2-2016

Efficient collective spatial keyword query processing on road networks

Yunjun GAO

Jingwen ZHAO

Baihua ZHENG
Singapore Management University, bhzheng@smu.edu.sg

Gang CHEN

DOI: https://doi.org/10.1109/TITS.2015.2477837

Efficient Collective Spatial Keyword Query Processing on Road Networks

Yunjun Gao, *Member, IEEE*, Jingwen Zhao, Baihua Zheng, *Member, IEEE*, and Gang Chen

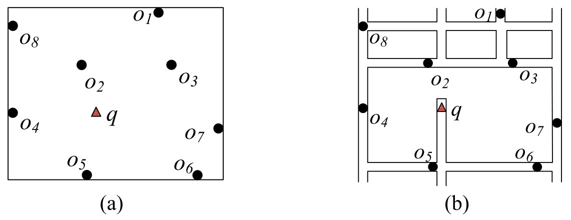
***Abstract*—The *collective spatial keyword query* (CSKQ), an important variant of spatial keyword queries, aims to find a set of the objects that collectively cover users' queried keywords, and those objects are close to the query location and have small inter-object distances. Existing works only focus on the CSKQ problem in the Euclidean space, although we observe that, in many real-life applications, the closeness of two spatial objects is measured by their road network distance. Thus, existing methods cannot solve the problem of network-based CSKQ efficiently. In this paper, we study the problem of *collective spatial keyword query processing on road networks*, where the objects are located on a predefined road network. We first prove that this problem is *NP-complete*, and then we propose two *approximate* algorithms with provable approximation bounds and one *exact* algorithm, for supporting CSKQ on road networks efficiently. Extensive experiments using real datasets demonstrate the efficiency and accuracy of our presented algorithms.**

***Index Terms*—Algorithm, collective, road network, spatial keyword query.**

I. Introduction

MOBILE social media with geo-location applications are catching on fast and are bringing the technology to the next level. The availability of both location information and rich contents of textual information creates a large number of new applications, such as recommender systems. The spatial keyword query has received much attention from both industry and research communities, due to the popularity of geo-positioning technologies.

Collective spatial keyword query (CSKQ), which aims to retrieve a set of the objects that collectively cover users' queried keywords with the minimum cost, has attracted lots of attention from academia [4], [16]. Nonetheless, to the best of our knowledge, existing efforts on CSKQ find the objects in the Euclidean space, and thus, they cannot be directly applied to retrieve the objects on the road network. In real life applications, Point of Interests (POIs) are located on a predefined road network, and the computation cost of the road network distance is much higher than that in the Euclidean space. Therefore, it is critical to handle this problem on road networks. In this paper, we study a new form of CSKQ, namely, *collective spatial keyword queries on road networks* (CSKQ on road networks). Given a set of objects with each associated with a set of keywords, a CSKQ on road network returns a set of the objects that collectively cover queried keywords, with those objects being close to a query location and meanwhile close to each other. An example is shown in Fig. 1, where Fig. 1(a) and (b) depicts eight objects in a Euclidean space and on a road network respectively, with the keywords associated with each object listed in Fig. 1(c). Given a query q having the query position denoted by a triangle and the queried keywords set as t_2 and t_5, the result set in the Euclidean space is $\{o_2, o_3\}$, which is totally different from the result set $\{o_5, o_6\}$ in the network space. This is because the distance metrics utilized to evaluate the proximity between objects in the Euclidean space and in the network space are distinct, and the objects that are close in the Euclidean space are not necessarily close to each other in the road network.



POI	Keywords	POI	Keywords
o_1	t_1, t_2	o_5	t_2, t_4
o_2	t_2, t_3	o_6	t_5, t_6
o_3	t_3, t_5	o_7	t_5, t_7
o_4	t_1, t_2	o_8	t_3, t_6

(c)

Fig. 1. Example of CSKQ in Euclidean spaces and road networks respectively. (a) POIs in a Euclidean space. (b) POIs on a road network. (c) POI description.

CSKQ on road networks is useful in real life applications, especially in decision making and travel planning. As an example, Hilton would like to attract more customers by promoting a few hotel packages, e.g., the relaxing package includes five days hotel stayed at Hilton, world-class shopping experiences, memorable dining experiences, and unique garden spa experiences within working distance. In this case, users' need cannot be satisfied by a single object, but a set of objects. As another example, a conference organizer intends to impress the participants of the conference by offering various packages to satisfy

Manuscript received December 19, 2014; revised May 14, 2015; accepted September 4, 2015. Date of publication November 6, 2015; date of current version January 29, 2016. This work was supported in part by the 973 Program under Grant 2015CB352502, by NSFC under Grants 61522208, 61379033, and 61472348, and by the Fundamental Research Funds for the Central Universities. The Associate Editor for this paper was Q. Zhang.

Y. Gao, J. Zhao, and G. Chen are with the College of Computer Science, Zhejiang University, Hangzhou 310027, China (e-mail: gaoyj@zju.edu.cn; zhaojw@zju.edu.cn; cg@zju.edu.cn).

B. Zheng is with the School of Information Systems, Singapore Management University, Singapore 178902 (e-mail: bhzheng@smu.edu.sg).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TITS.2015.2477837

different participants' needs, e.g., an economy package contains a budget hotel and a few low-cost restaurants close to the conference venue, and a luxurious package includes a five-star hotel and a few high-end restaurants close to the conference venue. It is also observed that the conference organizer should find out a set of objects to meet various needs. Motivated by these, we dedicate this paper to process CSKQ on road networks.

Existing efforts on CSKQ is Euclidean space based methods, and they utilize a Euclidean space index (e.g., IR-tree) to store the objects, which is not suitable for network based CSKQ. The reason behind is that the Euclidean based methods have to calculate all the distances among POIs and the distances from the query point to the POIs, as well as the computation cost $(O(n \log n))$ of the road network distance is much higher than that $(O(1))$ in the Euclidean space. In view of this, we adopt a Connectivity-Clustered Access Method [22] (CCAM) index to address our studied problem, and propose efficient pruning strategies and algorithms based on the road network to reduce the computation times of the road network distances. As demonstrated in our extensive empirical study, they show excellent performance.

In brief, the key contributions of this paper are threefold:

- We formally define the problem of CSKQ on road networks. With strict verification, we show that this problem is *NP-complete*. There is, to our knowledge, no priori work on the problem.
- We develop two approximate algorithms with proved approximation bounds and one exact algorithm to support CSKQ processing on road networks.
- We conduct comprehensive experiments on real datasets to verify the accuracy and efficiency of our algorithms.

The rest of this paper is organized as follows. Section II reviews related work. Section III formalizes our problem, and presents the index structure employed to store the objects and the road network. In Sections IV and V, we elaborate two approximate algorithms and one exact algorithm, respectively. Considerable experimental results and our findings are reported in Section VI. Finally, Section VII concludes the paper with some directions for future work.

II. Related Work

In this section, we survey the existing work related to spatial keyword search, including indexes and query models for spatial keyword queries.

A. *Spatial Keyword Indexes*

Spatial keyword retrieval has been intensively studied in recent years. In general, there are three types of spatial keyword queries, including Boolean spatial keyword query, ranked spatial keyword query, and Boolean range spatial keyword query. Please refer to [1], [6] for comprehensive surveys. A Boolean spatial keyword query finds the objects that contain all the queried keywords, sorted by their distances (e.g., the Euclidean distance) to a specified query object. To answer this query, many efficient index structures have been developed in the literature, e.g., spatial keyword index [5], information retrieval R-tree [10], BR-tree [15], inverted linear quad-tree [33], and spatial inverted index [23]. They tightly integrate spatial index (e.g., R-tree) and textual indexing (e.g., bitmap). In these indices, every entry in a tree node stores a keyword summary field that concisely summarizes the keywords for the entry.

The ranked spatial keyword query is to rank the objects based on some ranking functions that consider both the spatial proximity from an object to a query point (evaluated by the distance) and the textual similarity between the object and the queried keywords. Representative indexes contain the IR-tree [14] and inverted R-tree [9], [25], which augment an R-tree [11] with inverted files, and the spatial inverted index [19] that maps every keyword to an aR-tree. After a simple transformation, these indices can also tackle Boolean spatial keyword retrieval. In addition, the Boolean range spatial keyword query, where the target is to retrieve all the objects that include the queried keywords and meanwhile are located into a given query region, is well studied in the literature [7], [8], [12], [21], [24].

It is worth noting that, all the approaches mentioned above are unsuitable for CSKQ processing on road networks because they focus on spatial keyword queries in Euclidean space instead of road networks.

B. *Query Models for Spatial Keyword Queries*

Various forms of spatial keyword queries have also been explored intensively [18], [28], [32]. A collective spatial keyword query (CSKQ) [4], [16], [29], [31] finds a set of the objects that collectively cover the queried keywords. Cao *et al.* [4] consider two cost functions to handle this query, and prove that both are NP-complete. They propose approximate algorithms with provable approximation bounds, and exact algorithms for two sub-problems. In [16], Long *et al.* present the improved algorithms for such query, and also investigate a new instantiation of the CSKQ.

Wu *et al.* [27] study continuously moving spatial keyword retrieval, which finds the objects that can meet users' spatio-textual constraints, and guarantees that users have an exact result at any time. Lu *et al.* [17] explore reverse spatial and textual nearest neighbor search, in which the goal is to retrieve all the objects that take a specified query object as one of their k most spatio-textual similar objects. A spatial keyword query on a road network [2], [20], [34] aims to find the objects based on their spatio-textual similarity to the query object, and those objects are located on the road network.

A joint spatial keyword query [26] explores the problem of jointly processing multiple spatial keyword queries, and employs the IR-tree and the WIBR-tree as the index structure. A location-aware top-k prestige-based text retrieval [3] aims to retrieve the k spatial web objects according to not only the spatial-textual information but also the presence of nearby objects relevant to the query.

Last but not least, there are still many variations of spatial keyword queries, such as preference-based top-k spatial keyword query [30], direction-aware spatial keyword search [13], to name just a few. It is worth mentioning that, all the aforementioned works are fundamentally different from our studied problem.

TABLE I
SYMBOLS AND DESCRIPTION

Notation	Description
d(*m*, *n*)	The network distance from *m* to *n*
OPT	Optimal result set for CSKQ on road networks
V	Approximate result set for CSKQ on road networks
$e_{i,j}$	The edge between nodes n_i and n_j
q.l	The spatial location where a query *q* is issued
q.key	A set of queried keywords w.r.t. a query *q*
q.key[*i*]	The *i*-th queried keyword w.r.t. a query *q*

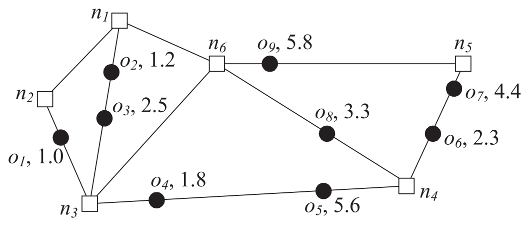


Fig. 2. Example of a road network.

III. PRELIMINARIES

In this section, we first formalize the problem of collective spatial keyword queries on road networks, and prove that it is NP-complete. Then, we introduce a CCAM [22] index structure employed in this paper as a disk-based index to store objects on a specified road network. Table I summarizes the symbols used frequently throughout the paper.

A. Problem Formulation

A road network is defined as an undirected weighted graph $G = (N, E, W)$, where N is a set of vertices/nodes, E is a set of edges, and W is a set of edge weights which represent the length of the edges. A point of interest (POI) is an object which contains both spatial and textual information. A spatial description of a POI captures its location on the road network, expressed in the form of tuple $(n_i, n_j, dist)$. Here, n_i and n_j denote the nodes of edge $e_{i,j}$ in which a POI is located on, and $dist$ refers to the distance from the POI to a node n_i along the edge $e_{i,j}$. In this paper, we suppose $i < j$. If the shortest path distance from a POI to the node is 0, then a node of the road network may include a POI; otherwise it does not include one. Given two objects m and n, the shortest path distance between m and n is denoted as $d(m, n)$. For instance, Fig. 2 shows an example of a road network. Nodes are denoted by squares, POIs are denoted by solid dots, and the digit next to a POI in an edge $e_{i,j}$ represents the distance from the POI to n_i. Take POI o_1 in Fig. 2 as an example. The position of o_1 can be denoted as $(n_2, n_3, 1.0)$, indicating that o_1 lies on the edge $e_{2,3}$, and it is 1.0 unit away from the node n_2. The distance of the shortest path from o_1 to o_6 is computed as $d(o_1, o_6) = d(o_1, n_3) + d(n_3, n_4) + d(n_4, o_6)$.

We then formally define the problem of *collective spatial keyword queries on road networks* in Definition 1.

Definition 1 (CSKQ on Road Networks): Given a set D of POIs and a query point $q = \{q.l, q.key\}$ where $q.l$ denotes a query location, and $q.key$ is a set of queried keywords, let $\cup V$ represent the union of object sets with each V consisting of a set of the objects that collectively cover the queried keywords, i.e., $\forall V \in \cup V, q.key \subseteq \cup_{o \in V} o.key \wedge \forall o \in V, q.key \not\subseteq \cup_{o' \in V - \{o\}} o'.key$. A *collective spatial keyword query on road networks* aims to find a set *OPT* of objects, such that $OPT \in \cup V \wedge \forall V \in \cup V - OPT, \text{COST}(OPT) \leq \text{COST}(V)$ with function COST( ) defined in Equation (1).[1]

$$\text{COST}(V) = \alpha \times \max_{o \in V} d(q, o) + (1 - \alpha) \times \max_{o_1, o_2 \in V} d(o_1, o_2) \tag{1}$$

■

In Equation (1), $\max_{o \in V} d(q, o)$ is the maximum network distance from a query point q to any object in V, and $\max_{o_1, o_2 \in V} d(o_1, o_2)$ is the maximum network distance between any two POIs in V. $\alpha (\in [0, 1])$ is a user specified parameter that balances the relative importance between $\max_{o \in V} d(q, o)$ and $\max_{o_1, o_2 \in V} d(o_1, o_2)$. Intuitively, if users prefer the objects in V that are close to each other, α is assigned smaller than 0.5. In contrast, if users prefer the objects in V that are close to the query location, α is assigned bigger than 0.5. However, users may feel confused if we ask them for the value of α when they issue a query. It could be more intuitively to give a distance constraint (e.g., retrieve the POIs that are within a maximum distance $d_{\max}$) for the users. In this case, we explain how the value of α should be chosen based on the distance constraint $d_{\max}$. Specifically, given a CSKQ q, we first assign $\alpha = 0.5$ and then find a set V of objects for q. Its cost is $\text{COST}(V) = 0.5 \times \max_{o \in V} d(q, o) + 0.5 \times \max_{o_1, o_2 \in V} d(o_1, o_2)$ according to Definition 1. If users prefer all the objects that are inside the maximum distance $d_{\max}$, in the extreme case, a set V' of objects is returned such that the shortest path distances from all the objects in V' to q (namely, $\max_{o \in V'} d(q, o)$) are $d_{\max}$, and the inter-object distance (i.e., $\max_{o_1, o_2 \in V'} d(o_1, o_2)$) is 0. Hence, we have $\text{COST}(V') = \alpha \times \max_{o \in V'} d(q, o) = \alpha \times d_{\max}$. If V' is a better choice than V, we have $\text{COST}(V) > \text{COST}(V')$, i.e., $\text{COST}(V) > \alpha \times d_{\max}$. Finally, we have $\alpha < \text{COST}(V)/d_{\max}$. In other words, if users choose a value smaller than $\text{COST}(V)/d_{\max}$ for α, it is guaranteed that our algorithms return the proper result for q.

Without loss of generality, in this paper, we give equal weights to the two components as with [4], [16], indicating that $\max_{o \in V} d(q, o)$ and $\max_{o_1, o_2 \in V} d(o_1, o_2)$ are equally important. Nonetheless, the proposed algorithms remain applicable when α is enabled. Based on Definition 1, we prove that the CSKQ on road networks is *NP-complete* in Lemma 1 below.

Lemma 1: The collective spatial keyword query on road networks is NP-complete.

Proof: Consider the Boolean satisfiability (SAT) problem. The SAT problem is the problem of determining whether there is an interpretation satisfying a given Boolean formula. Consider a conjunction normal form (CNF) $\Phi = C_1 \wedge C_2 \wedge \cdots \wedge C_i \wedge \cdots \wedge C_n$. An instance of clause C_i can be denoted as $C_i = x_{i1} \vee -x_{i2} \vee \cdots \vee -x_{ij} \vee \cdots \vee x_{im}$, in which x_{ij} is a variant of set $\{x_1, x_2, \ldots, x_m\}$ denoted as a positive character, and $-x_{ij}$ is denoted by a negative character. The length of the

[1]COST( ) function is application-dependent, and it can vary accordingly to serve different application needs.

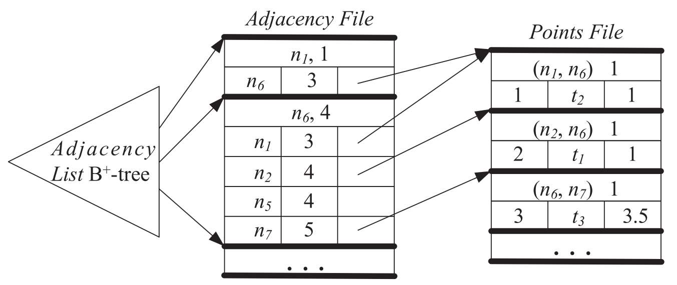
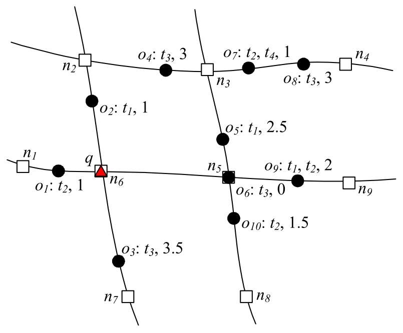


Fig. 3. The CCAM index structure on the road network shown in Fig. 4.



(a)

Edge	Length
$e_{1,6}$	3
$e_{2,3}$	4
$e_{2,6}$	4
$e_{3,4}$	4
$e_{3,5}$	4
$e_{5,6}$	4
$e_{5,8}$	5
$e_{5,9}$	4
$e_{6,7}$	5

(b)

Fig. 4. A running example. (a) Road network and POIs. (b) The keywords of objects.

clause C_i is denoted as m, meaning that C_i has m characters. For any clause C_i, if there is a positive character x_{ij}, we say C_i is satisfied. An SAT problem is actually the problem of determining whether there is an interpretation satisfying the CNF.

Next, we reduce the SAT problem to CSKQ on road networks. Each clause C_i is reduced to a queried keyword, and x_j is reduced to a POI $o \in V$. Thus, x_{ij} represents the i-th keyword in the j-th object. In this case, a CNF can be denoted as: $q.key = (x_{11} \vee -x_{12} \vee x_{13} \vee \cdots \vee x_{1m}) \wedge (-x_{21} \vee x_{22} \vee x_{23} \vee \cdots \vee x_{2m}) \wedge \cdots \wedge (x_{n1} \vee x_{n2} \vee x_{n3} \vee \cdots \vee x_{1m})$, indicating that the first keyword exists in $o_1, o_3, \ldots, o_m$, the second keyword exists in $o_2, o_3, \ldots, o_m$, and so on. Hence, there is a solution for the SAT problem that satisfies the CNF if and only if there is a solution to our problem studied in this paper. ∎

B. Index Structure

Although memory space becomes bigger and cheaper, we still employ a disk-based CCAM index structure to store the objects on road networks such that we can tackle the CSKQ on road networks for larger datasets. The CCAM index structure is to group network nodes based on their connectivity and distances, as proposed in [22]. Fig. 3 depicts a graphical illustration of an adjacency file, and a points file along with the index for our example road network is shown in Fig. 4. CCAM index structure allows efficient access to the adjacency lists and POIs that are stored in the adjacency file and the points file, respectively. A B^+-tree is employed to facilitate efficient access to adjacency files.

All the POIs on the same edge form one group, and the points file is used to collect and store POI groups. For every group, we need to maintain the edge where the group of POIs are located and the number of POIs. Subsequently, for each POI p on this edge, we store p's ID, p's associated set of keywords, and the distance from p to the node with smaller ID. A group of POIs are preserved in ascending order of their offset distances to the node having smaller ID. Given a node n_i, all its adjacent nodes form n_i's adjacency list. At the beginning of the adjacency list, we maintain the total number of n_i's adjacent nodes. Then, for every adjacent node n_j, we store its ID, the edge distance between nodes n_i and n_j, and a pointer to its POI group in the points file. If there is *no* POI on this edge, a NULL pointer is kept.

IV. Approximate Algorithms

Recall that the CSKQ on road networks is NP-complete. Thus, it is expected that the performance of an *exact* algorithm might not be good, in terms of running time and I/O cost. In light of this, we present two *approximate* algorithms with guaranteed approximation errors in this section.

A. Network Expansion Based Algorithm

For a specified query object $q = \{q.l, q.key\}$, the main idea of the *network expansion based* (NEB) *algorithm* is to find a set of POIs that are close to the given query location $q.l$ and cover the queried keywords $q.key$. Algorithm 1 shows the pseudo-code of NEB algorithm. NEB first locates the edge $e_{i,j}$ that q is located (line 2). A min-priority queue U is utilized to keep tracks of the edges that have been visited, and such edges are sorted in ascending order of their distances to q. Whenever a POI o having $o.key \cap q.key \neq \varnothing$ is retrieved, o is added to a result set V, and $q.key$ is updated to $(q.key - o.key)$ (lines 8–10). The expansion proceeds until $q.key$ is empty, meaning that all the queried keywords have been fully covered by the POIs in the result set V (line 11), and the approximate result set V is returned (line 12). We proceed to illustrate an example to show the procedure of NEB algorithm.

Algorithm 1 NEB Algorithm: NEB $(\boldsymbol{C}, \boldsymbol{q})$

```
Input: a CCAM index structure $C$ on a dataset $D$, a query
       object $q$ in the form of $\{q.l, q.key\}$
Output: an approximate result set $V$
 1: $V \leftarrow \varnothing$
 2: locate the edge $e_{i,j}$ that $q$ is located on (assume $q$ is closer
    to $n_i$)
 3: $U \leftarrow \{(e_{q,i}, e_{q,i}.d), (e_{q,j}, e_{q,j}.d)\}$ // edges in $U$ are sorted
    in ascending order of their distance to $q.l$
 4: while not $U$.IsEmpty( ) do
 5:     $e_{a,b} \leftarrow U$.dequeue( )
 6:     for each edge $e_{b,c}$ unvisited in the edge set $E$ do
 7:         $U$.enqueue($e_{b,c}, d(q, n_c)$)
 8:     for each POI $o$ on $e_{a,b}$ do
 9:         if $q.key \cap o.key \neq \varnothing$ then
10:             $V \leftarrow V \cup \{o\}$ and $q.key \leftarrow q.key - o.key$
11:     if $q.key = \varnothing$ then break
12: return $V$
```

Example 1: In Fig. 4, we depict a road network and the lengths of edges. Each rectangle represents a node, and every circle denotes the POI together with its keywords and the distance to the node with smaller ID. Given a query object q located at node n_6 with queried keywords $q.key = \{t_1, t_2, t_3\}$, the closest POI to $q.l$ is o_1, which contains keyword t_2. Thus, o_1 is added to V, and $q.key$ is updated to $\{t_1, t_3\}$. The next nearest object is o_2, and $q.key$ is updated to $\{t_3\}$. Then, o_3 is found, and $q.key$ is updated to $\varnothing$. Here, NEB algorithm stops with $\text{COST}(V) = 3.5 + 6.5 = 10$. Finally, the approximate result set $V = \{o_1, o_2, o_3\}$ is returned. ■

Next, we prove that $\text{COST}(V)$ of the approximate result set V returned by NEB algorithm is at most 3 times of $\text{COST}(OPT)$, and the time complexity of NEB algorithm is $O(|q.key| \cdot \log |E|)$, as stated in Lemma 2 and Lemma 3, respectively.

Lemma 2: The cost of the approximate result set V returned by NEB algorithm is at most 3 times of the cost of the optimal result set OPT, i.e., $\text{COST}(V) \le 3 \times \text{COST}(OPT)$.

Proof: Let $d = \max_{o \in V} d(q, o)$. For any two objects $o_1, o_2 \in V$, the shortest path from o_1 to q and the shortest path from q to o_2 form a path from o_1 to o_2 with the length bounded by $2d$. Consequently, we have $\max_{o_1, o_2 \in V} d(o_1, o_2) \le 2d$ and $\text{COST}(V) = \max_{o \in V} d(q, o) + \max_{o_1, o_2 \in V} d(o_1, o_2) \le 3d$. On the other hand, we have $\text{COST}(OPT) \ge d$. This is because, if $\text{COST}(OPT) = \max_{o \in OPT} d(q, o) + \max_{o_1, o_2 \in OPT} d(o_1, o_2)$ is smaller than d, we have $\max_{o \in OPT} d(q, o) < d$, meaning that NEB algorithm terminates before reaching a POI o with $d(o, q) = d$, which contradicts with our assumption that $d = \max_{o \in V} d(q, o)$. Thus, $\text{COST}(V) \le d + 2 \times d \le 3 \times \text{COST}(OPT)$ holds. ■

Lemma 3: The time complexity of NEB algorithm is $O(|q.key| \cdot \log |E|)$.

Proof: For every queried keyword t, NEB algorithm finds the nearest POI containing t, and all those POIs form a result set V. Since NEB algorithm expands the road network only once and it utilizes a link list to store the edges of a road network, the time complexity of NEB algorithm is $O(|q.key| \cdot \log |E|)$. ■

NEB minimizes the distances from a query location to answer POIs. Nevertheless, it does not take into account the proximity between POIs, and hence, $\max_{o_1, o_2 \in V} d(o_1, o_2)$ in Equation (1) might not be optimized. The intuition behind NEB algorithm is that POIs that are close to the query location may be close to each other as well.

B. Iterative NEB Based Algorithm

Without the consideration of the proximity of the objects in V, NEB algorithm has a loose approximation bound. In this section, we propose another approximate algorithm, namely, *iterative NEB based* (INB) *algorithm*, which has a tighter approximation bound. INB considers both the road network distance from a query location to each object in V and the inter-POI network distances between the objects in V.

The pseudo-code of INB algorithm is shown in Algorithm 2. Given a query object q, INB first invokes NEB algorithm to obtain an original result set V, and $\text{COST}(V)$ serves as the initial minimal cost (lines 1–2). Let o_e be a POI in V that is the farthest to q, and key_e be the set of queried keywords exclusively contributed by o_e, i.e., $\max_{o \in V} d(q, o) = d(o_e, q)$ and $key_e = o_e.key \cap (q.key - \cup_{o \in (V - o_e)} o.key)$ (lines 3–4). Note that, based on NEB, POI o_e is actually the closest object to q that contains the set of queried keywords key_e. INB then issues a new query q_e at the location of o_e with original queried keywords, i.e., $q_e.l = o_e.l$ and $q_e.key = q.key$. NEB is called again to find the result set V_e for the new query q_e, and the cost of V_e is denoted as $\text{COST}(V_e)$. If $\text{COST}(V_e) < \text{COST}(V)$, V_e becomes the current best result set for the query object q (lines 8–9). Then, the algorithm retrieves incrementally the next nearest POI o'_e to $q.l$ with its keywords crossing key_e. For each such o'_e, INB issues a new query q'_e at o'_e with $q'_e.l = o'_e.l$ and $q'_e.key = q.key$, and uses NEB to find the approximate result set V'_e. It then compares V'_e against V, and replaces V with V'_e if $\text{COST}(V'_e) < \text{COST}(V)$ (lines 5–12). The above process repeats until all the POIs o having $o.key \cap key_e \ne \varnothing$ and $d(o, q) < \text{COST}(V)$ are evaluated. Finally, INB returns the result set V (line 13).

Algorithm 2 INB Algorithm: INB (C, q)

```
Input: a CCAM index structure $C$ on a dataset $D$, a query
       object $q$ in the form of $\{q.l, q.key\}$
Output: an approximate result set $V$
 1: $V \leftarrow$ NEB$(C, q)$
 2: COST$(V) \leftarrow$ the cost of $V$
 3: $o_e \leftarrow$ the furthest POI to $q.l$ in $V$
 4: $key_e \leftarrow o_e.key \cap q.key - \cup o.key$ (for all $o (\ne o_e) \in V$)
 5: while $d(q, o_e) <$ COST$(V)$ do
 6:   $q_e.l \leftarrow o_e.l$ and $q_e.key \leftarrow q.key$
 7:   $Ve \leftarrow$ NEB$(C, q_e)$ and COST$(V_e) \leftarrow$ the cost of $V_e$
 8:   if COST$(V_e) <$ COST$(V)$ then
 9:     COST$(V) \leftarrow$ COST$(V_e)$ and $V \leftarrow V_e$
10:   if all the POIs $o$ with $o.key \cap key_e \ne \varnothing$ have been
      evaluated then
11:     break
12:   else $o_e \leftarrow$ the next nearest POI $o$ to $q.l$ with $o.key \cap$
      $key_e \ne \varnothing$
13: return $V$
```

Example 2: Recall that the query object q and the road network in Example 1. $\{o_1, o_2, o_3\}$ is returned by NEB algorithm as the initial result set V with $o_e = o_3$ and $key_e = \{t_3\}$. A new query object q_e is created at the position of o_3, and NEB(C, q_e) is invoked. $V_e = \{o_1, o_2, o_3\}$ is returned, and $\text{COST}(V_e) = 3.5 + 6.5 = 10$. o_6 is the next nearest POI to $q.l$ containing keyword t_3, and thus, the next result set for q is $V_e = \{o_5, o_6, o_{10}\}$ with $\text{COST}(V_e) = 5.5 + 3 = 8.5$. Result set V and $\text{COST}(V)$ are updated. In the next iteration, o_e is updated to o_4, and $\{o_4, o_5, o_7\}$ is returned as V_e with $\text{COST}(V_e) = 9 + 3.5 = 12.5$. Then, o_8 is retrieved as the next o_e. Since $d(q, o_8) = 11 > \text{COST}(V)$, the algorithm terminates. $V = \{o_5, o_6, o_{10}\}$ is returned as the final query result. ■

Next, we study the approximation bound of INB algorithm.

Lemma 4: For any two POIs, o_1 and o_2, if they both are connected with a POI o, we have $d(o_1, o_2) \le d(o_1, o) + d(o, o_2)$.

Proof: Obviously, if the shortest path from o_1 to o_2 passes o, we have: $d(o_1, o_2) = d(o_1, o) + d(o, o_2)$. Otherwise, we have $d(o_1, o_2) \leq d(o_1, o) + d(o, o_2)$ according to the triangle inequality. ■

Lemma 5: Let V be the result set returned by NEB algorithm with COST(V). For any POI $o \in D$, if $d(q, o) \geq$ COST(V), o cannot contribute to *OPT*.

Proof: Assume, to the contrary, that o with $d(q, o) \geq$ COST(V) belongs to *OPT*, i.e., $d(q, o) \geq$ COST(V) and $o \in OPT$. Then, we have COST$(OPT) > \max_{o' \in OPT} d(q, o') \geq d(q, o) \geq$ COST(V), which contradicts the fact that COST $(OPT) \leq$ COST(V). Consequently, our assumption is invalid, and the proof completes. ■

Based on Lemma 5, we can prune the objects with their network distances to q no less than COST(V), which offers the theoretical explanation of the code depicted in line 5 of Algorithm 2. It is also clear that COST$(OPT) \leq$ COST(V). In the following, we prove the lower bound of COST(OPT).

Lemma 6: Given a query object q and an approximate result set V found by NEB algorithm, let o_e be a POI in V that is the farthest to q, and key_e be the set of queried keywords exclusively contributed by o_e, i.e., $key_e \subseteq (q.key \cap o_e.key) \wedge \forall\, o(\neq o_e) \in V, o.key \cap key_e = \varnothing$. Assume that o_a is a POI in *OPT* with $o_a.key \cap key_e \neq \varnothing$, and o_f is the furthest POI to o_a in the set V_a that is returned by NEB$(C, q(o_a.l, q.key))$. It holds that COST$(OPT) \geq d(q, o_a) + d(o_a, o_f)$.

Proof: Obviously, $d(q, o_a) \leq \max_{o \in OPT} d(q, o)$, and it refers to the lower bound of the maximum network distance from q to any POI in OPT. Then, we explain why $d(o_a, o_f)$ represents the lower bound of the inter-POI network distance of OPT. V_a is returned by NEB$(C, q(o_a.l, q.key))$, and POI o_f in V_a must contain at least one exclusive queried keyword t_f, i.e., $t_f \in q.key \wedge t_f \subseteq o_f.key \wedge \forall\, o(\neq o_f) \in V, t_f \notin o.key$. Based on NEB, POI o_f is the one closest to o_a that has the keyword t_f. In other words, let object o_b be the POI in OPT that contains the queried keyword t_f, and we have $\max_{o_1, o_2 \in OPT} d(o_1, o_2) \geq d(o_a, o_b) \geq d(o_a, o_f)$. As a result, $d(o_a, o_f)$ is the lower bound of the maximum inter-POI network distance. Hence, COST$(OPT) \geq d(q, o_a) + d(o_a, o_f)$ holds, and the proof completes. ■

Next, we analyze the lower and upper bounds of COST(V_e).

Lemma 7: Given a query object q and an approximate result set V found by NEB algorithm, let o_e be the POI in V that is the farthest to q, and key_e be the set of queried keywords exclusively contributed by o_e, i.e., $key_e \subseteq q.key \cap o_e.key \wedge \forall\, o(\neq o_e) \in V, o.key \cap key_e = \varnothing$. Let V_e denote the approximate result set retrieved by NEB when the query object is located at the position of o_e, and o_f be the furthest POI from o_e in V_e. It holds that:

$$d(q, o_e) + d(o_e, o_f) \leq \text{COST}(V_e) \leq d(q, o_e) + 3 \times d(o_e, o_f).$$

Proof: We prove this inequation in two steps below:

(1) As $d(q, o_e) \leq \max_{o \in V_e} d(q, o)$, it is the lower bound of the maximum network distance from q to any POI in V_e, and $d(o_e, o_f) \leq \max_{o_1, o_2 \in V_e} d(o_1, o_2)$, and thus, it is the lower bound of the maximum inter-POI network distance. Hence, $d(q, o_e) + d(o_e, o_f) \leq$ COST(V_e) holds.

(2) Since both o_f and o_e are connected to q, we have $d(q, o_e) + d(o_e, o_f) \geq d(q, o_f)$ according to Lemma 4. Similarly, for any POI o in V_e, we have $d(q, o_e) + d(o_e, o) \geq d(q, o)$. Therefore, we have $\max_{o \in V_e} d(q, o) \leq d(q, o_e) + d(o_e, o) \leq d(q, o_e) + d(o_e, o_f)$, indicating that $d(q, o_e) + d(o_e, o_f)$ is the upper bound of the maximum network distance from any POI in V_e to q. Also, we know that the upper bound of inter-POI network distance is $2 \times d(o_e, o_f)$, because o_f is the furthest POI from o_e in the approximate result set V_e returned by NEB when the query object is located at the position of o_e, i.e., $\max_{o_1, o_2 \in V_e} d(o_1, o_2) \leq 2 \times d(o_e, o_f)$. Consequently, COST$(V_e) = \max_{o \in V_e} d(q, o) + \max_{o_1, o_2 \in V_e} d(o_1, o_2) \leq d(q, o_e) + d(o_e, o_f) + 2 \times d(o_e, o_f) = d(q, o_e) + 3 \times d(o_e, o_f)$. The proof completes. ■

We have proved the lower and upper bounds of both approximate result set and optimal result set. In the sequel, we show that INB algorithm gives a 2-factor approximation for the CSKQ on road networks.

Lemma 8: Given a query object q on a road network, let V be the approximate result set returned by INB algorithm, and OPT be the optimal result set, it is guaranteed that COST$(V) \leq 2 \times$ COST(OPT).

Proof: Assume that V_1 is the result set returned by NEB algorithm. Let o_e in V_1 be the furthest POI to q, and key_e the set of exclusively queried keywords covered by o_e, i.e., $key_e \subseteq q.key \cap o_e.key \wedge \forall\, o(\neq o_e) \in V_1, o.key \cap key_e = \varnothing$. NEB algorithm guarantees that o_e is the nearest POI to q that contains keywords set key_e. Let V_e be the approximate result set returned by NEB for the same query issued at the POI o_e. In addition, we assume that OPT is the optimal result set, and V is the approximate result set for the query object q returned by INB. Based on INB, we have COST$(V) \leq$ COST(V_1). On the other hand, we are for sure that COST$(V_1) \leq 3d$ with $d = \max_{o \in V_1} d(q, o)$. Thus, we have COST$(V) \leq$ COST$(V_1) \leq 3d(q, o_e)$ as $o_e \in V_1$.

Based on Lemma 6, we are certainly that COST$(OPT) \geq d(q, o_a) + d(o_a, o_f)$, with o_a the POI in OPT having $o_a.key \cap key_e \neq \varnothing$, and o_f the furthest POI to o_a in V_a returned by NEB algorithm for the same query object issued at o_a.

Based on what we present above, we have:

$$\frac{\text{COST}(V)}{\text{COST}(OPT)} \leq \frac{3d(q, o_e)}{d(q, o_a) + d(o_a, o_f)} = 2 + \frac{3d(q, o_e) - 2\,(d(q, o_a) + d(o_a, o_f))}{d(q, o_a) + d(o_a, o_f)}$$

(1) If $3d(q, o_e) \leq 2(d(q, o_a) + d(o_a, o_f))$, then COST$(V)/$ COST$(OPT) \leq 2$.

(2) Otherwise, $3d(q, o_e) > 2(d(q, o_a) + d(o_a, o_f))$. Based on NEB algorithm, it is guaranteed that the POI o_e is the closest POI to q containing keyword set key_e, i.e., $d(q, o_a) \geq d(q, o_e)$. In other words, $3d(q, o_e) > 2(d(q, o_a) + d(o_a, o_f)) \rightarrow 3d(q, o_a) \geq 3d(q, o_e) > 2(d(q, o_a) + d(o_a, o_f)) \rightarrow d(q, o_a) > 2d(o_a, o_f)$. On the other hand, we know that $d(q, o_a) \leq$ COST$(OPT) \leq$ COST(V), and hence, it is certainly that POI satisfies the condition listed in lines 10–12 of Algorithm 2, and NEB is invoked for the query object located at o_a. Let V_a denote the approximate result set returned by NEB for the query object located at o_a, and thus, we have COST$(OPT) \leq$ COST$(V) \leq$ COST(V_a).

According to Lemma 7, we have $\text{COST}(V_a) \le d(q, o_a) + 3d(o_a, o_f)$. Based on the above discussion, we have

$$\begin{aligned}\frac{\text{COST}(V)}{\text{COST}(OPT)} &\le \frac{d(q,o_a) + 3 \times d(o_a,o_f)}{d(q,o_a) + d(o_a,o_f)}\\ &= 2 + \frac{d(o_a,o_f) - d(q,o_a)}{d(q,o_a) + d(o_a,o_f)}\\ &\le 2 + \frac{0.5 \times d(q,o_a) - d(q,o_a)}{d(q,o_a) + d(o_a,o_f)}\\ &= 2 - \frac{0.5 \times d(q,o_a)}{d(q,o_a) + d(o_a,o_f)} \le 2\end{aligned}$$

Therefore, the proof completes. ■

INB algorithm gives a 2-factor approximation for the CSKQ on road networks. Specifically, INB algorithm enhances NEB algorithm by iterating NEB algorithm and obtaining multiple approximate result sets. Among all the approximate result sets, it returns the one with the smallest cost. Thus, the approximate solution returned by INB algorithm is no worse than the one returned by NEB algorithm. In addition, the time complexity of INB algorithm is as follows.

Lemma 9: The time complexity of INB algorithm is $O(|o_t + 1| \cdot |q.key| \cdot \log |E|)$.

Proof: Given a query object q and an approximate result set V found by NEB, let o_e be the POI in V that is the farthest to q, and key_e be the set of queried keywords exclusively contributed by o_e. Assume that there are $|o_t|$ POIs in the dataset whose keyword sets are overlapped with key_e. In the worst case, the *while-loop* in INB algorithm (lines 5–12) has to be executed $|o_t|$ times. Based on Lemma 3, we know that the time complexity of NEB algorithm is $O(|q.key| \cdot \log |E|)$. Hence, the total cost of INB is $O(|o_t + 1| \cdot |q.key| \cdot \log |E|)$, with 1 for NEB called in line 2 of Algorithm 2. The proof completes. ■

V. Exact Algorithm

We have proved that the CSKQ on road networks is NP-complete. Therefore, it is a challenge to develop an exact algorithm. However, the time cost of exact algorithm is acceptable when the queried keyword set is small (e.g., $|q.key| = 3$ or $|q.key| = 4$), and it is desirable to find the optimal result set in some applications. As a result, in this section, we propose an exact algorithm, i.e., the *sliding window* (SW) *algorithm*, for processing the CSKQ on road networks.

Before presenting SW algorithm, we first give a brute force algorithm since it serves as the basis of SW. A brute-force algorithm is to first retrieve the POIs that contain at least one queried keyword, then consider all the combinations formed by those POIs, and finally return the result set with the minimum cost. To be more specific, in the first step, NEB algorithm is called to find an approximate result set V, and $\text{COST}(V)$ serves as the upper bound of $\text{COST}(OPT)$. A min-priority queue Q_{tmp} is used to maintain the POIs having the network distances from the POIs to q are smaller than $\text{COST}(V)$ and meanwhile their keywords overlap with the queried keywords, i.e., $\forall\, o \in Q_{\text{tmp}}$, $d(o, q) < \text{COST}(V) \wedge o.key \cap q.key \neq \varnothing$. In addition, we also maintain another auxiliary list set, i.e., $L[\ ]$, with each list $L[i]$ storing the POIs o in Q_{tmp} whose keyword sets include the i-th queried keyword, i.e.,$\forall\, o \in L[i]$, $q.key[i] \in o.key$. Here, we use $q.key[i]$ to refer to the i-th queried keyword in $q.key$. A set $V' = \cup_{i \in [1, |q.key|]} o_i$ with $o_i \in L[i]$ forms a result set. We evaluate all possible V's and the one with the smallest cost is guaranteed to be the optimal result. Assume that there are $|q.key|$ queried keywords, and for every queried keyword, there are n POIs containing that keyword and meanwhile having their network distances to q bounded by $\text{COST}(V)$. Then, this brute-force algorithm needs to evaluate as many as $n^{|q.key|}$ result sets. This also inspires us that we have to cut down the value of n if we want to improve search performance, since $|q.key|$ is specified by the query. One efficient way to reduce the value of n is to use a tighter bound $\text{COST}(V)$. Our SW algorithm is motivated by this observation.

The basic idea of SW algorithm is that, the POIs in the min-priority queue Q_{tmp} are ordered based on their network distances to q, and the adjacent POIs in Q_{tmp} shall have similar distances to q. According to this property, we assume that these adjacent POIs might form a result set with few cost. SW algorithm is to scan the POIs in Q_{tmp} based on sliding windows even though the size of sliding window in SW algorithm is not fixed. The main objective is to find a result set V with relatively small cost early in order to prune away all the POIs with their distances to q larger than $\text{COST}(V)$. After checking all the result sets formed by adjacent POIs in Q_{tmp}, we evaluate the remaining combinations like the brute-force algorithm does. As demonstrated in our experimental evaluation (presented in Section VI), SW algorithm can reduce the number of the combinations evaluated.

Algorithm 3 shows the pseudo-code of SW algorithm. It first initializes all the parameters (line 1). Q_{tmp} is a min-priority queue that keeps all the POIs that may contribute to the result set, sorted in ascending order of their network distances to q. $L[\ ]$ is a set of $|q.key|$ lists, and each list $L[i]$ maintains the set of POIs in Q_{tmp} whose keyword sets contain the i-th queried keyword $q.key[i]$. S_k is a keyword set that is the union of the keywords of all the POIs in current result set V'. Parameters *start* and *end* refer to the start and end positions of current sliding window along the queue Q_{tmp}, respectively. Parameter *flag* is a Boolean indicator whose value decides whether the network expansion shall be stopped. The algorithm then locates the query object q on one edge $e_{i,j}$, and inserts edge $e_{q,i}$ and edge $e_{q,j}$ as initial edges into a queue U. U is a min-priority queue that maintains the set of edges we are going to explore next in order to expand the network (lines 2–3). Later, SW invokes NEB algorithm to find an approximate result set V, and the cost of V serves as the upper bound of $\text{COST}(OPT)$ (line 4). Thereafter, SW employs a while-loop to perform sliding window based scanning for the POIs maintained in Q_{tmp} (lines 5–27). It first checks whether the current sliding window has reached the end of Q_{tmp} via checking $Q_{\text{tmp}}[end]$. If the end is reached (i.e., $Q_{\text{tmp}}[end] = \text{NULL}$), the network expansion is triggered (lines 7–18). SW expands the network, and visits the edges that have the distances to q shorter than $\text{COST}(V)$. If at least one POI o having $o.key$ overlapped with the queried keywords is retrieved (i.e., $q.key \cap o.key \neq \varnothing$), the expansion can be terminated (lines 13–15).

Algorithm 3 Sliding Window algorithm: SW(C, q)

```
Input: a CCAM index structure $C$ on a dataset $D$, a query
       object $q$ in the form of $\{q.l, q.key\}$
Output: an optimal result set $OPT$
1: $Q_{\text{tmp}} \leftarrow \varnothing$, $L[\ ] \leftarrow \varnothing$, $S_k \leftarrow \varnothing$, $start \leftarrow 0$, $end \leftarrow 0$,
   $flag \leftarrow false$, $V' \leftarrow \varnothing$
2: locate the edge $e_{i,j}$ that $q$ is located on(assume $q$ is closer
   to $n_i$)
3: $U \leftarrow \{(e_{q,i}, e_{q,i}.d), (e_{q,j}, e_{q,j}.d)\}$ // edges in $U$ are sorted
   in ascending order of their distances to $q.l$
4: $V \leftarrow$ NEB$(C, q)$ and COST$(V) \leftarrow$ the cost of $V$
5: while $d(q, \text{head}(U)) <$ COST$(V)$ do
6:   while $q.key \not\subset S_k$ do
7:     if $Q_{\text{tmp}}[end] =$ NULL then // network expansion
8:       while $U$ is not empty and $flag$ is $false$ do
9:         $e_{a,b} \leftarrow U$.dequeue()
10:        if $d(q, n_a) \geq$ COST$(V)$ and $d(q, n_b) \geq$ COST$(V)$
           then break
11:        for each unvisited edge $e_{b,c}$ that is adjacent to
           edge $e_{a,b}$ do
12:          $U$.enqueue$(e_{b,c}, d(q, n_c))$
13:        for each POI $o$ on $e_{a,b}$ do
14:          if $d(q, o) <$ COST$(V)$ and $q.key \cap o.key \neq \varnothing$
             then
15:            $Q_{\text{tmp}}$.enqueue$(o)$ and $flag \leftarrow true$
16:            for each queried keyword $t_i$ in $q.key$ do
17:              if $t_i \in o.key$ then $L[i]$.enqueue$(o)$
18:        $flag \leftarrow false$
19:    else // sliding window based POIs scanning
20:      $o \leftarrow Q_{\text{tmp}}[end]$ and $end \leftarrow end + 1$
21:      if $(q.key - S_k) \cap o.key \neq \varnothing$ then
22:        $V' \leftarrow V' \cup o$ and $S_k \leftarrow o.key \cup S_k$
23:        if $q.key \not\subset S_k$ then continue
24:        else if (COST$(V') \leftarrow$ the cost of $V'$) $<$ COST$(V)$
           then
25:          COST$(V) \leftarrow$ COST$(V')$ and $V \leftarrow V'$
26:        else break
27:  $start \leftarrow start + 1$, $end \leftarrow start$, $S_k \leftarrow \varnothing$, $V' \leftarrow \varnothing$
28: while not all the combinations are considered do
29:   pick one POI from each $L[t]$ to form one result set $V'$
30:   if the result set $V'$ has not been considered then
31:     if COST$(V') <$ COST$(V)$ then
32:       COST$(V) \leftarrow$ COST$(V')$ and $V \leftarrow V'$
33: $OPT \leftarrow V$
34: return $OPT$
```

Otherwise, the end of Q_{tmp} is not yet reached, and SW incrementally grows *end* to increase the size of current sliding window (lines 19–26). Note that, initially, parameters *start* and *end* both refer to the same element in Q_{tmp}, and the size of the sliding window is only one. Then, every time, when the algorithm invokes the inner while-loop (lines 6–26) to increase *end* value by 1, it also grows the size of the sliding window by 1. This process proceeds until the POIs retrieved during the current sliding window can cover all the queried keywords (i.e., $q.key \subseteq S_k$). It then verifies the current result set V', and

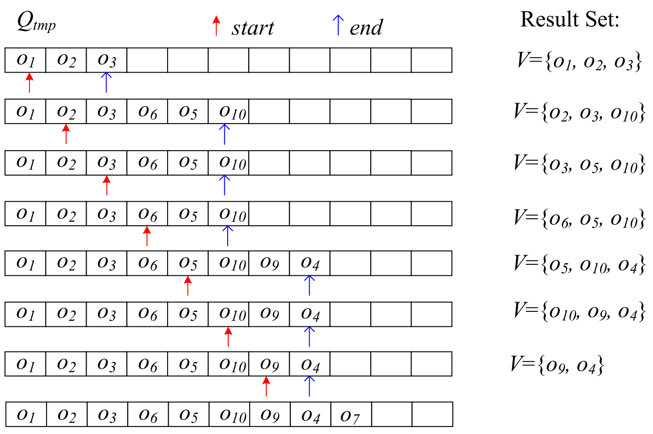


Fig. 5. The procedure of SW algorithm.

V' replaces the currently best result set V if COST$(V') <$ COST(V) (lines 24–25). We re-start the above process by trying a new sliding window started at position $start + 1$ (line 27), and repeat the previous process.

After checking the adjacent POIs in Q_{tmp}, we also need to check other result sets in order to guarantee the accuracy of SW algorithm. The algorithm picks one POI from each list in $L[\ ]$ to form a result set V' that has not yet been evaluated, and replaces currently best result set V if COST$(V') <$ COST(V) (lines 28–32). Here, we adopt an idea to reduce the times of network expansion when computing the maximal road network distances among objects. Specifically, for each object $o \in L[i]$, SW algorithm expands the road network at the location of o until all the other objects $o' \in \cup_{j=1,2\ldots|q.key|\wedge j\neq i} L[j]$ are visited. In this manner, SW algorithm maintains all the road network distances between the objects in $L[\ ]$, and decreases the times of network expansion from $\prod_{j=1}^{|q.key|} L[j]$ to $\sum_{j=1}^{|q.key|} L[j]$. Finally, the result set OPT is returned (line 34), and it is guaranteed to be the optimal solution since the algorithm has evaluated all the possible result sets.

Example 3: We utilize an example to illustrate how SW algorithm can support exact CSKQ processing on road networks. Assume that a query q is issued, as shown in Example 1. SW first initializes all the parameters, locates q on node n_6, and invokes NEB algorithm to find an approximate result set $V(=\{o_1, o_2, o_3\})$ with COST$(V) = 10$ serving as the upper bound. Then, it starts the while-loop to gradually expand the network and adds POIs to Q_{tmp}. Fig. 5 depicts the contents of Q_{tmp} at different iterations, where we also label the *start* and *end* positions of the sliding window for every iteration to facilitate understanding.

Next, the algorithm retrieves the POIs that contain the queried keywords and inserts them into Q_{tmp}. As listed in the first line in Fig. 5, the first result set is $\{o_1, o_2, o_3\}$ and COST$(V) =$ 10. As $d(q, o_3) = 3.5 < 10$, SW continues. *start* points to the next POI in Q_{tmp}, and *end* points to o_3. However, the queried keywords are not covered by the keywords of the objects that lie between *start* and *end*. Thus, the next nearest POIs, i.e. o_6, o_5, o_{10}, are inserted into Q_{tmp}, and *end* points to o_{10}. It is observed that the keywords of the result set $\{o_2, o_3, o_{10}\}$ cover the queried keywords, and COST$(V') = 5.5 + 9 = 14.5$. The next result sets are $\{o_3, o_5, o_{10}\}$ with $\text{COST}(V') = 5.5 + 9 = 14.5$ and $\{o_6, o_5, o_{10}\}$ with $\text{COST}(V') = 5.5 + 3 = 8.5$. Since $8.5 < \text{COST}(V)$, $\{o_6, o_5, o_{10}\}$ is updated as the current optimal result set. In the following steps, we have the result set $\{o_5, o_{10}, o_4\}$ with $\text{COST}(V') = 7 + 6.5 = 13.5$, $\{o_{10}, o_9, o_4\}$ with $\text{COST}(V') = 7 + 7 = 14$, and $\{o_9, o_4\}$ with $\text{COST}(V') = 7 + 7 = 14$. In the next iteration, o_7 is added to Q_{tmp}. Nevertheless, $d(q, o_7) = 9 > 8.5$, and hence, SW algorithm dequeues the last POI. It divides Q_{tmp} into $L[i]$ such that $L[\text{keyword} = t_1] = \{o_2, o_5, o_9\}$, $L[\text{keyword} = t_2] = \{o_1, o_{10}, o_9\}$, and $L[\text{keyword} = t_3] = \{o_3, o_6, o_4\}$. SW algorithm computes the cost of the rest of the combinations, e.g., $\{o_2, o_1, o_3\}$, $\{o_2, o_1, o_6\}$, and so on. Finally, the SW algorithm chooses the set $\{o_6, o_9\}$ as the optimal result set with the smallest $\text{COST}(OPT) = 8$. ■

Compared with the aforementioned brute-force algorithm, SW algorithm is expected to find a tighter upper bound earlier, which helps to limit the portion of the network we have to explore and cut down the number of result sets we have to evaluate. Also, it does not introduce any additional cost. However, due to the complexity of the query, it is inevitable to conduct extensive search. Last but not least, we analyze the time complexity of SW algorithm below.

Lemma 10: The time complexity of SW algorithm is $O(|D|^{|q.key|} \cdot |E| \log |E|)$.

Proof: In the worst case, the time complexity of SW algorithm is $O(|D|^{|q.key|} \cdot |E| \log |E|)$, which corresponds to the size of the result set containing all possible qualified sets, and thus, the proof completes. ■

VI. Experiments

In this section, we evaluate the performance of our proposed algorithms in terms of both efficiency and accuracy. In what follows, we first describe our experimental settings in Section VI-A, and then, we present the experimental results of our approximate algorithms NEB and INB and our exact algorithm SW in Section VI-B.

A. Experimental Setup

We use three real road network datasets, namely, Cities of California (CAL), Oldenburg (OL), and San Joaquin County (TG), from the Digital Chart of the World Serve. CAL contains 21048 nodes and 21693 edges, OL consists of 6105 nodes and 7029 edges, and TG contains 18257 nodes and 18263 edges. We then generate randomly the points of interest (POIs), i.e., 86772 POIs for CAL, 14058 POIs for OL, and 23875 POIs for TG. Since each dataset contains both spatial and textual information, we extract the textual information from every dataset to form the vocabulary of that dataset. Vocabularies w.r.t. CAL, OL, and TG have 64, 64, and 80 keywords, respectively. We assume that each POI contains one to five keywords selected randomly from the corresponding vocabulary, with the average number of keywords per POI being 2.5. We fix the page size to 4096 bytes, and we assume that the server maintains a buffer having 200 pages with LRU being cache replacement policy.

We verify the performance of our proposed algorithms under different number of queried keywords via considering the number of the objects returned by each query, query time, the number of edge/node accesses, the number of page accesses, and the accuracy of NEB and INB algorithms. All algorithms are implemented in C++, and all experiments are conducted on the PC with an Intel core 2 DUO 2.93 GHz and 4 GB RAM, running Code::Blocks 12.11 under Linux operating system. Note that, in each experiment, we report the average over 50 queries generated randomly.

TABLE II
NUMBER OF QUERIED KEYWORDS VS. NUMBER OF OBJECTS RETURNED

	CAL			OL			TG		
$\|q.key\|$	NEB	INB	SW	NEB	INB	SW	NEB	INB	SW
3	2.94	2.88	2.9	2.96	2.86	2.9	2.92	2.82	2.9
4	3.88	3.82	3.9	3.84	3.72	3.74	3.92	3.84	3.9
5	4.8	4.8	4.8	4.64	4.44	4.62	4.88	4.8	4.9
6	5.78	5.78	5.8	5.4	5.4	5.52	5.84	5.64	5.9

B. Results on CSKQ Processing on Road Networks

First, we investigate the number of the objects returned by every query when the number of queried keywords (i.e., $|q.key|$) varies. As depicted in Table II, the number of returned objects grows with the increment of queried keywords. As the number of queried keywords ascends from 3 to 6, the number of objects returned changes in the range [3, 6], meaning that one object only contains one queried keyword. Note that, the proposed algorithms might find the objects that contain 2 or more queried keywords, but those objects can not form a result set with smaller $\text{COST}(V)$, and they are pruned away.

The next set of experiments is to verify the efficiency of the approximate algorithms NEB and INB and the exact algorithm SW for CSKQ processing on road networks, with respect to various number of queried keywords. Fig. 6 plots the query time (in seconds) when the number of queried keywords grows. It is observed that, NEB algorithm is several orders of magnitude faster than INB algorithm, and both of them are better than SW algorithm, which is consistent with our expectation. The reason is that for the approximate algorithm, NEB only needs to evaluate one result set, while INB needs to evaluate multiple result sets. For the exact algorithm, SW has to find all the possible result sets and choose the one with the smallest $\text{COST}(V)$ as the final answer. From the analysis in Section V, we know that the number of possible result sets ascends exponentially with the growth of queried keywords. Thus, SW algorithm takes the most query time, and it is more sensitive to the number of queried keywords. Another observation is that all the query time of NEB, INB and SW algorithms increases as the number of queried keywords grows. This is because, when the number of queried keywords ascends, these algorithms need to retrieve more objects to cover the queried keywords. It is worth noting that these phenomena of the algorithms are all consistent with their time complexities presented previously.

Then, we study the impact of the number of queried keywords (i.e., $|q.key|$) on the number of edges or nodes expanded. Fig. 7 illustrates the experimental results on three datasets. As expected, for SW algorithm, the number of edges and nodes expanded increases exponentially with the growth of the number of keywords, while NEB and INB algorithms perform

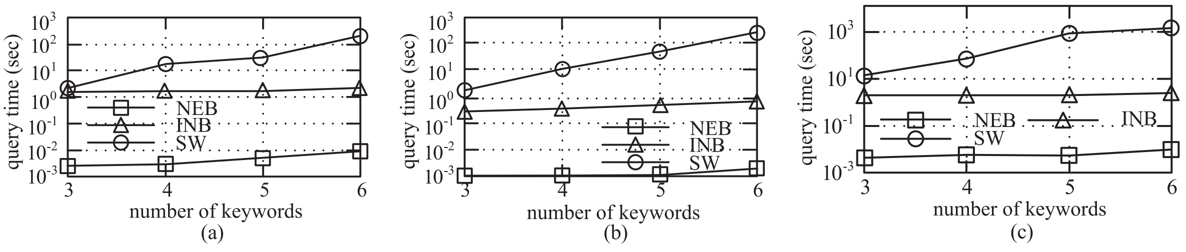


Fig. 6. Number of queried keywords vs. query time. (a) CAL. (b) OL. (c) TG.

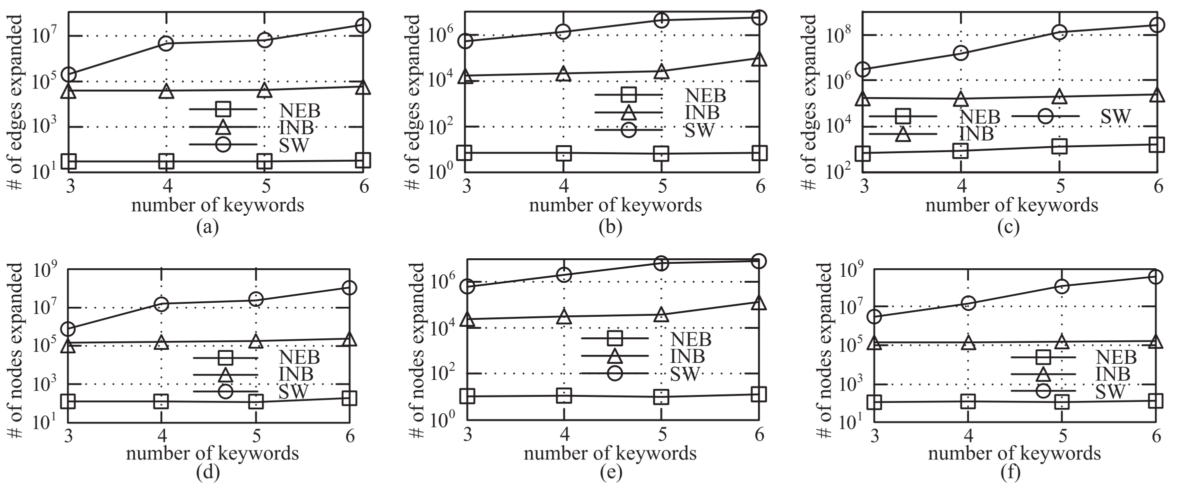


Fig. 7. Number of queried keywords vs. number of edges/nodes expanded. (a) CAL. (b) OL. (c) TG. (d) CAL. (e) EO. (f) TG.

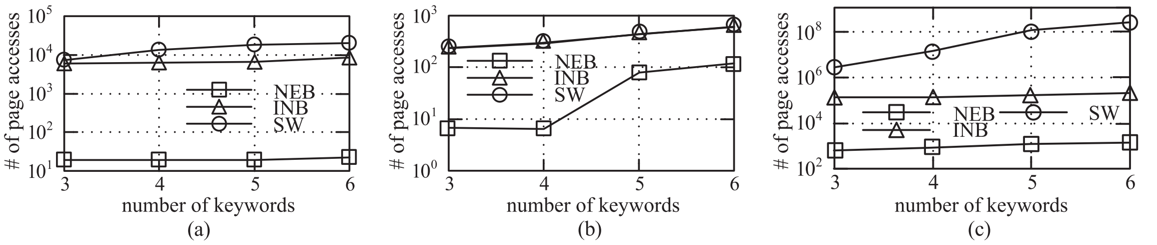


Fig. 8. Number of queried keywords vs. number of page accesses. (a) CAL. (b) OL. (c) TG.

relatively stable. The reason is that, SW has to verify all the combinations while NEB and INB only need to evaluate one and part of all the possible result sets, respectively.

Next, we explore the influence of the number of queried keywords on the number of page accesses, with experimental results shown in Fig. 8. We can observe that INB and SW algorithms access nearly the same number of pages. This is because, both INB and SW algorithms take the value of COST(V) returned by NEB algorithm as an upper bound, and they expand the road network within this upper bound. Although SW algorithm needs to evaluate all the possible result sets while INB only needs to evaluate some of them, the page accesses of the two algorithms are similar with each other since we maintain a buffer to store the pages with LRU being cache replacement policy. It is also observed that NEB algorithm needs the fewest page accesses because it expands the road networks only once.

Finally, we inspect the accuracy of approximate algorithms. Here, accuracy is defined as the ratio of COST(OPT) to COST(V). As depicted in Fig. 9, both NEB and INB algorithms achieve good accuracy, compared against the optimal result set returned by SW algorithm. It is observed that, the accuracy of NEB on CAL and OL grows with the growth of queried keywords, whereas that on TG remains stable. The reason is that NEB algorithm retrieves a set of the objects that are close to the query location and cover the queried keywords. Due to the randomness, the experimental results on TG are different from those on CAL and OL. Another observation is that INB algorithm is able to find more accurate result set when the number of queried keywords increases. As mentioned in Section IV, INB algorithm first expands the road network within the distance of COST(V) returned by NEB algorithm to find a set of POIs o_e. Then, it issues a new query q_e at every POI, and invokes NEB to find all the possible result sets.

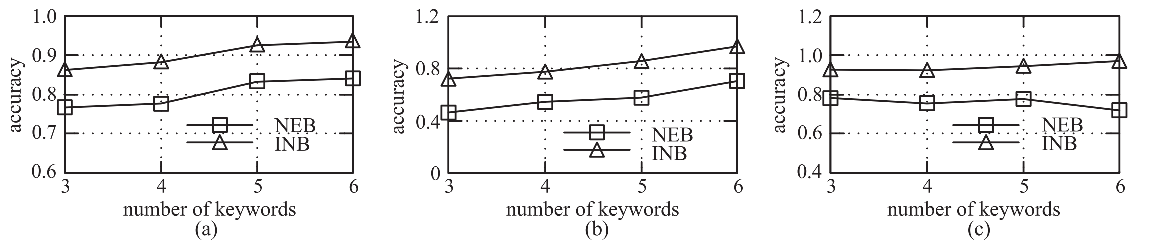


Fig. 9. Accuracy of NEB and INB algorithms. (a) CAL. (b) OL. (c) TG.

Among result sets, INB algorithm returns the one with the minimum value of COST(V'). If the number of queried keywords ascends, the value of COST(V) grows, and INB algorithm retrieves more POIs accordingly. Thus, INB algorithm calls NEB algorithm more times and finally returns the result set having the minimum value of COST(V'). As a consequence, the accuracy of INB algorithm becomes better as the number of queried keywords grows.

VII. Conclusion

In this paper, we study the problem of collective spatial keyword queries on road networks (i.e., CSKQ on road networks), which retrieves a set of POIs that collectively cover the queried keywords and have the lowest cost, measured by their shortest path distances to a specified query position, and the inter-POI distances between POIs in the set. With strict verification, we prove that this problem is NP-complete. To address this, we propose two approximate algorithms with provable approximation bounds and one exact algorithm for efficiently processing CSKQ on road networks, and present their corresponding time complexity. Finally, extensive experiments using real datasets demonstrate the performance of our proposed algorithms. In the future, we plan to develop more efficient algorithms for answering CSKQ on road networks by using pre-computation.

References

[1] X. Cao *et al.*, "Spatial keyword querying," in *Proc. 31st Int. Conf. ER*, 2012, 16–29.

[2] X. Cao, L. Chen, G. Cong, and X. Xiao, "Keyword-aware optimal route search," *Proc. VLDB Endowment*, vol. 5, no. 11, pp. 1136–1147, Jul. 2012.

[3] X. Cao, G. Cong, and C. S. Jensen, "Retrieving top-k prestige-based relevant spatial web objects," *Proc. VLDB Endowment*, vol. 3, no. 1/2, pp. 373–384, Sep. 2010.

[4] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi, "Collective spatial keyword querying," in *Proc. ACM SIGMOD*, 2011, pp. 373–384.

[5] A. Cary, O. Wolfson, and N. Rishe, "Efficient and scalable method for processing top-k spatial Boolean queries," in *Proc. SSDBM*, 2010, pp. 87–95.

[6] L. Chen, G. Cong, C. S. Jensen, and D. Wu, "Spatial keyword query processing: An experimental evaluation," *Proc. VLDB Endowment*, vol. 6, no. 3, pp. 217–228, Jan. 2013.

[7] Y.-Y. Chen, T. Suel, and A. Markowetz, "Efficient query processing in geographic web search engines," in *Proc. ACM SIGMOD*, 2006, pp. 277–288.

[8] M. Christoforaki, J. He, C. Dimopoulos, A. Markowetz, and T. Suel, "Text vs. space: Efficient geo-search query processing," in *Proc. ACM CIKM*, 2011, pp. 423–432.

[9] G. Cong, C. S. Jensen, and D. Wu, "Efficient retrieval of the top-k most relevant spatial web objects," *Proc. VLDB Endowment*, vol. 2, no. 1, pp. 337–348, Jan. 2009.

[10] I. D. Felipe, V. Hristidis, and N. Rishe, "Keyword search on spatial databases," in *Proc. IEEE ICDE*, 2008, pp. 656–665.

[11] A. Guttman, "R-Trees: A dynamic index structure for spatial searching," in *Proc. ACM SIGMOD*, 1984, pp. 47–57.

[12] R. Hariharan, B. Hore, C. Li, and S. Mehrotra, "Processing spatial-keyword (SK) queries in geographic information retrieval (GIS) systems," in *Proc. SSDBM*, 2007, pp. 1–16.

[13] G. Li, J. Feng, and J. Xu, "DESKS: Direction-aware spatial keyword search," in *Proc. IEEE ICDE*, 2012, pp. 474–485.

[14] Z. Li *et al.*, "IR-Tree: An efficient index for geographic document search," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 4, pp. 585–599, Apr. 2011.

[15] G. Li, J. Xu, and J. Feng, "Keyword-based k-nearest neighbor search in spatial databases," in *Proc. ACM CIKM*, 2012, pp. 2144–2148.

[16] C. Long, R. C. Wong, K. Wang, and A. W. Fu, "Collective spatial keyword queries: A distance owner-driven approach," in *Proc. ACM SIGMOD*, 2013, pp. 689–700.

[17] J. Lu, Y. Lu, and G. Cong, "Reverse spatial and textual k nearest neighbor search," in *Proc. ACM SIGMOD*, 2011, pp. 349–360.

[18] S. Luo *et al.*, "Distributed spatial keyword querying on road networks," in *Proc. EDBT*, 2014, pp. 235–246.

[19] J. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Norvag, "Efficient processing of top-k spatial keyword queries," in *Proc. SSTD*, 2011, pp. 205–222.

[20] J. Rocha-Junior and K. Norvag, "Top-k spatial keyword queries on road networks," in *Proc. EDBT*, 2012, pp. 168–179.

[21] C. Shahabi and C. Li, "SKIF-P: A point-based indexing and ranking of web documents for spatial-keyword search," *GeoInformatica*, vol. 16, no. 3, pp. 563–596, Jul. 2012.

[22] S. Shekhar and D. Liu, "CCAM: A connectivity-clustered access method for networks and network computations," *IEEE Trans. Knowl. Data Eng.*, vol. 9, no. 1, pp. 102–119, Jan./Feb. 1997.

[23] Y. Tao and C. Sheng, "Fast nearest neighbor search with keywords," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 4, pp. 878–888, Apr. 2014.

[24] S. Vaid, C. B. Jones, H. Joho, and M. Sanderson, "Spatio-textual indexing for geographical search on the web," in *Proc. SSTD*, 2005, pp. 218–235.

[25] D. Wu, G. Cong, and C. S. Jensen, "A framework for efficient spatial web object retrieval," *VLDB J.*, vol. 21, no. 6, pp. 797–822, Dec. 2012.

[26] D. Wu, M. L. Yiu, G. Cong, and C. S. Jensen, "Joint top-k spatial keyword query processing," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 10, pp. 1889–1903, Oct. 2012.

[27] D. Wu, M. L. Yiu, C. S. Jensen, and G. Cong, "Efficient continuously moving top-k spatial keyword query processing," in *Proc. IEEE ICDE*, 2011, pp. 541–552.

[28] B. Yao, M. Tang, and F. Li, "Multi-approximate-keyword routing in GIS data," in *Proc. SIGSPATIAL GIS*, 2011, pp. 201–210.

[29] D. Zhang, Y. M. Chee, A. Mondal, A. Tung, and M. Kitsuregawa, "Keyword search in spatial databases: Towards searching by document," in *Proc. IEEE ICDE*, 2009, pp. 688–699.

[30] J. Zhang, D. Liu, and X. Meng, "Preference-based top-k spatial keyword queries," in *Proc. MLBS*, 2011, pp. 31–39.

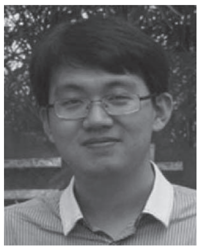
[31] D. Zhang, B. C. Ooi, and A. Tung, "Locating mapped resources in web 2.0," in *Proc. IEEE ICDE*, 2010, pp. 521–532.

[32] L. Zhang, X. Sun, and H. Zhuge, "Density based spatial keyword querying," *Future Gener. Comput. Syst.*, vol. 32, pp. 211–221, Mar. 2014.

[33] C. Zhang, Y. Zhang, W. Zhang, and X. Lin, "Inverted linear quadtree: Efficient top-k spatial keyword search," in *Proc. IEEE ICDE*, 2013, pp. 901–912.

[34] C. Zhang *et al.*, "Diversified spatial keyword search on road networks," in *Proc. EDBT*, 2014, pp. 367–378.

Yunjun Gao received the Ph.D. degree in computer science from Zhejiang University, Hangzhou, China, in 2008. He is currently an Associate Professor with the College of Computer Science, Zhejiang University. His research interests include spatial and spatiotemporal databases, metric and uncertain/incomplete data management, and spatiotextual data processing. Dr. Gao is a member of the Association for Computing Machinery and a Senior Member of the China Computer Federation.



Jingwen Zhao received the B.S. degree in computer science from Northeastern University, Shenyang, China, in 2013. He is currently working toward the Ph.D. degree with the College of Computer Science, Zhejiang University, Hangzhou, China. His research interest includes spatiotextual data management.

Baihua Zheng received the Ph.D. degree in computer science from Hong Kong University of Science and Technology, Kowloon, Hong Kong, in 2003. She is currently an Associate Professor with the School of Information Systems, Singapore Management University, Singapore. Her research interests include mobile/pervasive computing and spatial databases.

Gang Chen received the Ph.D. degree in computer science from Zhejiang University, Hangzhou, China. He is currently a Professor with the College of Computer Science, Zhejiang University. His research interests range from relational database systems to large-scale data management technologies supporting massive Internet users. Prof. Chen is a member of the Association for Computing Machinery and a Senior Member of the China Computer Federation.