Continuous Monitoring of Top-k Spatial Keyword Queries in Road Networks*

YANHONG LI1, GUOHUI LI2, LIHCHYUN SHU3, QUN HUANG4 AND HONG JIANG5

1 College of Computer Science
South-Central University for Nationalities
Wuhan, 430074 P.R. China

2 School of Computer Science and Technology
Huazhong University of Science and Technology
Wuhan, 430074 P.R. China,

3 College of Management
National Cheng Kung University
Tainan City, 701 Taiwan

4 709th Research Institute
China Shipbuilding Industry Corporation
Wuhan, 430079 P.R. China

5 Naval University of Engineering
Wuhan 430033 P.R. China

E-mail: liyanhong@mail.scuec.edu.cn; anddylee@163.com

Recently, spatial keyword queries (SKQ) have become a hot topic in database field. However, Most of the existing SKQ methods are limited in Euclidean space or assume that objects (and queries) are static. This paper addresses the issue of processing continuous top-k spatial keyword queries over moving objects (CMTkSK) in road networks. To efficiently index moving geo-textual objects in road networks, a novel index structure called TPR$$^0$$-tree is proposed. Based on the index, an efficient CMTkSK query processing method which includes three main phases, namely generating initial result set phase, pruning phase, and continuous monitoring phase, is proposed. The proposed method can deal with the situation where the query client and geo-textual objects move continuously in the road network. By finding the result change time points, the method can continuously monitor CMTkSK queries and keep the query result set up-to-date with a small price. Finally, experiment results show that the proposed method is much more efficient and precise than its competitor.

Keywords: top-k spatial keyword query, moving object, road network, continuous monitoring, algorithm

1. INTRODUCTION

With the popularization of the geographical applications and services, spatial data query issues are becoming more and more important [1-5]. In recent years, spatial keyword queries (SKQ), which consider both spatial proximity and textual relevance between the query client and geo-textual objects, have become a new research topic in database area. Researchers have started to address SKQ processing, and some important research results have been published, e.g., [6-20].

Received July 31, 2014; revised November 5, 2014; accepted March 18, 2015.
Communicated by Vincent S. Tseng.

* This work was supported by the National Natural Science Foundation of China under Grant No. 61309002.
Zhou [12] et al. discussed the issue of SKQ query processing and proposed two geo-textual indices that integrates inverted files and R*-trees loosely, i.e., inverted file-R*-tree (IF-R*) and R*-tree-inverted file (R*-IF). IF-R* is a spatial-first index, while R*-IF is a text-first geo-textual index. Felipe et al. [13] proposed an index structure called IR2-tree which integrates an R-tree and signatures files. The signature file, in the form of bitmap, is stored for each node of the IR2-tree. Cong et al. [14] proposed an index structure called IR-tree which incorporates the inverted files and R*-tree. The IR-tree combines these two indexes to jointly prune the search space, thus it is more efficient than the methods in [12]. Wu et al. [15] and Huang et al. [16] studied continuously moving top-k spatial keyword queries. The former proposed an efficient algorithm for computing safe zones that guarantee correct result, and the latter calculated a safe region such that if a new query falling into the safe region the answer set remains the same. Chen [17] presented the first all-around evaluation of geo-textual indices, and offered a new insight into the properties and relative merits of the geo-textual indices.

However, these above algorithms are restricted to Euclidean space. In Euclidean space, the distance between two objects is decided by their coordinates. While in road networks the object-object distance is determined by the connectivity of the road network. Thus, the query methods in Euclidean space cannot be applied to road networks. João et al. [18] first studied top-k spatial keyword query processing in road networks and described how to rank objects with respect to both network distance and text relevance. The author proposed the indexing structure and utilized overlay network for efficient query processing. However, they focused on snapshot SKQ queries in road networks. In [19], the authors investigated continuous top-k spatial keyword queries in road network, and proposed two methods that can monitor moving queries in an incremental manner. However, they assumed all the geo-textual objects are static.

Up to now, there is still few research effort on continuous monitoring of spatial keyword queries in road networks, where both the query and geo-textual objects can move continuously in road networks which arises naturally in a travel environment. Consider the following scenario: Bob was walking on the street on one Wednesday afternoon and wanted to take a taxi to downtown. Due to heavy traffic in the city nowadays, many new traffic rules are being rolled out. For example, on Monday, Wednesday and Friday, only vehicles whose license numbers end with odd numbers are allowed to travel on the three bridges that lead to downtown. Thus, Bob would submit a spatial keyword query with keywords “odd vehicle license number, taxi” as he walked on the street. If continuous spatial keyword queries were supported, he could keep walking and receiving up-to-date results until a satisfactory taxi appears.

This paper addresses the issue of processing continuous top-k spatial keyword queries over moving objects (CMTkSK) in road networks and can deal with the situation where the query client and geo-textual objects move continuously within the road network. Moreover, the textual information of the object (or query) may change during the movement. To efficiently index moving geo-textual objects in road networks, a novel index structure called TPRgt-tree is proposed. TPRgt-tree is a two-level structure. Its top level gives the spatial information of the road network. The second level of TPRgt-tree consists of three tables, which are edge table T_{edge}, node table T_{node}, and geo-textual object table T_{obj}. Even the geo-textual objects move continuously in the road network, we can maintain the correctness of our index structure by simply modifying the pointers.
between the objects and the edges where they move.

Based on the index, an efficient CMTkSK query processing method which includes three main phases, namely generating initial result set phase, pruning phase, and continuous monitoring phase, is proposed. In the first phase, an efficient method is used to search qualified objects of query $q$. Specifically, starting from $q$, it expands the road network for searching top-$k$ spatial keyword ($TkSK$) objects and examines nodes and edges in the exact order they are encountered. In the second phase, based on the network distance calculating model and the formula for calculating the ST score of geo-textual object which considers both road network distance and text relevance, a pruning network distance $ND_{pruning}$ is calculated. We are sure that if an object $o$ whose network distance at time $t_i$ is larger than $ND_{pruning}$, $o$ is impossible to be the TkSK object of query $q$ within the monitoring time period. Finally, in the third phase, an efficient algorithm called MonitorCMTkSK is proposed. By finding the result change time points ($t_{change}$), we can continuously monitor CMTkSK queries and keep TkSK result set up-to-date with a small price.

The major contributions of this paper are as follows:

1. Our work remedies the major drawbacks of the past related works and provides a more practical and efficient solution for the continuous top-$k$ spatial keyword query processing in road networks.
2. A novel index called TPR$^{gt}$-tree is proposed to efficient index the moving geo-textual objects in the road network.
3. A continuous top-$k$ spatial keyword query processing algorithm (CMTkSK) in the road network is proposed, to efficiently find the TkSK objects of the moving query client within the monitoring time period.
4. Simulation experiments are conducted to evaluate the performance of the CMTkSK algorithm on a real road network and a geo-textual object set.

2. PROBLEM DEFINITION AND DATA STRUCTURES

2.1 Problem Definition

It is assumed that each geo-textual object has a point location and a set of keywords, and the issue of processing continuous top-$k$ spatial keyword queries over moving objects (CMTkSK) in road networks on such objects is considered here.

Dataset Setting Let $D$ be a set of geo-textual objects, where each object $o \in D$ has spatial location $o.l$ and a textual description (or a set of keywords) $o.\psi$.

Top-$k$ Spatial keyword Query (TkSK) Given a TkSK query $q = \langle l, \psi, k \rangle$ on the road network, where $q.l$ is $q$’ location, $q.\psi$ is a set of query keywords, and $q.k$ is the number of requested objects, $TkSK(q)$, contains $k$ spatial-textual objects ranked according to the following score (ST) which considers both road network distance and text relevance.

Continuous top-$k$ Spatial keyword query over moving objects in road networks (CMTkSK) Given a CMTkSK $q=(l, \psi, k, [t_s, t_e])$, where $q.l$, $q.\psi$ and $q.k$ have the same
meanings with that in TkSK, and \([t_i, t_e]\) is a query time period, the result of \(q\), CMT\(kSK\) contains several tuples \(<\{t_i, t_j\}, D_i>\) \((i = 1, 2, 3, \ldots)\). In particular, \(t_i, t_j \in [t_i, t_e]\), and \(D_i\) consists of \(k\) spatial-textual objects ranked according to the following score (ST) within the sub-period \([t_i, t_e]\).

\[
ST(o, q) = \frac{\theta(o, q)}{1 + \alpha \delta(o, q)}
\]

where \(\delta(o, q)\) reflects the network proximity between \(o\) and \(q\), \(\theta(o, q)\) reflects the text relevance between \(o\) and \(q\), and \(\alpha \in [0, +\infty]\) is a preference parameter to define relative importance of one measure over the other. For example, \(\alpha > 1\) increases the weight of textual relevance over network proximity.

The network proximity can be defined as the network distance between \(o\) and \(q\).

\[
\delta(o, q) = d_N(o, q)
\]

As for the textual relevance, the well-known cosine similarity model [20] is adopted.

\[
\theta(o, q) = \frac{\sum_{w_{t-o, q}} w_{t-o, q}}{\sqrt{\sum_{w_{t-o, q}} (w_{t-o, q})^2}}
\]

In particular, the weight \(w_{t-o, q} = 1 + \ln(f_{t-o, q})\), where \(f_{t-o, q}\) is the number of occurrences (frequency) of term \(t\) in \(o\); and the weight \(w_{o, q} = \ln(1+\|O\|/df_t)\), where \(\|O\|\) is the cardinality of object set and \(df_t\) is the number of objects in \(O\) whose description containing term \(t\). The value of \(\theta\) is in the range of \([0, 1]\) (property of cosine). There are many other relevance measures for textual relevance, such as the language model [4] and Okapi BM25 [10]. Our method can also support these measures.

To illustrate this CMT\(kSK\) problem clearly, we consider an example in Fig. 1, where a set of geo-textual objects \(o_1\) to \(o_8\) and a query object \(q\) move continuously in a road network. Here both query clients (queries for short) and geo-textual objects (objects for short) belong to the data set \(D\). Assume that a moving query \(q = (l, v, k, [t_i, t_e])\), where \(q.v = \{\text{pizza, cheap}\}\) and \(q.k = 2\). As shown in Fig. 1 (a), object \(o_1\) and \(o_2\) are the
two nearest objects of \( q \) whose text description contains all the query keywords. Thus, \( o_1 \) and \( o_2 \) are the top-2 objects according to ST value, and the query result at time \( t_0 \) is \( \{o_1, o_2\} \). Similarly, as shown in Fig. 1 (b), \( o_2 \) moves closer to \( q \) than \( o_1 \) at time \( t_1 \). Thus, the query result at time \( t_1 \) is \( \{o_2, o_1\} \). Finally, the CMT\( k \)SK query result will consist of several tuples \(<t_0, t_1>, \{o_1, o_2\}>, <t_1, t_2>, \{o_2, o_1\}>,..., \( t_i (i = 0, 1, \ldots) \) is a time point.

### 2.2 Data Structure

In our system, we use graph model to simulate road networks to process CMT\( k \)SK queries. In particular, the road network is represented as an undirected weighted graph consisting of a set of nodes and edges. We maintain a set of moving CMT\( k \)SK queries (queries for short) and a set of moving geo-textual objects (objects for short) in the road network. Here, each object (or query) moves with fixed speed in the road network and the textual information of the object (or query) may change during the movement.

People often use TPR-tree [21] like index to keep the information of objects to support location based query processing over moving objects. In our system, an index structure called TPR\( ^{gt} \)-tree is proposed to efficiently index moving geo-textual objects with fixed speed in the road network, where \( gt \) indicates geo-textual objects.

As shown in Fig. 2, TPR\( ^{gt} \)-tree is a two-level structure. Its top level gives the spatial information of the road network, where each leaf node consists of the edges included in

---

**Table 1: Terms and term frequencies**

<table>
<thead>
<tr>
<th>Obj</th>
<th>Terms and term frequencies</th>
<th>v</th>
<th>Obj</th>
<th>Terms and term frequencies</th>
<th>v</th>
</tr>
</thead>
<tbody>
<tr>
<td>( o_1 )</td>
<td>(restaurant, 3)(pizza, 5)(cheap, 2)</td>
<td>1</td>
<td>( o_5 )</td>
<td>(pizza, 4)(Italian, 5)(coffee, 2)</td>
<td>1</td>
</tr>
<tr>
<td>( o_2 )</td>
<td>(pizza, 5)(Italian, 5)(cheap, 1)</td>
<td>1</td>
<td>( o_6 )</td>
<td>(Italian, 4)(coffee, 3)(cheap, 1)</td>
<td>2</td>
</tr>
<tr>
<td>( o_3 )</td>
<td>(coffee, 4)(Italian, 2)(cheap, 4)</td>
<td>2</td>
<td>( o_7 )</td>
<td>(Italian, 4)(restaurant, 3)(cheap, 2)</td>
<td>-1</td>
</tr>
<tr>
<td>( o_4 )</td>
<td>(coffee, 4)(Italian, 4)(cheap, 3)</td>
<td>-2</td>
<td>( o_8 )</td>
<td>(pizza, 4)(restaurant, 3)(cheap, 2)</td>
<td>1</td>
</tr>
</tbody>
</table>

---

(c) Text information of objects.

Fig. 1. (Cont’d) An example of CMT\( k \)SK query in road network.

---

Fig. 2. The structure of TPR\( ^{gt} \)-tree.
the corresponding MBR. Thus, we can identify the edge where an object (or query) lies according to the position of the object (or query).

The second level of TPR*-tree consists of three tables, which are edge table $T_{edge}$, node table $T_{node}$, and geo-textual object table $T_{obj}$. For each edge in the leaf node of the top level of TPR*-tree, there is a pointer pointing to the entry corresponding to this edge in $T_{edge}$. In this way, the detail information of an edge where an object (or query) lies can be retrieved. Each entry of $T_{edge}$ consists of the edge id ($e.id$), the id of its starting node ($e.startid$), the pointer to the entry of its starting node in $T_{node}$, the id of its ending node ($e.endid$), the pointer to the entry of its ending node in $T_{node}$, its weight ($e.w$), its velocity limitation ($e.vlimit$), and the list of objects on it ($e.objlist$). In particular, for each item in $e.objlist$, it includes the object id and the pointer to the entry of this object in $T_{node}$. Secondly, each entry of $T_{node}$ consists of the node id ($n.id$) and the set of edges adjacent to the node $n$ ($n.eadjlist$), where each entry of this adjacent list includes the id and the address of the edge in $T_{edge}$. Thirdly, each entry of $T_{obj}$ consists of the object id ($obj.id$), the id of the edge where the object lies ($obj.edgeid$), the address of the edge where $obj$ lies in $T_{edge}$ ($obj.edgep$), the distance from $obj$ to the start node of the edge where it resides ($obj.dist$), its moving velocity ($obj.v$), and the set of keywords of $obj$ ($obj.keywords$).

Based on the TPR*-tree structure, it is efficient to update the moving object information. Here considers two common situations: (1) object keyword update. Then we only need to modify the keywords of the object in $T_{obj}$ correspondingly, which is rather easy to do; (2) an object $o$ moves from one edge $e_i$ to another edge $e_j$. Then, (a) locate the entry for object $o$ in $T_{obj}$, and modify the fields $edgeid$ and $edgep$ to the id and the address for edge $e_j$ in $T_{edge}$, respectively; (b) delete $o$ from the $objlist$ of edge $e_i$ in $T_{edge}$; (c) insert $o$ together with its address in $T_{obj}$ into the $objlist$ of edge $e_j$ in $T_{edge}$.

**Definition 1**: Given two different edges $e_i$ and $e_j$, if the shortest distance between every pair of two points, one from $e_i$ and another from $e_j$, is always determined by the same path, then we say $e_i$ and $e_j$ are distance determinate [22].

To efficiently evaluate these predicates, we construct a matrix $DD$, with each element $DD_{ij}$ having one of the following five values: the value $\Phi$ means that $e_i$ and $e_j$ are distance indeterminate; otherwise, $e_i$ and $e_j$ are distance determinate and there is a shortest path connecting $e_i$ and $e_j$. There are four sub cases, in particular, the value $<0, 0>, <0, -1>, <-1, -1>, <-1, 0>$, resp.) means the two endpoints of the shortest path are $e_i.startid$ ($e_i.endid$, $e_j.startid$, $e_j.endid$, resp.) and $e_j.startid$ ($e_i.endid$, $e_j.startid$, resp.)

To further speed up the network distance calculation, we construct another matrix $ND$ where $ND_{ij}$ is the shortest distance between nodes $n_i$ and $n_j$.

**2.3 Network Distance Calculation**

When an object $o$ moves with a fixed speed, its location at time $t$, denoted as $o(t)$, is calculated as $o(t) = o.dist + o.v \cdot (t - t_0)$, where $o.dist$ is the start location of $o$ represented as its distance from the starting node of the edge it resides, $o.v$ is the moving speed, and $t_0$ is the start time. Given a moving object $o$ and a query $q$, there are two possible cases for the distance between $o$ and $q$, that is $d_{oq}(t)$, depending on whether they are moving on the same edge:

$$d_{oq}(t) = \min(d_{oq1}, d_{oq2})$$

where $d_{oq1}$ and $d_{oq2}$ are the distance between $o$ and $q$ at time $t$ when $o$ and $q$ are moving on the same edge and on different edges, respectively.

$$d_{oq1}(t) = \sqrt{(x_o(t) - x_q(t))^2 + (y_o(t) - y_q(t))^2}$$

$$d_{oq2}(t) = \min(d_{oq11}, d_{oq12})$$

where $d_{oq11}$ and $d_{oq12}$ are the distance between $o$ and $q$ at time $t$ when $o$ and $q$ are moving on the same edge and on different edges, respectively.

$$d_{oq11}(t) = \min(d_{oq111}, d_{oq112})$$

where $d_{oq111}$ and $d_{oq112}$ are the distance between $o$ and $q$ at time $t$ when $o$ and $q$ are moving on the same edge and on different edges, respectively.
Case 1: \(q\) and \(o\) move on the same edge \(e\)

Assume both \(q\) and \(o\) are moving on the edge \((n_i, n_j)\), \(d_{q,o}(t)\) can be easily calculated.

\[
d_{q,o}(t) = |(o.\text{dist} - q.\text{dist}) + (o.v - q.v)(t - t_0)|
\] (4)

Case 2: \(q\) and \(o\) move on two different edges.

Assume that \(q\) is moving on edge \(e_i\) which starts from \(n_k\) and ends at \(n_l\), and \(o\) is moving on edge \(e_j\) which starts from \(n_m\) and ends at \(n_n\). There are two different sub cases. Based on the notion of distance determinate proposed before, even if \(o\) and \(q\) move on two different edges, it becomes much easier to calculate their distance in most cases.

(1) \(e_i\) and \(e_j\) are distance determinate.

By Definition 1, there is a shortest path connecting \(e_i\) and \(e_j\). Without loss of generality, we assume the two nodes connecting this shortest path are \(n_k\) and \(n_l\).

We can see that no matter \(o\) and \(q\) move toward or away from the shortest path connecting \(e_i\) and \(e_j\), \(d_{q,o}(t)\) is equal to the sum of \(d(q.l(t), n_k)\), \(DN_{l,n_k}\), and \(d(o.l(t), n_l)\), where \(d(q.l(t), n_k)\) is the distance from \(q.l(t)\) to \(n_k\), \(ND_{k,n_k}\) is the network distance between \(n_k\) and \(n_l\). Recall our use of the matrix \(DD\) in which the entry \(DD_{ij}\) with <value\(_0\), value\(_1\)> is used to represent different relations between \(e_i\) and \(e_j\) in terms of distance determinate. Note that when \(n_i\) is \(e_i.\text{startid}\) (the value\(_0\) of \(DD_{ij}\) is 0), \(d(q.l(t), n_i) = q.\text{dist} + q.v*(t-t_0)\); when \(n_i\) is \(e_i.\text{endid}\) (the value\(_0\) of \(DD_{ij}\) is -1), \(d(q.l(t), n_i) = e_{i.w} - (q.\text{dist} + q.v*(t-t_0))\). Thus \(d(q.l(t), n_i)\) can be uniformly represented by \([q.\text{dist} + q.v*(t-t_0)] + DD_{ij}.\text{value}\_0*\text{\(e_i.w\)}\), no matter whether \(n_i\) is \(e_i.\text{startid}\) or \(e_i.\text{endid}\). As a result, we have

\[
d_{q,o}(t)=q.\text{dist}+q.v*(t-t_0)+DD_{ij}.\text{value}\_0*\text{\(e_i.w\)}+\text{\(o.\text{dist}\)}+\text{\(o.v\)}*(t-t_0)\]
\[
|o.\text{dist}+o.v*(t-t_0)+DD_{ij}.\text{value}\_0*\text{\(e_i.w\)}|
\] (5)

(2) \(e_i\) and \(e_j\) are distance indeterminate.

Note that the possible locations of \(q\) and \(o\) are within two edges, i.e., \(e_i\) and \(e_j\) respectively. Thus, the shortest path between any pair of points in these two edges should pass through their end nodes, i.e., \(n_k\) (or \(n_l\)) and \(n_m\) (or \(n_n\)). Therefore, we need to take into account four network distances, i.e.,

\[
d_{q,o}(t)=d(n_k, q)+ND_{k,n_m}+d(n_m, o)\]
\[
d_{q,o}(t)=d(n_l, q)+ND_{l,n_m}+d(n_m, o)\]
\[
d_{q,o}(t)=d(n_k, q)+ND_{k,n_m}+d(n_m, o)\]
\[
d_{q,o}(t)=d(n_l, q)+ND_{l,n_m}+d(n_m, o)\]

Now we use the objects in Fig. 1 as an example to illustrate the calculation of \(d_{q,o}(t)\). Here we consider three different cases: object \(o_1\) and query \(q\) that are on the same edge; object \(o_2\) and query \(q\) that are on two different edges which are distance determinate; object \(o_3\) and query \(q\) that are on two different edges which are distance indeterminate. We obtain the following functions of \(d_{q,o}(t)\) with respect to different pairs of objects and query during different time intervals:

\[0 \leq t \leq 4.5 \quad dq, o_1(t) = 8 + 3t\]
\[ 0 \leq t \leq 4.5 \quad dq, \, o(t) = 15 - t \]
\[ 0 \leq t \leq 1 \quad dq, \, o(t) = 36 + 3t \]
\[ 1 \leq t \leq 4.5 \quad dq, \, o(t) = 42 - 3t \]

Observe that \( dq, \, o(t) \) is a linear function of time \( t \), except that when \( q \) and \( o \) move on two edges which are distance indeterminate, \( dq, \, o(t) \) could be a poly-line which consists of one segment with an upward slope and another segment with a downward slope.

### 3. CMT\( k \)SK ALGORITHM

In this section, we present the CMT\( k \)SK query monitoring algorithm in the road network. The query in question may choose a different edge in the road network to travel or change its direction at road intersection, and in this case the distance functions of all related objects will change. Thus, we only consider the time period from \( t_s \) to the earliest time instance \( t_e \), when the query reaches an intersection. For the time after \( t_e \), we consider a new time period to be monitored. For the same reason, we also consider the time point, when \( q \) change its moving speed, as the beginning of a new period. Besides, when a query changes its query keywords, the ST functions for every candidate objects will change, thus it is considered as a new query of course. However, when one of the related object arrives at a road intersection, changes its moving speed, or changes its keywords, its distance function (or ST function) is modified correspondingly. Next, we propose a continuous \( TSK \) query monitoring method which is composed of three phases, generating initial result set phase, pruning phase and continuous monitoring phase.

#### 3.1 Phase 1: Generating the Initial \( TkSK \) Query Result

In order to get the initial query result set, we use an algorithm called \( \text{InitTkSK} \) to search qualified objects of query \( q \). Specifically, starting from \( q \), \( \text{InitTkSK} \) expands the road network for searching \( TkSK \) objects and examines nodes and edges in the exact order they are encountered.

\( \text{InitTkSK} \) uses the heaps \( \text{nodeL} \) and \( \text{optL} \), which are both initialized to empty, to organize the nodes and \( TkSK \) candidate objects met during network expansion, respectively. By using the TPR*-tree, \( \text{InitTkSK} \) first locates the edge \( e \) where query \( q \) locates. Then, (1) it locates the entry for edge \( e \) in edge table \( T_{edge} \), and retrieves the detail information of \( e \); (2) by using the \( \text{objids} \) and pointers in \( e.objlist \), it retrieves the location and the set of keywords of each object on edge \( e \) from object table \( T_{obj} \); (3) calculates ST score for each object \( o \) on edge \( e \), by using formula 1 in section 2.1, and inserts object \( o \) together with its ST score into \( \text{optL} \) in descending order of ST value.

If there are \( k \) objects in \( \text{optL} \) (\( o_k \) is used to represent the object ranked \( k \)-th in the list), and the distance from \( q \) to the two end points of edge \( e \) is both larger than \( (1 - \text{ST}(o_k)) / (\alpha \cdot \text{ST}(o_k)) \), then the network expansion is stop and the first \( k \) objects in \( \text{optL} \) form the query result set. Since ST value is depended on both the network distance and keyword reference between query \( q \) and objects, we enlarge the expansion distance to \( (1 - \text{ST}(o_k)) / (\alpha \cdot \text{ST}(o_k)) \), which is denoted as \( d_k \), to avoid pruning candidate objects by mistake. In this way, we ensure that for an object \( o \) whose distance from \( q \) is larger than \( d_k \), the ST value of \( o \), \( \text{ST}(o) \), cannot be larger than \( \text{ST}(o_k) \), even its keyword set includes all the
keywords of query q. Otherwise, we insert e.startid and e.endid into nodeL in ascending order of distance from q.

Next, InitTkSK iteratively de-heaps nodes from nodeL. For each de-heaped node n:
(1) for each adjacent node n_adj of n except its predecessor (lines 11-22): ① calculates d(n_adj, q); ② further checks whether d(n_adj, q)<d_i. If true, 1) for each object o on this edge e(n, n_adj), calculate its ST value (ST(o)), inserts o together with ST(o) into optL in descending order of ST value; 2) inserts n_adj into nodeL together with d(n_adj, q); Otherwise, 1) calculates the point n’ in edge e(n, n_adj) where d(n’, q) = d_i; 2) for each object o on this sub edge e(n, n’), calculate its ST value, inserts o together with ST(o) into optL in descending order of ST value. The iteration continues until nodeL is empty and the first k objects in optL form the query result set. The detail step of the phase is shown in Algorithm 1.

Algorithm 1: InitTkSK (q)
1. Heap optL=∅, nodeL=∅; float d_i=∞, ST(o_i)=0;
2. Search TPR^k-tree to locate the edge e containing q;
3. For each object o on edge e {  
4. Calculate ST(o);
5. Insert (o, ST(o)) into optL in ascending order of ST value;
6. Let ST(o_j) to be ST value of the kth object in optL;
7. d_i=(1-ST(o_j))/(α* ST(o_j));
8. Insert e.startid (e.endid) to its distance to q into nodeL, if the distance value is smaller than d_i;
9. While nodeL is not empty {  
10. De-heap node n from nodeL;
11. For (each adjacent node n_adj of n except its predecessor) {  
12. d(n_adj, q)=d(n, q)+n_adj.w;
13. if (d(n_adj, q)<=d_i) {  
14. For each object o in edge e(n, n_adj) {  
15. Insert the objects o together with ST(o) into optL;}
16. d_i=(1-ST(o_j))/(α* ST(o_j));
17. Insert n_adj into nodeL with d(n_adj, q);}
18. else {  
19. Calculate the point n’ in edge e(n, n_adj) where d(n’, q)=d_i;
20. For each object o in sub-edge e(n, n’) {  
21. Insert the objects o together with ST(o) into optL;}
22. d_i=(1-ST(o_j))/(α* ST(o_j));}
23. Choose the first k objects in optL to form the TkSK_Set;
24. Output TkSK_Set;

Fig. 1 gives an example of CMTkSK query. As shown in Fig. 1 (a), query q which is denoted by a solid triangle is moving on edge e_k. Assume that q.k is 2 and q.w={pizza, cheap}. For ease of presentation, we assume that α=1 and the textual relevance of an object o (o.θ) is the number of occurrences of the query keywords in o.θ divided by the number of query keywords in q.w. The algorithm first accesses the network R-tree to find that query q is moving on edge e_k. There are two objects, o_1 and o_2, on edge e_k. Thus,
the algorithm calculates their ST values as follows.

\[ ST(o_1) = \frac{\theta(o, q) \cdot d_k(o, q)}{1 + \alpha \cdot d_k(o, q)} - \frac{1}{1 + 8} \]

\[ ST(o_2) = \frac{\theta(o, q) \cdot d_k(o, q)}{1 + \alpha \cdot d_k(o, q)} - \frac{0}{1 + 5} = 0 \]

\[ o_1 \text{ and } o_2 \text{ together with their ST values are inserted into optL in descending order of ST value. Thus optL equals } \{(o_1, 1/9), (o_2, 0)\}, \text{ and } d_k = (1 - ST(o_2))/(\alpha \cdot ST(o_2)). \]

Since the distance values from both the starting node and the end node of edge \( e_1 \) to query \( q \) are smaller than \( d_k \) which equals infinite at this moment, these two nodes together with their distances to query \( q \) are inserted into nodeL sequentially (line 8). Here nodeL equals \{\((n_3, 9), (n_5, 15)\)\}. Then, the first element of nodeL, which is \( n_3 \), is de-heaped, and edge \( e_1 \) and \( e_2 \) are processed. The ST values of object \( o_1 \) on edge \( e_1 \) and \( o_2 \) on edge \( e_2 \) are calculated, respectively. In particular, ST\((o_3) = 1/40 \) and ST\((o_2) = 1/16 \). Next, \( o_2 \) and \( o_1 \) are inserted into optL = \{\((o_1, 1/9), (o_2, 1/16), (o_3, 1/40) (o_5, 0)\} \). Since each adjacent node of \( n_2 \) is either outside the distance range \( d_k = (1 - ST(o_2))/(\alpha \cdot ST(o_2)) \) of query \( q \), which is 15 at this moment, or is a boundary node of the road network, neither of them is inserted into nodeL.

Next, \( n_5 \) is de-heaped from nodeL. Since the distance of \( n_5 \) to query \( q \) is equal to \( d_k \), none of its adjacent edges is processed. Now, nodeL is empty and the processing is stopped, and the first two objects in optL forms the TSKS_Set = \{\((o_1, 1/9), (o_2, 1/16)\)\}.

### 3.2 Phase 2: Pruning Phase

The main goal of this phase is to find a pruning network distance, denoted as NDpruning, to ensure that if an object \( o \) whose network distance at time \( t_s \) is larger than NDpruning, then \( o \) is impossible to be the TSKS object of query \( q \) within the interval \([t_s, t_e]\). In Phase 1, we have gotten ST\((o_1) \) which is the ST value of the 4th result object of query \( q \) at time \( t_s \), and \( d_k \) equals \((1 - ST(o_1))/(\alpha \cdot ST(o_1)) \). In particular, for an object \( o \) whose distance from \( q \) is larger than \( d_k \) at time \( t_s \), the ST value of \( o \), ST\((o) \), cannot be larger than ST\((o_1) \) at this time point, even if its keyword set includes all the keywords of query \( q \).

Since objects and the query are moving continuously in the road network, the objects outside the pruning distance of \( q \) at time \( t_s \) may move into the distance range \( d_k \) within the period \([t_s, t_e]\), thus we enlarge the pruning distance by \((q.v + v_{\text{max}})^*(t_e - t_s) \). Here, \( v_{\text{max}} \) is the largest velocity of the edges within the distance range \( d_k \) of \( q \). Then, the pruning distance, NDpruning, is set to be \( d_k = (q.v + v_{\text{max}})^*(t_e - t_s) \). Thus, for each object \( o \) within NDpruning of query \( q \) at time \( t_s \), if \( o.q \) includes any query keyword in q, o is regarded as the candidate objects and put into Cand_Set; otherwise, o is regarded as the monitored objects and put into Monitor_Set. All other objects can be pruned safely.

Continue the example in Fig. 1, we have already gotten the value of \( d_k \) which is 15. As shown in Fig. 3, query \( q \) moves with a speed of -2m/sec toward \( n_2 \) and its distance to node \( n_2 \) is 9, it takes 4.5 sec for \( q \) to reach node \( n_2 \). Hence, \( t_e \) is set to be 4.5sec. Moreover we add \((2+2)*4.5 = 18 \) to NDpruning which increases NDpruning from 15 to 33. Note that both the moving speed of \( q \) and the largest speed of the edges within NDpruning of \( q \) are 2 here. There are six moving objects \( o_1, o_2, o_3, o_6, o_7 \) and \( o_8 \) within the pruning range and \( o_7.q \) doesn’t include any query keyword. Thus the Cand_Set is \{\( o_1, o_2, o_3, o_6, o_7 \)\} and Monitor_Set is \{\( o_7 \)\}. 
3.3 Phase 3: Continuous Monitoring for CMTkSK Queries

Since queries and objects can move continuously in the road network, the TkSK_Set for a query \( q \) may be overdue after some time. To keep the TkSK_Set correct continuously, an algorithm called Monitor-CMTkSK is proposed to continuously monitor CMTkSK queries and keep TkSK_Set up-to-date. Remember that for query \( q \) at time instance \( t_s \), there is an object that ranks \( k \)-th among all the objects in Cand_Set in terms of their ST values, from largest to smallest; we call this object \( k \)-th object. The goal of phase 3 is to determine the query result change time points \( (t_{\text{Changes}}) \) within \([t_s, t_e]\) such that the query has the same TkSK result within two consecutive \( t_{\text{Changes}} \), and find the corresponding TkSK_Set at each \( t_{\text{Change}} \). Algorithm 2 is for this purpose.

Firstly, Algorithm MonitorCMTkSK (1) sets two variables, \( t_a \) and \( t_b \), which are both initialized to \( t_s \), to record the beginning and the end of the sub-periods being processed within \([t_s, t_e]\); respectively; and (2) uses set CMTkSK_Set to organize the TkSK result which includes a set of tuples: <sub-period, the corresponding TkSK_Set>. Then, we consider each object \( o \in \text{Cand}_\text{Set} \). Remember that \( \text{Cand}_\text{Set} \) includes all the objects \( o \) whose \( ND_{pruning}(t) \) is smaller than \( ND_{pruning} \) at \( t_s \) and \( o.\phi \) includes any query keyword. If the ST function of \( o \) intersects with that of \( k \)-th object at a time point \( t \in [t_s, t_e] \), we use variable \( t_c \) to keep this time point. If there are several this kind of time points, \( t_c \) equals the earliest one.

Then we replace \( t_s \) with \( t_b \) and \( t_b \) with \( t_c \) to form a new time period (line 7). Since TkSK_Set remains unchanged within the newly formed time period, the tuple \([t_a, t_b], \{ \text{TkSK}_\text{Set}\}\) represents part of the query result and is inserted into CMTkSK_Set (line 8). Then the TkSK_Set is modified discriminately: if \( o \) is not in current TkSK_Set, \( o \) is included into TkSK_Set (line 10); otherwise switch the order of \( o \) and the old \( o_k \) in Cand_set and TkSK_Set to let \( o \) be the new \( k \)-th object (lines 12-13). The modified TkSK_Set will be used in the next sub-period. Next, repeat the steps in lines 3-5 to get the next \( t_{\text{Change}} \). The processing repeats until all the \( t_{\text{Changes}} \) are found and processed, and then the final CMTkSK_Set is obtained.

Algorithm 2: MonitorCMTkSK

Input: TkSK_Set, Cand_Set, \( d_t \) and ST\((o_k)\) at time instance \( t_s \), and the time period \([t_s, t_e]\)

Output: CMTkSK_Set
1. Begin
2. set $CMTkSK\_Set=\emptyset$; $t_a=t_0$; $t_i=t_i$; $t_i=t_i$;
3. For (each object $o \in Cand\_Set-o_k$)
4. {If ($ST_{q,o}(t)$ intersects $ST_{q,ok}(t)$ at a time instance $t \in [t_i, t_a]$ and ($t > t_i$))
5. {t_i=t_i;}
6. While ($t_i < t_a$)
7. {t_i=t_i; t_j=t_j;}
8. Insert the tuple $<[t_i, t_j], \{TkSK\_Set\}>$ into $CMTkSK\_Set$ ;
9. If (o is not in $TkSK\_Set$)
10. {Replace the $k$th object in $TkSK\_Set$ with $o$;}
11. Else
12. {Switch the position of o and the old $o_i$ in $Cand\_Set$ and $TkSK\_Set$;
13. let $o_i$ be $k$th object in $Cand\_Set$;}
14. Repeat lines 3-5;}
15. Insert the tuple $<[t_0, t_i], \{TkSK\_Set\}>$ into $CMTkSK\_Set$ ;
16. Return $CMTkSK\_Set$;}

As shown in Fig. 4, at time $t_0$, $TkSK\_Set$ is $\{o_1, o_2\}$ and $o_3$ is $o_2$. Then, the ST functions of $o_1$ and $o_2$ intersect with each other at time $t=1.75$, thus $o_1$ replaces $o_2$ as the $k$th object. Next, the ST function of $o_3$ intersects that of $o_1$ which is the $k$th object at time $t=3.18$, thus $o_3$ replaces $o_1$ as the $k$th object and $TkSK\_Set$ becomes $\{o_2, o_3\}$. Here time instances $t=1.75$ and $3.18$ are $t_{\text{changes}}$ and all these $t_{\text{changes}}$ within time period $[0, 4.5]$ divide this time period into several sub-periods. Finally, all the sub-periods together with their $TkSK\_Sets$ form the $CMTkSK\_Set$ of query $q$, which consists of four tuples: $<[0, 1.75], \{o_1, o_2\}>$, $<[1.75, 3.18], \{o_2, o_1\}>$, $<[3.18, 4], \{o_2, o_3\}>$, and $[4, 4.5], \{o_3, o_2\}$.

4. PERFORMANCE EVALUATION

4.1 Experimental Settings

This section presents the performance evaluation of our $CMTkSK$ query processing.
method. To simulate the real world road network, we use the real data of the traffic network of Oldenburg in Germany [23], which consists of 6105 nodes and 7035 edges. We use the generator proposed in [24] to obtain a set of geo-textual objects and queries. The description of the objects (queries) is obtained from Twitter (http://twitter.com), one tweet per object. Fig. 5 depicts the real road network of Oldenburg and the date objects in it, with roads and data objects represented by blue lines and red points, respectively. Remember that the index and data structure of our proposed method consists of a road network R-tree and three relational tables. Fig. 6 depicts the storage cost when we vary the number of objects in the system, and this figure shows that the storage space for keeping the index and data structure is small and can be kept in memory. Moreover, these three tables include some pointer fields to make them interrelated, thus the time for searching related items (edges, nodes, or objects) within the tables can be saved. To further speed up the network distance calculation, we construct two matrixes $DD$ and $ND$ (refer to section 2.2. The total storage space needed for these two matrices constructed for Oldenburg is 93.1M, thus can also be kept in memory). As a result, the total query processing time can be low.

![Fig. 5. Oldenburg and data set.](image1)
![Fig. 6. Storage cost for index and data structure.](image2)

Here, we will compare our method with the CMA method [25], which uses a combined expansion tree to keep the monitoring area of a TkSK query. Each query requires continuously monitoring of their $TkSK\_Sets$ for 100 timestamps. The CMA algorithm re-evaluates the snapshot $TkSK$ query when location updates of objects (and/or the query) occur. The update interval ($UI$) of CMA is set to 5 and 10 time units in this experiment which are denoted as CMA($UI=5$) and CMA($UI=10$), respectively. Besides, we use $CMTkSK$ to denote our proposed method. We measure the average running time for processing $CMTkSK$ queries in road networks. Moreover, we investigate the precision of these two algorithms by evaluating the percentage of retrieved $TkSK$ objects that are real. Let $TkSK\_get$ be the set of $TkSK$ objects which are retrieved by these two algorithms. Besides, we let $TkSK\_real$ be the set of objects which are the real $TkSK$s of query $q$. Then, precision is represented as follows.

\[
\text{Precision} = \frac{\#(TkSK\_get \cap TkSK\_real)}{\#(TkSK\_real)}
\]  

The input and distance-calculations of these two compared methods are the same.
Table 1 includes the parameters under investigation and the values in bold face are the default values in the following experiments.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query time interval</td>
<td>10, 30, 50, 70, 100</td>
</tr>
<tr>
<td>Number of keywords</td>
<td>1, 2, 3, 4, 5</td>
</tr>
<tr>
<td>Value of $k$</td>
<td>10, 15, 20, 25, 30</td>
</tr>
<tr>
<td>Number of objects</td>
<td>15k, 20k, 25k, 30k, 35k</td>
</tr>
</tbody>
</table>

### 4.2 Experimental Results

Firstly, Fig. 7 evaluates the effect of query interval length on the CPU time and the precision of CMTkSK and CMA. As shown in Fig. 7 (a), the CPU time of these two algorithms increases as query interval length becomes longer. For our CMTkSK, this is because a longer query interval length implies more queries and objects reaching the network nodes and/or changing their keywords, hence more queries are launched or modified. For CMA, this is due to the fact that a larger query interval length incurs more location updates of objects and queries, resulting in more maintenance cost. Clearly, CMTkSK has a better performance at all time intervals, compared to CMA (Both for UI = 5 and 10). Fig. 7 (b) shows that the precision of CMTkSK is always 100% under different query interval lengths. If CMA is adopted to answer the query, the precision is at best 54% and a large part of the query results are unknown due to the nature of CMA’s discrete location updates. Moreover, as shown in the figures, if UI increases, which means the time span between two location updates becomes longer, the total update processing cost will decrease. However, the precision will decrease too.

The second set of experiments measures the effect of $k$ value on the performance of the methods. Fig. 8 (a) shows that the CPU time for both algorithms grows as $k$ increases. This is because that as $k$ becomes larger, the number of candidate objects increases so that the monitoring range and ST value comparisons between these candidate objects also increase. Fig. 8 (b) studies how the value of $k$ affects the precision of the methods.
The precision of the CMTkSK algorithm remains 100% for different \( k \) values. However, the precisions of CMA is low no matter UI=5 or UI=10.

Fig. 9 plots the CPU time and precision as a function of keyword number. As shown in Fig. 9 (a), the CPU cost of these two methods increases as the number of keywords increases. This is because that more query keywords result in more candidate objects and more ST value comparisons. Fig. 9 (b) shows that similar to Figs. 7 (b) and 8 (b), the precision of our CMTkSK maintains 100% for different keyword numbers, and the precision of CMA is still low and changes slightly when the keyword number varies.

Finally, Fig. 10 evaluates the effect of object cardinality on the CPU time and the
precision of CMTkSK and CMA. As shown in Fig. 10 (a), the CPU time of these two algorithms decreases as the number of objects becomes larger. For CMA, as the number of objects increases, the object density increases correspondingly, thus the network expansion needed for searching candidate objects decreases. As a result, the running time decreases. For CMTkSK, as the object density increases, the monitoring range of a query decreases, thus the running time decreases. As shown in Fig. 10 (b), the precision of CMA is at best 56% and decreases slightly as the number of moving objects increases.

5. CONCLUSIONS

This paper addressed the issue of processing continuous top-$k$ spatial keyword queries over moving objects in road networks (CMTkSK). The TPR$^6$-tree index is proposed to efficiently index the moving geo-textual objects in road networks. Based on the index, an efficient CMTkSK query processing method is proposed. Finally, experimental study on a real road network demonstrates the efficiency of our proposed method. The result shows that our method is about 1.28 times more efficient than the CMA method. Moreover, the precision of our method maintains 100% at any time instance, while the precision of the compared method is at best 56%.

REFERENCES

Yanhong Li (李艳红) received her Ph.D. degree in Computer Science from Huazhong University of Science and Technology, China in 2011. She has been on the faculty at South-Central University for Nationalities, and she is currently an Associate Professor in the Department of Computer Science. Her research interest is spatial information and communication.

Guohui Li (李国徽) received his Ph.D. degree in Computer Science from Huazhong University of Science and Technology in 1999. He is a Full Professor in School of Computer Science and Technology, Huazhong University of Science and Technology. His research interests mainly include advanced data management, mobile computing and real-time computing.

LihChyun Shu (徐立群) received her Ph.D. degree in Computer Science from Purdue University in 1994. He is a Fully Professor in the College of Management, National Cheng Kung University. His research interest is methods for design and analysis of software, especially software for concurrent and distributed systems, as well as real-time systems.

Qun Huang (黄群) has been on the faculty at 709th Research Institute, China Shipbuilding Industry Corporation, and she is currently a Senior Engineer in the Department of Engineering. Her research interests include spatial information and communication, and multimedia network communication technology.

Hong Jiang (蒋宏) received his Master degree in Automation from Huazhong University of Science and Technology, China in 2008. He has been on the faculty at Naval University of Engineering, and he is currently an Associate Professor in the Department of Electrical and Electronic Engineering. His research interest is advanced data management.