Available online at www.ijpe-online.com

International Journal of Performability Engineering

Vol. 13, No. 4, July 2017, pp. 501-510
DOI: 10.23940/ijpe.17.04.p17.501510

*k*NN Research based on Multi-Source Query Points on Road Networks

Jia Liu[a,b], Wei Chen[a,b], Lin Zhao[a], Junfeng Zhou[a], Ziyang Chen[c,a,*]

[a]*School of Information Science and Engineering, YanShan University, Qinhuangdao and 066004, P.R.China*
[b]*Department of Information Engineering, Hebei University of Environmental Engineering, Qinhuangdao and 066102, P.R.China*
[c]*School of Information and Management, Shanghai Lixin University of Accounting and Finance, Shanghai and 201620, P.R.China*

Abstract

Given a query point set and an object point set, a multi-source query of *k* nearest neighbors (MQ-*k*NN) returns the query set for its *k* closest objects. However, most existing nearest neighbor query algorithms are based on a single-source query point (SQ-*k*NN), and the query point is often the user's location. However, in some cases, a query can be a point set. For example, a user wants to choose a house from the existing idle houses (query points) and hopes that its surrounding facilities (object points) are best. For this kind of application, we study the problem of MQ-*k*NN on road networks and try to solve MQ-*k*NN query problems. A basic algorithm based on Dijkstra algorithm is proposed as an original algorithm by calculating SQ-*k*NN repeatedly. Then, two improved algorithms are proposed by taking all query points as a whole and adopting the effective pruning strategy. Comprehensive experiments on five road network datasets clearly demonstrate the efficiency of this method.

Keywords: Road Networks; single-source query of *k*NN; multi-source query of *k*NN

(Submitted on February 13, 2017; Revised on May 5, 2017; Accepted on June 25, 2017)

1. Introduction

K nearest neighbors (*k*NN) query on road networks is a fundamental problem in location based service (LBS). Given a query location and a set of static objects (e.g., supermarkets or stations) on the road networks, the user can find *k* nearest objects to the query location [1,2]. Many *k*NN-based methods were proposed successively, such as the aggregate nearest neighbor query (ANN) [3,4,5], the merged aggregate nearest neighbor query (MANN) [6], the spatial *k*NN query methods combining with keywords [7,8,9], and the nearest neighbor of moving objects [10,11,12]. Most of these methods are the traditional *k*NN-based query methods, in which the *k* closest targets to a query point are found by taking the query point as the center. Additionally, in these methods, a query point is usually the location of the user or the starting point of a path, which is the shortest Euclidean distance of road networks. However, in many important cases, the query is not a point but a set of discrete points. The following lists two examples that illustrate these cases.

Example 1. An express company will hold annual sweepstakes to thank some customers, so we need to choose *k* customers who are the nearest neighbors to *n* agents. Here, the query set is composed of *n* delivery agents, and the target point set is composed of all the customers.

Example 2. An investor wants to open a new children's clothing store, and he hopes that the store must be the nearest neighbor to some brand children's clothing stores or women's clothing stores (under the assumption that women are typically

* Corresponding author. Tel.: +86-13731342999.
*E-mail address:*zychen@ysu.edu.cn.

the ones buying clothes for their children). Here, the query set is existing children's clothing stores or women's clothing stores, and the target point set includes all the sites that can be rented.

In order to deal with the similar problems, we propose three algorithms; one is a basic algorithm which processes the query points one by one, and the other two are the improved algorithms which consider the search of multiple query points as a whole and extend nodes simultaneously. We conduct an experimental study on five datasets to evaluate the efficiency of the proposed algorithms on real road networks. The results demonstrate that our algorithms are feasible and efficient on the question of the MQ-*k*NN.

2. Preliminaries and Related Work

2.1. Preliminaries

A road network can be modeled as a weighted graph $G=(V,E,W)$, where V is the set of vertices, E is the set of edges, and W is the set of weights (the network distance or the traveling time) that are associated with each edge. A vertex $n_1 \in V$ represents a road intersection or an end-point in the road network, an edge $(n_1,n_2) \in E$ represents a road segment which connects two road nodes n_1 and n_2, and a non-negative weight $w(n_1,n_2) \in W$ associated with each edge (n_1,n_2) represents the length that can be the network distance or the time. We assume that all facility instances (query points and object points) lie on the road.

A query $Q=\{q_1,q_2,\cdots,q_m\}$ represents a point set of locations. An objects set $O=\{o_1,o_2,\cdots,o_n\}$ represents a series of interest points, such as gas stations, supermarkets, shopping malls, etc. We use $d_E(q,o)$ to denote the Euclidean distance between q and o, which is the linear distance between q and o. We also use $d_N(q,o)$ to denote the network distance between q and o, which is the sum of the edge weights along the least costly path from q to o. Given a query point q lying on the edge (n_1,n_2), the following equation shows how to derive network distance between a query point q and an object point o.

(1) $d_N(q,o)=\min(d_N(q,n_1)+w(n_1,o), d_N(q,n_2)+w(n_2,o))$ if o does not lie on the edge (n_1, n_2)
(2) $d_N(q,o)=w(q,o)$ if both q and o lie on the same edge (n_1, n_2)

Definition 1. SQ-*k*NN. Given a query point q and an object set $O=\{o_1,o_2,\cdots,o_n\}$, a SQ-*k*NN retrieves $k(\geq 1)$ object(s) $\{o_1,o_2,\cdots,o_k\}=D \subset O$, if $d_N(q,o_1) \leq d_N(q,o_2) \leq \cdots \leq d_N(q,o_k)$, so that $\forall o \in D$, $o' \in O \backslash D$, $d_N(q,o) \leq d_N(q,o')$.

Definition2. MQ-*k*NN. Given a query set $Q=\{q_1,q_2,\cdots,q_m\}$ and an object set $O=\{o_1,o_2,\cdots,o_n\}$, a MQ-*k*NN retrieves an object set D which is composed of the k tuples (q_i,o_j,d_N), so that $\forall$ $o \in D$, $o' \in O \backslash D$, there exists $q \in Q$, $d_N(q,o) \leq d_N(q,o') \wedge d_N(q,o) \leq d_N(q',o)$, where $q' \in Q \backslash \{q\}$.

2.2. Related Work

We focus on MQ-*k*NN based on road networks, which is an extension of the SQ-*k*NN query instead of an overlay of the SQ-*k*NN. Two algorithms are widely used to process SQ-*k*NN in spatial networks [2]. One is incremental network expansion (INE) and the other is incremental Euclidean restriction (IER). INE extends the Dijkstra algorithm [13] by expanding neighbor vertices from the query location until *k*NN answers have been found, so it must visit all the nodes of G within the k values. By applying the multi-step *k*NN methodology and spatial pruning techniques, IER improves INE. IER and INE are 'blind' algorithms, since they can neither capture the global distance from objects to the query location nor efficiently prune unnecessary objects. Like IER, the Closest-Pairs Euclidean Restriction (CPER) algorithm and the Closest-Pairs Network Expansion (CPNE) algorithm apply the muli-step *k*NN methodology in literature [2]. CPER performs an incremental closest-pairs query on the R-tree of query set (S) and object set (T) and retrieves the Euclidean closest pairs. Then, the network distance $d_N(s,t)$ provides an upper bound d_{Emax} for all candidate pairs in the Euclidean space. Subsequent candidate pairs are retrieved incrementally, continuously updating the result and d_{Emax}, until no candidate pairs can be found within the limits of d_{Emax}. In CPNE, the only option is to use as sources all the data points of one dataset (the one with the smallest cardinality of S and T). The CPNE method takes d_{Emax} of k Closest-Pairs as maximum network distance at present. If the next Closest-Pairs queried satisfies the condition $d_N(s_i,t) < d_{Emax}$, then the set of k Closest-Pairs and d_{Nmax} are updated. The disadvantage of CPER is that it has fixed Closest-Pairs based on Euclidean distance in advance—that is, $<s_i,t_j>$—whereas the nearest point in s_i may be t_k based on Road Network distance and not t_j. The disadvantages of CPNE are that it does not restrict all the query points in the query set and that such queries have great blindness and may lead to many useless nodes being accessed.

Existing methods [3-6] have similarly discussed multiple query points, but their queries return an object that minimizes the aggregate distance for a given query point set. However, our problem is how to efficiently find *k*NN of all the query points. G-Tree [14] is a balanced search tree index. It is constructed by recursively partitioning the road network into sub-networks and each G-Tree node corresponds to a sub-network. It has a high maintenance cost.

In our algorithms, we calculate the distance of the road network and implement the effective pruning strategy by the theory of A* [15]. Unlike the CPER method, our algorithm does not fix Closest-Pairs based on Euclidean distance in advance. Unlike the CPNE method, our algorithm extends the query point in the query set at the same time, and searches for the current nearest point to extend, so that both the number of visited nodes is reduced and the global optimality is maintained.

3. The Basic Algorithm

Given a query set $Q=\{q_1,q_2,\cdots,q_m\}$ and an object set $O=\{o_1,o_2,\cdots,o_n\}$, the direct processing method separates the query point set to deal with the query point one by one and find *k*NN of all the query points. In the query process, we use it to update the value of the candidate set when one of the *k*NN is found and search the next one by taking the *k*th value of the candidate set as the upper bound. Algorithm 1 shows the calculating process in detail.

Algorithm 1: S-MQ-*k*NN- Dijkstra /$Q=\{q_1,q_2,\cdots,q_n\}$/
1 $C\leftarrow\{c_1,c_2,\cdots,c_k\}$ Initialize $c_i(o_i,d)\leftarrow(o_i,\infty)$
2 For each $q_i\in Q$ do
3 S-MQ-*k*NN- Dijkstra(q_i)
4 return candidate
5 Function S-MQ-*k*NN- Dijkstra(q)
6 Initialize Priority queue *PQ*
7 Insert(q,0) into *PQ*
8 count=0
9 while *PQ* is not empty do
10 PQEntry(n,d)=DequeueHead(*PQ*)
11 if($d>=c_k.d$) then return
12 if n is not expanded
13 if($n\in O$)
14 Update Candidate set
15 count++
16 if(count==k)then return
17 For each adjacent vertice n_a of n
18 if n_a is not expanded
19 $d\leftarrow d+$weightof(n,n_a)
20 if($d<c_k.d$)
21 Insert(n_a,d) into *PQ*
22 n is expanded
23 return

As shown by Algorithm 1, the baseline algorithm for MQ-*k*NN, namely S-MQ-*k*NN- Dijkstra, uses a heap *C* to buffer *k* candidates, where $c_i\in C$ is initialized to two variables, $c_i.o$ represents an object, and $c_i.d$ is the road network distance between q to $c_i.o$. Algorithm computes *k*NN of each query point q and updates the candidate set *C* by calling Function S-MQ-*k*NN-Dijkstra (q) in line 5. In Function, Lines 6-7 initialize the priority queue *Q*, which stores the nodes of the road network in the expansion and their distance to the query point on the road network while a counter is also initialized. When *Q* is not empty, the point where d_N is shortest is processed first (lines 8-9). If $c.d\geq c_k.d$, the algorithm terminates (line 10). If the node is not expanded and it belongs to *O*, the *C* is updated and the counter is increased by 1, and algorithm is terminated if count=k (lines 11-15). Lines 16-21 process adjacency nodes that are not visited if their distance is less than $c_k.d$.

Example. Letting k=3, consider the query set $Q=\{q_1,q_2,q_3\}$ and the object set $O=\{o_1,o_2,\cdots,o_6\}$ on the road network *G* in Figure 1, and find *k*NN of *Q* from *O*. Algorithm 1 first processes q_1. As the distance of q_1's adjacency nodes to q_1 is less than $c_k.d=\infty$, q_1's adjacency nodes, i.e., o_1, v_1 and v_2, are inserted into *Q*. After that, $d_N(q_1,o_1)=d_N(q_1,v_1)=2<d_N(q_1,v_2)=4$, so o_1 or

v_1 is first removed from Q and marked as visited. Because v_1 has no adjacency nodes, and o_1 belongs to O, we find the first object o_1 and use it to update the candidate set C, then $C=\{<o_1,2>,<o_i,\infty>,<o_j,\infty>\}$. Next, v_2's adjacency node v_3 and o_1's adjacency nodes (v_4,v_{11}) are processed. Because they are not the object points, they are removed from Q and their adjacency nodes are inserted into Q. The following processing is similar. After processing q_1, the candidate set is updated to $C=\{<o_1,2>,<o_2,7>,<o_5,10>\}$. The next processed query point is q_2, and if we find that the nearest object of q_2 is o_3 and $d_N(q_2,o_3)=4$, then $C=\{<o_1,2>,<o_3,4>,<o_2,7>\}$. In the same way, q_3 is processed. In q_3's adjacency nodes, o_6 is the nearest object. As $d_N(q_3,o_6)=5$, the candidate set is updated to $C=\{<o_1,2>,<o_3,4>,<o_6,5>\}$. After processing all query points of Q, the candidate set $C=\{<o_1,2>,<o_3,4>,<o_6,5>\}$ is the final result.

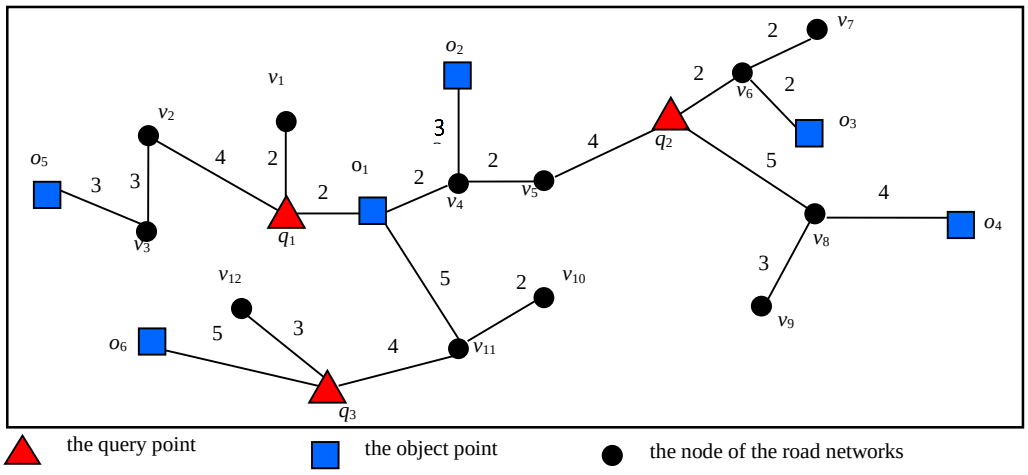

Figure 1. An example of a MQ-kNN

Algorithm 1 has to visit all nodes of V as long as its distance is less than the maximum distance of the candidate set. Especially in the worst cases, all the nodes on road networks are inserted into Q as the adjacency point. Here, the time complexity of each query point is $O(|E|\log|V|)$ and the time complexity of m query point is $O(m|E|\log|V|)$, in which $|V|$ is the number of nodes and $|E|$ is the number of sides in the road network.

4. The Optimized Algorithm

Algorithm 1 uses Dijkstra algorithm idea to process all the query points one by one. Although the method is simple, it does not reflect the integrity of multi-source. If we want to maintain the global optimality, we should regard the query set as a whole and extend query points simultaneously. Using this method, we can effectively reduce the number of node accesses.

4.1. M-MQ-kNN-Dijkstra Algorithm

M-MQ-kNN-Dijkstra algorithm searches all the query points at the same time and selects the current node nearest to Q to extend firstly. When an object is found, it will be inserted into C until all the kNN are found. Algorithm 2 shows the calculating process in detail.

As is shown by Algorithm 2, lines 1-2 initialize the candidate set C, the priority queue Q, and a counter. The structure of the priority queue is (n,q_i,d_N), where n is a network node, q_i is a query point, and d_N is the network distance. All the query points are inserted into the priority queue (lines 3-4). The node is extended in the queue if it belongs to the object set O and added to C, and the algorithm terminates when all k objects are found (lines 5-11). Lines 12-16 process all the adjacent points of n if the adjacent point has not been visited. Then, the adjacent point is either inserted into the priority queue or marked as already visited. The algorithm terminates if the queue is empty.

Algorithm 2: M-MQ-kNN-Dijkstra /$Q=\{q_1,q_2,\cdots,q_n\}$/
1 $C\leftarrow\{c_1,c_2,\cdots,c_k\}$ Initialize $c_i(o_i,d)\leftarrow(o_i,\infty)$
2 Initialize Priority queue PQ
3 count=0
4 For each $q_i\in Q$ do
5 Insert(q_i,q_i,0) into PQ
6 while PQ is not empty do

```
7   PQEntry(n,d)=DequeueHead(PQ)
8   If n is not expanded
9   if(n∈O)
10      Update Candidate set
11      count++
12      if(count==k) then return
13   For each adjacent vertex n_a of n
14      if n_a is not expanded
15         d←d+weightof(n, n_a)
16         Insert(n_a,d) into PQ
17   n is expanded
18  return
```

According to the above example, algorithm 2 first inserts all the query points and the distance to themselves $\{(q_1,q_1,0),(q_2,q_2,0),(q_3,q_3,0)\}$ into Q, which are visited from Q in turn, and inserts their adjacent nodes$\{(v_1,q_1,2), (o_1,q_1,2), (v_2,q_1,4)\},\{(v_6,q_2,2), (v_5,q_2,4), (v_8,q_2,5)\}, \{(v_{12},q_3,3), (v_{11},q_3,4), (o_6,q_3,5)\}$ into Q. As $d_N(q_1,v_1)=d_N(q_1,o_1)=d_N(q_2,v_6)=2$ and the distance is at its shortest, v_1, o_1 and v_6 will be visited first. v_1 has no adjacent nodes, so it is removed from Q. Because $o_1\in O$, it is added to the candidate set $C=\{(o_1,q_1,2),(o_i,q_j,\infty),(o_i,q_j,\infty)\}$. In o_1's adjacent nodes, the node $(v_4,q_1,4)$ is not extended, but instead inserted into Q with the two adjacent nodes $(o_3,q_2,4),(v_7,q_2,4)$ of v_6 together. Next, the node $(v_{12},q_3,3)$ is visited. As $(v_{12},q_3,3)$ has no adjacent nodes, it is removed from Q. After $(v_2,q_1,4),(v_5,q_2,4),(v_{11},q_3,4),(v_4,q_1,4),(o_3,q_2,4)$ and $(v_7,q_2,4)$ are visited, their respective adjacent points are inserted into Q, and the candidate set is updated $C=\{(o_1,q_1,2),(o_3,q_2,4),(o_i,q_j,\infty)\}$. The following process is similar. In the end, the candidate set is updated to $C=(o_1,q_1,2),(o_3,q_2,4),(o_6,q_3,5)\}$.

Algorithm 2 carries out Dijkstra extension only once, so its time complexity is $O(|E|\log|V|)$. This is superior to the time complexity of algorithm 1, which is $O(m|E|\log|V|)$.

4.2. M-MQ-kNN-IER Algorithm

Algorithm 2 improves algorithm 1 and reduces the number of nodes to access, but it still carries out extension based only on the d_N. Although this method extends query points simultaneously, no pruning operations are set, and there are still some useless nodes that have been accessed. In order to solve this problem, Algorithm 3 takes Euclidean distance d_E as the lower bound value of the distance for pruning operation and constructs a R-Tree index for all the query objects. According to R-Tree index, all query points search the Euclidean nearest neighbor and then calculate the corresponding road networks distance d_N by A* algorithm. When distance from the query point to its Euclidean nearest neighbor $d_E \geq d_N$ (the kth road networks distance in the candidate set), the algorithm terminates.

Algorithm 3: M-MQ-*k*NN-IER / $Q=\{q_1,q_2,\cdots,q_n\}$/

```
1  C←{c_1,c_2,⋯,c_k}  Initialize c_i(o_i,d)←(o_i,∞)
2  Initialize Priority queue PQ
3  For each q_i∈Q do
4    For each entry e in root(R)
5      Enqueue(PQ,e, d_E(q_i,e))
6  while PQ is not empty do
7    PQEntry(e,q_i,d_E)=DequeueHead(PQ)
8      if(d_E=c_k.d) then  return Candidate
9      if e is a leafnode
10       d=A*(e,q_i)
11       if(d<c_k.d)
12         Update Candidate set
13     else
14       For each child entry e' of e
15         If (d_E(q,e')<d)
16           Insert(e',d_E(q,e')) into PQ
17 return
```

In Algorithm 3, the structure of the priority queue is (e,q_i,d_E), where e is the node entity of R-Tree, q_i is the query point, d_E is Euclidean distance. According to the structure of the priority queue, lines 3-5 insert relevant information into the queue. If d_E (the distance from the query point to rectangular entity)≥d_N (the maximum network distance of the candidate set), then the algorithm terminates (line 8).If e is a leaf node and d_N≤c_k.d, then the candidate set is updated (lines 9-12).On the contrary, if e is not a leaf node and d_E from the query to its child node is less than c_k.d, then e is inserted into Q (lines 14-16).

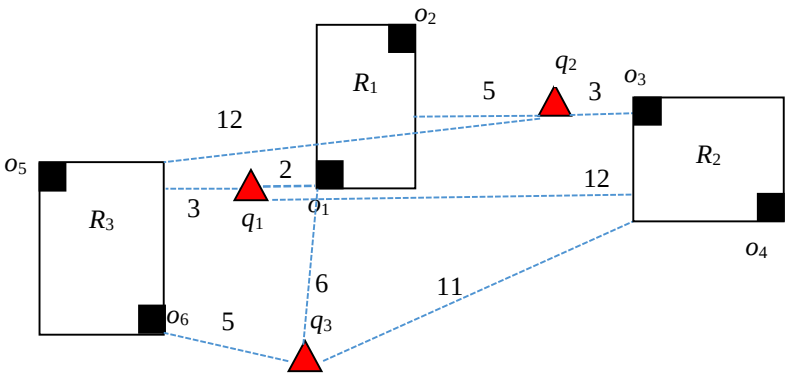

Figure 2. The distance from the query to rectangular frame

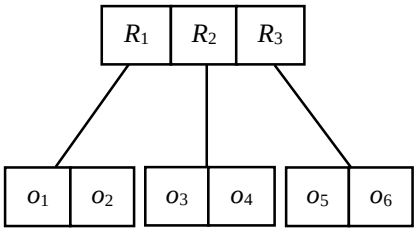

Figure 3. The R-Tree of query objects

As is shown in Figure 3, algorithm 3 searches the R-tree of three query points at the same time, and the node where its d_E is the shortest is extended first. Figure 2 shows the Euclidean distance d_E between the three query points to the R-tree root node entity. First, (R_1,q_1,2) (R_2,q_1,12) (R_3,q_1,3),(R_1,q_2,5) (R_2,q_2,3) (R_3,q_2,12) and (R_1,q_3,6) (R_2,q_3,11) (R_3,q_3,5) are inserted into Q. Then, the node (R_1,q_1,2) is visited, and its child node entities (o_1,q_1,2) and (o_2,q_1,6) are inserted into Q. Now, the node (o_1, q_1,2) is visited, so it is the first nearest neighbor of all the query points. As d_N(o_1,q_1)=2, the candidate set is updated with {(o_1,q_1,2), (o_i,q_j,∞), (o_i,q_j,∞)}. Then, the node (R_3,q_1,3) is visited, and its child node entities (o_5,q_1,6) and (o_6,q_1,5) are inserted into Q. Next, the node (R_2,q_2,3) is visited, and its child node entities (o_3,q_2,3) and (o_4,q_2,8) are inserted into Q. Now, the node (o_3,q_2,3) in Q is visited. As d_N (o_3,q_2)=4, the candidate set C is updated to{(o_1,q_1,2), (o_3,q_2,4), (o_i,q_j,∞)}. Then, the child nodes (o_1,q_2,7) and (o_2,q_2,5) of the node (R_1,q_2,5) are inserted into Q. Similarly, the child nodes (o_5,q_3,9) and (o_6,q_3,5) of the node (R_3,q_3,5) are inserted into Q. Now, all the unvisited object nodes in Q whose d_E are the shortest are (o_6,q_1,5), (o_2,q_2,5), and (o_6,q_3,5). Their corresponding network distances are 16, 9 and 5 respectively. As d_N(o_6,q_3)=5, the candidate set C is updated to {(o_1,q_1,2), (o_3,q_2,4), (o_6,q_3,5)}. Now, the d_E of all the unvisited node entities in the Q are larger than the maximum d_N of C, so the algorithm terminates, and {(o_1,q_1,2), (o_3,q_2,4), (o_6,q_3,5) } in C is the final result.

In the processing of extended nodes, the algorithm 3 considers the global optimality, reduces the access to useless nodes, overcomes the blindness of the Dijkstra algorithm, and increase the efficiency of multi-source query of kNN.

5. Experiment

5.1. Experimental Setup

All experiments are conducted on a PC with AMD Athon™ II X2 270 3.4GHZ CPU, 2GB memory, and Windows 7. The three algorithms are implemented in VC++ with the important data structures (R-tree).

We evaluate the performances of the three algorithms on both real and synthetic datasets. The five datasets are deployed in the experiment as shown by Table 1.

Table 1. Information of 5 road network data sets in experiments

DataSet	number of nodes	number of edges
California (CA)	21047	21692
Oldenburg (OL)	6105	7034
San Joaquin (TG)	18262	23873
North America(NA)	175813	179178
San Francisco (SF)	174955	223000

Some of these datasets, such as San Francisco and North America, are large, while others, such as Oldenburg, are small. Through the horizontal comparison of the 5 datasets, we can see the performance of the algorithm in different scale data sets, which can verify the scalability of the algorithm.

The parameters in the experiment are shown in Table 2.

Table 2. Parameter setting in experiment

parameter	default	min	max
k	5	10	25
s	1000	100	10000
m	10	10	50

k is the number of nearest neighbors to be returned. *s* is denoted as $|V|/|O|$, where *V* is the set of nodes of the given road network *G*. *O* is the set of objects in *G*. *m* is the number of query points.

We make comparisons based on the following metrics: (1) running time and (2) the number of visited nodes to process each query. Additionally, we test the performances of the three algorithms by observing changing parameters (*k*,*m*,*s*).

5.2. Performance Comparison

Table 3 compares the running times of the three algorithms in different datasets. The number of parentheses is a multiple relationship between the execution time of each algorithm and the execution time of the M-MQ-*k*NN-IER algorithm.

Relatively large data sets are SF and NA, and the smallest data set is OL. In Table 3, we do not observe similarities in execution times among the five datasets, as California's execution time is lower than those of the other four datasets. This is because query time is independent of the size of the road network data set as long as the density of the query object s is fixed. The algorithm M-MQ-*k*NN-IER is the fastest, regardless of which data set is computed.

Table 3. The execution time of tree algorithms (ms)

Algorithms	(CA)	(OL)	(TG)	(NA)	(SF)
M-SQ-*k*NN-Dijkstra	1.3978	3.2511	3.7924	2.2417	3.8314
	(6.053)	(4.48)	(9.88)	(5.94)	(11.59)
M-MQ-*k*NN-Dijkstra	0.6006	0.7488	1.2792	0.6989	1.2215
	(2.60)	(1.10)	(3.33)	(1.85)	(3.69)
M-MQ-*k*NN-IER	0.2309	0.677	0.3837	0.3775	0.3307
	(1.00)	(1.00)	(1.00)	(1.00)	(1.00)

Similarly, in Table 4, we can see that, when the parameters are constant, the number of extended network nodes is not directly related to the size of the data set, and the algorithm M-MQ-*k*NN-IER is optimal in the number of road network nodes.

5.3. Influence of Parameter Values

Next, we illustrate the experimental situation of each algorithm in the case of different parameters. The experiment uses the San Francisco road network datasets. In each experiment, one parameter is changed while the rest of the parameters are fixed.

Table 4. The number of extended nodes in 5 different road network data sets

Algorithms	(CA)	(OL)	(TG)	(NA)	(SF)
M-SQ-*k*NN-Dijkstra	14595	24121	19256	14347	15820
	(10.09)	(8.10)	(16.38)	(10.60)	(22.28)
M-MQ-*k*NN-Dijkstra	4632	4727	5594	4534	4923
	(3.20)	(1.65)	(4.76)	(3.35)	(6.93)
M-MQ-*k*NN-IER	1446	2873	1175	1354	710
	(1.00)	(1.00)	(1.00)	(1.00)	(1.00)

(1) Impacts of *k*'s value

It can be seen from Figure 4 that the execution time of each algorithm increases with an increasing *k* value, but the algorithm M-MQ-*k*NN-IER is better than the other two algorithms. We can also see that the number of extended nodes in each algorithm increases with an increasing *k* value. This is because as more results are returned, more algorithm is called repeatedly and more nodes are extended.

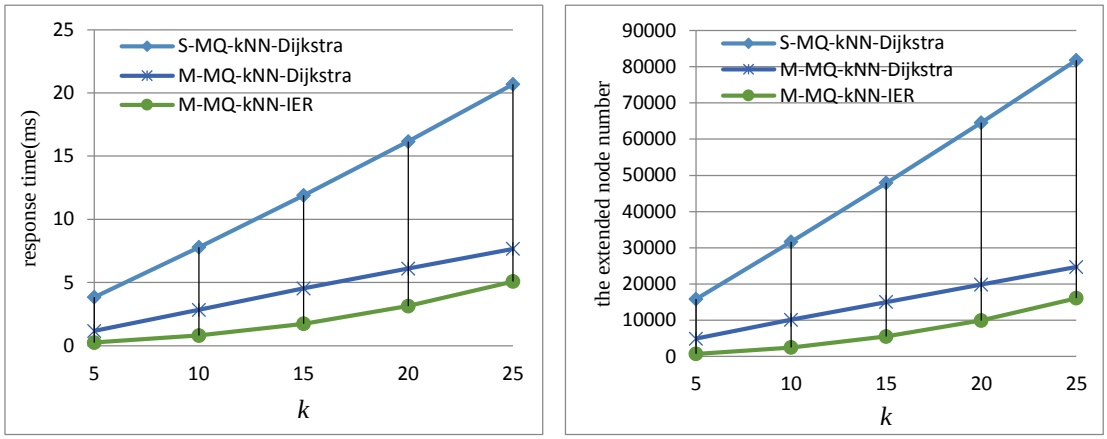

Figure 4. The response time and number of extended nodes of three algorithms in different *k*

(2) Impacts of *s*'s value

In this experiment, we set the range of the *s* value to 3 levels: 100, 1000 and 10000. As shown in Figure 5, we see that the execution time of each algorithm increases with an increase of *s*. When the density is 100, the algorithm M-MQ-*k*NN-Dijkstra is more efficient than the other algorithms. When the density is small, the query object is very dense. We know that the Dijkstra algorithm is blind, but when the query object is very close and very dense, the network nodes to be extended will be relatively small. It does not need to carry out multiple A* verification and search the R-tree, so it is relatively simple and efficient. However, when the *s* value increases gradually, its execution time is not as efficient as that of the other two algorithms. With an increasing *s* value, the number of extended nodes of all algorithms also increases. However, M-MQ-*k*NN-IER remains the most optimal.

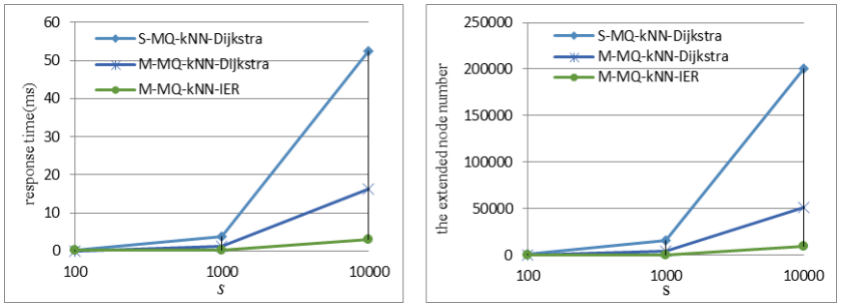

Figure 5. The response time and number of extended nodes of three algorithms in different *s*

(3) Impacts of *m*'s value

We set the value of *m* to 10, 20, 30, 40, and 50. As shown in Figure 6, we can see that the execution time of S-MQ-*k*NN-Dijkstra algorithm significantly increases with an increase of *m*, while the other two algorithms are only slightly increased (can be negligible). This is because the algorithm S-MQ-*k*NN-Dijkstra expands the query point one by one, so more queries leads to more iterations. Whether the number of queries is greater or lesser, the other two algorithms are not as affected.

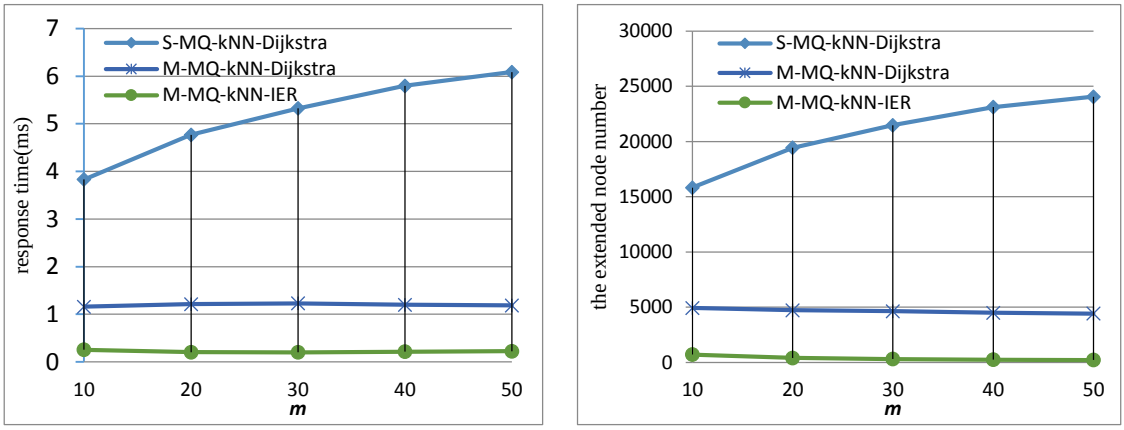

Figure 6. The response time and number of extended nodes of three algorithms in different *m*

6. Conclusions

In this paper, a basic algorithm and two improved algorithms are proposed to solve the MQ-*k*NN problem in road networks. The nodes are extended based on Dijkstra algorithm and IER algorithm respectively, and the operation efficiency is improved by applying exquisite pruning strategy. In the experiments, the two improved algorithms are proved to be instance optimal in terms of the response time.

In the future, we will study the influences between *k*NN to find the optimal neighbors and form the optimal areas with *k*NN of high influences. This work will be combined with the keyword query and distance query in order to find the optimal balance and provide a better choice for inquirers.

Acknowledgements

This work is supported by the National Natural Science Foundation of China under Grants No. 61272124 and No. 61572421, the Project of Youth Foundation for science and technology in Hebei higher education institutions under Grant No. QN2017055, and the Hebei University of Environmental Engineering Foundation under Grant No. ZRZD201602. The authors also gratefully acknowledge the helpful comments and suggestions of reviewers, who have improved the presentation.

References

1. C. S. Jensen, J. Kolárvr, T. B. Pedersen, and I. Timko, "Nearest neighbor queries in road networks," in *Proceedings of the 11th ACM international symposium on Advances in geographic information systems* (*GIS*), pp.1–8, 2003
2. D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, "Query processing in spatial network databases," *in Proceedings of International Conference on Very Large Data Bases* (*VLDB*), pp.802–813, 2003
3. M. Yiu, N. Mamoulis, and D. Papadias, "Aggregate nearest neighbor queries in road networks," *IEEE Transactions on Knowledge and Data Engineering archive* (*TKDE*), vol. 17, no. 6, pp.820–833, 2005
4. D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis, "Group nearest neighbor queries," in *Proceedings of 20th International Conference on Data Engineering* (*ICDE*), pp.301–312, 2004
5. D. Yan, Z. Zhao, and W. Ng, "Efficient algorithms for finding optimal meeting point on road networks," in *Proceedings of the VLDB Endowment* (*PVLDB*), vol. 4, no. 11, pp.968–979, 2011
6. W. Sun, C. Chen, B. Zheng, C. Chen, L. Zhu, W. Liu, "Merged Aggregate Nearest Neighbor Query," in *Proceedings of the 22nd ACM Conference of Information and Knowledge Management* (*CIKM*), pp.2243–2248, 2013
7. J. B. Rocha-Junior and K. Nørvåg, "Top-k spatial keyword queries on road networks," in *Proceedings of International Conference on Extending Database Technology* (*EDBT*). pp.168–179, 2012

8. G. Li, J. Feng, and J. Xu, "Desks: Direction-aware spatial keyword search," in *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering* (*ICDE*), pp.474–485, 2012
9. R. Zhong, J. Fan, G. Li, K.-L.Tan, and L. Zhou, "Location-aware instant search," in *Proceedings of the 21st ACM Conference of Information and Knowledge Management* (*CIKM*), pp.385–394, 2012
10. C. Deng, Y. Hu, T. Zhou, B. Wang, J. Suo, "A method of range nearest neighbor query of moving objects with uncertain velocity in road network," *Journal of Yanshan University*, vol. 36, no. 6, pp.526–533, 2012 (in Chinese)
11. G. Li, Y. Li, J. Li, L. Shu, F.Yang, "Continuous reverse k nearest neighbor monitoring on moving objects in road networks," *Information Systems*, pp.860–883, 2010
12. B. Lu, N. Liu, "Snapshot K neighbor query processing on moving objects in road networks," *Journal of Computer Applications*, vol.31, no. 11, pp.3078–3083,2011
13. E. W. Dijkstra, "A note on two problems in connection with graphs," *Numerisce Mathematik*, vol.1, no.1, pp.269–271,1959
14. R. Zhong, G. Li, K. Tan, and L. Zhou, "G-Tree: An Efficient Index for KNN Search on Road Networks," in *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*(*CIKM*). pp.39–48, 2013
15. K. Deng, X. Zhou, H. T. Shen, S. W. Sadiq, and X. Li, "Instance optimal query processing in spatial networks," *VLDB Journal*, vol. 18, no. 3, pp.675–693, 2009