Efficient Reverse Top-*k* Boolean Spatial Keyword Queries on Road Networks

Yunjun Gao, *Member, IEEE*, Xu Qin, Baihua Zheng, *Member, IEEE*, and Gang Chen

Abstract—Reverse *k* nearest neighbor (R*k*NN) queries have a broad application base such as decision support, profile-based marketing, and resource allocation. Previous work on R*k*NN search does not take textual information into consideration or limits to the Euclidean space. In the real world, however, most spatial objects are associated with textual information and lie on road networks. In this paper, we introduce a new type of queries, namely, *reverse top-k Boolean spatial keyword* (R*k*BSK) *retrieval*, which assumes objects are on the road network and considers both spatial and textual information. Given a data set *P* on a road network and a query point *q* with a set of keywords, an R*k*BSK query retrieves the points in *P* that have *q* as one of answer points for their top-*k* Boolean spatial keyword queries. We formalize the R*k*BSK query and then propose *filter-and-refinement framework* based algorithms for answering R*k*BSK search with *arbitrary k* and *no any pre-computation*. To accelerate the query process, several novel *pruning heuristics* that utilize both spatial and textual information are employed to shrink the search space efficiently. In addition, a new data structure called *count tree* has been developed to further improve query performance. A comprehensive experimental evaluation using both real and synthetic data sets demonstrates the effectiveness of our presented pruning heuristics and the performance of our proposed algorithms.

Index Terms—Boolean spatial keyword query, reverse top-*k* Boolean spatial keyword query, road network, query processing

1 Introduction

R*k*NN retrieval has received lots of attention from the database research community in the past decade, due to its importance in a wide spectrum of applications such as decision support, profile-based marketing, and resource allocation [9], [21], [22]. Given a set P of data points and a query point q in a euclidean space, a reverse k nearest neighbor (RkNN) query finds the points in P that have q as one of their k nearest neighbors (NNs). Consider the example shown in Fig. 1, where two RNN ($k = 1$) queries are issued at q_1 and q_2 respectively in the euclidean space. The RNN of q_1 is Ø, as none of the objects takes q_1 as its nearest neighbor; and the RNN of q_2 is A as A's NN is q_2. RkNN search and its variants (e.g., [4], [8], [10]) have been well-studied in the literature. In this work, we enhance traditional RkNN retrieval from two aspects. First, different from existing RkNN search that assumes a euclidean space, we consider a road network. We believe this setting is more realistic since spatial objects in the real world are always restricted to the road network. Second, in addition to objects' spatial properties that are considered by existing RkNN queries, we also take into account textual characteristics of objects. The combination of spatial and textual properties offers greater flexibility to its users when looking for interesting objects. It also aligns nicely with the industry practice. For example, more and more real life applications call for new forms of queries that satisfy both spatial and textual constraints. In view of this, we propose a new type of queries, namely, *reverse top-k Boolean spatial keyword* (RkBSK) *query*, which assumes objects on the road network, and returns the objects having a specified query point q as one of the answer objects for the top-k Boolean spatial keyword **(TkBSK)** query.[1]

RkBSK queries constitute a suite of interesting and practical problems from not only the research point of view but also the application point of view. For instance, as illustrated in Fig. 1, assume that Hard Rock Cafe plans to open a new restaurant that serves *pizza*, *coffee*, and *steak* (represented as a set of keywords) in a new industry park. If there are two places (e.g., q_1, q_2) available to host the new restaurant, we need to identify a better one. One common strategy is to choose the place with fewer competitors. Obviously, if restaurant C takes the new restaurant as its nearest neighbor and all the items served by C will be served by the new restaurant as well, the restaurant C is considered as a competitor for the new restaurant. By taking into account both textual information and distance (i.e., the shortest path), the RkBSK query can find the location out of a given set of potential places that have the fewest competitors. In this case, q_2 offers a better choice, since it has fewer competitors compared with q_1. As another example, suppose all the customers subscribing to a coupon service specify their shopping interests via keywords (e.g., baby, clothing, mobile devices, etc.).

• Y. Gao, X. Qin, and G. Chen are with the College of Computer Science, Zhejiang University, 38 Zheda Road, Hangzhou 310027, China. E-mail: {gaoyj, ccrsno1, cg}@zju.edu.cn.
• B. Zheng is with the School of Information Systems, Singapore Management University, 80 Stamford Road, Singapore 178902, Singapore. E-mail: bhzheng@smu.edu.sg.

Manuscript received 18 Feb. 2014; revised 11 Aug. 2014; accepted 29 Sept. 2014. Date of publication 29 Oct. 2014; date of current version 27 Mar. 2015. Recommended for acceptance by G. Das.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2014.2365820

1. To be detailed later, a top-k Boolean spatial keyword query retrieves the k objects that are the closest to a given query point among all the objects containing all the query keywords.

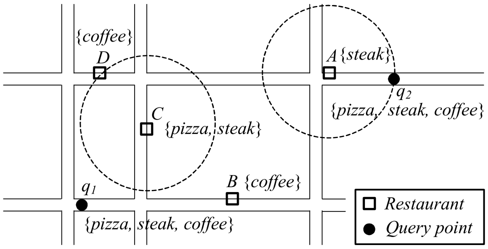


Fig. 1. Illustration of a motivating example.

The service provider can issue an R*k*BSK query at every shopping mall m with the textual keyword set to the products available in m. All the customers whose shopping interests could be satisfied by m and meanwhile have m as their closest shopping mall will be returned as the potential customer base for m. The service provider can send shopping coupons of the shopping mall m to m's potential customer base as they are more likely to shop in m, compared with other customers.

A simple way to answer R*k*BSK queries is to issue a top-k Boolean spatial keyword query at every data point $p \in P$, and those have q in their corresponding result sets form the answer set for R*k*BSK search. It is straightforward but very *inefficient*. It needs to traverse the *whole* dataset *multiple times* (i.e., at worst case $(|P|+1)$ times, 1 for fetching data points and $|P|$ times for verification), incurring *high* I/O overhead and *expensive* CPU cost.

Motivated by the significance of R*k*BSK queries and the lack of efficient search algorithm for processing R*k*BSK retrieval, in this paper, we propose efficient algorithms based on *filter-and-refinement framework* to support R*k*BSK search. Our solution utilizes both spatial and textual information to prune the search space significantly. Moreover, it can tackle *exact* R*k*BSK retrieval with an *arbitrary k*, without any pre-computation. In brief, our key contributions in this paper are summarized as follows:

- We identify the problem of R*k*BSK queries on road networks. To the best of our knowledge, this is the first work to address this problem.
- We propose efficient R*k*BSK search algorithms based on a *filter-and-refinement* framework, which can handle *arbitrary k* and has *no* any pre-computation.
- We develop several novel pruning heuristics for the filtering phase and the refinement phase, to effectively prune unqualified objects. In addition, we design a new data structure so-called *count tree* to further boost query performance.
- We conduct extensive experiments using both real and synthetic data sets to demonstrate the effectiveness of our presented pruning heuristics and the performance of our proposed algorithms.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 formulates the problem, introduces the index structure and reveals its characteristics. Sections 4 and 5 propose two efficient algorithms for processing R*k*BSK queries. Extensive experimental evaluation and our findings are reported in Section 6. Finally, Section 7 concludes the paper with some directions for future work.

2 Related Work

In this section, we overview the existing work related to R*k*BSK queries, focusing mostly on R*k*NN search and spatial keyword retrieval.

2.1 Conventional Spatial Queries

Since the concept of RNN was first introduced in [9], many algorithms have been proposed in answering RNN/R*k*NN query and its variants in euclidean spaces [8], [10], [21], [22], [27]. R*k*NN retrieval in road networks has also received significant attention. Safar et al. [19] deploy the network Voronoi diagram and apply a progressive incremental network expansion for processing RNN queries. Yiu et al. [29] present two methods, namely, eager algorithm and lazy algorithm, to tackle RNN search in a large graph. Cheema et al. [4] adopt a filter-and-refinement technique to solve continuous R*k*NN (CR*k*NN) search in euclidean spaces and road networks, respectively. Their approach does not require expensive pre-computation, by assigning each object and query with a safe region. Li et al. [13] also explore the CR*k*NN query on road networks. They present a new data structure, called DLM tree, to represent the whole monitoring region of a CR*k*NN query. However, it is worth noting that all the above approaches are unsuitable for R*k*BSK search because they only focus on spatial geometric information without considering any textual information.

Recently, the reverse top-k query is attracting much attention. Vlachou et al. [24] first indentify and solve reverse top-k queries. Later, they [25] also propose a new branch-and-bound algorithm called *BBR* to address the bichromatic reverse top-k query. Nevertheless, it is worth mentioning that, their work differs from ours in at least two aspects. First, their work is based on the weighting vector offered by users. Second, they do not take the textual constraint into consideration.

2.2 Spatial Keyword Queries

Combining traditional spatial queries with keywords has received considerable attention in the last few years [1], [2], [3], [5], [12], [31]. Boolean spatial keyword query and score based spatial keyword query are two important types of spatial keyword queries.

The Boolean spatial keyword query is to find the k objects nearest to the users' location among the set of objects whose textual description contains the query keyword set. Felipe et al. [7] augment the R-tree with a signature file, termed as *IR^2-tree*, to facilitate the top-k spatial keyword query. Unfortunately, the IR^2-tree inherits a drawback of false hits from the signature file. To overcome it, Tao and Sheng [23] develop a new access method, i.e., *spatial inverted* (SI) *index*, which extends the conventional inverted index to cope with this problem. As demonstrated in [23], SI index outperforms IR^2-tree. There are some other efforts on Boolean spatial keyword queries. Cary et al. [3] study the Boolean spatial keyword query under different logical semantics. Wu et al. [28] utilize an IR-tree to solve the problem of joint spatial keyword query processing. Cao et al. [2] investigate collective spatial keyword search, a variant of Boolean spatial keyword queries, which retrieves a group of spatial web objects such that the group's keywords cover query keywords and meanwhile the objects are closest to the query location and have the minimal inter-object distances. In particular, due to the complexity of this problem (i.e., NP-complete), they present solutions for both exact search and approximate search. Nonetheless, all the above queries differ from the R*k*BSK query as they only aim at the euclidean space.

Score based spatial keyword query aims to retrieve the *k* objects with the highest ranking scores, measured as a combination of their distances to the query location (a point) and the relevance of their textual descriptions to the query keywords. To address this, Cong et al. [6] propose an index, i.e., IR-tree, which combines an R-tree and an inverted file, to find the query result. Rocha-Junior et al. [17] develop a novel index named spatial inverted index (S2I) to boost the performance of the top-*k* spatial keyword query.

More recently, Lu et al. [14] investigate reverse spatial and textual *k* nearest neighbor (RST*k*NN) search, which takes into account textual similarity in R*k*NN retrieval. An RST*k*NN query is to find the objects that take a specified query object as one of their *k* most spatial-textual similar objects. It is worth noting that, their work is also different from ours. First, their rank function is based on the similarity score, which combines the spatial distance with textual similarity. Second, they only consider the euclidean space, and their algorithms use some geometric properties that are only valid in euclidean spaces but not road networks.

Last but not least, spatial keyword queries on road networks have also been studied in the literature [15], [18], [30], [32]. Rocha-Junior and Norvag [18] employ spatio-textual indexes that combine R-trees and inverted files to process the top-*k* spatial keyword query on road networks. Zhang et al. [32] explore the problem of diversified spatial keyword search on the road network, which takes into account both the textual relevance and the spatial diversity of the results. Zhang et al. [30] develop a spatial keyword query evaluation system that is comprised of keyword constraint filter, keyword and spatial refinement, and spatial keyword ranker for processing spatial keyword *k* nearest neighbor and spatial keyword range queries. It is worth pointing out that, these approaches are designed only for top-*k* spatial keyword queries on road networks, without considering the reverse version. Thus, they are not capable of supporting efficient R*k*BSK retrieval.

3 Preliminaries

In this section, we first formally define the R*k*BSK query on the road network, and then, we introduce the disk based storage model, and propose a baseline method (BM) which performs better than the naive approach mentioned in Section 1. Table 1 summarizes the symbols used frequently in this paper.

3.1 Problem Statement

In this paper, we model a road network by an undirected weighted graph $G = (V, E, W)$, in which V is a set of vertices (i.e., road conjunctions or road borders), E is a set of edges, and W is a set of weights that map every edge (n_i, n_j) in E to a positive real number (indicating the road distance or the travel time). Without loss of generality, we suppose bidirectional traffic, which is ubiquitous in real life. We also assume that a set of spatial objects *loc* (e.g., restaurants, hotels, etc.) associated with a set of keywords *key* (e.g., the menu of restaurants) lies on the road network. These spatial points are referred to as the points of interest (POIs), with each denoted by a two-vector tuple (*loc*, *key*). For two POIs p_1 and p_2, the path from p_1 to p_2 with the shortest distance represents the shortest path. The network distance between p_1 and p_2, denoted as $\|p_1, p_2\|$, is the length of the corresponding shortest path.

TABLE 1
Symbols and Description

Notation	Description
P	a set of points with keywords on a road network
q	a spatial query point with keywords
$\|p, p'\|$	the network distance between two points p and p'
SP_{qp}	The set of elements including vertexes, POIs, and edges located on the shortest path between q and p
S_c	the candidate set of POIs including all R*k*BSK points
S_r	the result set of an R*k*BSK query
$n_i[key].cnt$	the count # of the keyword set *key* of a node n_i
$TkBSK(q)$	the result set of a T*k*BSK query issued at q
$RkBSK(q)$	the result set of an R*k*BSK query issued at q

Definition 1 (top-k Boolean spatial keyword query on the road network). *Given a query q(loc, key), a parameter k, and a data set P with each POI* $p \in P$ *in the form of (loc, key), let* $P_{q.key}$ *be the set of POIs in P that contain q.key, i.e.,* $P_{q.key} = \{p \in P \mid q.key \subseteq p.key\}$. *A TkBSK query (on the road network) issued at q, denoted as TkBSK(q), returns the k POIs in* $P_{q.key}$ *having the minimal network distances to q, formally,* $TkBSK(q) = \{S \subseteq P_{q.key} \mid |S| = k \wedge \forall s \in S, \forall p \in (P_{q.key} - S), \|q, s\| \leq \|q, p\|\}$. *For any data point in TkBSK(q), we say that it is one of the Boolean spatial keyword nearest neighbors of q.*

Definition 2 (reverse top-k Boolean spatial keyword query on the road network). *Given a query q(loc, key), a parameter k, and a data set P, an RkBSK query (on the road network) issued at q, denoted as RkBSK(q), retrieves all the POIs in P whose top-k Boolean spatial keyword queries include q, formally,* $RkBSK(q) = \{p \in P \mid q \in TkBSK(p)\}$.

As an example, in Fig. 2, *T1BSK*(q) is p_1 as $q.key \subseteq p_1.key$ and p_1 is the nearest neighbor of q. Also, p_1 is an answer point of *R1BSK*(q) due to $q \subseteq T1BSK(p_1)$.

After formulating the R*k*BSK query, we are ready to reveal its important properties, as stated in Property 1 and Property 2, which can be utilized to handle R*k*BSK search. Property 1 states that the size of a result set for an R*k*BSK query could be very different from *k*. As shown in Fig. 2, given an R3BSK ($k = 3$) query issued at q with a keyword set $q.key = \{a, b\}$, the result set $R3BSK(q) = \{p_1\}$ whose size is different from *k*. Property 2 states that the relationship between an answer point p for the R*k*BSK query issued at q and the query point q, in terms of their keywords set. As depicted in Fig. 2, POI p_9 cannot be an answer point for

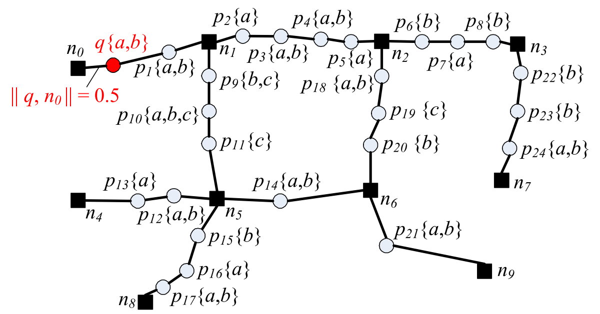


Fig. 2. Example of the road network with POIs.

RkBSK(q) as $p_9.key \not\subseteq q.key$ while p_1 is a potential answer point due to $p_1.key \subseteq q.key$.

Property 1. *Given an RkBSK query issued at q with a fixed k, its result set (i.e., RkBSK(q)) varies, which depends on the position of q, the keyword set of q, and the distribution of data points.*

Property 2. *Given a query point q with a keyword set q.key, for any point* $p \in$ *RkBSK(q), we have* $p.key \subseteq q.key$.

Proof. Assume the above statement is not valid, and there is at least one point $p \in$ *RkBSK*(q) with $p.key \not\subseteq q.key$. Thus, based on Definition 2, $q \in$ *TkBSK*(p). According to Definition 1, for $q \in$ *TkBSK*(p), we have $p.key \subseteq q.key$, which contradicts with our assumption $p.key \not\subseteq q.key$. Hence, property 2 holds, and the proof completes. □

3.2 Disk Based Storage Model

In real-life applications, the size of a road network and its POIs could be very large. Therefore, we assume that the road network and its POIs are too large to be fit in main memory, and we design a disk-based storage model to support our algorithms seamlessly. The model we adopt is to group network nodes based on their connectivity and distances, as proposed in [20]. A graphical illustration of an adjacency file and a point file along with the index for our example road network is shown in Fig. 3. Our model allows efficient access to the adjacency lists and points which are stored in the adjacency file and the point 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 the 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, the distance between p and the edge node with smaller ID, and p's associated set of keywords. A group of POIs are stored in ascending order of their offset distances to the node with smaller ID. The adjacency file stores an adjacency list for each node. 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, we store ID, the edge distance between node n_i and n (i.e., $\|n_i, n\|$), and a pointer to its POI group in the point file. If there is no POI on this edge, a NULL pointer is kept. Take the node n_1 in Fig. 2 as an example. As shown in Fig. 3, it has three adjacent nodes,

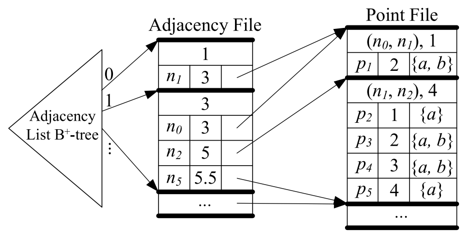


Fig. 3. Example of the disk based storage model.

and thus, we store 3 at the beginning of $n_1's$ adjacent list. Thereafter, three adjacent nodes (i.e., n_0, n_2, n_3) are stored. For each adjacent node, we store its ID (e.g., n_0), the edge length (e.g., $\|n_0, n_1\| = 3$), and a pointer to POIs on the edge (e.g., p_1 is located on the edge (n_0, n_1)).

3.3 Baseline Method

As mentioned in Section 1, a naive solution for the R*k*BSK query is to invoke $|P|$ times T*k*BSK queries to form a R*k*BSK result, i.e., for each POI p in P we need to expand the road network around p to form *TkBSK*(p) and judge whether q belongs to *TkBSK*(p). In worst case, the whole data set has to be traversed ($|P|+1$) times, i.e., one for fetching data points and $|P|$ times for verification using T*k*BSK search, resulting in high I/O overhead and expensive CPU cost, especially when $|RkBSK(q)| \ll |P|$. To improve performance, we develop a non-trivial *baseline method* that performs much better than the above naive solution. It is worth noting that, BM utilizes the properties presented in Section 3.1 to prune unqualified data points effectively.

To facilitate R*k*BSK retrieval, we adopt a filter and refinement framework. In the filtering step, we expand the road network from q based on Dijkstra's algorithm. During the expansion, we preserve all the data points p encountered that satisfy the keyword constraint (i.e., $p.key \subseteq q.key$) in a candidate set S_c. The filtering step stops only when the whole road network has been explored. In the refinement step, we verify all the candidate points preserved in S_c. A data point $p \in$ *RkBSK*(q) iff it satisfies $q \in$ *TkBSK*(p). Instead of issuing a T*k*BSK query at p like the naive approach does, we adopt a *Boolean Verification* (BV) method as presented in Algorithm 1. The basic idea is to count the number of points p' with $p.key \subseteq p'.key$ and $\|p, p'\| \leq \|p, q\|$, denoted as *count*. If *count* $< k$, it is guaranteed that $q \in$ *TkBSK*(p) and the algorithm returns TRUE; otherwise, it returns FALSE. To simplify our discussion, given two adjacent nodes n_i and n_j with n_i being visited before n_j during the network expansion, we call the edge (n_i, n_j) as *n_j's previous edge*, and refer to the edge(s) from n_j to its adjacent node(s) visited after (n_i, n_j) as *n_j's next edge*(s). Take the road network illustrated in Fig. 2 as an example. Assume that we expand the road network in the order of $n_0, n_1, n_2, n_5, n_3, n_6, \ldots$. Then, for n_1, its previous edge is (n_0, n_1), and its next edges are (n_1, n_2) and (n_1, n_5). Similarly, for n_2, its previous edge is (n_1, n_2), and its next edges are (n_2, n_3) and (n_2, n_6).

4 RKBSK ALGORITHM

In this section, we propose our R*k*BSK algorithm that is based on two newly developed lemmas to shrink the search space. In the following, we first present two lemmas and then describe our R*k*BSK algorithm.

Algorithm 1. Boolean Verification Algorithm

Input: a data point p to be evaluated, a query point q, a parameter k
Output: TRUE if $q \in TkBSK(p)$, otherwise FALSE

locate the edge (n_i, n_j) that p locates and initialize $count = 0$
priority queue $U = \{\langle (p, n_i), \|n_i, p\| \rangle, \langle (p, n_j), \|n_j, p\| \rangle\}$ // edges in U are sorted in ascending order of their distances to p
while U is not empty **do**
  edge $(n, n') =$ de-queue (U)
  $count \mathrel{+}= |\{p'$ on edge $(n, n') \wedge p.key \subseteq p'.key \wedge \|p, p'\| \leq \|p, q\|\}|$
  if $count \geq k$ **then return** FALSE
  for each unvisited adjacent nodes n'' of n' **do**
    en-queue $\langle (n', n''), \|n'', p\| \rangle$ to U
  if $\|p, n'\| > \|p, q\|$ **then break**
return TRUE

4.1 Theoretical Foundation

The main drawback of BM algorithm can be summarized as follows. First, it needs to explore the entire road network, even if the answer points are all located near the query point. Second, it has to verify all the data points that satisfy the keyword constraint (i.e., all the points in S_c). When the keyword constraint is common and/or the data set is huge, the size of the candidate set S_c might be large (i.e., $|S_c| >> |RkBSK(q)|$). Evaluating all the candidate points in S_c using BV algorithm (presented in Algorithm 1) could be costly. To address these, we develop two lemmas to prune away unqualified data points and terminate the network expansion earlier.

Lemma 1. *Let $q(loc, key)$ be a query point, p be a data point with $p.key \subseteq q.key$, SP_{qp} be the shortest path from p to q, and S_{l1} be the set of data points (including p) located on SP_{qp} with their keyword sets the same as $p.key$, i.e., $S_{l1} = \{p' \in P \mid p'.key = p.key \wedge p' \in SP_{qp}\}$. Then, we have $p \in RkBSK(q) \rightarrow |S_{l1}| \leq k$.*

Proof. Assume, on the contrary, that Lemma 1 is not valid, and we have $p \in RkBSK(q)$ and meanwhile $|S_{l1}| > k$. Without loss of generality, we assume that points in S_{l1} (i.e., $p_1, p_2, \ldots, p_k, p_{k+1}, \ldots$) are sorted in ascending order of their distances to q (i.e., $\|p_i, q\| \leq \|p_{i+1}, q\|$), and let the point p be the last data point. Obviously, these k points in S_{l1} have their minimal distances to q smaller than $\|p, q\|$ and meanwhile have their keywords covered by $q.key$, i.e., $\forall i \in [1, k]$, $p_i.key = p.key \subseteq q.key$ and $\|p_i, q\| \leq \|p, q\|$. As all the points in S_{l1} lie on the shortest path SP_{qp}, we have $\|p_i, p\| = \|p, q\| - \|p_i, q\| < \|p, q\|$. Consequently, $q \notin TkBSK(p)$ and $p \notin RkBSK(q)$, which contradicts with our assumption that $p \in RkBSK(q)$. Thus, our assumption is invalid, and the proof completes. □

In order to illustrate Lemma 1, let us consider the example shown in Fig. 4. Assume that an R2BSK ($k = 2$) query is issued at a query point q with $q.key = \{a, b\}$. Let the path depicted in Fig. 4 be the shortest path from a node n_1 to q, denoted as SP_{qn1}. Given the fact that points p_1 and p_2 are located on SP_{qn1} and meanwhile $p_1.key = p_2.key = \{a\} \subseteq q.key$, all the points p with $p.key = \{a\}$ located on SP_{qn1} after p_1 and p_2 cannot be the actual answer point(s) for the R2BSK query according to Lemma 1. In other words, during the network expansion from q, the expansion via SP_{qn1} can safely ignore any point p with $p.key = \{a\}$, which helps to reduce the size of S_c.

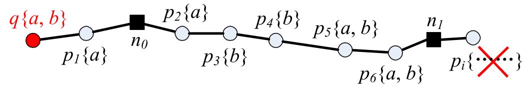


Fig. 4. Example of the shortest path SP_{qn1} on a road network.

Lemma 2. *Given a query point $q(loc, key)$ and the shortest path SP_{qn} from q to a node n, let set S_{key_i} preserve all the candidate points p for $RkBSK(q)$ located on SP_{qn} and having $p.key = key_i \subseteq q.key$. If $\forall$ $key_i \subseteq q.key$, we have $|S_{key_i}| = k$ (i.e., $\sum_{\forall key_i \subseteq q.key} |S_{key_i}| = (2^{|q.key|} - 1) \times k$), and the network expansion via SP_{qn} can be safely terminated.*

Proof. Assume that the above statement is not valid, and there is at least one point p that belongs to $RkBSK(q)$ but its shortest path to q bypasses the node n, i.e., $\|q, n\| < \|q, p\|$. As $p \in RkBSK(q)$, $p.key \subseteq q.key$. Without loss of generality, let $p.key = key_i \subseteq q.key$, i.e., point p will be included in set S_{key_i}. In other words, $|S_{key_i}| = k + 1$, which contradicts with Lemma 1. Hence, our assumption is invalid, and the proof completes. □

Continue our example illustrated in Fig. 4. Given $q.key = \{a, b\}$, there are in total three (i.e., $2^{|q.key|} - 1 = 3$) possible subsets (i.e., $key_1 = \{a\}$, $key_2 = \{b\}$, and $key_3 = \{a, b\}$). On the shortest path SP_{qn1} from q to n_1, as depicted in Fig. 4, we have $S_{key_1} = \{p_1, p_2\}$, $S_{key_2} = \{p_3, p_4\}$, and $S_{key_3} = \{p_5, p_6\}$. Thus, as guaranteed by Lemma 2, the network expansion via SP_{qn_1} can be safely stopped at the node n_1.

4.2 Algorithm Details

Based on the aforementioned two Lemmas, we present an algorithm called *RkBSK algorithm* to retrieve the exact result of an R*k*BSK query. In particular, our R*k*BSK algorithm improves the filtering step of BM algorithm by terminating the network expansion earlier as guided by Lemma 2. Next, we detail two steps of R*k*BSK algorithm.

In general, R*k*BSK algorithm shares the same filtering step as BM algorithm. It expands the road network from q based on Dijkstra's algorithm to form the candidate set S_c. The only difference is that R*k*BSK algorithm is more selective in both candidate set formation and network expansion. First, it does not blindly insert all the data points p with $p.key \subseteq q.key$ into S_c like BM algorithm does. As shown in Algorithm 2 (lines 10-11), whenever a data point p satisfying the textual constraint is encountered, it checks the number of data points in S_c that share the same keywords as p and meanwhile lie on the shortest path from q to p (and then to node n'), which is preserved by $n'[p.key].cnt$. Point p is a potential answer point for $RkBSK(q)$ only if $n'[p.key].cnt < k$, as guaranteed by Lemma 1. Second, it enables an early termination for the network expansion while BM algorithm has to explore the whole road network. As depicted in Algorithm 2 (lines 12-14), the expansion at node n is necessary only when the condition listed in Lemma 2 is not satisfied, i.e., at least for one keyword set $key_i \subseteq q.key$, the number of data points p with $p.key = key_i \subseteq q.key$ located on the shortest path from q to n is smaller than k. As demonstrated in our experiments, these two lemmas can significantly boost the search performance.

Algorithm 2. Filter for R*k*BSK Algorithm (R*k*BSK-Filter)

Input: $q(loc, key)$, k, a set P of data points on a road network
Output: the candidate set S_c of an R*k*BSK query

1: locate the edge (n_i, n_j) that q is located (suppose q is closer to n_i)
2: $U = \{\langle(q, n_i), \|n_i, q\|\rangle, \langle(q, n_j), \|n_j, q\|\rangle\}$ // edges in U are sorted in ascending order of their distances to q
3: **while** U is not empty **do**
4: $e = (n, n') =$ de-queue (U)
5: **if** q is located on the edge e **then**
6: **for** each subset $key \subseteq q.key$ **do** $n'[key].cnt = 0$
7: else
8: **for** each subset $key \subseteq q.key$ **do** $n'[key].cnt = n[key].cnt$
9: **for** every point p on e **do** // visit points based on ascending order of their distances to q
10: **if** $p.key \subseteq q.key$ and $n'[p.key].cnt < k$ **then** // Lemma 1
11: $n'[p.key].cnt{+}{+}$ and $S_c = S_c \cup \{p\}$
12: **if** $\exists n'[^*].cnt < k$ **then** // Lemma 2
13: **for** each unvisited edge (n', n'') in the edge set E **do**
14: en-queue $\langle(n', n''), \|n'', q\|\rangle$ to U
15: **return** S_c

Algorithm 2 presents the pseudo-code of the filtering step of R*k*BSK algorithm. It first locates the edge (n_i, n_j) where q locates (line 1). The priority queue U maintains all the edges (n, n') to be examined sorted based on ascending order of their distances to q, and initially it has two edges (q, n_i) and (q, n_j) (line 2). Thereafter, the network expansion starts by continuously de-queuing the first element of U until U is empty (lines 3-14). For each de-queued edge (n, n'), the algorithm needs to initialize the count list of n' to facilitate the checking of lemmas. To be more specific, for a given node n', each element of its count list corresponds to one subset keyword key_i of $q.key$, and it records the number of data points p located on the shortest path from q to n' with $p.key = key_i$, denoted as $n'[key_i].cnt$ (lines 5-8). Note that, we treat the edge that q is located on different from other edges. For the edge (n_i, n_j) containing q, the count lists of nodes n_i and n_j are initialized to zero since there is no other node located on the shortest paths from q to n_i or n_j (lines 5-6). For all the other edges (n, n'), the algorithm initializes the count list of n' by copying the count list of node n (lines 7-8). Then, it checks the data points located on (n, n') and updates the count list if necessary (lines 9-11). Note that, the algorithm only enrolls a data point p into the candidate set S_c if it cannot be discarded by Lemma 1. Once the examination of edge (n, n') finishes, the algorithm needs to en-queue the next edge(s) of n', if any, to U in order to expand the network. Again, as guided by Lemma 2, n' requires expansion only if the shortest path from q to n' does not contain sufficient candidate points (lines 12-14). Finally, the algorithm returns the candidate set S_c to complete the filter step.

For the refinement step, our R*k*BSK algorithm does exactly what BM algorithm does, and thus is omitted. It validates every data point p in the candidate set S_c using BV algorithm (depicted in Algorithm 1).

Example 1. For ease of understanding, we illustrate how R*k*BSK algorithm works using an example. Based on the road network shown in Fig. 2, we assume that an R2BSK ($k = 2$) query with keywords $\{a, b\}$ is issued at a query point q on edge (n_0, n_1) with $\|q, n_0\| = 0.5$. Initially, the priority queue U contains $\{\langle(q, n_0), 0.5\rangle, \langle(q, n_1), 2.5\rangle$. Then, the network expansion starts. The first de-queued edge is (q, n_0). As it is the edge where q locates, an empty count list is initialized for the node n_0 (i.e., $n_0[a].cnt = n_0[b].cnt = n_0[a, b].cnt = 0$). Since there is no any data point on the edge (q, n_0) and n_0 does not have any not-yet-marked adjacent node, no action is taken. The second de-queued edge is (q, n_1), and again an empty count list is initialized (i.e., $n_1[a].cnt = n_1[b].cnt = n_1[a, b].cnt = 0$). As there is one point $p_1(\{a, b\})$ located on (q, n_1), its count list is updated (i.e., $n_1[a, b].cnt = 1$), and p_1 is enrolled into the candidate set S_c ($= \{p_1\}$). The algorithm then en-queues $\langle(n_1, n_2), 7.5\rangle$ and $\langle(n_1, n_5), 8\rangle$ into U to complete the evaluation of the edge (q, n_1). Next, the algorithm evaluates edge (n_1, n_2). It locates three candidate points $p_2(\{a\})$, $p_3(\{a,b\})$, and $p_5(\{a\})$ that updates S_c to $\{p_1, p_2, p_3, p_5\}$, and the count list of n_2 is updated accordingly, with $n_2[a].cnt = 2$, $n_2[b].cnt = 0$, $n_2[a, b].cnt = 2$. Note that, $p_4(\{a, b\})$ is not a candidate point as $n_2[a, b].cnt$ has already reached 2. Since $n_2[b].cnt$ is not yet 2, n_2 requires further expansion, and both $\langle(n_2, n_3), \| q, n_3\|\rangle$ and $\langle(n_2, n_6), \| q, n_6\|\rangle$ are en-queued. Then, it verifies edge (n_1, n_5). As the algorithm does not locate any data point satisfying the textual constraint, it en-queues $\langle(n_5, n_4), \|q, n_4\|\rangle$, $\langle(n_5, n_6), \|q, n_6\|\rangle$, and $\langle(n_5, n_8), \|q, n_8\|\rangle$ into U. Next, edge (n_2, n_3) is de-queued, and it has two candidate points $p_6(\{b\})$ and $p_8(\{b\})$ which update $n_3[b].cnt = 2$. Here, $n_3[a].cnt = n_3[b].cnt = n_3[a, b].cnt = 2$. That is to say, the network expansion at node n_3 can be safely terminated, even if n_3 has not-yet-visited adjacent nodes. The evaluation proceeds until $U = \emptyset$.

In the R*k*BSK refinement step, it evaluates all candidates using BV algorithm, in the same way as BM algorithm does.

5 Enhanced R*k*BSK Algorithm

Compared with Baseline algorithm, our proposed R*k*BSK algorithm actually shrinks the expanded network area and meanwhile reduces the size of candidate set S_c. However, the candidate set formed by R*k*BSK during the filtering step is still much larger than the real result set *RkBSK*(q), especially when $|q.key|$ is large. Furthermore, R*k*BSK has to evaluate all the candidate points in S_c as it does not implement any further pruning for the candidate points. In the sequel, we first present Heuristic 1 and Heuristic 2 which can efficiently cut down the size of the candidate set; and Heuristic 3 to enable candidate point pruning. Then, we introduce a new data structure, *count tree*, to facilitate the implementation of our newly proposed heuristics. Finally, we propose an *enhanced RkBSK algorithm* with better search performance.

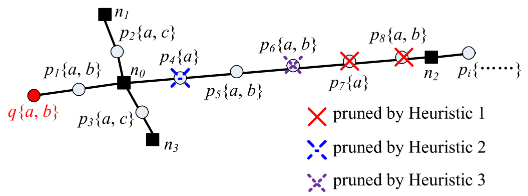
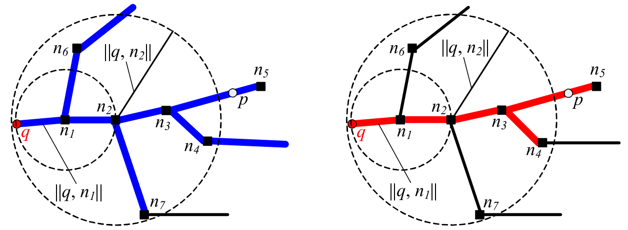


Fig. 5. Illustration of Heuristics 1, 2, and 3.



(a) Original Heuristic 2 (b) Simplified Heuristic 2

Fig. 6. Illustration of simplified Heuristic 2.

5.1 Pruning Heuristics

Heuristic 1. *Given a query point q(loc, key) and a data point p located on the road network, let* S_{H1} *be the set of data points* p' *located on the shortest path* SP_{qp} *from q to p (i.e.,* $p' \in SP_{qp}$*) that have their keywords covering p.key, i.e.,* $S_{H1} = \{p' \in P \mid p.key \subseteq p'.key \wedge p' \in SP_{qp}\}$. *If* $|S_{H1}| \geq k$, *it is certain that* $p \notin RkBSK(q)$; *otherwise* $|S_{H1}| < k$ *if* $p \in RkBSK(q)$.

Proof. First, we prove the first statement, i.e., $|S_{H1}| \geq k \rightarrow p \notin RkBSK(q)$, via contradiction. Assume that this statement is not valid, i.e., $|S_{H1}| \geq k \wedge p \in RkBSK(q)$. If a T*k*BSK query is issued at p, based on the fact that $\forall p' \in S_{H1}$, $p.key \subseteq p'.key \wedge \|p, q\| > \|p', q\| \wedge |S_{H1}| \geq k$, and thus, q cannot be an answer point for $TkBSK(p)$, which contradicts with our assumption that $p \in RkBSK(q)$. Hence, our assumption is invalid, and the statement $|S_{H1}| \geq k \rightarrow p \notin RkBSK(q)$ holds.

Next, we prove the second statement that $|S_{H1}| < k$ if $p \in RkBSK(q)$ via contradiction as well. Assume that there is at least one answer point $p \in RkBSK(q)$ having its $|S_{H1}| \geq k$. Similar as the above proof, we have a T*k*BSK query issued at p. Based on the fact that $\forall p' \in S_{H1}$, $p.key \subseteq p'.key \wedge \|p, q\| > \|p', q\| \wedge |S_{H1}| \geq k$, and hence, we are certain that $q \notin TkBSK(p)$, which contradicts with our assumption that $p \in RkBSK(q)$. The proof completes. □

Compared with Lemma 1 and Lemma 2 used by our R*k*BSK algorithm, Heuristic 1 implies a stronger pruning criterion. Lemma 1 prunes away a point p based on those data points located on the shortest path SP_{qp} from q to p and having exactly the same keywords as p; while Heuristic 1 discards the point p based on those data points located on SP_{qp} and having their keywords covering $p.key$. Besides, Heuristic 1 also serves as an *early termination condition*. For example, if we have found at least k points bounding each non-empty subset of $q.key$ on SP_{qn}, we can safely terminate examination because no qualified data points will have the shortest path to q passing n.

Consider, for instance, the example shown in Fig. 5. Assume that an R3BSK ($k = 3$) query is issued at a query point q, and currently we are evaluating the data points on edge (n_0, n_2), based on ascending order of their distances to q (i.e., in the order of $p_4, p_5, p_6, p_7, p_8, \ldots$). When point p_7 is evaluated, we have its corresponding $S_{H1} = \{p_1, p_4, p_5, p_6\}$. Since $|S_{H1}| > k$, the point p_7 can be safely pruned by Heuristic 1. Similarly, for point p_8, we have its corresponding $S_{H1} = \{p_1, p_5, p_6\}$. As $|S_{H1}| > k$, the point p_8 can also be safely discarded by Heuristic 1.

Heuristic 2. *Given a query point q(loc, key), let n be one of the vertices passed by the shortest path* SP_{qp} *from q to p, and* S_{H2} *be the set of data points that have their distances to n smaller than the distance from q to n and meanwhile have their keywords covering p.key, i.e.,* $S_{H2} = \{p' \in P \mid \|p', n\| < \|q, n\| \wedge p.key \subseteq p'.key\}$. *If* $|S_{H2}| \geq k$, *then* $p \notin RkBSK(q)$.

Proof. Since n is one of the vertices passed by the shortest path SP_{qp} from p to q, we have $\|p, q\| = \|p, n\| + \|n, q\|$. On the other hand, based on the triangle inequality, for $\forall p' \in S_{H2}$, we have $\|p', p\| \leq \|p', n\| + \|n, p\| < \|q, n\| + \|n, p\| = \|q, p\|$. As $|S_{H2}| \geq k$, it is certain that q cannot be an answer point for $TkBSK(p)$, and hence, $p \notin RkBSK(q)$. The proof completes. □

Heuristic 2 considers not only those data points located on a specified shortest path, but also all the points located around any node vertex on the shortest path. During the network expansion, Heuristic 2 can serve as a supplement to Heuristic 1. In the following, we first illustrate how Heuristic 2 can help to prune away unqualified data points using an example, and then, we will present a new structure to implement Heuristic 2 in Section 5.2.

Consider the example depicted in Fig. 5 again, and suppose an R3BSK query is issued at q. Assume that the network expansion reaches point p_4, and its shortest path from q passes node n_0. If the network expansion has already identified three data points p_1, p_2, and p_3 around the node n_0 with $p_1.key = \{a, b\}$, $p_2.key = \{a, c\}$, and $p_3.key = \{a, c\}$. Clearly, all these three data points have their distances to n_0 smaller than $\|q, n_0\|$, and their keyword sets all contain $p_4.key = \{a\}$, i.e., $S_{H2} = \{p_1, p_2, p_3\}$. As $|S_{H2}| \geq k = 3$, p_4 can be safely pruned away by Heuristic 2.

In order to prune a point p, Heuristic 2 actually considers the points located around each node on the shortest path SP_{qp} from q to p. Nonetheless, this kind of checking might be expensive, and it does not align with our network expansion order. In this paper, we adopt an approximated implementation of Heuristic 2, and try to integrate Heuristic 2 with our network expansion. Instead of considering all the nodes located on SP_{qp}, we only take into account the node n closest to p; instead of considering all the points with their distances to n bounded by $\|q, n\|$ and meanwhile having their keywords satisfying the textual constraint, we only take a subset into consideration, i.e., those points located on SP_{qp} and those points located on n's next edge(s). We illustrate the difference between our implementation and Heuristic 2 in Fig. 6. Assume that the network expansion just reaches point p, which is located on edge (n_3, n_5). Now we need to check whether p can be discarded by Heuristic 2. The original Heuristic 2 needs to examine all the nodes located along SP_{qp}, i.e., nodes n_1, n_2, and n_3. For each node n, we need to find its corresponding $S_{H2} = \{p' \mid \|p', n\| < \|q, n\| \wedge p.key \subseteq p'.key\}$. In other words, we have to issue a range query around every node n

along SP_{qp} in order to identify those points p' with $\|p', n\| < \|q, n\|$. As shown in Fig. 6a, those bold edges represent the set of edges Heuristic 2 has to scan. Our implementation simplifies the processing. We only consider *one* node along SP_{qp} that is closest to p, e.g., node n_3 in this example. For node n_3, we do not take into account *all* the points p' with $\|p', n_3\| < \|q, n_3\|$. Instead, we only consider those points located on the shortest path from q to n_3, and those located on n_3's next edges, i.e., edge (n_3, n_4) and edge (n_3, n_5). In other words, the bold edges in Fig. 6b denote the set of edges our simplified checking needs to scan.

Heuristic 3. *Given a query point q(loc, key) and a data point p with p.key $\subseteq$ q.key, let S_{H3} be the set of points p' whose keywords are subsets of p and whose shortest paths $SP_{qp'}$ actually pass p, i.e., $S_{H3} = \{p' \mid p'.key \subseteq p.key \wedge p \in SP_{qp'}\}$. If $p \notin$ RkBSK(q), it is certain that all the points in S_{H3} cannot be the answer points for RkBSK(q), i.e., $p \notin RkBSK(q) \rightarrow \forall\, p' \in S_{H3}, p' \notin RkBSK(q)$.*

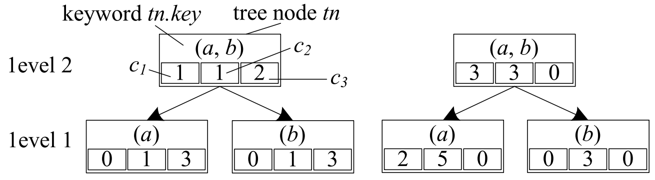
Proof. Assume that the above statement is invalid, i.e., there is at least one point $p' \in S_{H3}$ that belongs to the result set $RkBSK(q)$. Based on the definition of R*k*BSK search, $p' \in RkBSK(q) \rightarrow q \in TkBSK(p')$. As $|TkBSK(p')| = k$, q and another $(k-1)$ points form the result set $TkBSK(p')$. On the other hand, we know that $p \notin RkBSK(q)$, and thus, $q \notin TkBSK(p)$. Since $|TkBSK(p')| = |TkBSK(p)| = k$, $q \notin TkBSK(p)$ and $q \in TkBSK(p')$, i.e., there is at least one point p'' ($\neq q$) such that $p'' \in TkBSK(p)$ and $p'' \notin TkBSK(p')$. As $p'' \notin TkBSK(p')$, $q \in TkBSK(p')$, $p'.key \subseteq q.key$, and $p'.key \subseteq p''.key$, we have $\|p', q\| < \|p', p''\|$. Since the shortest path $SP_{qp'}$ from q to p' actually passes by p, we have $\|p', q\| = \|p, q\| + \|p, p'\|$. Based on the triangle inequality, we have $\|p', p''\| \leq \|p, p''\| + \|p, p'\|$. Therefore, the above inequation $\|p', q\| < \|p', p''\|$ can be converted to $\|p, q\| + \|p, p'\| < \|p, p''\| + \|p, p'\|$, i.e., $\|p, q\| < \|p, p''\|$, which contradicts with the fact that $p'' \in TkBSK(p)$ but $q \notin TkBSK(p)$ with $p.key \subseteq q.key$ and $p.key \subseteq p''.key$. Consequently, our assumption is invalid and the proof completes. □

Back to our example shown in Fig. 5 with an R3BSK query issued at q. As discussed earlier, points p_4, p_7, and p_8 can be pruned by Heuristic 1 and Heuristic 2. In other words, only points p_1, p_5, and p_6 are in the candidate set, and we need to invoke BV algorithm to verify each of them. However, using Heuristic 3, once we know p_5 is not a real answer point for *R3BSK(q)*, we can discard p_6 without any further evaluation. This is because the shortest path from q to p_6 passes p_5 and $p_6.key \subseteq p_5.key$.

To sum up, three Heuristics developed in this section can help to prune points p_4, p_6, p_7, and p_8, as illustrated in Fig. 5. Their pruning power will be also verified through extensive experiments to be presented in Section 6.

5.2 The Count Tree

In order to further improve search performance, we also propose a novel data structure so-called *count tree* as a replacement of the count list used in our R*k*BSK algorithm. The main drawback of the count list is that it has no fast access method to fetch all subsets of a given keyword set, which definitely affects the search efficiency.



(a) $CT(n_1)$ (b) $CT(n_2)$

Fig. 7. Example of count trees.

As shown in Fig. 7, the count tree is comprised of $2^{|q.key|}$-1 nodes, and each tree node *tn* in the count tree corresponds to a non-empty subset *tn.key* of *q.key*, i.e., $tn.key \subseteq q.key$. In addition to the keyword *tn.key*, it also maintains three counters, namely, c_1, c_2, and c_3. Here, c_1 represents the number of points p' located on the shortest path SP_{qn} from q to n with $p'.key = tn.key$, c_2 denotes the number of points p' located on SP_{qn} with $tn.key \subseteq p'.key$, and c_3 represents the number of points located on n's next edges. In other words, counter c_1 is to serve Lemma 1, counter c_2 is to serve Heuristic 1, and counter c_3 is to serve Heuristic 2. We will utilize Example 2 to further explain these three counters later. The height of the count tree is set to $|q.key|$. Assume that all the leaf nodes are at level 1, and the root node is at level $|q.key|$. Then, nodes in the lth level of the count tree correspond to the keywords with length l, e.g., a leaf node at level 1 only contains one keyword of *q.key*, a node at level 2 includes two keywords of *q.key*, and so forth. A tree node tn_1 at level $(l+1)$ is a parent of a tree node tn_2 at level l if the keywords of tn_2 are a subset of the keywords corresponding to node tn_1, i.e., $tn_2.key \subseteq tn_1.key$.

Example 2. Take the road network depicted in Fig. 2 as an example. Assume that an R3BSK query is issued at a query point q with $q.key = \{a, b\}$, and it expands the road network in order of $n_0, n_1, n_2, n_5, n_3, n_6, \ldots$. When node n_1 is encountered, a new count tree $CT(n_1)$ is created. As shown in Fig. 7a, it has two levels as $|q.key| = 2$. For each tree node *tn*, its counters c_1 and c_2 are initially set to 0, and they are increased only when a point p' located on the shortest path from q to n_1 (i.e., edge (q, n_1)) with $tn.key = p'.key$ or $tn.key \subseteq p'.key$ is found. Given the fact that there is only one point p_1 ($\{a, b\}$) located on the edge (q, n_1), the counter c_1 of $\{a, b\}$ is increased by 1, and the counters c_2 of all three tree nodes are increased by 1. Similarly, for every tree node *tn*, its counter c_3 is initially set to 0, and it is increased only when a point p' located on n_1's next edge(s) with $tn.key \subseteq p'.key$ is found. Table 2 lists the located data points that trigger the updates of the counters, and Fig. 7a illustrates the final count tree $CT(n_1)$. Note that, in Table 2, 'ø' means there is *zero qualified point* on the edge that can trigger any update of corresponding counters, and '–' indicates the point(s) on the edge does (do) not trigger any update on corresponding counter(s).

Next, we explain the count tree $CT(n_2)$ of node n_2. First of all, a tree similar as $CT(n_1)$ is created, with all the c_1s and c_2s of $CT(n_2)$ copy the values of corresponding c_1s and c_2s in $CT(n_1)$, and all the c_3s of $CT(n_2)$ are set to 0. This reason behind is that, c_1 and c_2 of a tree node *tn* in $CT(n_2)$ represent the number of points p' located on the shortest path SP_{qn2} from q to n_2 with $p'.key = tn.key$ and $tn.key \subseteq p'.key$, respectively. As node n_1 is located on SP_{qn2}, SP_{qn2} actually can be divided into two sub-paths,

TABLE 2
Trace of N_1's Count Tree

key	counter	edge (q, n_1)	edge (n_1, n_2)	edge (n_1, n_5)
a,b	c_1 (0→1)	p_1	–	–
	c_2 (0→1)	p_1	–	–
	c_3 (0→2)	–	p_3	p_{10}
a	c_1 (0→0)	ø	–	–
	c_2 (0→1)	p_1	–	–
	c_3 (0→3)	–	p_2, p_3	p_{10}
b	c_1 (0→0)	ø	–	–
	c_2 (0→1)	p_1	–	–
	c_3 (0→3)	–	p_3	p_9, p_{10}

TABLE 3
Trace of N_2's Count Tree

key	counter	edge (n_1, n_2)	edge (n_2, n_3)	edge (n_2, n_6)
a,b	c_1 (1→3)	p_3, p_4	–	–
	c_2 (1→3)	p_3, p_4	–	–
	c_3 (0→0)	–	unvisited	unvisited
a	c_1 (0→2)	p_2, p_5	–	–
	c_2 (1→5)	p_2, p_3, p_4, p_5	–	–
	c_2 (0→0)	–	unvisited	unvisited
b	c_1 (0→0)	ø	–	–
	c_2 (1→3)	p_3, p_4	–	–
	c_3 (0→0)	–	unvisited	unvisited

i.e., the shortest path SP_{qn1} from q to n_1 and edge (n_1, n_2). Since c_1 and c_2 of $CT(n_1)$ actually capture those qualified data points located on SP_{qn1}, we only need to focus on edge (n_1, n_2) by initializing c_1s and c_2s in $CT(n_2)$ to corresponding c_1s and c_2s in $CT(n_1)$. As listed in Table 3, on edge (n_1, n_2), we locate a few qualified points which help to update the values of c_1 and c_2. Then, on edge (n_2, n_3) and edge (n_2, n_6), we locate another set of qualified points which help to update the values of c_3 if the algorithm does not terminate at n_2. The final $CT(n_2)$ is depicted in Fig. 7b. We would like to highlight that, although we separate the count tree formation for node n_1 and node n_2 in the above explanation, they are actually formed in a parallel fashion. As shown in Tables 2 and 3, when points on edge (n_1, n_2) are evaluated, it updates the c_3s of $CT(n_1)$ as well as the c_1s and c_2s of $CT(n_2)$.

Last but not least, we explain the idea of count tree reuse. Every time, when we evaluate a new vertex n, it is not always necessary to create a new count tree $CT(n)$ because we might be able to reuse some existing count tree $CT(v)$ if $CT(v)$ no longer needed. In the following, we explain when a count tree $CT(v)$ corresponding to a vertex v is no longer needed. Given a vertex v, its count tree $CT(v)$ is to preserve the information related to the points located on SP_{qv} and to initiate count trees $CT(v')$s with v' being n's adjacent vertices. Consequently, if vertex v and all its adjacent vertices v' have been visited, the information related to points located on SP_{qv} is actually preserved by count trees $CT(v')$s and $CT(v)$ is no longer needed. In this case, we can re-use $CT(v)$ for another newly visit vertex. The reason we promote the reuse of count trees is that all count trees share the same structure, which is dependent on keywords specified by the query point. Although the reuse technique is simple, it is efficient which will be further demonstrated in our experiments.

5.3 Algorithm Details

Now, we are ready to present our *Enhanced RkBSK* (ER*k*BSK) *algorithm* that fully utilizes the above pruning heuristics. In the sequel, we present the ER*k*BSK-Filter process which prunes unnecessary data points based on not only Lemma 1 and Lemma 2 but also Heuristic 1 and Heuristic 2, and then explain the ER*k*BSK-Refinement process using Heuristic 3.

5.3.1 Filtering for ERkBSK Algorithm

ER*k*BSK algorithm shares the same framework as R*k*BSK algorithm. It expands the road network from a specified query point q, and only inserts potential answer points to the candidate set S_c after applying certain pruning rules (e.g., Lemma 1).

Algorithm 3 presents the pseudo-code of the *filtering step for ERkBSK algorithm* (ER*k*BSK-Filter). Different from R*k*BSK-Filter algorithm (mentioned in Section 4.2), ER*k*BSK-Filter integrates new pruning heuristics, i.e., Heuristic 1 and Heuristic 2. It first locates edge (n_i, n_j) that q is located (line 1). We assume that q is closer to node n_i, and split edge (n_i, n_j) into two edges (q, n_i) and (q, n_j) which are en-queued into the priority queue U (line 2). Like R*k*BSK algorithm, edges (n, n') in U are sorted based on ascending order of the network distances from node n' to q. Note that, CT(key, c_1, c_2, c_3) is a constructor function to create a new count tree, with the parameter *key* determining the height and keys of the tree, and c_1, c_2, c_3 determining the initial values of the counters. An empty count tree is initiated for node q (line 3).

Algorithm 3. Filter for ER*k*BSK Algorithm (ER*k*BSK-Filter)

Input: $q(loc, key)$, k, a set P of data points on a road network
Output: the candidate set S_c of an R*k*BSK query
1: locate the edge (n_i, n_j) that q is located (assume q is closer to n_i)
2: $U = \{\langle(q, n_i), \|n_i, q\|\rangle, \langle(q, n_j), \|n_j, q\|\rangle\}$ // edges in U are sorted in ascending order of their distances to q
3: $CT(q)$ = new CT($q.key$, 0, 0, 0) // q is regarded as a road vertex
4: **while** U is not empty **do**
5: $e = (n, n')$ = de-queue (U)
6: $CT(n')$ = new CT($q.key$, $CT(n).tn[*].c_1$, $CT(n).tn[*].c_2$, 0)
7: **for** each data point p on e **do** // visit points in ascending order of their distances to q
8: **if** $CT(n').tn[p.key].c_1 < k$ and $CT(n').tn\,[p.key].c_2 < k$ and $(CT(n).tn[p.key].c_2 + CT(n).tn[p.key].c_3) < k$ **then**
9: $S_c = S_c \cup \{p\}$ // Lemma 1, Heuristics 1 and 2
10: **if** $p.key \subseteq q.key$) **then** $CT(n').tn[p.key].c_1$ ++ // Lemma 1
11: **if** $(key = p.key \cap q.key) \neq$ ø **then**
12: **for** each $k_j \subseteq key$ **do** $CT(n').tn[k_j].c_2$ ++ // Heuristic 1
13: **if** $\|p, n\| < \|n, q\|$ **then**
14: **for** each $k_j \subseteq key$ **do** $CT(n).tn[k_j].c_3$ ++ // Heuristic 2
15: **if** $\exists CT(n').tn\,[*].c_1 < k$, $\exists CT(n').tn\,[*].c_2 < k$, and $\exists(CT(n).tn\,[*].c_2 + CT(n).tn[*].c_3) < k$ **then** // Lemma 2, Heuristics 1 and 2
16: **for** each unvisited edge (n', n'') in the edge set E **do**
17: en-queue $\langle(n', n''), \|n'', q\|\rangle$ to U
18: **return** S_c

TABLE 4
Trace of ER*k*BSK-Filter

Step	Action	U	S_c
1	de-queue $\langle(q,n_0), 0.5\rangle$	$\langle(q, n_1), 2.5\rangle$	ø
2	de-queue $\langle(q,n_1), 2.5\rangle$	$\langle(n_1, n_2), 7.5\rangle$ $\langle(n_1, n_5), 8\rangle$	p_1
3	de-queue $\langle(n_1, n_2), 7.5\rangle$	$\langle(n_1, n_5), 8\rangle$	p_1, p_2, p_3, p_4
4	de-queue $\langle(n_1, n_5), 8\rangle$	ø	p_1, p_2, p_3, p_4

Thereafter, the network expansion starts. For each de-queued edge (n, n'), ER*k*BSK-Filter first needs to initialize the count tree for node n'. It needs to copy the values of counter c_1 and counter c_2 from the count tree $CT(n)$. For counter c_3, ER*k*BSK-Filter needs to check the qualified points located on edge (n, n') which fits nicely with our network expansion strategy. Then, it scans all the points located on edge (n, n') one by one, based on their distances to q. For each located point p, the algorithm first examines if it can be pruned by our pruning heuristics, and it enrolls p to the candidate set iff it cannot be discarded by Lemma 1, Heuristic 1, and Heuristic 2 (lines 8-9). Next, ERkBSK-Filter updates counters of $CT(n')$ and $CT(n)$ based on $p.key$. First, using Lemma 1, counter c_1 of $CT(n')$ w.r.t. $p.key$ is increased by one if $p.key$ is bounded by $q.key$ (i.e., $p.key \subseteq q.key$) (line 10). Second, based on Heuristic 1, counter c_2 of $CT(n')$ w.r.t. any non-empty subset of $p.key \cap q.key$ is increased by one if $p.key$ overlaps with $q.key$ (i.e., $p.key \cap q.key \neq ø$) (lines 11-12). Third, using Heuristic 3, counter c_3 of $CT(n)$ w.r.t. any non-empty subset of $p.key \cap q.key$ is increased by one if $p.key$ overlaps with $q.key$ (i.e., $p.key \cap q.key \neq ø$) and meanwhile $\|p, n\| < \|n, q\|$ (lines 13-14). Once all the points located on edge (n, n') are evaluated, the algorithm en-queues n''s next edges into U to expand the network based on Lemma 2, Heuristic 1, and Heuristic 2 (lines 15-17). Once the expansion finishes, the candidate set is returned to complete the algorithm (line 18).

Algorithm 4. Refinement for ER*k*BSK Algorithm (ER*k*BSK-Refinement)

Input: a candidate set S_c, a query point q, a parameter k
Output: the result set S_r of an R*k*BSK query
Initialize $S_r = ø$
for each candidate point p in S_c **do**
  if BV(p, q, k) = TRUE **then** $S_r = S_r \cup \{p\}$
  else $S_c = S_c - \{p' \mid p'.key \subseteq p.key \wedge p \in SP_{qp'}\}$ // Heuristic 3
return S_r

Example 3. We illustrate ER*k*BSK-Filter algorithm using the dataset in Fig. 2. Assume that an R3BSK query is issued at a query point q with $q.key = \{a, b\}$, and the road network is expanded in order of $n_0, n_1, n_2, n_5, n_3, n_6, \ldots$. The algorithm starts by locating q and initializing U and $CT(q)$. We depict the trace of the filtering step in Table 4, and the changes of count trees are shown in Fig. 8.

5.3.2 Refinement for ERkBSK Algorithm

The *refinement step of ERkBSK algorithm* (ER*k*BSK-Refinement) applies Heuristic 3, which is different from that of R*k*BSK algorithm. Specifically, it has three tasks, i.e., verifying data points in S_c based on the BV algorithm, pruning false candidates in S_c based on Heuristic 3 without incurring other verification procedure, and returning the final answer points. Initially, ER*k*BSK-Refinement sorts the candidate points in S_c based on ascending order of their distances to q, and then evaluates them one by one. For each evaluated point p, if p is validated to be an actual answer point, p is added to the result set S_r; otherwise, ER*k*BSK-Refinement discards p, together with all the other not-yet-checked candidates p' in S_c whose shortest paths to q contain p and $p'.key \subseteq p.key$. The algorithm terminates when all the candidate points in S_c have either been evaluated or discarded, and the final query result set S_r is returned.

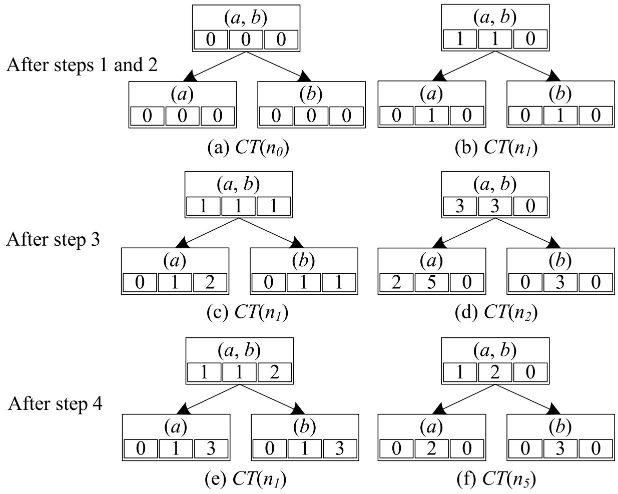


Fig. 8. Trace of count trees after each step.

Example 4. Continue Example 3. After the filter step, $S_c = \{p_1, p_2, p_3, p_4\}$. ER*k*BSK-Refinement then verifies them based on ascending order of their distances to q. First, p_1 is evaluated and is reported as a real answer point with $S_r = \{p_1\}$. Next, p_2 is verified, and is reported as a false answer point. As S_{H3} w.r.t. p_2 is empty, it does not help to prune any other candidate point. Then, p_3 is checked and also reported as a false answer point, and $p_4 \in S_{H3}$ w.r.t. p_3. Hence, both p_3 and p_4 are discarded. Finally, the refinement step stops with $S_r = \{p_1\}$.

5.4 Discussion

In a 2D space, like existing TPL and RST*k*NN methods for RNN search and its variants, the proposed ER*k*BSK algorithm with several pruning heuristics does not require any pre-computation, and it can return the exact result. However, compared with TPL and RST*k*NN, ER*k*BSK algorithm incurs a higher query cost, especially when data points are sparse and $|q.key|$ is large. This is because ER*k*BSK algorithm needs to consider the distance of the shortest path and the keyword constraint. In what follows, we first briefly discuss the cost of ER*k*BSK algorithm, and then prove its correctness.

Lemma 3. *If m shortest paths ended in a specified query point q have been expanded in the filtering step, the ERkBSK algorithm traverses the dataset P at most $(|S_c| + 1)$ times with S_c being the candidate set and $|S_c| \leq (2^{|q.key|} - 1) \times mk$.*

TABLE 5
Statistics of Real Datasets

Data	Vertex	Edge	Objects
CA	21,048	21,693	0.17M
NA	175,813	179,179	1.4M
SF	174,956	223,001	1.7M

Proof. As shown in Algorithm 3, ER*k*BSK-Filter algorithm only traverses a given data set P at most once to form a candidate set S_c. Since ER*k*BSK algorithm uses Lemma 2 to set the upper bound of S_c, the number of points in S_c is no more than $(2^{|q.key|} - 1) \times mk$. Then, ER*k*BSK-Refinement algorithm invokes BV algorithm once for every point in S_c in the worst case (i.e., if Heuristic 3 does not help to prune away any candidate point). Consequently, ER*k*BSK algorithm traverses P at most $((2^{|q.key|} - 1) \times mk + 1)$ times. □

Theorem 1. *The ERkBSK algorithm returns exactly the result set RkBSK(q), i.e., the algorithm has no false negative and no false positive.*

Proof. First, ER*k*BSK algorithm only prunes away those unqualified points or network area in the filtering step, by using our proposed pruning heuristics. Therefore, no answer points are missed (i.e., no false negative). Second, every candidate point $p \in S_c$ either is verified in the refinement step by BV algorithm or is discarded by Heuristic 3, which ensures no false positive. Consequently, the proof completes. □

6 Experimental Evaluation

In this section, we experimentally evaluate the effectiveness and efficiency of our proposed algorithms for R*k*BSK search through extensive experiments on both real and synthetic data sets. We describe the experimental settings in Section 6.1, verify the pruning power of the developed heuristics in Section 6.2, compare the efficiency of count list and count tree in Section 6.3, and report the performance of our proposed algorithms for R*k*BSK queries in Section 6.4. All the algorithms were implemented in C++, and all the experiments were conducted on a PC with an Intel Core 2 Duo 2.93 GHZ E7500 CPU and 3 GB RAM, running Ubuntu 13.04 desktop edition.

6.1 Experimental Setup

We deploy both real and synthetic data sets. As summarized in Table 5, we employ three real road networks[2] *CA*, *NA*, and *SF* as our data sets. For these datasets, POIs with real keyword sets are randomly generated in a way similar as [18]. We also generate two synthetic datasets. The first data set is to study the impact of the keyword set size per data point on the search performance. We preserve the road network and the data points of *NA* but change the keyword settings to generate five sets, viz., *NA-K2*, *NA-K4*, *NA-K6*, *NA-K8* and *NA-K10* as [18]. The average distinct number of keywords per data point in each dataset is roughly 2, 4, 6, 8, and 10, respectively. The second data set is to explore the impact of data point density on the search performance. We preserve the road network of *NA* but change the number of data points per edge to generate five datasets, i.e., *NA-C4*, *NA-C6*, *NA-C8*, *NA-C10*, and *NA-C12*. For every dataset *NA-Ci*, the average number of data points per edge is approximately set to *i*.

TABLE 6
Parameter Settings

Parameter	Range	Default
# of query keywords (i.e., $\|q.key\|$)	1, 3, 5, 7, 9	5
k	10, 20, 30, 40, 50	30
avg # of POI (p) keywords (i.e., $\|p.key\|$)	2, 4, 6, 8,10	4
avg # of POIs per edge (i.e., $\|$POIs$\|$)	4, 6, 8, 10, 12	8

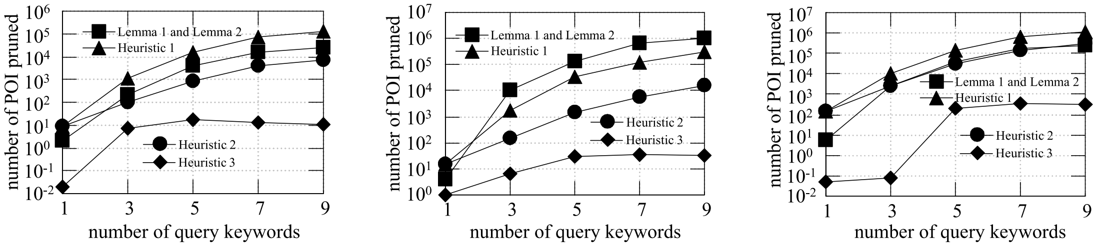
The experiments investigate the performance of the proposed algorithms under a variety of parameters which are listed in Table 6. In the experiments, we measure (i) the response time (i.e., the average response time in processing a workload); (ii) the number of page accesses by various algorithms during the search; and (iii) the number of data points pruned by each pruning heuristic. All data sets are indexed by our modified CCAM model (discussed in Section 3.2) with the disk page size fixed to 4,096 bytes. In each experiment, we vary only one parameter and fix other parameters at their defaults. Hundred random queries are evaluated in every experiment (as with [18]), and their average performance is reported. To be more specific, the query is randomly located in one edge of the road network, and the query keywords are randomly extracted from the vocabulary of the data set. We assume that the server maintains a buffer of 200 pages with LRU as the cache replacement policy.

6.2 Effectiveness of Pruning Heuristics

The first set of experiments is to verify the effectiveness of our presented pruning heuristics based on the number of data points pruned. As different lemmas/heuristics are introduced for different purposes, the points they try to prune are different. For Lemma 1 and Lemma 2, they are integrated simultaneously in R*k*BSK algorithm, and hence we report their joint pruning power based on the number of data points they, but not BM, can discard. For Heuristic 1 and Heuristic 2, we refer to the data points they, but not newly developed lemmas, can prune. For Heuristic 3, we measure those candidate points in the candidate set S_c that can be discarded without invoking BV algorithm.

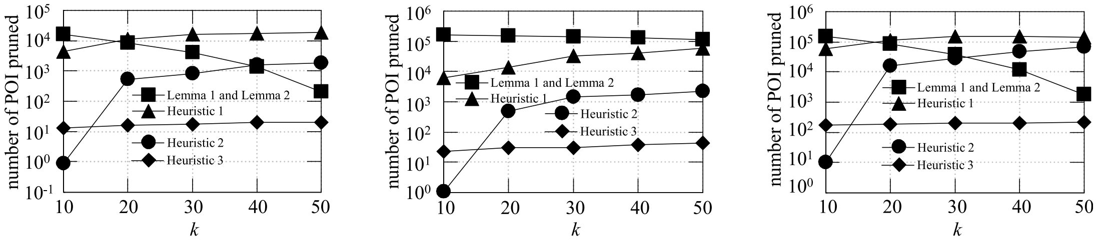
We change $|q.key|$ from 1 to 9 and depict the results in Fig. 9. We also vary k and show the results in Fig. 10. Evidently, all the lemmas and heuristics have excellent pruning power. Take Heuristic 1 for *NA* as an example. It saves the detailed examination of about 33,026 points when $|q.key| = 5$. Based on our experiments, Heuristic 3 is not as effective as others. This is because Heuristic 3 is only applied to the candidate points in S_c. Since the majority of data points have already been pruned away by Lemmas, Heuristic 1, and Heuristic 2, the candidate set is not big, which leaves

2. *CA*, *NA* and *SF* are available at http://www.cs.utah.edu/~lifeifei/.



(a) *CA* (b) *NA* (c) *SF*

Fig. 9. Pruning efficiency of heuristics versus $|q.key|$.



(a) *CA* (b) *NA* (c) *SF*

Fig. 10. Pruning efficiency of heuristics versus *k*.

the room for improvement brought by Heuristic 3 very small. It is observed that although heuristics perform differently as parameters change, their overall effectiveness is significant.

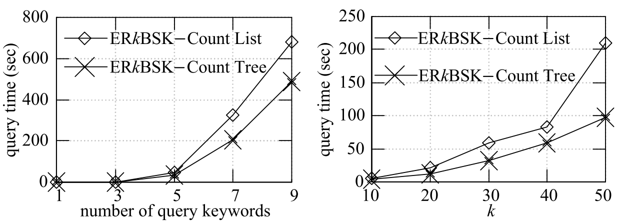
6.3 Effectiveness of Count Tree

The second set of experiments evaluates the efficiency of count tree. Towards this, we deploy *NA* dataset, and vary the number *k* and $|q.key|$. We demonstrate the efficiency of count tree by comparing it with count list using the same algorithm, i.e., ER*k*BSK algorithm. The experimental results are illustrated in Fig. 11. It is observed that, the cost of ER*k*BSK using count list (denoted as ER*k*BSK-Count List) increases more significantly with the growth of both *k* and $|q.key|$. Furthermore, when the values of *k* and $|q.key|$ grow, the gap of ER*k*BSK using different data structures is also enlarged. The reason is that the cost of maintaining count list ascends with *k* and $|q.key|$, since finding count information using count list is more costly than that under count tree.

6.4 Results on RkBSK Queries

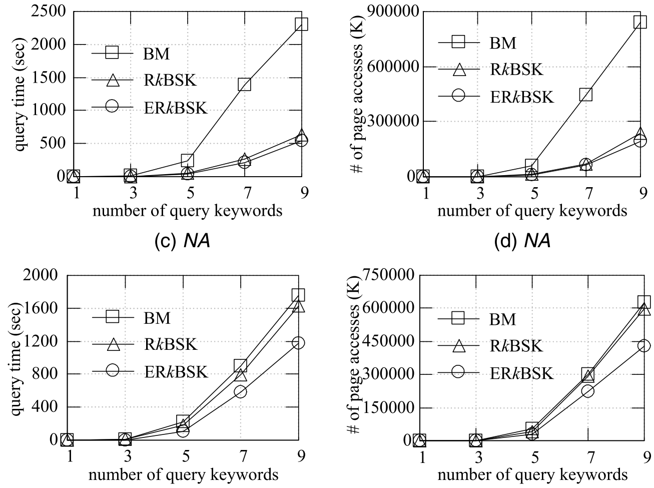
In this set of experiments, we evaluate the performance of BM, R*k*BSK, and ER*k*BSK algorithms. We study the influence of various parameters, including (i) the number of query keywords (i.e., $|q.key|$), (ii) *k*, (iii) the average number of keywords per point *p* in the dataset (i.e., $|p.key|$), and (iv) the average number of POIs per edge (i.e., $|\text{POIs}|$).

First, we investigate the impact of $|q.key|$ on the efficiency of the algorithms, and show the results for $k = 30$ in Fig. 12. As observed, ER*k*BSK exceeds BM and R*k*BSK in all cases. The reason is that, as mentioned in Section 4, BM needs to verify all data points *p* with $p.key \subseteq q.key$, and R*k*BSK, although performing better than BM, still needs to



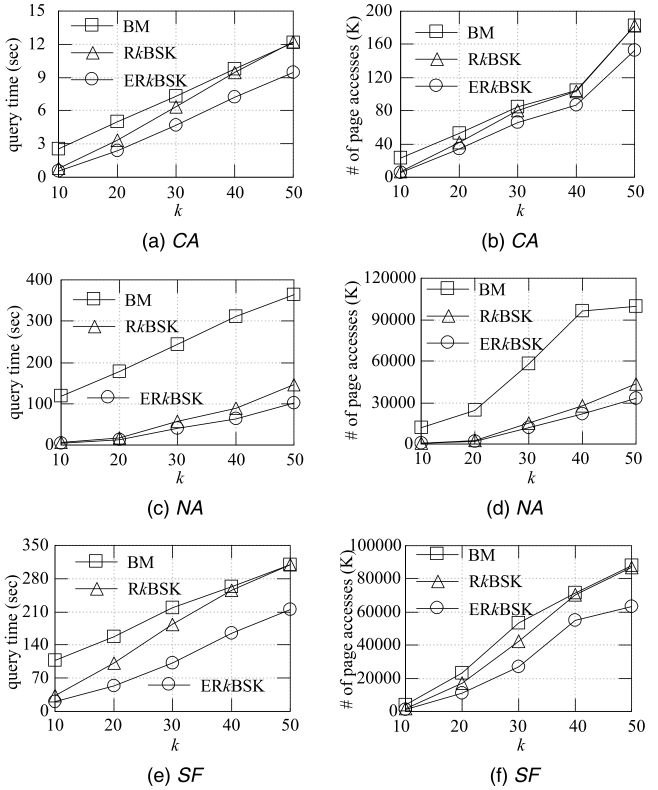
(a) *NA* (b) *NA*

Fig. 11. Efficiency of count tree and count list.



(e) *SF* (f) *SF*

Fig. 12. Query cost versus $|q.key|$.

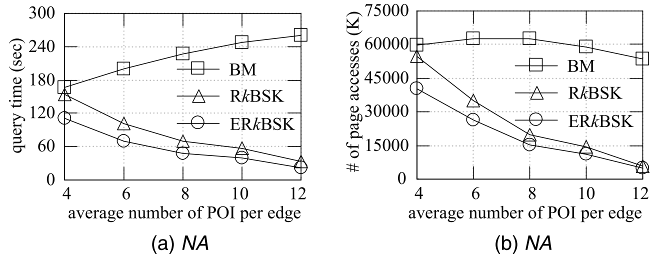


(a) *CA* (b) *CA*

(c) *NA* (d) *NA*

(e) *SF* (f) *SF*

Fig. 13. Query cost versus *k*.



(a) *NA* (b) *NA*

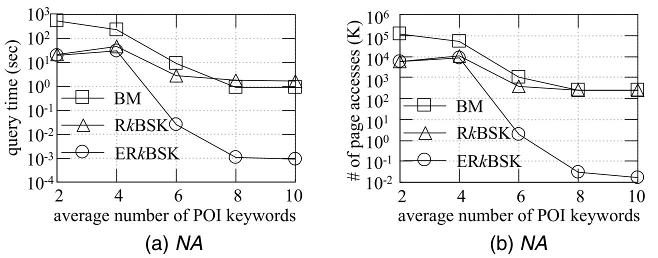
Fig. 15. Query cost versus |POIs|.

evaluate a large number of data points. We also observe that the cost of R*k*BSK retrieval increases with |*q.key*|. This is expected, as |*q.key*| grows, more points satisfy the keyword constraint, and thus, the candidate set is bigger.

We then explore the impact of *k* with results depicted in Fig. 13. Similar to what has been observed previously, ER*k*BSK performs the best, followed by R*k*BSK, BM is the worst. Also, the value of *k* affects the performance.

Next, we evaluate the performance of R*k*BSK algorithms under different average number of keywords per data point *p* (i.e., |*p.key*|), with results plotted in Fig. 14. As expected, ER*k*BSK outperforms BM and R*k*BSK significantly, especially for larger |*p.key*|. When |*p.key*| changes from 2 to 10, the cost of R*k*BSK retrieval decreases. This is because the number of candidate objects drops as |*p.key*| ascends, which helps to reduce the cost. Another observation is that the cost of ER*k*BSK decreases dramatically as |*p.key*| grows. The reason is that the pruning power of Heuristics increases with the growth of |*p.key*|.

Last but not least, we study the effect of |POIs| (i.e., the average number of POIs per edge) on the performance of the algorithms, with the results shown in Fig. 15. Increasing |POIs| has different impacts on these three algorithms. As depicted in Fig. 15, when |POIs| grows, BM increases its costs significantly, while R*k*BSK and ER*k*BSK actually decrease their costs. This is because, as |POIs| ascends, BM needs to evaluate more data points, whereas R*k*BSK and ER*k*BSK actually expand a smaller area in the road network with the help of efficient lemmas and heuristics.



(a) *NA* (b) *NA*

Fig. 14. Query cost versus |*p.key*|.

7 Conclusions

In this paper, we identify and solve a novel type of queries, namely, R*k*BSK search, on road networks by considering spatial and textual constraints. Although both R*k*NN retrieval and spatial keyword search on road networks have been studied, there is no previous work that takes into account both the reverse spatial proximity between objects on road networks and the textual constraint. On the other hand, R*k*BSK retrieval is useful in many decision support applications involving keywords and road networks. Two efficient algorithms are developed to support R*k*BSK queries on road networks, assuming that the road network is modeled by a large graph. An extensive experimental evaluation with both real and synthetic data sets has been conducted to verify the performance of our proposed algorithms in answering R*k*BSK queries.

This work also inspires several directions for future work. First, we only focus on Boolean spatial keywords search in this work. Thus, how to extend our solutions to score based spatial keyword retrieval needs further study. Second, we plan to explore the multi-source R*k*BSK query that is issued with respect to multiple query points, which can be regarded as the group version of R*k*BSK search. Finally, in addition to road networks, how to use R*k*BSK queries on social networks is also interesting.

Acknowledgments

Yunjun Gao was supported in part by NSFC Grant No. 61379033, the National Key Basic Research and Development Program (i.e., 973 Program) No. 2015CB352502, and the Cyber Innovation Joint Research Center of Zhejiang University.

References

[1] X. Cao, L. Chen, G. Cong, C. S. Jensen, Q. Qu, A. Skovsgaard, D. Wu, and M. L. Yiu, "Spatial keyword querying," in *Proc. Int. Conf. Conceptual Model.*, 2012, pp. 16–29.

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

[3] A. Cary, O. Wolfson, and N. Rishe, "Efficient and scalable method for processing top-*k* spatial Boolean queries," in *Proc. Int. Conf. Sci. Statistical Database Manage.*, 2010, pp. 87–95.

[4] M. A. Cheema, W. Zhang, X. Lin, Y. Zhang, and X. Li, "Continuous reverse *k* nearest neighbors queries in euclidean space and in spatial networks," *VLDB J.*, vol. 21, no. 1, pp. 69–95, Feb. 2012.

[5] 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.

[6] 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, Aug. 2009.

[7] I. D. Felipe, V. Hristidis, and N. Rishe, "Keyword search on spatial databases," in *Proc. Int. Conf. Data Eng.*, 2008, pp. 656–665.

[8] Y. Gao, B. Zheng, G. Chen, W.-C. Lee, K. C. Lee, and Q. Li, "Visible reverse *k*-nearest neighbor query processing in spatial databases," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 9, pp. 1314–1327, Sep. 2009.

[9] F. Korn and S. Muthukrishnan, "Influence sets based on reverse nearest neighbor queries," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2000, pp. 201–212.

[10] K. C. Lee, B. Zheng, and W.-C. Lee, "Ranked reverse nearest neighbor search," *IEEE Trans. Knowl. Data Eng.*, vol. 20, no. 7, pp. 894–910, Jul. 2008.

[11] G. Li, J. Feng, and J. Xu, "Desks: Direction-aware spatial keyword search," in *Proc. Int. Conf. Data Eng.*, 2012, pp. 474–485.

[12] Z. Li, K. C. Lee, B. Zheng, W.-C. Lee, D. Lee, and X. Wang, "IR-tree: An efficient index for geographic document search," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 4, pp. 585–599, Apr. 2011.

[13] G. Li, Y. Li, J. Li, L. Shu, and F. Yang, "Continuous reverse *k* nearest neighbor monitoring on moving objects in road networks," *Inf. Syst.*, vol. 35, no. 8, pp. 860–883, Dec. 2010.

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

[15] S. Luo, Y. Luo, S. Zhou, G. Cong, and J. Guan, "Distributed spatial keyword querying on road networks," in *Proc. Int. Conf. Extending Database Technol.*, 2014, pp. 235–246.

[16] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, "Query processing in spatial network databases," in *Proc. Int. Conf. Very Large Data Bases*, 2003, pp. 802–813.

[17] J. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Norvag, "Efficient processing of top-*k* spatial keyword queries," in *Proc. Int. Symp. Advances Spatial Temporal Databases*, 2011, pp. 205–222.

[18] J. B. Rocha-Junior and K. Norvag, "Top-*k* spatial keyword queries on road networks," in *Proc. Int. Conf. Extending Database Technol.*, 2012, pp. 168–179.

[19] M. Safar, D. Ibrahimi, and D. Taniar, "Voronoi-based reverse nearest neighbor query processing on spatial networks," *Multimedia Syst.*, vol. 15, no. 5, pp. 295–308, Oct. 2009.

[20] S. Shekhar and D.-R. 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. 1997.

[21] I. Stanoi, D. Agrawal, and A. El Abbadi, "Reverse nearest neighbor queries for dynamic databases," in *Proc. ACM SIGMOD Workshop DMKD*, 2000, pp. 44–53.

[22] Y. Tao, D. Papadias, and X. Lian, "Reverse *k*NN search in arbitrary dimensionality," in *Proc. Int. Conf. Very Large Data Bases*, 2004, pp. 744–755.

[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] A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Norvag, "Reverse top-*k* queries," in *Proc. Int. Conf. Data Eng.*, 2010, pp. 365–376.

[25] A. Vlachou, C. Doulkeridis, K. Norvag, and Y. Kotidis. "Branch-and-bound algorithm for reverse top-*k* queries," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013, pp. 481–492.

[26] L. Wu, X. Xiao, D. Deng, G. Cong, A. D. Zhu, and S. Zhou, "Shortest path and distance queries on road networks: An experimental evaluation," *Proc. VLDB Endowment*, vol. 5, no. 5, pp. 406–417, 2012.

[27] W. Wu, F. Yang, C.-Y. Chan, and K.-L. Tan, "Finch: Evaluating reverse *k*-nearest-neighbor queries on location data," *Proc. VLDB Endowment*, vol. 1, no. 1, pp. 1056–1067, 2008.

[28] 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.

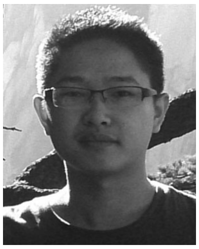
[29] M. L. Yiu, D. Papadias, N. Mamoulis, and Y. Tao, "Reverse nearest neighbors in large graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 4, pp. 540–553, Apr. 2006.

[30] J. Zhang, W.-S. Ku, X. Jiang, X. Qin, and Y.-L. Hsueh, "Evaluation of spatial keyword queries with partial result support on spatial networks," in *Proc. Int. Conf. Mobile Data Manage.*, 2013, pp. 279–282.

[31] D. Zhang, K. L. Tan, and A. K. H. Tung, "Scalable top-*k* spatial keyword search," in *Proc. Int. Conf. Extending Database Technol.*, 2013, pp. 359–370.

[32] C. Zhang, Y. Zhang, W. Zhang, X. Lin, M. A. Cheema, and X. Wang, "Diversified spatial keyword search on road networks," in *Proc. Int. Conf. Extending Database Technol.*, 2014, pp. 367–378.

Yunjun Gao received the PhD degree in computer science from Zhejiang University, China, in 2008. He is currently an associate professor in the College of Computer Science, Zhejiang University, China. Prior to joining the faculty, he was a postdoctoral fellow at the Singapore Management University, during 2008-2010, and a visiting scholar or a research assistant at the Nanyang Technological University, Simon Fraser University, and City University of Hong Kong, respectively. His research interests include spatial and spatiotemporal databases, uncertain and incomplete databases, and spatiotextual data management. He has published papers in Journals and conferences including *TODS*, *VLDBJ*, *TKDE*, SIGMOD, VLDB, ICDE, and SIGIR. He is a member of the ACM and the IEEE, and a senior member of the CCF.



Xu Qin received the BS degree in network engineering from Nanchang University, China, in 2012. He is currently working toward the MS degree in the College of Computer Science, Zhejiang University, China. His research interest includes spatial keyword queries.

Baihua Zheng received the PhD degree in computer science from Hong Kong University of Science & Technology, China, in 2003. She is currently an associate professor in the School of Information Systems, Singapore Management University, Singapore. Her research interests include mobile/pervasive computing and spatial databases. She has published papers in Journals and conferences including *TODS*, *VLDBJ*, *TKDE*, SIGMOD, VLDB, and ICDE. She is a member of the IEEE.

Gang Chen received the PhD degree in computer science from Zhejiang University. He is currently a professor in the College of Computer Science, Zhejiang University, China. He has successfully led the investigation in research projects which aim at building China's indigenous database management systems. His research interests range from relational database systems to large-scale data management technologies supporting massive Internet users. He has published papers in Journals and conferences including *TODS*, *VLDBJ*, *TKDE*, SIGMOD, VLDB, and ICDE. He is a member of the ACM and senior member of the CCF.