Evaluation of Spatial Keyword Queries with Partial Result Support on Spatial Networks

Ji Zhang, Wei-Shinn Ku, Xiao Qin
Dept. of Computer Science and Software Engineering, Auburn University, Auburn, AL, USA
Email: {jizhang, weishinn, xqin}@auburn.edu,

***Abstract*—Numerous geographic information system applications need to retrieve spatial objects which bear user specified keywords close to a given location. For example, users can search for hotels in a certain city with preferred features and amenities by using hotel reservation websites. In this research, we present efficient approaches to answer spatial keyword queries on spatial networks. In particular, we systematically introduce formal definitions of Spatial Keyword k Nearest Neighbor (SKkNN) and Spatial Keyword Range (SKR) queries. Then, we present the framework of a spatial keyword query evaluation system which is comprised of Keyword Constraint Filter (KCF), Keyword and Spatial Refinement (KSR), and the spatial keyword ranker. KCF employs an inverted index to calculate keyword relevancy of spatial objects, and KSR refines intermediate results by considering both spatial and keyword constraints with the spatial keyword ranker. In addition, we design novel algorithms for evaluating SKkNN and SKR queries. These algorithms employ the inverted index technique, shortest path search algorithms, and network Voronoi diagrams. Finally, we apply both real-world and synthetic data sets to evaluate the performance of the proposed solutions. Our extensive simulations show that the proposed SKkNN and SKR algorithms can answer spatial keyword queries effectively and efficiently.**

I. Introduction

A Spatial Keyword (SK) query is an approach of searching qualified spatial objects by considering both the query requester's location and user specified keywords. Taking both spatial and keyword requirements into account, the goal of a spatial keyword query is to efficiently find results that satisfy all the conditions of a search. However, most existing solutions for SK queries are designed based on Euclidean distance [6], [8], [20], [19], which is not realistic since most users move on spatial networks. Moreover, most current approaches of SK queries are limited to finding objects that fully match the given keywords. Nevertheless, the objects with fully matched keywords could be *far away* from the query point. In this research, we design novel SK query techniques based on spatial networks. In addition, we take both fully and partially matched query results into account in the process of keyword searching. This new SK query mechanism enables users to not only retrieve qualified results on spatial networks, but also obtain partially matched objects when there are not enough fully matched results *in the vicinity* of the requester.

Figure 1 illustrates an example: a tourist who flies to Atlanta may want to search for two hotels which provide both "Internet" and "Breakfast" amenities and have the shortest driving distance to the Atlanta airport. In addition, the tourist may also search for all the hotels which are within 10 miles of the airport and provide the two amenities in order to compare the hotels' reviews and prices. For retrieving the qualified hotels, the tourist will launch a Spatial Keyword k Nearest Neighbor (SKkNN) query with ranking parameters for the first search; the query results are hotels 1 and 3. A Spatial Keyword Range (SKR) query will be executed for the second inquiry, and the answers are hotels 1, 3, and 6. In this paper, we focus on solving the two aforementioned spatial query types by devising three novel solutions which employ the inverted index technique, shortest path search algorithms, and network Voronoi diagrams. Particularly, the inverted index is used to maintain the relationships between spatial objects and their attached keywords for quickly retrieving spatial objects whose features match the given keywords. In addition, we propose both a network expansion-based approach and a Voronoi diagram-based approach to efficiently answer SKkNN queries on spatial networks. The contributions of this study are as follows:

1) We provide formal definitions of spatial keyword kNN and range queries on spatial networks.
2) We develop three novel approaches for efficiently processing SKkNN and SKR queries on spatial networks.
3) Our SKkNN solution is able to return partially matched query results based on the output of the spatial keyword ranker.
4) We evaluate the performance of the proposed SKkNN and SKR algorithms through extensive experiments with both real-world and synthetic data sets.

The rest of this paper is organized as follows. Section II surveys related works. The proposed query types are formally defined in Section III. In Section IV, we introduce the spatial keyword query evaluation algorithms. The experimental validation of our design is presented in Section V. Section VI

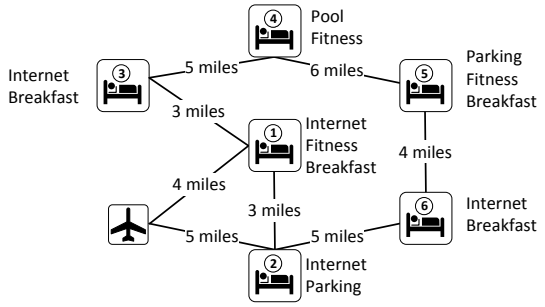


Fig. 1. A sample spatial network of hotels close to an airport.

concludes the paper with a discussion of future work.

II. Related Work

A. k Nearest Neighbor Queries

In spatial databases, k nearest neighbor (kNN) and range queries are fundamental query types. These two types of spatial queries have been extensively studied and applied in various location-based services (LBS) applications. For answering spatial queries on road networks, Papadias *et al.* [15] developed a Euclidean restriction and a network expansion framework to efficiently prune the search space. Based on the proposed frameworks, solutions for nearest neighbor queries are designed in the context of spatial network databases. In addition, a network Voronoi diagram-based solution for kNN searches in spatial network databases is presented in [10] by partitioning a large network to small Voronoi regions and pre-computing distances both within and across the regions. Because most Dijkstra's algorithm-based kNN solutions have been shown to be efficient only for short distances, Hu *et al.* [9] proposed an efficient index (distance signature) for distance computation and query processing over long distances. Their technique discretizes the distances between objects and network nodes into categories and then encodes these categories to accelerate the kNN search process. Furthermore, in order to speed up kNN searches, Samet *et al.* [17] designed an algorithm to explore the entire network by pre-computing the shortest paths between all the vertices in the network and employing a shortest path quadtree to capture spatial coherence. With the algorithm, the shortest paths between all possible vertices can be computed only once to answer various kNN queries on a given spatial network. Nevertheless, all the aforementioned techniques mainly focused on the distance metric. They did not consider text description (keywords) of spatial objects in their query evaluation processes.

B. Text Retrieval

Text retrieval is another important topic related to spatial keyword queries. There are two main indexing techniques, inverted files and signature files, widely utilized in text retrieval systems. According to experiments made by Zobel *et al.* [22], [21], signature files require a much larger space to store index structures, and are more expensive to construct and update than inverted files. In addition, Baeza-Yates and Ribeiro-Neto [1] also stated that inverted files outperform signature files in most cases.

Although these aforesaid methods perform quite well in text retrieval applications, none of them can efficiently process spatial keyword queries. In other words, it is impractical to answer spatial keyword queries by simply employing approaches introduced in this or the previous subsection. An effective way to handle spatial keyword queries is to combine the two groups of techniques as discussed in the following subsection.

C. Spatial Keyword Query

As local search services become more and more popular, many solutions [2], [4], [6], [8], [3], [20], [19], [16] have been developed to evaluate spatial keyword queries by integrating index techniques previously used in spatial queries and text search.

Location-based web search is studied by Zhou *et al.* [20] to find web pages related to a spatial region. They described three different hybrid indexing structures of integrating inverted files and R*-trees together. According to their experiments, the best scheme is to build an inverted index on the top of R*-trees. In other words, the algorithm first sets up an inverted index for all keywords, and then creates an R*-tree for each keyword. This method performs well in spatial keyword queries in their experiments; however, its maintenance cost is high. When an object insertion or deletion occurs, the solution has to update the R*-trees of all the keywords of the object. Cong *et al.* [4] illustrated a hybrid index structure, the IR-tree, which is a combination of an R-tree and inverted files to process location-aware text retrieval and provide k best candidates according to a rank system. They also proposed the DIR-tree and the CIR-tree, two extensions of the IR-tree, which take both minimizing areas of enclosing rectangles and maximizing text similarities into account during construction procedures. Recently, Cary *et al.* [2] proposed an efficient approach of answering top-k spatial boolean queries. They combined an R-tree with an inverted index to search the k best candidates which satisfy a group of boolean constraints. However, with their method, only candidates which completely meet boolean constraints will be found. The ones merely matching part of the constraints will be discarded because of strict constraints or an input error.

Felipe *et al.* [6] developed a novel index, IR^2-Tree which integrates an R-tree and signature files together, to answer top-k spatial keyword queries. They record signature information in each node of R-trees in order to decide whether there is any object which satisfies both spatial and keyword constraints simultaneously. However, the size of space for storing signatures in each node is decided before IR^2-Tree construction. Once the IR^2-Tree has been built, it is impossible to enlarge the space unless the tree is reconstructed. If the number of keywords grows quickly, a system will spend a lot of time repeatedly rebuilding the IR^2-Tree. Hariharan *et al.* [8] proposed an indexing mechanism, *KR*-tree*, which combines an R*-tree and an inverted index. The difference between their solution and [6] is that they only store related keywords in each node of an R*-tree in order to avoid merging operations to find candidates containing all keywords. Consequently, the number of keywords that appear in each node varies. However, such a complicated indexing technique has a high maintenance cost as well. If an object with new keywords is inserted, the method not only has to add new keywords to corresponding nodes from leaf to root of the R*-tree, but also update the inverted index (*KR*-tree* List).

Although there are a number of previous studies on spatial keyword queries, most of their solutions can only evaluate queries in Euclidean spaces. This limitation is due to the adoption of the R-tree (or its variants), which cannot index spatial objects based on network distances, into their hybrid

TABLE I A sample data set of hotels.

Name	$D_n(q,.)$	Amenities
H_1	3	Internet, Fitness Center, Pets Allowed, Parking
H_2	8	Pool, Parking, Room Service
H_3	13	Internet, Fitness Center, Pets Allowed, Parking
H_4	10	Parking, Airport Shuttle
H_5	8	Pets Allowed, Breakfast, Hot Tub, Restaurant Onsite
H_6	15	Internet, Pets Allowed, Restaurant Onsite
H_7	8	Fitness Center, Hot Tub, Parking

index structures. The work in [16] is the most relevant research to this paper. However, it is infeasible to provide partial results with the solution in [16].

III. Query Type Definition and Background

In order to explain definitions and algorithms in the following sections, we prepare a sample data set of hotels in Table I and an example spatial network in Figure 2. All the hotels have three attributes which include their names, amenities, and distances from a specific location q. In Figure 2, road segments are assigned weights that stand for their individual costs (e.g., distance or time). The location q and hotels are symbolized with a triangle and squares on road segments, respectively.

A. Foundation

In this subsection, we introduce the foundation of spatial keyword queries. In a SK query, a spatial object p is defined as a pair $<l, t>$, where l is a location in the search space and t is a text description (e.g., amenities and features of a hotel) of the corresponding object. Table II summarizes notations used in this paper.

1) Distance on Spatial Networks: Spatial networks are composed by undirected weighted graphs $G = (V, E)$, where V is a set of vertices and E is a set of edges. In general, the weight of each edge is determined by a metric measured in physical distance or time cost for traveling the road segment [12], [11]. The distance between two objects $D_n(.,.)$ on spatial networks is the summation of all segment weights on the shortest path connecting the two objects. For example, in Figure 2, $D_n(H_6, H_7) = D_n(H_6, n_8) + D_n(n_8, H_7)$ = 7.

2) Matched Keywords: Matched-keywords is a set of keywords which are in both sets of $p.t$ and K, where $p.t$ is the text description of a given spatial object, and K is a set of keywords specified by a user. For example, given a hotel H_2 with keywords {"Pool", "Parking", "Room Service"}

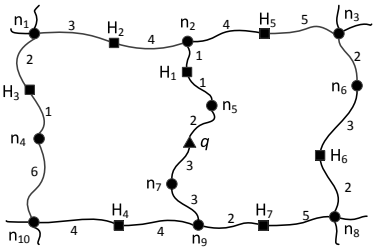


Fig. 2. An example spatial network.

TABLE II Symbolic notations.

Symbol	Meaning
P	A set of spatial objects
Q	A spatial keyword query
K	A set of search keywords
q	The location of a requester
I	An inverted index
k	The requested number of objects in the result of a SKkNN query
r	The search range of a SKR query
s	The ranking score of an object
$\|\mathbb{S}\|$	The number of elements in set $\mathbb{S}$
$d(.,.)$	The Euclidean distance between two points
$D_n(.,.)$	The shortest network distance between two points
$\mathbb{R}$	The result set of a query
$\mathbb{E}$	The explored region of a VDkNN query
$\mathbb{C}$	The set of candidate spatial objects

and a set of keywords K {"Pool", "Parking", "Breakfast"}, $MK(H_2, K)$ is an intersection of $H_2.t$ and K, {"Pool", "Parking"}. The formal definition of the MK function is shown in Equation (1).

$$MK(p, K) = \{\, k_i \in K \mid k_i \in p.t \,\} \tag{1}$$

3) Fully Matched Keyword Search: With a given data set, the purpose of Fully Matched Keyword Search (FMKS) is to find objects whose descriptions completely match with a set of keywords K specified by a requester. As shown in Equation (2), the descriptions of search results of FMKS may be either identical to K or a superset of K. For example, given the keywords "Internet" and "Pets Allowed" and the data set in Table I, the result set of the FMKS is $\{H_1, H_3, H_6\}$.

$$FMKS(P, K) = \{\, p_i \in P \mid K \subseteq p_i.t \,\} \tag{2}$$

4) Partially Matched Keyword Search: With a given data set, the purpose of Partially Matched Keyword Search (PMKS) is to retrieve objects which match at least one keyword in the user defined keyword set as shown in Equation (3). For example, given the keywords "Internet" and "Pets Allowed" and the data set in Table I, the results of the PMKS are $\{H_1, H_3, H_5, H_6\}$. The difference in search results from the previous FMKS is H_5, which matches only one keyword ("Pets Allowed") and is a valid answer of this PMKS.

$$PMKS(P, K) = \{\, p_i \in P \mid \exists k_j \in p_i.t \text{ and } k_j \in K \,\} \tag{3}$$

5) Weighted Keyword Relevancy: We use a weight function TR to calculate keyword relevancy of a specific spatial object p [18]. We assume that each keyword k_i in a keyword set K is assigned with a weight $w(k_i)$, which indicates its importance in queries. Consequently, given an object p and a keyword set K, we have the following equation:

$$TR(p, K) = \sum_{k_i \in MK(p,K)} w(k_i) \tag{4}$$

For special cases where all keywords share identical weight, Equation (5) can be derived from Equation (4) where $w(k_i) = 1$ and $|MK(p, K)|$ is the number of keywords in $MK(p, K)$.

$$TR(p, K) = \sum_{k_i \in MK(p,K)} 1 = |MK(p, K)| \tag{5}$$

B. Spatial Keyword Ranker

A spatial keyword ranker is designed to determine the ranking of a given spatial object in a SKkNN query by employing both metrics, spatial network distance and keyword relevancy. We utilize a ranking function RK to compute how well an object matches a SKkNN query. Given a query Q <l, K> and an object p <l, t>, the ranking function is defined as follows:

$$RK(Q,p) = \theta_1 \cdot TR(p.t, Q.K) - \theta_2 \cdot D_n(p.l, Q.l) \quad (6)$$

In Equation (6), θ_1 and θ_2 are parameters of each part of the function [8], and their values depend on user preferences. For example, if a user is more concerned about keyword match, θ_1 can be set to a larger value than θ_2 in order to make keyword relevancy dominant in the RK function. Moreover, intuitively, an object with either a shorter distance or a higher keyword relevancy would have a higher ranking in query results. Therefore, TR has a positive influence on the RK function while D_n has a negative one.

C. Spatial Keyword **k**NN Queries

Based on the spatial keyword ranker, the purpose of a spatial keyword kNN query is to retrieve k objects which have top k ranking values.

Definition Given a SKkNN query Q and an object set P, we define SKkNN(P, Q, k) as follows:

$$RK(p_i) \geq RK(p_j)\ where\ p_i \in SKkNN(P,Q,k) \wedge p_j \in P \backslash \{SKkNN(P,Q,k)\} \wedge |SKkNN(P,Q,k)| = k \quad (7)$$

We utilize the data set in Table I and spatial network in Figure 2 to demonstrate a SKkNN query example. Assume a visitor wants to find the two nearest hotels that have the amenities, "Internet" and "Pets Allowed" from q. Partially matched results are acceptable when there are not enough fully matched objects in the vicinity. In addition, all keywords have identical weight and the values of θ_1 and θ_2 are 0.8 and 0.2, respectively. The result set for this query is $\{H_1, H_5\}$ where H_5 has only one matched keyword. If 4 hotels are requested instead of 2, the result set will be $\{H_1, H_5, H_3, H_6\}$.

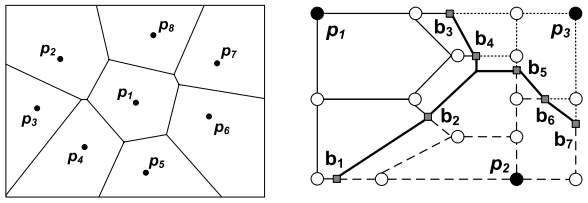
D. Spatial Keyword Range Queries

The purpose of an SK Range query is to find all the objects that fully match the given keywords within a user specified distance.

Definition Let P be a set of objects. Given a query location q, a search range r, and a set of keywords K, an SK range query is defined as follows:

$$SKR(P,q,r,K) = \{p_i \in P | K \subseteq p_i.t \wedge D_n(p_i,q) \leq r\} \quad (8)$$

Assume a tourist wants to find all the hotels bearing the keywords "Internet" and "Pets Allowed" within 10 miles of q on the sample spatial network. The answer is $\{H_1\}$ based on the example data set (Table 1). Furthermore, if the range is extended to 20 miles, the result set will be $\{H_1, H_3, H_6\}$.



(a) Ordinary VD. (b) Network VD.

Fig. 3. Voronoi diagram examples.

E. Network Voronoi Diagram

We employ network Voronoi diagrams in our approach for efficiently evaluating spatial keyword queries. A Voronoi diagram divides a metric space into disjoint polygons (Voronoi polygons) based on the distances to a specified set of points (generators) in the space. The nearest neighbor of any point inside a polygon is the generator of the polygon. The Voronoi Polygon (VP) and the Voronoi Diagram (VD) in the Euclidean plane can be formally defined as follows. Given a set of generators $P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^2$, where $2 \leq n < \infty$ and $p_i \neq p_j$ for $i \neq j, i, j \in I_n = \{1, \ldots, n\}$. The region given by:

$$VP(p_i) = \{p \mid d(p,p_i) \leq d(p,p_j)\}\ for\ j \neq i, j \in I_n\} \quad (9)$$

is called the Voronoi polygon associated with p_i where $d(p,p_i)$ denotes the minimum Euclidean distance between p and p_i. In addition, the set given by:

$$VD(P) = \{VP(p_1), \ldots, VP(p_n)\} \quad (10)$$

is called the Voronoi diagram generated by P. Figure 3(a) demonstrates a Voronoi diagram in the Euclidean plane.

The Network Voronoi Diagram (NVD) is defined based on a planar geometric graph where the locations of objects are restricted to the links that connect the nodes of the graph. Distances between objects are defined as the length of the shortest path in the graph (network distance) [14]. In our problem, spatial networks can be modeled as a geometric graph where the intersections are symbolized by nodes of the graph and edges are represented by the links connecting the nodes. Furthermore, the weights of links are the distances between corresponding nodes.

The network Voronoi diagram can be formally defined as follows. Consider a geometric graph $G(N, L)$ consisting of a set of nodes $N = \{p_1, \ldots, p_n, p_{n+1}, \ldots, p_l\}$, where the first n elements are the generators (i.e., $P = \{p_1, \ldots, p_n\}$), and a set of links $L = \{l_1, \ldots, l_k\}$ which form a connected network. We define the distance from a point p on a link in L to a node p_i in N, $D_n(p, p_i)$, by the length of the shortest path from p to p_i. For all $j \in I_n \backslash \{i\}$, let

$$Dom(p_i,p_j) = \{p | p \in \bigcup_{i=1}^{k} l_i, D_n(p,p_i) \leq D_n(p,p_j)\} \quad (11)$$

$$b(p_i,p_j) = \{p | p \in \bigcup_{i=1}^{k} l_i, D_n(p,p_i) = D_n(p,p_j)\} \quad (12)$$

We call the set $Dom(p_i, p_j)$ the dominance region of p_i over p_j on links in L, and the set $b(p_i, p_j)$ the bisector (border)

points between p_i and p_j on links in L. Accordingly, the Voronoi link set associated with p_i and the network Voronoi diagram are defined as follows, respectively:

$$V_{link}(p_i) = \bigcap_{j \in I_n \setminus \{i\}} Dom(p_i, p_j) \quad (13)$$

$$NVD(P) = \{V_{link}(p_1), \ldots, V_{link}(p_n)\} \quad (14)$$

where $V_{link}(p_i)$ specifies all the points in all the links in L that are closer to p_i than any other generator point in N. By properly connecting adjacent bisector points of a generator to each other without crossing any of the links, we can generate a bounding polygon, named Network Voronoi Polygon (NVP) [14], [10]. Figure 3(b) shows an NVD example where each line style corresponds to a Voronoi link set of a generator (NVPs are created by connecting adjacent bisector points).

IV. System Design

In this section, we design a spatial keyword query evaluation system which is comprised of Keyword Constraint Filter (KCF), Keyword and Spatial Refinement (KSR), and the spatial keyword ranker. For the proposed spatial keyword query algorithms, if two or more objects have the same ranking score, our algorithms will sort the objects based on their distances to the query point (i.e., in an ascending order). In addition, in order to simplify the explanation, we assume all keywords have the same weight.

A. Framework of Query Evaluation

Before presenting the details of our spatial keyword query algorithms, we briefly introduce the framework of our system. As illustrated in Figure 4, the spatial keyword query evaluation system comprises three main components which are Keyword Constrain Filter (KCF), Keyword and Spatial Refinement (KSR) and the spatial keyword ranker. This system receives both spatial data sets and spatial keyword constraints as inputs and produces results after a two-step computation.

The system employs a filter-and-refine strategy to answer SK queries. The two key steps are KCF and KSR. KCF receives spatial data sets and keyword constraints and filters out objects that do not match any user specified keyword. Because spatial network distance computation is expensive, we do not take spatial constraints into account in this step. The main purpose of KCF is to reduce the number of candidate objects in order to decrease computation costs in the next step. In the second step, KSR receives inputs from KCF and refines the intermediate results based on both keyword and spatial constraints. Afterward, KSR returns the qualified objects sorted by their ranking scores provided by the ranker.

B. Keyword Constraint Filter

1) Inverted Indexing Structure: Inverted indexes are primarily designed to support keyword searches from a set of text files. In our system, we utilize inverted indexes to search for objects related to specific keywords from spatial databases. An index of terms is maintained in our system where each term

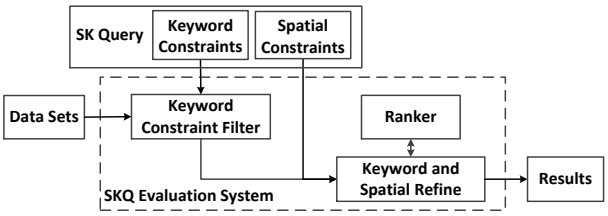


Fig. 4. Framework of the proposed system.

is a unique keyword, and each postings list contains a number of object identifiers. Each postings list is in sorted order (based on object identifiers) to facilitate the efficient search of objects related to a specific keyword. If an object has multiple keywords, its identifier will appear in each corresponding postings list. In addition, inverted indexes are independent of other dedicated index structures, such as R-trees or grids, in spatial databases.

2) Keyword Match Algorithm: Based on the proposed problem, we design a keyword match algorithm by employing the inverted index-based merge technique [13] to calculate the keyword relevancy of spatial objects. With the keyword match algorithm, we measure the keyword relevancy of a spatial object by counting the number of matched-keywords. The more matched-keywords the object has, the higher its keyword relevancy is. This algorithm receives an inverted index and a set of keywords as input parameters, and then returns the keyword relevancy of objects which match with at least one keyword.

In Algorithm 1, mergeList is a list, of which each element comprises a pair $<id, occurrence>$ where id is an object identifier and $occurrence$ is the corresponding keyword relevancy. With the for loop in line 1, the algorithm iteratively retrieves object lists of matched-keywords from the inverted index structure and merges these object lists into mergeList. This merge process, illustrated in lines 5 to 22, is an essential part which supports partially-matched-keyword searches. The worst-case time complexity of Algorithm 1 is $O(|K| * |P|)$, where $|P|$ is the number of spatial objects in the data set and $|K|$ stands for the number of search keywords.

Figure 5 illustrates how KeywordMatch works. We utilize the data set in Table I and spatial network in Figure 2 for explanation. Assume a query with keywords "Internet", "Pets Allowed", and "Parking" is evaluated. Algorithm 1 first finds object lists that are related to these keywords by searching the inverted index. As shown in Figure 5, three object lists, $\{H_1, H_3, H_6\}$, $\{H_1, H_3, H_5, H_6\}$ and $\{H_1, H_2, H_3, H_4, H_7\}$, are retrieved from the inverted index. Then the algorithm merges these lists into a mergeList.

In the first round, KeywordMatch simply copies objects in the "Internet" list to mergeList, and each object is marked by one occurrence. Then, in the second round, KeywordMatch compares objects in the "Pets Allowed" list with mergeList. If an object appears in the "Pets Allowed" list but does not exist in mergeList, it will be inserted into mergeList with occurrence marked by one. However, if an object already exists in mergeList, its counter will be increased by one. The third round of merging the "Parking" list is processed in the same

Algorithm 1 KeywordMatch(I, K)

```
1: for each term in the input inverted index I do
2:   if (term ∈ K) then
3:     iterator iterA = mergeList.begin
4:     iterator iterB = term.idList.begin
5:     while (iterA != mergeList.end and iterB != term.idList.end) do
6:       if (iterA.id > iterB.id) then
7:         newNode ← {iterB.id, 1}
8:         insert newNode at previous position of iterA
9:         iterB++
10:      else if (iterA.id == iterB.id) then
11:        iterA.occurrence += 1
12:        iterA++
13:        iterB++
14:      else
15:        iterA++
16:      end if
17:    end while
18:    while (iterB != term.idList.end) do
19:      newNode ← {iterB.id, 1}
20:      append newNode to the end of mergeList
21:      iterB++
22:    end while
23:  end if
24: end for
25: return mergeList
```

way. After the iterations, mergeList contains the final result with seven objects and their keyword relevancy shown at the bottom of the dashed rectangle in Figure 5.

C. Network Expansion-Based SKkNN Query Algorithm

In this section, we explain our algorithm for processing spatial keyword k nearest neighbor queries based on network expansion techniques [15], [5]. As present in Section 3.3, the algorithm receives an inverted index I, a query point q, the value of k, and a set of keywords K as input parameters and returns the top k objects by considering both keyword and spatial constraints.

For searching the shortest path between objects on spatial networks, Dijkstra's algorithm-based approaches [5], [7] have been widely utilized in various applications. Given a source point and a group of destinations, the algorithm recursively expands the unvisited paths and records distances of intermediate nodes. During the search, a distance record of a node will be updated if there is a shorter path than the present one.

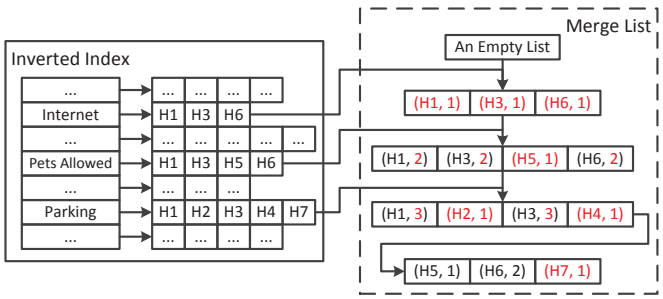


Fig. 5. An example of KeywordMatch.

Such a process is continued until all the destinations have arrived, and the distances of all other possible paths are longer than their current distances. In addition, a solution named Incremental Network Expansion (INE) is presented in [15] by extending Dijkstra's algorithm to compute k nearest neighbors in a network space. Specifically, INE first locates the network segment e_i, which covers the query point q, and retrieves all objects on e_i. If any object p_i is found on e_i, p_i will be inserted into the result set. Furthermore, the endpoint of e_i, which is closer to q, will be expanded while the second endpoint of e_i will be placed in a priority queue Q_p. INE repeats the process by iteratively expanding the first node in Q_p and inserting newly discovered nodes into Q_p until k objects are retrieved.

Algorithm 2 NEkNN(I, q, k, K)

```
1. mergeList = KeywordMatch(I, K)
2. if mergeList == ∅ then
3.   return ∅
4. end if
5. mark each object in mergeList in the spatial network
6. n_i n_j = the segment covers q
7. if n_i n_j covers candidate objects then
8.   calculate their ranking scores and insert them into ℝ
9. end if
10. {p_1, ..., p_k} are the top k objects in ℝ sorted in descending order of their ranking scores
11. s_min = p_k.s   // if p_k = NULL, s_min = −∞
12. Q_p = <(n_i, D_n(q, n_i)), (n_j, D_n(q, n_j))>   // sorted in ascending order of their distance to q
13. de-queue the first node n_f in Q_p
14. if |ℝ| ≥ k then
15.   calculate n_f.s by assuming that n_f fully matches K
16. else
17.   n_f.s = 0
18. end if
19. while s_min < n_f.s or |ℝ| < k do
20.   for each non-visited adjacent node n_x of n_f do
21.     search n_x n_f
22.     if n_x n_f covers candidate objects then
23.       for each candidate object p_i do
24.         calculate p_i.s
25.         if |ℝ| < k then
26.           ℝ ∪ {p_i}
27.         else
28.           if p_i.s > p_k.s then
29.             replace p_k by p_i
30.           end if
31.         end if
32.       end for
33.     end if
34.     en-queue (n_x, D_n(q, n_x))
35.   end for
36.   s_min = p_k.s   // if p_k = NULL, s_min = −∞
37.   de-queue the first node n_f in Q_p
38.   if |ℝ| ≥ k then
39.     calculate n_f.s by assuming that n_f fully matches K
40.   else
41.     n_f.s = 0
42.   end if
43. end while
44. return ℝ
```

We develop a Network Expansion-based SKkNN (NEkNN)

solution by leveraging INE. There are two main steps in the NEkNN algorithm. The first step is to filter out objects which do not match any user specified keywords by employing Algorithm 1. Then, we mark all the objects in mergeList in the spatial network as candidates (e.g., set a bit of these points of interest). The next step is to expand the network from q with INE and the ranking function (Section 3.2). When an object p_i is discovered, NEkNN verifies that p_i is a candidate object. If p_i is a candidate object, NEkNN calculates its ranking score s by executing the ranking function (otherwise the algorithm ignores p_i). Meanwhile, NEkNN keeps a result set $\mathbb{R}$ which is sorted in descending order based on the ranking score. If $\mathbb{R}$ has fewer than k objects and p_i is a candidate object, p_i is inserted into $\mathbb{R}$. Otherwise, NEkNN compares the ranking score of p_i with the last object p_j in $\mathbb{R}$. p_j will be replaced by p_i if $p_i.s > p_j.s$. In addition, when $|\mathbb{R}| \geq k$, NEkNN calculates ranking scores for network nodes as well by assuming that they match all the search keywords to restrict the search space. In other words, any spatial object p_i, which is further away from q than a network node n_i, must have a lower ranking score than n_i even if p_i matches all the search keywords. Consequently, NEkNN iterates the search process until $\mathbb{R}$ contains k objects and the next network node to be expanded in Q_p has an equal or lower ranking score than the last object in $\mathbb{R}$.

By employing the data set in Table I and spatial network in Figure 2, we demonstrate an example to retrieve the two nearest hotels that have the amenities, "Internet" and "Pets Allowed" from q with NEkNN. We assume that all keywords have an identical weight and the values of θ_1 and θ_2 are 0.5 and 0.5, respectively. First, NEkNN executes Algorithm 1 and marks candidate hotels H_1, H_3, H_5, and H_6 on the network. Then, NEkNN locates the segment n_5n_7 that covers q. Since no hotel is covered by n_5n_7, the node (n_5) closer to q is expanded and the other endpoint n_7 is placed in Q_p. On n_2n_5, H_1 is discovered and inserted into $\mathbb{R}$ with $s = -0.5$. Meanwhile, n_2 is inserted into $Q_p = <(n_7, 3), (n_2, 4)>$. The expansion of n_7 reaches n_9 and $Q_p = <(n_2, 4), (n_9, 6)>$. Next, the expansion of n_2 reaches n_1 and n_3, after which $Q_p = <(n_9, 6), (n_1, 11), (n_3, 13)>$ and H_5 is found on n_2n_3. Afterward, H_5 is inserted into $\mathbb{R}$ with $s = -3.5$. Subsequently, n_9 is expanded and $Q_p = <(n_1, 11), (n_3, 13), (n_8, 13), (n_{10}, 14)>$. The ranking score of the next node in Q_p (n_1) is -4.5; the algorithm terminates because $\mathbb{R}$ contains two hotels and $H_5.s > n_1.s$. The complete algorithm of NEkNN is formalized in Algorithm 2.

D. Voronoi Diagram-Based SKkNN Query Algorithm

Although NEkNN is able to restrict the search space and retrieve the top k objects based on their ranking scores, the main limitation of NEkNN is that it has to explore a large portion of the network when candidate objects are not densely distributed in the network. Therefore, we propose a Voronoi diagram-based SKkNN (VDkNN) solution by leveraging the network Voronoi diagram (NVD) [10] to improve performance. In order to be independent of the density and distribution of candidate objects, we first partition the spatial network into small regions by generating a network Voronoi diagram over all the spatial objects (points of interest). Each cell of the NVD is centered by one spatial object and contains the nodes that are closest to that object in network distance. Afterward for each NVD cell, we pre-compute the distances between all the edges of the cell to its center as well as the distances only across the border points of the adjacent cells. Consequently, for a new cell, we can quickly extend the searched region to the border points without expanding all the internal network segments.

With the NVD of the search space, for a SKkNN query, VDkNN first filters out unqualified objects with Algorithm 1 and marks all the objects in mergeList in the NVD as candidates. Then, VDkNN finds the network Voronoi polygon NVP(p_i) that contains q where p_i is the generator of the polygon. This step can be accomplished by employing a spatial index (e.g., the R-tree), which is generated based on the NVD cells. Next, we verify that p_i is a candidate object. If p_i is a candidate object, VDkNN calculates its ranking score by running the ranking function. In addition, VDkNN maintains a result set $\mathbb{R}$ which is sorted in descending order according to the ranking score. When $\mathbb{R}$ contains fewer than k objects, newly discovered candidate objects are inserted into $\mathbb{R}$. However, if $\mathbb{R}$ already includes k objects, VDkNN replaces the k^{th} object p_k of $\mathbb{R}$ when a newly retrieved candidate object has a higher s than p_k. Also, VDkNN keeps a queue Q_n which stores the neighbors (adjacent cells) of p_i and a set $\mathbb{E}$ which consists of all the searched cells (i.e., $\mathbb{E}$ covers the current explored region).

Subsequently, VDkNN searches the adjacent cells of $\mathbb{E}$ (i.e., NVP(p_i)) stored in Q_n for the next candidate object. Every time after a cell NVP(p_j) been explored, the neighboring generators of p_j are unioned with Q_n, NVP(p_j) is unioned with $\mathbb{E}$, and $\mathbb{R}$ is updated according to the aforementioned rules if p_j is a candidate object. Moreover, when $|\mathbb{R}| \geq k$, VDkNN calculates the ranking score of all the border points of the current explored region by assuming that they match all the search keywords to restrict the search space. VDkNN iterates the search process until $\mathbb{R}$ contains k objects and the ranking scores of all the border points of $\mathbb{E}$ are equal or worse than the ranking score of the k^{th} object in $\mathbb{R}$ (i.e., there will not be any changes in $\mathbb{R}$ even if we search further).

Figure 6 illustrates an example of retrieving the two nearest points of interest (POI) which match keywords in K from q with VDkNN. First, VDkNN executes Algorithm 1 and marks candidate POIs on the NVD. Then, VDkNN locates the network Voronoi polygon, NVP(p_1), which contains q. Next, VDkNN verifies that p_1 is a candidate POI and inserts the

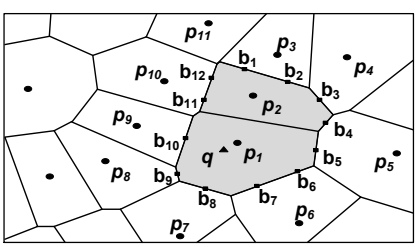


Fig. 6. A VDkNN query example.

Algorithm 3 SKR(I, q, r, K)

```
1. mergeList = KeywordMatch(I, K)
2. for each object o in mergeList do
3.     if o.occurrence == |K| then
4.         ℂ.append(o)
5.     end if
6. end for
7. if (|ℂ| != 0 ) then
8.     ℝ = ShortestPath(q, ℂ)
9. end if
10. filter out objects beyond r in ℝ
11. return ℝ
```

neighboring generators, p_2, p_5, p_6, p_7, p_8, p_9, and p_{10} into Q_n. Also, $\mathbb{E}$ covers NVP(p_1). Afterward VDkNN searches the objects in Q_n for the next candidate POI. Assume that NVP(p_2) is the second explored NVP and both p_1 and p_2 are candidate POIs. Then, $\mathbb{R}$ contains p_1 and p_2, $\mathbb{E}$ covers NVP(p_1) and NVP(p_2) (the shaded region in Figure 6), and Q_n comprises nine generators (p_3 to p_{11}). Since $\mathbb{R}$ covers two POIs, VDkNN computes the ranking score of all the border points of $\mathbb{E}$ (b_1 to b_{12}) by assuming that they match all the search keywords in K. Here we suppose that $s_{min} > b_{max}$ and the algorithm terminates. The algorithm of VDkNN is similar to Algorithm 2 except for the underlying index. We skip the algorithm here because of the space limit.

E. Spatial Keyword Range Query Algorithm

As defined in Section 3.4, given a query point q, a search range r and a set of keywords K, SKR query is to retrieve all the objects which fully match all the keywords within r. SKR query first calculates the keyword relevancy of objects by utilizing KeywordMatch (Algorithm 1). Then, it retrieves objects which fully match all the given keywords and stores the qualified objects in $\mathbb{C}$. Afterward, it calls Dijkstra's algorithm for calculating distances from q to all the candidate objects. Finally, SKR query removes objects which are out of the search range from $\mathbb{R}$. The complete SKR query algorithm is illustrated in Algorithm 3.

The worst-case running time of Algorithms 2 and 3 on a spatial network with a set of nodes N is $O(|K| * |P| + |N|^2)$ by considering both the keyword match and spatial network search subroutines.

V. Experimental Validation

In this section, we evaluate the performance of our spatial keyword query solutions with both real-world and synthetic data sets. We implemented the proposed algorithms and related experimental components in C++. The inverted index structure was loaded into the main memory during the execution of simulations. All the experiments were conducted on an Ubuntu Linux server equipped with an Intel Xeon 2.4GHz processor and 2GB memory. All simulation results were recorded after the system model reached a steady state.

TABLE III Default values of parameters.

Parameters	θ_1	θ_2	k	$\|K\|$	r	q
Value	0.048	0.952	5	2	20 km	random

A. Experimental Data Sets

In our experiments, a real-world data set was downloaded from edigitalz[1], which provides a wide range of general data sets for free. We retrieved 9,483 restaurants in the state of California and collected 34,091 keywords from their menus (e.g., pizza, steak, etc.) and cuisine (e.g., American, French, etc.) for searches. The data sets of road networks in both the state of California (containing 21,692 roads and 21,047 intersections) and the continental United States (containing 179,178 roads and 175,812 intersections) were downloaded from the US Census Bureau (TIGER/Line Shapefiles)[2].

For the synthetic data set, we generated around 160,000 restaurants, of which the density follows the real-world data set in order to investigate the scalability of our algorithms. In addition, each restaurant has a similar number of keywords (totally 575,200 keywords) to the real-world data set. The network of the continental United States is used with the synthetic data set.

Table III displays the default values of parameters in our experiments. We varied an essential parameter in each experiment set in order to evaluate its impact on the performance of the proposed algorithms. Other parameters were kept constant during all the experiments in the same set. The default values of parameters are used in experiments if we do not explicitly specify other values. The selection of θ_1 and θ_2 values depends on preference for keyword relevancy and distance of users. We fixed the ratio of θ_1 to θ_2 ($1/20$) in all the experiments.

B. Data Set Size Experiment

In this experiment, we evaluate NEkNN, VDkNN, and SKR queries with various data set sizes. The main purpose of this experiment is to analyze the influence of different data set sizes on query execution time. For both real-world and synthetic data sets, we generate five subsets of restaurants with an increasing number of data objects. The number of restaurants in consecutive subsets is increased by 2,000 for real-world data sets and 35,000 for synthetic data sets. Objects in smaller subsets are included in bigger ones.

The results of real-world and synthetic data sets are demonstrated in Figure 7(a) and Figure 7(b), respectively. VDkNN always outperforms NEkNN with the default parameters in all the experiments. From Figure 7, we observe that the execution time of most queries increases linearly with the increment of data set size. The reason is that more POIs and keywords have to be processed in these queries. In addition, the difference of execution time between NEkNN and VDkNN queries decreases gradually as the data set size grows. In other words, the time costs of these two solutions become close with a larger data set. The reason for rapid performance degradation in VDkNN is that it has an extra overhead of searching on

[1] http://www.edigitalz.com/

[2] http://www.census.gov/geo/www/tiger/

Voronoi diagrams in addition to the cost of processing POIs, which is suffered by both solutions. The higher density of POIs on spatial networks, the more border nodes are generated in Voronoi polygons. Hence, VDkNN has to spend more time on border node related calculation when it tests its stopping condition.

Another observation is that queries run faster on a bigger synthetic data set (e.g., 20,000 data objects) than a smaller real-world data set (e.g., 9,483 data objects). The reason is that the density of POIs is a dominant factor in these queries. Although there are more POIs involved in the keyword-match process in the synthetic data set, there are fewer candidates which are discovered in the search area due to lower POI density.

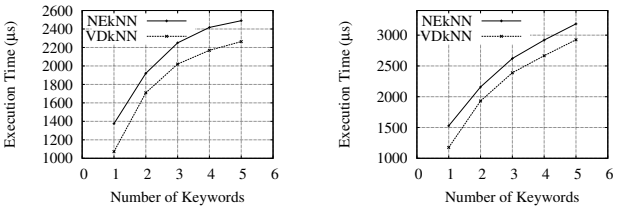
C. Number of Keywords Experiment

The number of keywords is an essential parameter of both NEkNN and VDkNN queries. In order to investigate the impact of the number of keywords on query performance, we vary the number of specified keywords on both data sets. We conduct experiments from queries with one keyword to ones with five keywords by adding a new keyword after each experiment. Figure 8 shows that the execution time of queries increases when more keywords are specified by users. In order to retrieve partially matched query results, POIs that match any of the given keywords have to be taken into account. Consequently, more keywords will increase the number of POIs to be processed in the keyword match and query evaluation processes.

The difference in execution time between NEkNN and VDkNN remains nearly constant in all queries. As the number of keywords becomes larger, more POIs are considered in the keyword match process in both solutions. Moreover, varying the number of keywords does not directly enlarge or shrink the search area of both methods (i.e., the score of a POI is determined by the ranking function). Therefore, no apparent change of the difference in query performance between the two methods is observed. However, VDkNN always exceeds NEkNN in execution time in this experiment.

D. Number of k Experiment

Next, we evaluate the impact of k on the performance of NEkNN and VDkNN queries with the two data sets. We vary the value of k from 5 to 30 with an increment of five. Figure 9 illustrates that the execution time of queries increases as the number of k becomes larger. With both



(a) Real-world data set (b) Synthetic data set

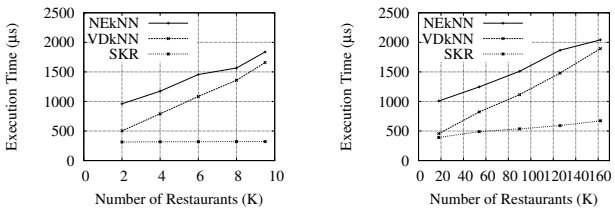
Fig. 8. Execution time over keyword number.

NEkNN and VDkNN, a larger search area has to be processed in order to retrieve more qualified results when we increase the k value. The performance difference between NEkNN and VDkNN becomes clear when k increases. Such a difference is proportional to the k value if POIs and networks are equally distributed. Apparently, given specific keywords, the cost of the keyword match process of NEkNN and VDkNN is identical. Therefore, the performance gain of VDkNN queries is from searches on the NVD where VDkNN can retrieve the top k candidates faster than NEkNN. When k increases, the search area is enlarged correspondingly and VDkNN is able to achieve more performance gains in the expanded search region.

E. Query Range Experiment

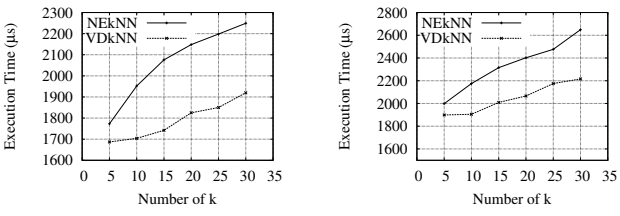
We examine the effect that varying the query range would have on the performance of SKR queries. In the experiments, SKR queries with various query ranges are evaluated in three different cases, which are queries with one (SKR-1), two (SKR-2), and three (SKR-3) keywords. Both Figures 10(a) and 10(b) illustrate that the execution time of queries grows exponentially with increasing query range. This is because the search area expands equally in all directions.

Interestingly, the execution time of the queries on real data sets are very close. Two factors mainly affect SKR. The first one is the number of POIs involved in the keyword match step. More POIs will be processed if more keywords are given. Furthermore, POIs that are fully keyword-matched are qualified candidates in SKR and a large number of partially keyword-matched POIs are filtered out by KCF. Consequently, fewer candidate POIs need to be processed in the range search phase. The two factors offset each other in range queries with relatively small search distances and data sets. However, when large amounts of POIs are searched with SKR, the overhead of the keyword match process becomes dominant in execution



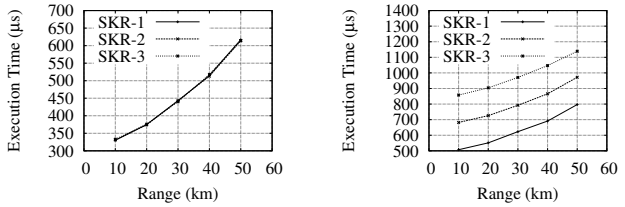
(a) Real-world data set (b) Synthetic data set

Fig. 7. Execution time over data set size.



(a) Real-world data set (b) Synthetic data set

Fig. 9. Execution time over k.



(a) Real-world data set (b) Synthetic data set

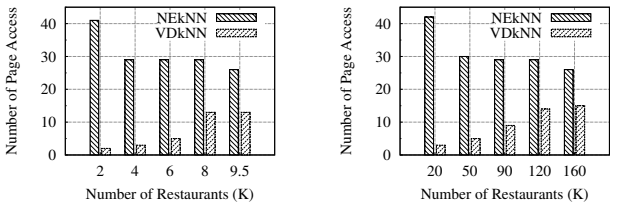
Fig. 10. Execution time over query range.

time. As shown in Figure 10(b), SKR-1 becomes the best and SKR-3 is the worst.

F. Page Access Experiment

Finally, we evaluate the number of page accesses by our proposed solutions. In these tests, we mainly focus on the comparison of NEkNN and VDkNN queries. Given a specific query, both solutions have the keyword match process. An identical number of keywords are retrieved from data sets. In addition, the POIs detected by NEkNN are required to be processed in VDkNN as well, and vice versa. The only difference is that NEkNN searches on spatial networks, whereas VDkNN explores on NVDs. Therefore, we evaluate the page access regarding network retrieval in these experiments. The page size is set to be 4 KB. The size of intersections or border nodes is 20 Bytes, containing their identifiers and coordinates. The road segments have a size of 20 Bytes as well, encompassing their identifiers, the identifiers of two endpoints, and the length of the road segment. A single page can accommodate either 200 nodes or road segments. The nodes and segments are stored continuously in pages. During a query process, each page is loaded only once.

Figure 11(a) and Figure 11(b) display the number of page accesses of NEkNN and VDkNN queries in real-world and synthetic data sets, respectively. The trend shared by the two figures is that as the data set size grows, the number of page accesses decreases in NEkNN, whereas it increases in VDkNN. The main reason is that NEkNN searches in a smaller area for qualified results in a larger dataset. Fewer intersections and road segments are retrieved by NEkNN. On the other hand, NVD becomes more complex when more border nodes and connections between borders are generated. Therefore, more page access is required in VDkNN with a larger POI data set.



(a) Real-world data set (b) Synthetic data set

Fig. 11. Page Access over data set size.

VI. Conclusion

Geographic information systems are becoming increasingly sophisticated, and spatial keyword search represents an important class of queries. Most existing solutions for evaluating spatial keyword queries are based on Euclidean distance and cannot provide partially matched results. In this research, we introduce efficient techniques to answer spatial keyword k nearest neighbor and spatial keyword range queries on spatial networks. We demonstrate the excellent performance of the proposed algorithms through extensive simulations.

References

[1] R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval - the concepts and technology behind search, Second edition*. Pearson Education Ltd., Harlow, England, 2011.

[2] A. Cary, O. Wolfson, and N. Rishe. Efficient and Scalable Method for Processing Top-k Spatial Boolean Queries. In *SSDBM*, pages 87–95, 2010.

[3] Y.-Y. Chen, T. Suel, and A. Markowetz. Efficient query processing in geographic web search engines. In *SIGMOD Conference*, pages 277–288, 2006.

[4] G. Cong, C. S. Jensen, and D. Wu. Efficient Retrieval of the Top-k Most Relevant Spatial Web Objects. *PVLDB*, 2(1):337–348, 2009.

[5] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[6] I. D. Felipe, V. Hristidis, and N. Rishe. Keyword Search on Spatial Databases. In *ICDE*, pages 656–665, 2008.

[7] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987.

[8] R. Hariharan, B. Hore, C. Li, and S. Mehrotra. Processing Spatial-Keyword (SK) Queries in Geographic Information Retrieval (GIR) Systems. In *SSDBM*, page 16, 2007.

[9] H. Hu, D. L. Lee, and V. C. S. Lee. Distance Indexing on Road Networks. In *VLDB*, pages 894–905, 2006.

[10] M. R. Kolahdouzan and C. Shahabi. Voronoi-Based K Nearest Neighbor Search for Spatial Network Databases. In *VLDB*, pages 840–851, 2004.

[11] W.-S. Ku, R. Zimmermann, H. Wang, and T. Nguyen. Annatto: Adaptive nearest neighbor queries in travel time networks. In *MDM*, page 50, 2006.

[12] W.-S. Ku, R. Zimmermann, H. Wang, and C.-N. Wan. Adaptive nearest neighbor queries in travel time networks. In *GIS*, pages 210–219, 2005.

[13] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*. Cambridge University Press, 2008.

[14] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Wiley Series in Probability and Statistics. John Wiley & Sons, Inc., 2^{nd} edition, 2000.

[15] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query Processing in Spatial Network Databases. In *VLDB*, pages 802–813, 2003.

[16] J. B. Rocha-Junior and K. Nørvåg. Top-k spatial keyword queries on road networks. In *EDBT*, pages 168–179, 2012.

[17] H. Samet, J. Sankaranarayanan, and H. Alborzi. Scalable network distance browsing in spatial databases. In *SIGMOD Conference*, pages 43–54, 2008.

[18] D. Wu, M. L. Yiu, C. S. Jensen, and G. Cong. Efficient Continuously Moving Top-K Spatial Keyword Query Processing. In *ICDE*, 2011.

[19] D. Zhang, Y. M. Chee, A. Mondal, A. K. H. Tung, and M. Kitsuregawa. Keyword Search in Spatial Databases: Towards Searching by Document. In *ICDE*, pages 688–699, 2009.

[20] Y. Zhou, X. Xie, C. Wang, Y. Gong, and W.-Y. Ma. Hybrid Index Structures for Location-based Web Search. In *CIKM*, pages 155–162, 2005.

[21] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Comput. Surv.*, 38(2), 2006.

[22] J. Zobel, A. Moffat, and K. Ramamohanarao. Inverted Files Versus Signature Files for Text Indexing. *ACM Trans. Database Syst.*, 23(4):453–490, 1998.