Journal of Computational Information Systems 11: 2 (2015) 759–767
Available at http://www.Jofcis.com

Search Continuous Spatial Keyword Range Queries over Moving Objects in Road Networks ⋆

Xiaokun DU [1], Yanhong LI [1,∗], Qun HUANG [2], Lihchyun SHU [3]

[1] College of Computer Science, South-Central University for Nationalities, Wuhan 430074, China

[2] 709th Research Institute, China Shipbuilding Industry Corporation, Wuhan 430074, China

[3] College of Management, National Cheng Kung University Tainan City, Taiwan 701, R.O.C.

Abstract

With the popularization of GPS-enabled devices and the arrival of the big data era, a significant amount of spatial documents have been generated every day. This development gives prominence to spatial keyword queries (SKQ), which consider both the distance and the keyword similarity of objects. However, Most of the existing SKQ methods are limited in Euclidean space which are unsuitable for SKQ processing in road networks. The paper addresses the issue of processing continuous spatial keyword range queries over moving objects (CMRSK) in road networks where both the query point and data objects can freely move within the road network. By using a range tree to bound the monitoring region of a CMRSK query, an efficient query processing method is proposed. Finally, simulation experiments are conducted to demonstrate the efficiency of our proposed method.

Keywords: Spatial Keyword Range Query; Moving Object; Road Network

1 Introduction

In recent years, with the rising popularity of the geographical applications and services, such as Google earth and yahoo map, spatial data query issues are becoming more and more important. Nowadays, query processing research has broke the limitation of pure location-based queries, such as nearest neighbor query [1, 2], reverse nearest neighbor query [3, 4], range queries [5], skyline query [6], etc. On the other hand, spatial keyword queries (SKQ), which consider both spatial proximity and textual relevance between the query and objects, have attract great concern in database area. And several techniques have been proposed for efficiently processing SkQ queries [7-14]. Recently, three types of SKQ queries are receiving particular attention, namely Boolean kNN query, top-k kNN query, and range query. We proceed to give the definition of these three types of queries.

⋆Project supported by the National Nature Science Foundation of China (No. 61309002).

∗Corresponding author.
Email address: anddylee@163.com (Yanhong LI).

1553–9105 / Copyright © 2015 Binary Information Press
DOI: 10.12733/jcis13181
January 15, 2015

Boolean kNN Query: Given a set of objects with a textual description, a query location, and a set of query keywords, it retrieves the k objects nearest to the query location such that each objects text description contains all the query keywords.

Top-k kNN Query: Given a set of objects with a textual description, a query location, and a set of query keywords, it retrieves k objects based on a ranking function that takes into account both the distance and the keyword similarity between the query and objects.

Range Query: Given a set of objects with a textual description, a query range, and a set of query keywords, it retrieve all objects whose text description contains all the query keywords and whose location is within the query range.

Zhou et al [10], proposed a hybrid index structure, which integrates inverted files and R*-trees, to handle spatial keyword queries. Three different combining schemes are studied: (1) inverted file and R*-tree double index, (2) first inverted file then R*-tree, (3) first R*-tree then inverted file. The text-first combination scheme can handle range queries and Boolean kNN queries, and the spatial first scheme can process range queries and Top-k kNN queries. Cong et al. [11] proposed an index structure called IR-tree which augments each node of the R-tree with an inverted file. The IR-tree based method can support all these three types of queries, namely, range queries, Boolean kNN queries, and Top-k kNN queries. Li et al. [12] discussed the issue of direction-aware spatial keyword query. This kind of query finds k spatially nearest neighbors of the query which are in the query direction and contain all the query keywords.

However, most of the existing SKQ query methods are limited in Euclidean space and there are few research works focusing on road networks. The SKQ methods in Euclidean space are unsuitable for SKQ processing in road networks, and it is therefore essential to examine SKQ methods suitable for use in real road networks. This paper addresses the issue of processing continuous spatial keyword range queries over moving objects (CMRSK) in road networks and can deal with the situation where the query point and data objects move freely in the road network. Our CMRSK method includes two main phases, namely *initial result computation phase* and *continuous monitoring phase*.

In the first phase, a range tree is worked as the monitoring region of a CMRSK query. We uses a method similar to Dijkstras algorithm to search qualified objects of a query q and examines nodes and edges according to the order they are encountered. Furthermore, we add the nodes and edges met during expansion into the range tree orderly. The expansion in each path stops once it reaches a point whose distance from the location of q is not smaller than a given distance limit or the border of the network is met. In the second phase, two basic kinds of updates, the query position updates and the object position updates, are processed, so as to keep the query result valid continuously.

2 Problem Difinition and Data Structrues

2.1 Problem definition

As many previous researchers do, we assume that each geo-textual object has a point location and a set of keywords, and we consider continuous spatial keyword range queries over moving objects ($CMRSK$) in road networks on such objects. Since we process CMRSK queries in road networks, whenever we refer to distance we mean the distance in the road network.

Dataset Setting Let D be a set of geo-textual objects. Each geo-textual object $o \in D$ is defined as a pair $(o.l, o.\psi)$, where $o.l$ is a location and $o.\psi$ is a set of keywords.

Spatial keyword range Query (RSK) Given a RSK query $q = (l, \psi, r, R)$, where $q.l$ is q' location, $q.\psi$ is a set of keywords, and $q.r$ is a distance value (thus, the search region $q.R$ consists of the points whose distance to q is not larger than $q.r$), the result of q, RSK(q) contains objects such that $\forall o \in RSK(q)(o.l \in q.R \wedge q.\psi \subseteq o.\psi)$.

Continuous Spatial keyword range Query over moving objects in road networks (CMRSK) Given a CMRSK query $q = (l, \psi, r, R, [t_s, t_e])$, where $q.l$ is q' location, $q.\psi$ is a set of keywords, $q.r$ is a distance value (thus, the search region $q.R$ consists of the points whose distance to q is not larger than $q.r$), and $[t_s, t_e]$ is a query time period, the result of q, CMRSK(q) consists of several tuples $< t_i, D_i >$ (i=1,2,3,...). In particular, $t_i \in [t_s, t_e]$, and D_i is a subset of D containing objects such that $\forall o \in D_i$ $(o.l \in q.R \wedge q.\psi \subset o.\psi)$ at time point t_i.

To illustrate this CMRSK problem clearly, we consider an example in Fig. 1, where a set of geo-textual objects o_1 to o_6 and a query object q move freely in a road network. Here both query objects (queries for short) and geo-textual objects (objects for short) belong to the data set D. Assume that a moving query $q = (l, \psi, r, R, [t_s, t_e])$, where $q.\psi = \{sushi, curry\}$, and query range $q.R$ consists of the points whose distance to $q.l$ is no larger than $q.r$). As shown in Fig. 1, $q.R$ is a dashed circle centered at $q.l$. In fact, the shape of $q.R$ is not always a circle since the distance used here is road network distance. However, we use a dashed circle in Fig. 1 to keep things simple. As shown in Fig. 1(a), there is only one object o_2 where $o_2.l \in q.R$ and $q.\psi \subseteq o2.\psi = \{sushi, soup, curry\}$. Thus, the query result at time t_0 is $\{o_2\}$. Similarly, as shown in Fig. 1(b), another object o_3 which includes all the keywords of query q moves into the query range at time t_1. Thus, the query result at time t_1 is $\{o_2, o_3\}$. As a result, the CMRSK query result will consist of several tuples $< t_0, \{o_2\} >, < t_1, \{o_2, o_3\} > \ldots$, where t_i is a time point.

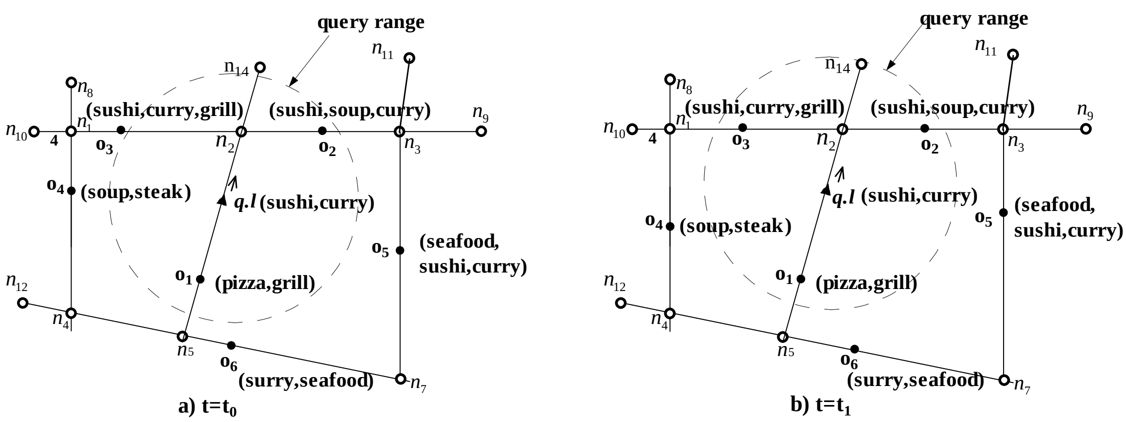


Fig. 1: An example of CMRSK query in road network

To get the query result at time t_0, we expand the road network from $q.l$ to find qualified objects. The expansion in each path stops once it reaches a point whose distance from $q.l$ equals r. In this way, all qualified objects (i.e., o_1, o_2,) are found. Then, for each qualified object p, we check whether $p.\psi$ includes all the query keywords. If this is the case, q is a result object and inserted into the RSK query result set. As for getting the RSK result set at time t_1, a straightforward method is to repeat the procedure executed at time t_0. However, this straightforward method will serious affect the query processing efficiency. Instead, we use a range tree to represent the whole monitoring region of query q so as to incrementally monitor the RSK query result.

2.2 Data structures

Graph model is used to simulate road networks here. In particular, we use an undirected weighted graph to represent the road network which includes an edge set and a node set. We maintain a set of moving $CMRSK$ queries (queries for short) and a set of moving geo-textual objects (objects for short) in the road network.

In our system, we use the following data structures to keep the information about the network structure and the moving objects and queries. Firstly, a PMR quad-tree is used to partition and keep the network structure. With this PMR quad-tree, we can identify the edge where an object (or query) lies according to the position of the object (or query). Secondly, we use the edge table T_{edge} and the node table T_{node} to represent the connectivity of the road network and to manage the moving objects on the roads. For each edge e, it has the following attributes in the edge table: (1) the edge id which is denoted as $e.id$; (2) the starting node of e which is denoted as $e.s$; (3) the ending node of e which is denoted as $e.e$; (4) the weight of e which is denoted as $e.w$; (5) the set of objects currently in edge e. Furthermore, the node table, T_{node}, stores for each node n: (1) the node id which is denoted as $n.id$; (2) the set of edges adjacent to the node n which is denoted as $n.e_{adj}$; (3) the location of the node n which is denoted as $n.l$. Thirdly, we use the query table T_{qry} to store for each query q: (1) the query id which is denoted as $q.id$; (2) the location of q which is denoted as $q.l$; (3) a set of keywords which denoted as $q.\psi$; (4) the distance range of query q which is denoted as $q.r$ (Note we can get the query range $q.R$ according to $q.l$ and $q.r$); (5) the current $CMRSK$ result set which is denoted as $q.CMRSK\_Set$.

3 Cmrsk Algorithm

This section presents the CMRSK query monitoring algorithm in the road network. The algorithm includes two main phases, namely, the *initial result computation phase* and *continuous monitoring phase*.

3.1 Phase 1: generating the initial RSK result

The initial phase has three main goals: (1) constructing the *range tree* of $CMRSK$ query q; (2) identifying a set of RSK objects which are termed RSK_Set hereinafter; (3) outputting the RSK_Set of q. Note that the range tree and RSK_Set will all be monitored in the continuous monitoring phase. An algorithm called $IniCMRSK$ is proposed to achieve these three goals.

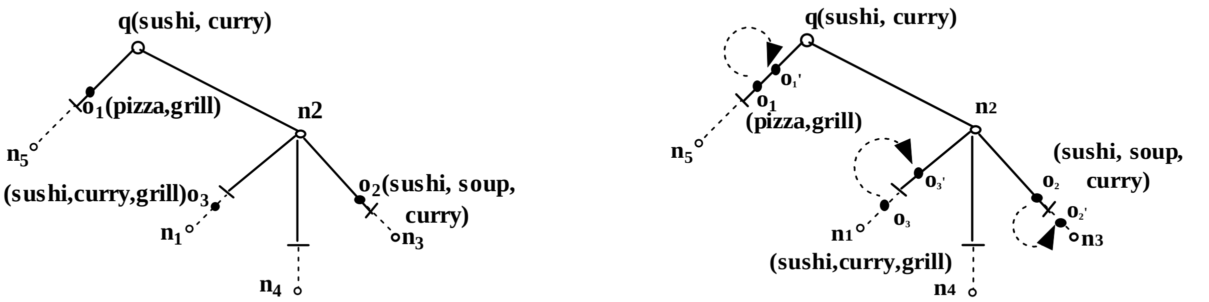
$IniCMRSK$ uses a method similar to Dijkstra's algorithm to search RSK objects of query q. Specifically, starting from q, $IniCMRSK$ expands the road network for searching RSK objects and examines nodes and edges according to the order they are encountered. Furthermore, it adds the nodes and edges met during expansion into the range tree orderly. The expansion in each path stops once it reaches a point whose the distance from $q.l$ is not smaller than $q.r$ or the border of the network is met.

$IniCMRSK$ uses the heap $Hnode$ and the set RSK_Set, which are both initialized to empty, to organize the nodes and RSK result objects met during network expansion, respectively. By using the PMR quad tree, $IniCMRSK$ first locates the edge e where query q locates. Then, it sets q as the root of the range tree, and divides e into two sub edges, i.e., e($e.s$, q) and e(q, $e.e$).

Next, lines 5-13 check if d$(q, e.s) < q.r$. If true, 1) for each object o on this sub edge e$(e.s, q)$, if $q.\psi \subseteq o.\psi$, then insert o into RSK_Set; 2) En-heap $e.s$ with the distance of d$(q, e.s)$, in ascending order of the distance from query q. Otherwise, 1) calculate the point n in sub-edge e$(q, e.s)$ where d(n, q) =$q.r$, and mark n as the boundary of the range tree; 2) for each object o on this sub edge e(q, n), if $q.\psi \subseteq o.\psi$, insert o into RSK_Set. Then, line 14 repeats the steps in lines 5-13 for sub-edge e$(q, e.e)$. Next, $IniCMRSK$ iteratively de-heaps nodes from $Hnode$. For each de-heaped node n: (1) it connects the edge e between n and its predecessor in the range tree; (2) for each adjacent node n_{adj} of n except its predecessor (lines 19-28): a) calculates d(n_{adj}, q); b) further checks whether (d$(n_{adj}, q) <= q.r$). If true, 1) for each object o on this sub edge e(n, n_{adj}), if $q.\psi \subseteq o.\psi$, inserts o into RSK_Set; 2) inserts n_{adj} into $Hnode$ together with d(n_{adj}, q); Otherwise, 1) calculates the point n in edge e(n, n_{adj}) where d$(n, q) = q.r$; and mark n as the boundary of the range tree; 2) for each object o on this sub edge e(n, n), if $q.\psi \subseteq o.\psi$, inserts o into RSK_Set.

Algorithm 1: IniCMRSK(q).

1 **begin**
2   Initialize($Hnode$); RSK_Set=∅;
3   let e be the edge containing q; divide e into two sub edges, i.e., e$(q, e.s)$,e$(q, e.e)$;
4   Set q as the root of the range tree of q;
5   **if** *d$(q, e.s) < q.r)$* **then**
6   **for** *each object o in sub-edge e$(q, e.s)$* **do**
7   Insert the objects o together with d(q, o) into RSK_Set, if $q.\psi \subseteq o.\psi$;
8   En-heap $e.s$ with the distance of d$(q, e.s)$;
9   **else**
10   Calculate the point n in edge e$(q, e.s)$ where d(n, q)=$q.r$;
11   **for** *each object o in sub-edge e(q, n)* **do**
12   Insert the objects o together with d(q, p) into RSK_Set, if $q.\psi \subseteq o.\psi$;
13   Mark n as the boundary of the range tree;
14   Repeat steps 5-13 for sub-edge e$(q, e.e)$;
15   **while** *not empty($Hnode$)* **do**
16   n=De-heap($Hnode$);
17   Connect the edge e between n and its predecessor in the range tree;
18   **for** *(each adjacent node n_{adj} of n except its predecessor* **do**
19   d(n_{adj}, q)= d(n, q)+$nn_{adj}.w$;
20   **if** *d(n_{adj}, q)* $<= q.r$ **then**
21   **for** *each object o in sub-edge e(n, n_{adj})* **do**
22   Insert the objects o together with d(q, o) into RSK_Set, if $q.\psi \subseteq o.\psi$;
23   Insert n_{adj} into Hnode with d(n_{adj}, q);
24   **else**
25   Calculate the point n in edge e(n, n_{adj}) where d(n, q)=$q.r$;
26   **for** *each object o in sub-edge e(n, n)* **do**
27   Insert the objects o together with d(q, o) into RSK_Set, if $q.\psi \subseteq o.\psi$;
28   Mark n as the boundary of the range tree;
29   Output every objects in RSK_Set;



(a) The original range tree (b) Object position updates of the range tree

Fig. 2: The range tree of CMRSK query q in Fig. 1

3.2 Phase 2: continuous monitoring for CMRSK queries

Since queries and objects can move freely in the road network, the RSK_Set for a query q may be overdue after some time. To keep the RSK_Set correct continuously, an algorithm called $MonitorCMRSK$ is proposed to continuously monitor $CMRSK$ queries and keep RSK_Set up-to-date. In the subsection, two basic kinds of updates, the query position updates and the object position updates, are processed.

(1) Object position Updates

As discussed before, the monitoring area of query q is a range tree which is rooted at $q.l$. We distinguish 3 types of object position updates affecting q. The first type is the incoming object which moves into the range tree of q from outside (e.g., object o_3 in Fig. 2b). In this case, we check whether $q.\psi \subseteq o.\psi$. If true, insert o into RSK_Set. The second type is the outgoing object which moves out of the range tree of q from inside (e.g., object o_2). In this case, we check whether $o \in RSK\_Set$. If true, delete o from RSK_Set . The last type is the object which moves within the range tree of q (e.g., object o_1). In this case, no processing is needed.

(2) Query position updates

As aforementioned, the movement of the query q is constrained within the road network and all RSK objects of q are bounded by its range tree. For clarity, we use q_n to represent the current position of q. There are two cases as follows:

Case 1: The current position (q_n) of q is within the former range tree of q. Let e(n_i, n_j) be the edge where q locates and e(n'_i, n'_j) be the edge where q_n locates. Thus, the new range tree which rooted at q_n can be got as follows: a) let q_n be the root node of the new range tree, b) set n'_i and n'_j as the two children of q_n; c) set other parts connected to n'_i (or n'_j) in the former range tree (except q_n) as the sub-tree of n'_i (or n'_j). Then, the boundaries of the new range tree will be adjusted according to the moving direction and distance of query q to ensure that the distance from each leaf node to query q maintained to be $q.r$.

Case 2: The current position of q is outside the former range tree. We process it as the incoming of a new query and the outgoing of an old one. We discard the old range tree of q, call $IniCMRSK$ to build the new range tree for q, and output q's RSK_Set.

3.3 MonitorCMRSK algorithm

Firstly, algorithm *MonitorCMRSK* checks whether q moves out of the range tree of q. If true, we discard the former range tree of q, call *IniCMRSK* to build the new range tree for q, and output its *RSK_Set*; Otherwise, it modify boundaries of the new range tree as discussed before.

Secondly, it processes object position updates about the range tree. We deal with three types of object position updates: 1) the incoming objects, 2) the outgoing objects, 3) objects moving within the range tree. The processing is similar to that discussed in Section 3.2. Note that only the object o satisfying that $q.\psi \subseteq o.\psi$ will really affect the query result set.

Algorithm 2: MonitorCMRSK.

```
begin
    if q moves out of the range tree of q then
        Discard the range tree of q and Call IniCMRSK to build the new range tree for q;
        Output its RSK_Set;
    else
        Modify boundaries of the new range tree;
    for each object update within the range tree of q do
        if the affected object o satisfy that q.ψ ⊆ o.ψ then
            if o is an incoming object then
                Insert o into RSK_Set; //case 1
            if o is an outgoing object then
                Delete o from RSK_Set; //case 2
    Output the objects in RSK_Set;
```

4 Performance Evaluation

In this section, we compare our method with a straight-forward method, which uses a method similar to Dijkstras algorithm for searching *RSK* objects at each time stamp. Hereinafter, we use *STF* denoting this straight-forward method and *CMRSK* denoting our method.

To simulate the real world road network, we use real data of the traffic network of Beijing city in China from [15], and construct a sub-network with 10K edges containing a set of queries and a set of objects which follow random distribution. Here queries require continuously monitoring of their *RSK_Sets* for 30 timestamps. At every timestamp, a percentage p_{obj} (or p_{qry}) of the objects (or queries) change their locations. Table 1 includes the parameters under investigation and the values in bold face are the default values in the following experiments. All algorithms were implemented in CPP and runs on Intel Core 2 Quad CPU Q8200 2.33GHz with 2GB RAM.

Firstly, Fig. 3(a) studies the effect of object cardinality on the performance of *STF* and *CMRSK*. The running time of both *STF* and our *CMRSK* increases slightly when we increase the number of objects in the system. It is because that as the number of objects increases, the number of objects within the query range also increases, so does the processing cost of these two algorithms. In Fig. 3(b), we measure the effects of object mobility (p_{obj}) on the CPU time

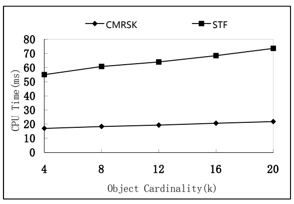
Table 1: Dataset parameters

Number of objects	4, 8, **12**, 16, 20(k)
Percent of update queries(P_{qry})	1, 5, **10**, 15, 20(%)
Percent of update objects(P_{obj})	1, 5, **10**, 15, 20(%)
Number of keywords	1, 2, **3**, 4, 5
Query range(r)	5, 10, **15**, 20, 25 (average edge length)

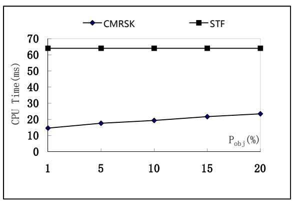
of STF and $CMRSK$. The figure tells us that the cost of our $CMRSK$ increases as object mobility increases. This is due to the fact that the update cost increases as the number of updated objects increases. STF on the other hand, its performance remains unchanged under different object mobility. The reason lies in that STF re-computes the RSK_Set from scratch at every timestamp.

Fig. 3(c) studies the effect of query mobility (p_{qry}) on the performance of these two methods. The cost of our $CMRSK$ increases for higher object mobility, since the movement of a query invalidates the entire or part of its range tree. STF on the other hand, its performance remains unchanged under different query mobility. The reason is the same as that in Fig. 5.

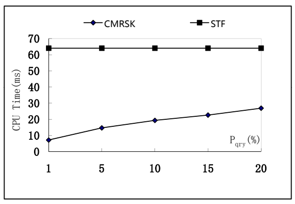
Fig. 3(d) plots the CPU time as a function of keyword number. The cost of these two methods increases slightly as the number of keywords increases, since more query keywords result in more keyword matching operation. Finally, we study the effects of the query range on the processing time of STF and $CMRSK$. Fig. 4(e) shows that the running time of both STF and our $CMRSK$ increase obviously as the query range gets larger. It's natural since much road network expansion and object comparison need be conducted as the query range becomes larger.



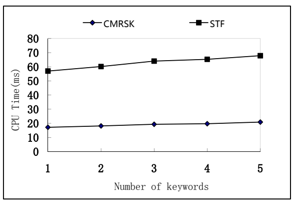
(a) Object cardinality



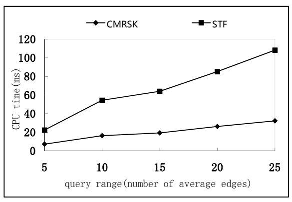
(b) Object mobility



(c) Query mobility



(d) Number of keywords



(e) Query range

Fig. 3: Experimental results

5 Conclusion

This paper addressed the issue of processing continuous spatial keyword range queries over moving objects in road networks (CMRSK). We use a range tree to represent the monitoring range of a CMRSK query. Thus, we can greatly reduce the query processing cost. The proposed method includes two main phases, namely initial result computation phase and continuous monitoring phase. Finally, experimental study on a real road network demonstrates the efficiency of our method. The result shows that our method is about 2.3 times more efficient than its competitor.

References

[1] YJ. Gao, BH. Zheng, GC. Chen, Q. Li, XF. Guo, Continuous visible nearest neighbor query processing in spatial databases, VLDB J., 2011, 20(3), pp. 371-396.

[2] YH. Li, JJ. Li, Lihchyun Shu, etc., Searching Continuous Nearest Neighbors in Road Networks on the Air, Information systems, 42(2014), pp. 177-194.

[3] XN. Yu, Y. Gu, TC. Zhang, G. Yu. A Method for Reverse k-Nearest-Neighbor Queries in Obstructed Spaces, Chinese Journal of Computers, 2011, 34(10), pp. 1917-1926.

[4] Y. Tao, D. Papadias, and X. Lian, Reverse kNN Search in Arbitrary Dimensionality, in: Proc of VLDB, 2004, pp. 744-755.

[5] J. Jin, N. An, A. Sivasubramaniam, Analyzing range queries on spatial data, in: Proc of ICDE, 2000, pp. 525-534.

[6] K. Deng, XF. Zhou, and HT. Shen, Multi-source skyline query processing in road networks, in: Proc of ICDE, 2007, pp. 796-805.

[7] J. Zhang, WS. Ku, X. Qin, Efficient Evaluation of spatial keyword queries on spatial networks, 2012, eng.auburn.edu.

[8] J. Lu, Y. Lu, and G. Cong, Reverse spatial and textual k nearest neighbor search, in: Proc of SIGMOD, 2011, pp. 349-360.

[9] D. Zhang, B.C. Ooi, and A. Tung, Locating mapped resources in web 2.0, in: Proc of ICDE, 2010, pp. 521-532.

[10] Y. Zhou, X. Xie, C. Wang, et al, Hybrid index structures for location-based web search, in: Proc of ACM CIKM, 2005, pp. 155-162.

[11] G. Cong, C.S. Jensen, and D. Wu, Efficient retrieval of the top-k most relevant spatial web objects, in: Proc of PVLDB, 2009, pp. 337-348.

[12] G. Li, J. Feng, J. Xu, DESKS: Direction-Aware Spatial Keyword Search, in: Proc of ICDE, 2012, pp. 474-485.

[13] W. Huang, G. Li, K.-L. Tan, and J. Feng, Efficient safe-region construction for moving top-k spatial keyword queries, in: Proc of CIKM, 2012, pp. 932-941.

[14] J. Wan, XK. Du, A Genetic Schema Matching Algorithm Based on Partial Functional Dependencies. Journal of Computational Information Systems, 2013, 9(12): 4803-4811.

[15] http://www.cs.fsu.edu/ lifeifei/SpatialDataset.html.