outside the expanding tree.
- Pattern 2: query object moves in the expanding tree. It indicates q_{new} is in the expanding tree.

2) PROCESSING OF QUERY OBJECT EXITS EXPANDING TREE

Example 7: In Fig. 2, q moves to n_5n_4. In this case, n_5n_4 is outside the current expanding tree (see Fig. 3) and the latest k nearest neighbor of q cannot be calculated based on the current expanding tree and $O_{encounter}$. Instead, q is deleted from $e.IL$ of all roads in the expanding tree and, its k nearest neighbor is re-initialized by calling Algorithm 1.

3) PROCESSING OF QUERY OBJECT MOVES IN EXPANDING TREE

Example 8: In Fig. 2, q moves to the road n_1n_7. In this case, q_{new} is on the road n_1n_7 and, the latest k nearest neighbor of q_{new} can still be calculated based on sub-trees with q_{new} as the root. After the query object updated, the maintenance of the road model, $O_{encounter}$ and k nearest neighbor results, shall be achieved by the ExTCKNN algorithm. The maintenance process is as follows:

a: MODIFICATION OF ROAD MODEL (T_{road})

The query object q should be deleted from $e.IL$ of roads that do not affect the query after q is updated. Herein, only those roads that are not in sub-trees with q_{new} as the root (e.g., n_1n_2 and n_1n_{11} in Fig. 3) shall be identified so that q is deleted from $e.IL$ of these roads.

b: MAINTENANCE OF $O_{encounter}$

First, objects that do not belong to sub-trees with q_{new} as the root are deleted from $O_{encounter}$. Then, distances of objects left in $O_{encounter}$ are updated. Specifically, the updated distance equals to the previous distance minus displacement distance of q.

c: MAINTENANCE OF THE QUERY MODEL (T_{query})

According to the expanding tree, all objects in sub-trees with q_{new} as the root are still k nearest neighbors of q_{new} in this case. After maintenance of $O_{encounter}$, updating information of these objects are stored in $O_{encounter}$. Also, objects in $O_{encounter}$ that were not k nearest neighbors of q are highly possible to become k nearest neighbors of q_{new}, and they are superior to objects not in $O_{encounter}$. So the following steps are executed:

(1) add first k objects in $O_{encounter}$ into the updated k nearest neighbor set;

(2) if objects in $O_{encounter}$ are less than k, sub-trees with q_{new} as the root are expanded by calling Function 1-2. Herein, information of these sub-trees should be updated at first: distances from q_{new} to nodes in the sub-tree are modified and H_{node} is initialized using leaf nodes of this sub-tree. Further expanding based on that results in the updated expanding tree.

C. UPDATING OF ROAD NETWORKS

1) ANALYSIS OF ROAD UPDATE PROCESSING

The update operation of road networks are mainly updates of road weights. For instance, a road can be set as one-way by defining the forbidden road weight as ∞. Hence, this study focuses on the update of road weights. Additionally, updating e_i will affect the query results only when these roads are in the expanding tree.

2) PROCESSING OF ROAD WEIGHT UPDATES

The update operation of road weights in the expanding tree can be classified into two categories:

a: DECREASING ROAD WEIGHT, SUCH AS n_1n_{11} IN Fig. 2

This update may lead to cases where objects that do not belong to the expanding tree (e.g., p_6 on $n_{11}n_{12}$) enter the expanding tree and replace other objects in k nearest neighbors set. For objects in other subtrees in T_q (except for n_1n_{11} and sub-trees with n_{11} as the root), both the shortest paths and shortest distances from q to them may vary. However, variations are only observed for objects on sub-trees with n as root in T_q, where $dist(q, n) > dist(q, n_{11})$. Hence, when the weight of $n_1n_{11}$is reduced, the following steps are executed:

(1) delete sub-trees with n ($dist(q,n) > dist(q,n_{11})$) as root from T_q, delete q from IL of roads contained in these sub-trees, and delete affected objects in $O_{encounter}$;

(2) update key of all objects and nodes on sub-tree with n_{11} as the root and modify objects information in $O_{encounter}$ accordingly;

(3) initialize H_{node} using leaf nodes of the current expanding tree and expand the expanding tree by employing Function 1-2.

b: INCREASING ROAD WEIGHT, SUCH AS n_1n_7 IN Fig. 2

Herein, the distance from q to o on n_1n_7 should be modified as the current distance plus weight variations. The shortest paths and distances from q to the object on roads in sub-tree with n_7 as the root will be affected as these objects can be reached from q without passing through n_1n_7. Therefore, sub-tree with n_7 as the root in the expanding tree should be deleted and, the expanding tree should be expanded by calling Function 1-2.

D. CONTINUOUS k NEAREST NEIGHBOR MONITORING ALGORITHM OF ExTCKNN

In real-world cases, the three pattern updates can be observed at any update moment. Herein, S_o, S_r, S_q denote the updated objects set, the updated roads set, and the updated query objects set transmitted to the central server at a specific update moment, respectively. The proposed nearest neighbor monitoring algorithm for moving objects in road networks is introduced as follows:

Assume two cases for transmission of moving object update information in road networks: the moving object moves from one road to another and speed of moving object significantly varies. If the object o' has no significant variations of moving speed, it is considered to move on the previous road in $o'.v_i$, which is close to the speed at the last update moment $(o'.t_u)$ and variations of the object can be calculated accordingly.

1) CONTINUOUS MONITORING ALGORITHM

Based on the description above, our proposed ExTCKNN algorithm is shown in Algorithm 2.

Algorithm 2 ExTCKNN_Monitoring (t'_u, S_o, S_r, S_q)

```
Input: update time:$t'_u$, $S_o$, $S_r$, $S_q$.
Output: updated road network model
Steps:
1)  $Q_{affected} \leftarrow null$; // $Q_{affected}$ stores query objects
    affected by this update
2)  HandleObjUpdate($S_o$, $t'_u$);
3)  HandleQueryRemoveUpdate($S_q$);
4)  HandleOtherUpdate($S_r$, $S_q$, $S_o$, $Q_{affected}$);
5)  for each query $q \in Q_{affected}$ do
6)      $q.kNN$ ←ExtractObjects($O_{encounter}$, $k$); // Extract k
objects from $O_{encounter}$
7)      if $q.kNN$.length < $k$ then
8)          AdjustTree($q$);   // Further expand $T_q$
```

First, update the location of the object in T_{obj} (Line 2).

Then, check whether the query object q leaves T_q: if yes, re-initiate and re-calculate k nearest neighbor of q using Algorithm 1 (Line 3) without considering other update information; if no, updates of roads affecting the query are processed from those with reducing weights to those with increasing weights. Then, an update of query object q is processed as described in Section V.B, followed by a pruning of the expanding tree. Finally, updates of moving objects are processed as described in Section V.A (Line 4).

After the above steps, objects stored in $q.kNN$ are the latest k nearest neighbors of q if the distances from q to leaf nodes of the expanding tree exceed $q.kNN\_dist$. If the number of objects in $q.kNN$ is less than k, the expanding tree is further expanded by Algorithm 1 (Lines 5-8).

Function 2-1 is employed to update T_{obj}. For an object in S_o, if it moves from one road to another, this object is deleted from the previous road and added to the current road. Meanwhile, information about this object in $O_{encounter}$ has been modified accordingly (Lines 1-2). For the object o that does not submit update information and is still in the previous road, its moving distances between two update moments are calculated using $o.v_i \times (t'_u - o.t_u)$, and its information is updated accordingly. Meanwhile, it is possible for those objects that do not submit update information enter the query tree and become k nearest neighbors. Therefore, the expanding tree and $O_{encounter}$ should be updated. Herein, any object entering roads in the expanding tree are added into $O_{encounter}$ (Lines 3-4).

Function 2-1 HandleObjUpdate(S_o, t'_u)

```
Input:updated object set $S_o$.
Output: updated road network model
Steps:
1)  for each $o \in S_o$ do
2)      UpdateTobjAndTroad($O_{encounter}$, $o$); // Update
        information about $O$ in $T_{obj}$ and $T_{road}$
3)  for each $o \in (T_{obj}$- $S_o)$ do
4)      UpdateTobj($o$, $t'_u$, $T_q$, $O_{encounter}$);
```

Function 2-2 handles the cases where the query object in S_q has moved outside the expanding tree and re-calculate the corresponding k nearest neighbors.

Function 2-2 HandleQueryRemoveUpdate(S_q)

```
Input: updated query object set $S_q$.
Output: updated road network model
Steps:
1)  for each  $q \in S_q$ do
2)      if $q$ is not inside $T_q$ then
3)      { DeleteQueryFromTroadIL ($q$);
4)        ExTCKNN_InitKNN($q$); } // Delete $q$ from the road
  that affects it ($IL$) and re-calculate k nearest neighbor of $q$
```

Function 2-3 handles cases where the query object in S_q is still in the expanding tree. Herein, Lines 1-4 handle updated roads. Line 3 employs Function2-3-1 to delete sub-tress according to variations of weights and maintain k

nearest neighbor results of q and the data model (Lines 2-9 in Function2-3-1). For updated query objects in S_q that are still in road n_in_j of the expanding tree, Line 6 employs Function2-3-1 to establish a new expanding tree (T'_q) based on the new location (q_{new}) of q and delete T_q, while maintaining other information (Lines 10-13 in Function2-3-1). Lines 8-12 maintain query results after the moving object updated. If an object belongs to k nearest neighbor set of query object q and leave T_q after updated, this object should be deleted from the k nearest neighbor set; otherwise no changes. During the updating process mentioned above, query objects affected by this updating pattern are recorded using $Q_{affected}$.

Function 2-3 HandleOtherUpdate(S_r, S_q, S_o, $Q_{affected}$)

```
Input: $S_o$, $S_r$, $S_q$, and $Q_{affected}$.
Output: updated road network model
Steps:
1)  for each road $n_in_j \in S_r$ do
2)      for each query $q \in n_in_j.IL$ do
3)      { DeleteSubTreeAndUpdateDS($q$, $n_in_j$);
4)          $Q_{affected} \leftarrow Q_{affected} \cup \{q\}$; }
5)  for each query $q \in S_q$ && q $\in n_in_j$ of $T_q$ do  // updated query is still in $T_q$
6)    {DeleteSubTreeAndUpdateDS($q$, $n_in_j$);
7)      $Q_{affected} \leftarrow Q_{affected} \cup \{q\}$; }
8)  for each $o \in S_o$ do
9)      for each query $q$ and $o \in q.kNN$ do
10)         if $o$ outside $T_q$ then
11)         {DeleteObjectFromkNN($q.kNN$, $O_{encounter}$, $o$);
12)           $Q_{affected} \leftarrow Q_{affected} \cup \{q\}$; }
```

E. k NEAREST NEIGHBOR QUERY OF ROAD NETWORKS

After processing by the ExTCKNN algorithm, the latest k nearest neighbors of q have been stored in $q.kNN$. When a query request is proposed, the current time t and query object set $\{q\}$ are used as parameters for the monitoring algorithm of ExTCKNN to obtain the newest k nearest neighbor set of $q(q.kNN)$ and return the result directly.

VI. EXPERIMENT AND PERFORMANCE EVALUATION

In order to evaluate the effectiveness and validity of the proposed algorithm, a series of experiments have been designed and re-partitioning of time intervals in the CKNN algorithm [13] and OCP algorithm [22] at each updating time is achieved by modifying the proposed data model so that these algorithms can be applied on the proposed data model. The performance comparison of the proposed algorithm and the CKNN and OCP algorithms demonstrates better efficiency and viability of the proposed algorithm, and it is considered to be applied to the real road networks.

Our ExTCKNN algorithm is implemented in Java and all the experiments are conducted on a PC running

Function 2-3-1 DeleteSubTreeAndUpdateDS(Query q, Road n_in_j)

```
Input: query object to be updated ($q$), road to be updated ($n_in_j$).
Output: expanding tree ($T_q$) of $q$
Steps:
1)  for each node $n \in T_q$ do
2)    if $n_in_j.w$ decreasing then {
3)      if $n \notin$SubTree($T_q$, $n_j$) && $dist(q,n)>dist(q, n_j)$ then
4)      { DeleteSubTree($T_q$,SubTree($n$));  //Delete sub-trees with $n$ as root
5)         DeleteObjectFromkNN(SubTree($n$),$q.kNN$,
        $O_{encounter}$); } }// Delete objects on sub-trees with $n$ as root in $k$ nearest neighbor set of $q$and $O_{encounter}$
6)    else if $n_in_j.w$ increasing then  {
7)      if $n \in$SubTree($T_q$, $n_j$)&&$dist(q,n)>q.kNN\_dist$then
8)      { DeleteSubTree($T_q$,SubTree($n$));
9)         DeleteObjectFromkNN(SubTree($n$),$q.kNN$,
        $O_{encounter}$); } }
10)   else{  // q is in expanding tree
11)       $T'_q.root \leftarrow null$; $T'_q.root \leftarrow q_{new}$;
12)       $q_{new}.child \leftarrow$ SubTree($T_q$, $n_j$); // $n_j$ is the updated node below $q_{new}$
13)       DeleteTree($T_q$); }
14) DeleteQueryFromTroadIL ($q$);
15) UpdateDistance($T_q$); // Update information about the rest nodes in $T_q$
16) UpdatekNN($q.kNN$, $O_{encounter}$); // Update distance from q to the rest objects of $q.kNN$ and $O_{encounter}$
```

Windows10 equipped with an Intel Dual E2620 V3 CPU 2.4GHz and 32GB memory.

A. EXPERIMENTAL DATASETS

In this study, the experimental dataset was generated using the road network based moving object generator [21], which uses practical road networks as inputs. The Oldenburg road map was used in the study, and 10∼50k moving objects were generated. Table 3 summarizes the characteristics of the dataset.

TABLE 3. Parameters of datasets.

Parameter	Oldenburg
Map width	23,572m
Map length	26,915m
Number of vertices	6105
Number of roads	7035
Number of moving objects	10k∼50k/default 30k

Based on the inputs mentioned above, the parameters of the proposed data model of road networks are set as follows: object speeds are in the range of −20∼20 m/s (negative speed indicates cases that objects move from end point to start

TABLE 4. Parameters of data model in road network.

Parameter	Default value	range
Speed of moving object	-12meter/sec(negative direction),12meter/sec	-20～20 meter/sec
Speed limit	20 meter/sec	0～20 meter/sec
Number of nearest neighbor(k)	30	10,20,30,40,50
Number of update objects	6%	2,4,6,8,10(%)
Number of query objects	6%	2,4,6,8,10(%)
Number of update road	3%	1,2,3,4,5(%)
The weight of road	±30%	±(10,20,30,40,50)(%)

point), the initial locations of moving objects follow unified distribution (or Gaussian distribution in the proposed algorithm), the speed limits of roads are in the range of 0∼20 m/s, rules of intersections in road networks are generated randomly. A random 2% of moving objects are regarded as query objects. At each update time, a random 2∼10% of moving objects are updated and a random 2∼10% of query objects are updated. The locations of updated moving objects and updated query objects are determined based on traffic rules and random walk at their current speeds. The initial weights of roads correspond to their lengths, and a random 1∼5% of the road networks are changed by a random value in $(-10\%)\sim(-50\%)$ and $(10\%)\sim(50\%)$ (negative changes denote the weight reduction of the road) at each update moment. A random 10% of varied roads are set to have weights of ∞ as certain roads will be set as one-way in cases such as peak hours. At the same updating time, the moving object set, the query object set, and the update road set shall have at least one set that is not null. Table 4 summarizes the parameters involved in our proposed model.

The effects of these parameters on algorithm performance and accuracy of query results, which refers to the ratio of moments that k nearest neighbor objects are accurately extracted, are evaluated. Only one parameter is a variable in each case (other parameters are set to be defaults) and the average of 10 rounds is defined as the ultimate value. The updating period is five-time units.

B. EFFECTS OF MOVING OBJECT NUMBER

Fig. 6(a) illustrates the effects of moving objects number on the query time. As we can see, the execution time of both CKNN and OCP algorithms increase while that of ExTCKNN algorithm decreases as the number of moving objects increases. With moving object number $<$ 20k, the moving object distribution is sparse and expanding tree in the ExTCKNN algorithm is relatively large to obtain k nearest neighbor. As a result, the performance of CKNN and OCP algorithm is slightly better than ours'; with moving object number $>$ 20k, the moving object distribution is concentrated and expanding tree in the ExTCKNN algorithm is relatively small (low maintenance cost and considerable candidate objects in $O_{encounter}$). Thus, the performance of ExTCKNN algorithm is significantly better than that of CKNN and OCP algorithm.

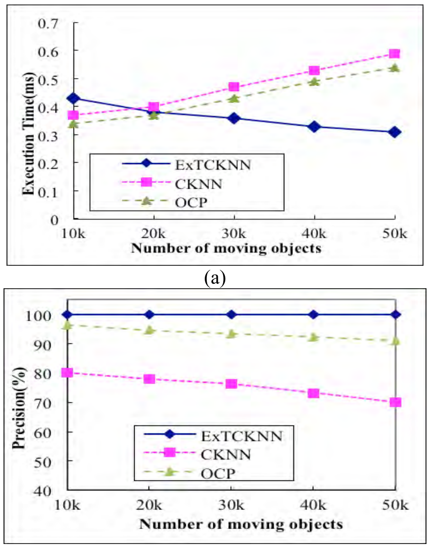


FIGURE 6. Effects of moving objects number. (a) Execution time. (b) Accuracy.

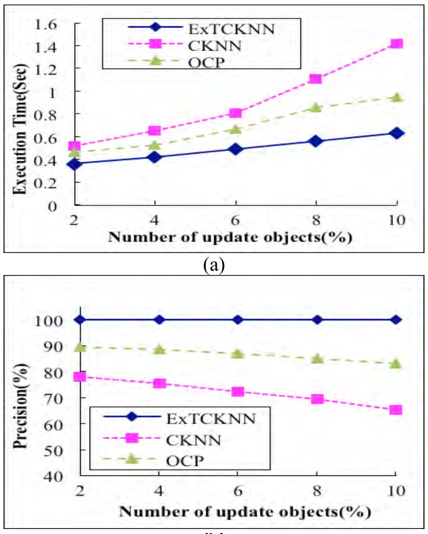


FIGURE 7. Effects of updated moving objects number (a) Execution time. (b) Accuracy.

Fig. 6 (b) shows the effects of moving objects number on the accuracy of the results. As we can see, the information of all moving objects is maintained in ExTCKNN algorithm. Therefore, the latest k nearest neighbor is always stored in $q.kNN$, and query results are perfectly accurate. In CKNN and OCP algorithm, however, object motions are viewed as constant speed, and no updates are processed, resulting in poor accuracy. But the accuracy of OCP is better than that of CKNN because the kNN set at each timestamp can be entirely determined by OCP when the moving objects keep fixed speed.

C. EFFECTS OF THE UPDATED MOVING OBJECT NUMBER

Fig. 7 illustrates the effects of the number of updated moving objects on algorithm performance. As we can see in Fig. 7(a), efficiencies of all the three algorithms decrease as the number of updated moving objects increases. In ExTCKNN, increasing the number of updated moving objects facilitates the expanding process of the expanding tree; in CKNN and OCP, time intervals of the updated objects need to be re-partitioned. Besides, the efficiency degradation is more severe for CKNN algorithm in this case. The objects entering the expanding tree and those exiting approached an equilibrium when the number of updated moving objects saturates. Hence, the maintenance cost of expanding tree in ExTCKNN algorithm shows no significant variations. Overall, the execution time of ExTCKNN is 30∼100% and 15%∼40% less than that of CKNN and OCP algorithms, respectively. As shown in Fig. 7(b), as the number of updated moving objects increases, the accuracy of ExTCKNN stays at 100%, while that of CKNN and OCP algorithms degrades significantly. The reason is that the accuracy of time interval partitioning in these two algorithms degraded while the number of updated moving objects grows.

D. EFFECTS OF UPDATED QUERY OBJECT NUMBER

Fig. 8 illustrates the effects of updated query objects number on algorithm performance. As shown in Fig. 8 (a), the effects of updated query objects number on execution time are significant for ExTCKNN algorithm. In most cases, query object updates will induce regeneration or an expansion of the expanding tree in ExTCKNN algorithm, resulting in reduced efficiency. However, the execution time of ExTCKNN algorithm is still less than that of CKNN and OCP algorithms. As shown in Fig. 8 (b), the accuracy of CKNN algorithm degrades to 65% as the number of updated query objects increases. During its updating, the query object is not maintained by CKNN algorithm, resulting in reduced accuracy. Moreover, the accuracy of CKNN algorithm further degraded as the updating frequency increases. In contrast, ExTCKNN algorithm maintains a high accuracy in all cases. Meanwhile, the accuracy of OCP algorithm degrades to 90% as the number of updated query objects increases. This can be explained that the moving objects processed increase slowly by their *moving_State* value, resulting in low accuracy.

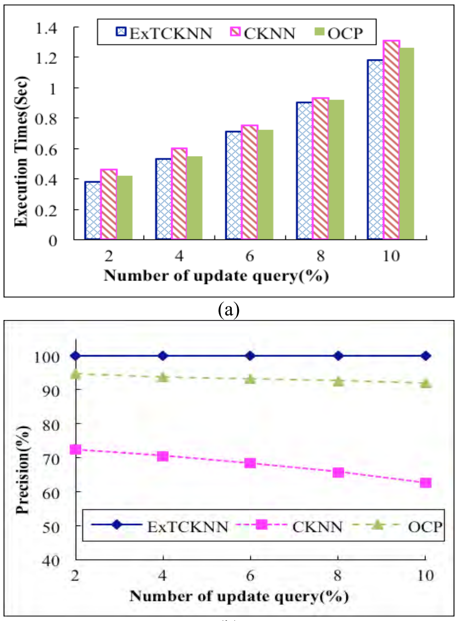


FIGURE 8. Effects of updated query objects number. (a) Execution time. (b) Accuracy.

E. EFFECTS OF UPDATED ROAD NUMBER

Fig. 9 illustrates the effects of the updated road number on algorithm performance. As observed in Fig. 9 (a), road update has a significant impact on the execution time of these three algorithms. This can be attributed to the fact that certain sub-trees in the expanding tree are deleted, and the expanding tree is further expanded. This is particularly true when the road weight is specified to ∞. For the CKNN and OCP algorithm, a significant number of time intervals shall be re-partitioned without previous information available. Meanwhile, the *moving_State* of all moving objects shall be updated in OCP algorithm. As observed in Fig. 9 (b), the accuracy of CKNN and OCP algorithms degrades significantly as the updated road number increases. One reason is that these two algorithms can barely process cases with road update. When the road weight is set to ∞, partitioning of the time intervals cannot be achieved by these algorithms.

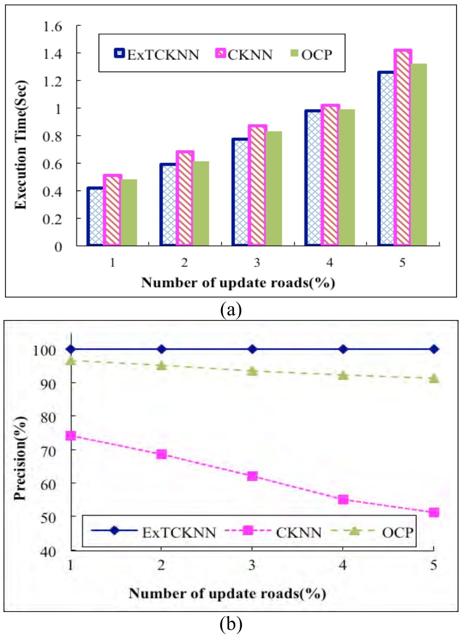


FIGURE 9. Effects of updated roads number. (a) Execution time. (b) Accuracy.

F. EFFECTS OF K ON ALGORITHM PERFORMANCE

Fig. 10 illustrates the effects of k on algorithm performance. As shown in Fig. 10, the performances of these three algorithms degrade as k increases, but it is more severe for both

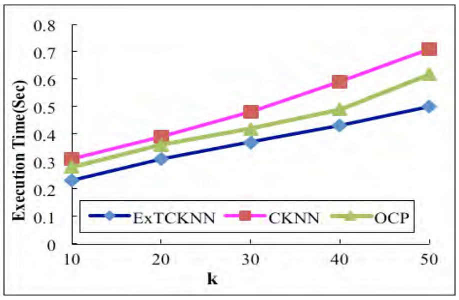


FIGURE 10. Effects of k.

CKNN and OCP algorithms. This can be attributed to the increasing of expanding tree in the ExTCKNN algorithm induced by increasing k. In this case, considerable candidate objects have been stored in $O_{encounter}$ so that the continuous query can be accelerated. However, there is no similar mechanism to stimulate query in CKNN and OCP algorithms.

The experimental results demonstrate that ExTCKNN algorithm is superior to OCP and it is optimized for CKNN algorithm. The mechanism of expanding tree is reliable and helpful for monitoring the continuous k nearest neighbor queries in road networks and complex transportation networks.

VII. CONCLUSIONS

The kNN query of moving objects in road networks has been an active topic in recent years. Most previous studies achieved kNN of moving object in road networks by simply converting Euclidean distance to the road network distance without considering practical conditions of road networks (e.g., road rules).

Highlighted by the discussion above, we propose a data model of road networks that combines practical road networks and a novel continuous k nearest neighbor monitoring algorithm for road networks based on this model. The proposed algorithm consists of three phases: the initial query results and expanding tree are obtained in the initialization phase, moving objects and road variations are processed in the monitoring phase (the k nearest neighbor updating is accelerated based on the information about expanding tree), and the latest k nearest neighbor of query object is obtained in the query phase based on its latest location. Extensive experiments have demonstrated that the proposed algorithm is a viable and effective query algorithm that is applicable in road networks and it exhibits great advantages on effectiveness and accuracy beyond the state-of-the-arts.

One important future extension of this work can be directed toward distributing the computation load at the expanding tree to multiple servers. Since the tree structure divides the network space into the non-overlapping region and our algorithm are simple to be implemented. Another extension may include the expansion of the proposed method to cases involving constraints-based nearest neighbor monitoring of road networks.

REFERENCES

[1] Y. Huang, F. Bastani, R. Jin, and X. S. Wang, "Large scale real-time ridesharing with service guarantee on road networks," in *Proc. 40th Int. Conf. Very Large Databases (VLDB)*, Hangzhou, China, 2014, pp. 2017–2028.

[2] J. M. Lee, "Fast k-nearest neighbor searching in static objects," *Wireless Pers. Commun.*, vol. 93, no. 1, pp. 147–160, Mar. 2017.

[3] S. Alamri, D. Taniar, and M. Safar, "A taxonomy for moving object queries in spatial databases," *Future Generat. Comput. Syst.*, vol. 37, pp. 232–242, Jul. 2014.

[4] K.-W. Lee, D.-W. Choi, and C.-W. Chung, "DART+: Direction-aware bichromatic reverse k nearest neighbor query processing in spatial databases," *J. Intell. Inf. Syst.*, vol. 43, no. 2, pp. 349–377, Oct. 2014.

[5] H.-J. Cho, S. J. Kwon, and T.-S. Chung, "A safe exit algorithm for continuous nearest neighbor monitoring in road networks," *Mobile Inf. Syst.*, vol. 9, no. 1, pp. 37–53, Jan. 2013.

[6] Y. Jing, L. Hu, W.-S. Ku, and C. Shahabi, "Authentication of k nearest neighbor query on road networks," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 6, pp. 1494–1506, Jun. 2014.

[7] H. Al-Khalidi, Z. Abbas, and M. Safar, "Approximate range query processing in spatial network databases," *Multimedia Syst.*, vol. 19, no. 2, pp. 151–161, Mar. 2013.

[8] S. Wang, M. A. Cheema, and X. Lin, "Efficiently monitoring reverse k-nearest neighbors in spatial networks," *Comput. J.*, vol. 58, no. 1, pp. 40–56, Jan. 2015.

[9] H.-J. Cho and C.-W. Chung, "An efficient and scalable approach to CNN queries in a road network," in *Proc. 31st Int. Conf. Very Large Data Bases (VLDB)*, Trondheim, Norway, 2005, pp. 865–876.

[10] K. Mouratidis, D. Papadias, and M. Hadjieleftheriou, "Conceptual partitioning: An efficient method for continuous nearest neighbor monitoring," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, Baltimore, MD, USA, 2005, pp. 634–645.

[11] G. Zhao *et al.*, "Voronoi-based continuous k nearest neighbor search in mobile navigation," *IEEE Trans. Ind. Electron.*, vol. 58, no. 6, pp. 2247–2257, Jun. 2011.

[12] K. Mouratidis, M. L. Yiu, D. Papadias, and N. Mamoulis, "Continuous nearest neighbor monitoring in road networks," in *Proc. 32nd Int. Conf. Very Large Data Bases (VLDB)*, Seoul, South Korea, 2006, pp. 43–54.

[13] Y.-K. Huang, Z.-W. Chen, and C. Lee, "Continuous K-nearest neighbor query over moving objects in road networks," in *Proc. 10th Int. Conf. Web-Age Inf. Manage.*, Suzhou, China, 2009, pp. 27–38.

[14] M. Hasan, M. A. Cheema, W. Qu, and X. Lin, "Efficient algorithms to monitor continuous constrained k nearest neighbor queries," in *Proc. 15th Int. Conf. Database Syst. Adv. Appl. (DASFAA)*, Tsukuba, Japan, 2010, pp. 233–249.

[15] Y. Gotoh, "A simple routing method for reverse k-nearest neighbor queries in spatial networks," in *Proc. 17th Int. Conf. Netw.-Based Inf. Syst. (NBiS)*, Salerno, Italy, Sep. 2014, pp. 615–620.

[16] H. Zhang, C. Reardon, and L. E. Parker, "Real-time multiple human perception with color-depth cameras on a mobile robot," *IEEE Trans. Cybern.*, vol. 43, no. 5, pp. 1429–1441, Oct. 2013.

[17] D. Delling and R. F. Werneck, "Customizable point-of-interest queries in road networks," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 3, pp. 686–698, Mar. 2015.

[18] X. Lian and L. Chen, "Trip planner over probabilistic time-dependent road networks," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 8, pp. 2058–2071, Aug. 2014.

[19] S. M. Yuen, Y. Tao, X. Xiao, J. Pei, and D. Zhang, "Superseding nearest neighbor search on uncertain spatial databases," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 7, pp. 1041–1055, Jul. 2010.

[20] J. Bao, C.-Y. Chow, M. F. Mokbel, and W.-S. Ku, "Efficient evaluation of k-range nearest neighbor queries in road networks," in *Proc. 11th Int. Conf. Mobile Data Manage.*, Kansas City, MO, USA, May 2010, pp. 115–124.

[21] T. Brinkhoff, "A framework for generating network-based moving objects," *GeoInformatica*, vol. 6, no. 2, pp. 153–180, Jun. 2002.

[22] P. Fan, G. Li, and L. Yuan, "Continuous K-nearest neighbor processing based on speed and direction of moving objects in a road network," *Telecommun. Syst.*, vol. 55, no. 3, pp. 403–419, Mar. 2014.

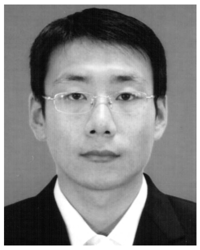
[23] K.-T. Yang and G.-M. Chiu, "Monitoring continuous all κ-nearest neighbor query in mobile network environments," *Pervas. Mobile Comput.*, vol. 39, pp. 231–248, Aug. 2017.

[24] A. Eldawy *et al.*, "Sphinx: Distributed execution of interactive SQL queries on big spatial data," in *Proc. 23rd SIGSPATIAL Int. Conf. Adv. Geograph. Inf. Syst.*, Bellevue, WA, USA, 2015, pp. 78–81.

[25] A. Eldawy and M. F. Mokbel, "The era of big spatial data: A survey," *Found. Trends Databases*, vol. 6, nos. 3–4, pp. 163–273, Dec. 2016.
[26] J. J. Levandoski, M. Sarwat, A. Eldawy, and M. F. Mokbel, "LARS: A location-aware recommender system," in *Proc. 28th Int. Conf. Data Eng. (ICDE)*, Washington, DC, USA, Apr. 2012, pp. 450–461.
[27] L. Guo, J. Shao, H. H. Aung, and K.-L. Tan, "Efficient continuous top-k spatial keyword queries on road networks," *Geoinformatica*, vol. 19, no. 1, pp. 29–60, Jan. 2015.
[28] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, pp. 269–271,1959.
[29] B. Liao, L. Hou U, M. L. Yiu, and Z. Gong, "Beyond millisecond latency kNN search on commodity machine," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 10, pp. 2618–2631, Oct. 2015.
[30] B. Shen *et al.*, "V-tree: Efficient kNN search on moving objects with road-network constraints," in *Proc. 33rd Int. Conf. Data Eng. (ICDE)*, San Diego. CA, USA, Apr. 2017, pp. 609–620.
[31] K. Bok, Y. Park, and J. Yoo, "An efficient continuous k-nearest neighbor query processing scheme for multimedia data sharing and transmission in location based services," *Multimedia Tools Appl.*, 2018, doi: 10.1007/s11042-018-6433-3.
[32] J. Bao, B. Wang, X. Yang, and H. Zhu, "Nearest neighbor query in road networks," (in Chinese), *Ruan Jian Xue Bao/J. Softw.*, vol. 29, no. 3, pp. 642–662, Mar. 2018.

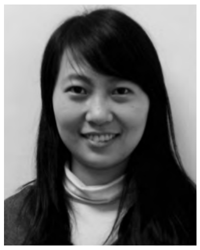
HONGJUN LI received the B.S. degree in computer science from Sichuan Normal University in 2000 and the M.S. and Ph.D. degrees from the School of Computer Science, Sichuan University, in 2007 and 2011, respectively. He is currently a Lecturer with the School of Information Science and Technology, Chengdu University of Technology, Chengdu, China. His research interests include spatial data mining, spatial query processing, and social computing.

BIAO CAI received the B.S. degree in computer application technology from Sichuan Normal University in 2003 and the M.S. degree in computer application technology from China West Normal University, China, in 2006, and the Ph.D. degree from the School of Computer Science, Sichuan University, in 2009. He has been with the Chengdu University of Technology since 2009, where he is currently an Associate Professor with the Digital Media Department. His research interests include social media mining, artificial intelligence, and complex networks.



SHAOJIE QIAO received the B.S. and Ph.D. degrees from Sichuan University, Chengdu, China, in 2004 and 2009, respectively. From 2007 to 2008, he was a Visiting Scholar with the School of Computing, National University of Singapore. He is currently a Professor with the School of Cybersecurity, Chengdu University of Information Technology, Chengdu, China. He has led several research projects in the areas of database and data mining. He has authored more than 40 high-quality papers and co-authored more than 90 papers. His research interests include complex networks and trajectory data mining.

QING WANG received the B.S. degree in computer science from Zhengzhou University in 2009 and the M.S. degree in computer science from Xidian University in 2013. She is currently pursuing the Ph.D. degree with the School of Computer Science, Florida International University. Her research interests include interactive recommender systems, multi-armed bandit, and large-scale data mining.



YAN WANG received the Ph.D. degree from the School of Computer Science, Fudan University, China, in 2015, and the B.S. and M.S. degrees from the School of Computer Science, Nanchang University, China, in 2003 and 2006, respectively. She is currently an Associate Professor with the School of Software, East China Jiaotong University, Nanchang, China. Her research interests include erasure codes and distributed storage systems.