Hindawi
Complexity
Volume 2022, Article ID 4091245, 19 pages
https://doi.org/10.1155/2022/4091245

Research Article

Path–Based Continuous Spatial Keyword Queries

Fangshu Chen,[1] Pengfei Zhang,[2] Chengcheng Yu,[1] Huaizhong Lin,[2] Shan Tang,[3] and Xiaoming Hu[1]

[1]*The College of Computer and Information Engineering, Shanghai Polytechnic University, Shanghai, China*
[2]*The College of Computer Science and Technology, Zhejiang University, Hangzhou, China*
[3]*1th Shanghai Zhi Pan Intelligent Technology Co.Ltd., 2th Shanghai Polytechnic University, Shanghai, China*

Correspondence should be addressed to Chengcheng Yu; ccyu@sspu.edu.cn

Received 30 October 2021; Accepted 26 April 2022; Published 10 June 2022

Academic Editor: Giacomo Fiumara

In this paper, we study the path based continuous spatial keyword queries, which find the answer set continuously when the query point moves on a given path. Under this setting, we explore two primitive spatial keyword queries, namely k nearest neighbor query and range query. The technical challenges lie in that: (1) retrieving qualified vertices in large road networks efficiently, and (2) issuing the query continuously for points on the path, which turns out to be inapplicable. To overcome the above challenges, we first propose a backbone road network index structure (BNI), which supports the distance computation efficiently and offers a global insights of the whole road network. Motivated by the safe zone technique, we then transform our queries to the issue of finding event points, which capture the changes of answer set. By this transformation, our queries are to be simple and feasible. To answer the queries, we propose a Two-Phase Progressive (TPP) computing framework, which first computes the answer sets for some crucial vertices on the path, and then identifies the event points by the retrieved answer sets. Extensive experiments on both real and synthetic data sets are conducted to evaluate the performance of our proposed algorithms, and the results show that our algorithms outperform competitors by several orders of magnitude.

1. Introduction

With the prevalence of smartphones and other mobile devices, continuous spatial queries have gained increasing attention from the research community [1–12]. Previous works can be roughly classified into two categories by the query model: (1) snapshot based model [2, 3, 7, 8]. In this model, snapshot query processing techniques are periodically invoked, which either yields excessive costs or outdated results as pointed out in [6]; (2) safe zone based model [1, 4, 6, 9, 12, 13]. When the query point moves within the safe zone, it can be guaranteed that the answer set will not change. This property encourages to identify the safe zone, rather than periodically invoking the query processing techniques.

Traditional spatial keyword query models usually suppose that the query point is static or moves randomly, however, in real life a user (query point) may be moving on a navigation path or a public transport line, thus an data point is relevant only if it covers all query keywords (capturing user needs) and is on the query path at the same time.

Consider the following scenario as shown in Figure 1, there are 10 vertices $\{v_1, v_2, \ldots v_{10}\}$) on the road network, specifically, each vertex is associated with a set of keywords (the abbreviations of the keywords are defined on the top right corner in Figure 1), and each edge is associated with a real-valued distance. Given a query path P (the red arrow in Figure 1), a tourist (described as a blue car in the figure) monitors Chinese food restaurant with car parking service within 1 km continuously when driving on path P, the demand of the tourist can be expressed by keyword set $\{C, P\}$. As another example, a passenger may continuously query the nearest Chinese food restaurant with car parking service. The first example can be answered by issuing the path based continuous range query, and the second one can be answered by issuing the path based continuous k NN query. As shown in 1, when the query point is q_1, the answer set for path based continuous range query is $\{v_3\}$, since v_3 is

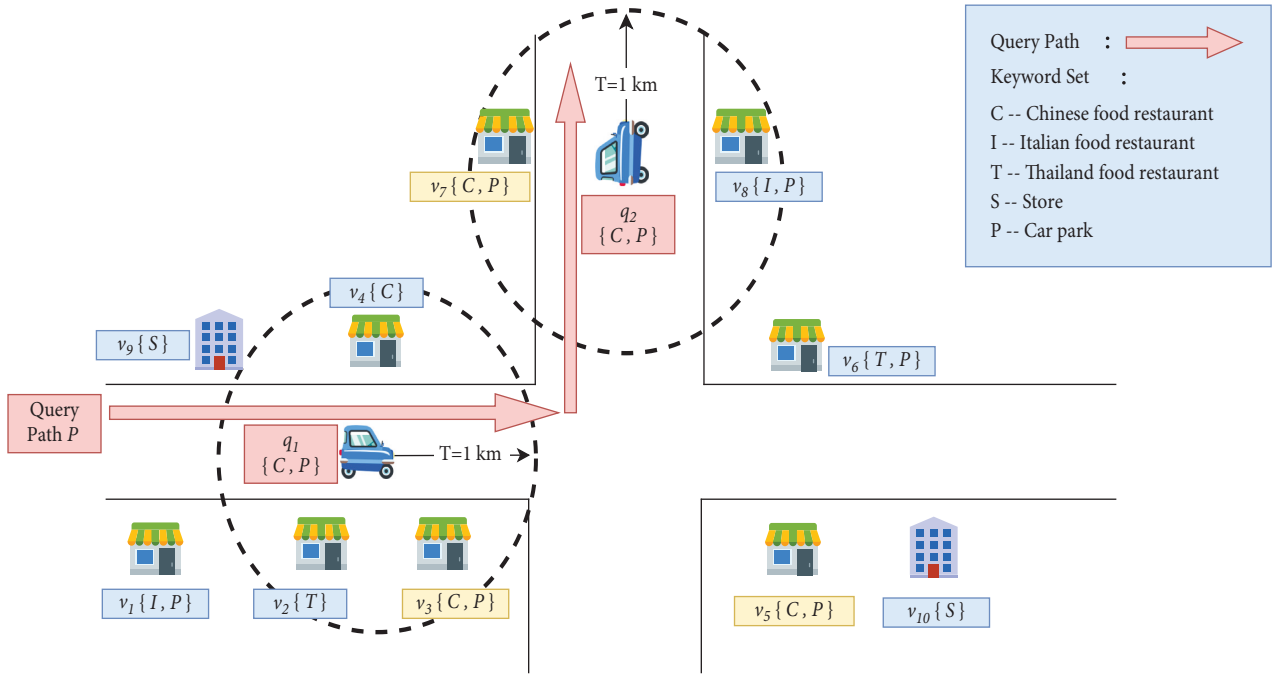


Figure 1: Example of the path based continuous spatial keyword query.

the only vertex within the query range and cover the query keywords $\{C, P\}$ at the same time. And as the car moving on the query path, the answer set keeps changing, when it moves to q_2, the answer set for path based continuous range query is $\{v_7\}$. And in the same way, we can find that the answer of k NN query ($k = 2$) for q_1 is $\{v_3, v_7\}$, since these two points are on the query path and cover the keywords at the same time. Notice that, v_5 is nearer to q_1 compared to v_7 and it also satisfies the keyword constraint, however, it is not on the query path P, thus it is not in the answer set.

Motivated by the above examples, we propose to study the continuous spatial keyword queries with path based paradigm. Particularly, we focus on studying two fundamental types of queries, namely path based continuous range keyword (PCR) query and path based continuous k NN keyword (PC k NN) query.

From the technical perspective, we encounter two major challenges. The first challenge is how to compute distance on large scale road networks, which incurs prohibitive computation overhead. Various techniques are developed [14–17] to facilitate the distance computation. However, they either incur prohibitive pre-computation overhead or are short of global insights of the whole network. To make the road network distance computation more efficiently, we propose a dual-index structure, namely backbone network index (BNI). Concretely, we first reduce the original network to a backbone network using a simple but effective strategy. We then maintain the skeleton and detailed information of the road network by two structures, namely the modified G-tree [18] (in memory) and modified adjacent list [19] (on disk). As pointed out in [18], G-tree enables to facilitate the distance computation by assembly-based method, whilst offering the global insights of the road network.

Another challenge arises when considering infinite number of query points on the path. Obviously, issuing query at each query point is infeasible. Inspired by the observation that the answer set will not change when query point moves within the safe zone, we target to find the event points on the path which capture the changes of answer set. This transformation makes our queries simple and feasible.

Given a query path, [20] investigates the continuous nearest neighbor queries. To enable efficient query processing, they precompute k NN sets for a fraction of intersection points. Nevertheless, one may request various k NN sets for different needs (expressed by different keywords set). As a result, computing all the k NN sets for different query keywords is infeasible. LARC [9] studies the continuous k nearest neighbor keyword queries on the road network. However, LARC suffers from two problems when it is applied to answer our queries. First, it has to precompute and reserve much information for each vertex, which incurs prohibitive computation and space overhead in large scale road networks. Second, the safe zone computed by LARC is approximate and redundant for our path based queries.

To answer the queries, we develop a Two-Phase Progressive query framework (TPP) on the top of our proposed index structure BNI. We first compute and maintain the answer sets for some crucial vertices on the given path when receiving queries. This can be achieved by only issuing one query. Then, we identify the event points on the path progressively with the reserved answer sets.

This paper is a significant extension to its preliminary conference version [21]. Compared to the preliminary version, we extend the path based continuous range keyword query to the k nn keyword query, and propose two new effective algorithms. And the previous work in [21] lacks of

theoretical analysis of the effectiveness and efficiency of the proposed index structure and algorithms.

Our major contributions (excluding the contributions in conference version [21]) can be summarized as follows:

(i) We extend the spatial keyword query to the k nearest neighbor query and propose two new effective algorithms including IssuingkNNQuery and PCkNN – IdentifyingEventPoint to solve the Path Based Continuous kNN Query (PCkNN) (in Section 4 and Section 5).

(ii) Some detailed proofs and definitions that have not been included int the conference version have been added here;

(iii) We add an analysis section to present the theoretical analysis of the effectiveness of proposed BNI index structure, and give out the time complexity and space complexity for all the proposed algorithms (in Section 6);

(iv) We extend the experimental part to verify the effectiveness and efficiency of our proposed PCkNN algorithms (in Section 7). And we also extend the experiment results to evaluate the algorithms' page access number for both queries.

The rest of this paper is organized as follows. Section 2 reviews the related work. Section 3 formally defines the problem and introduce the backbone network. We describe the proposed index structure in Section 4 and introduce our proposed algorithms in Section 5. Section 6 offers the theoretical analysis of proposed algorithms, and we give the experimental study in Section 7. This paper is concluded in Section 8.

2. Related Work

We review works related to the path based continuous spatial keyword queries, mainly focusing on the continuous/moving spatial queries and the road network index structures.

2.1. Continuous/Moving Spatial Queries. Spatial queries have been extensively studied in database community. Specifically, enormous focuses have shifted to range queries [16, 23–27] and k nearest neighbor queries [2, 4–6, 9, 18]. Next, we will introduce the works on them respectively.

Cheema et al. [22] explore the continuous range queries on both the Euclidean space and the road network. They adopt the concept of safe zone for monitoring the query position updating. Literature [23, 24] studied the problem in context of road network. Du et al. [23] answer the continuous spatial keyword range queries by two steps. They first build the range tree for the query, which maintains the shortest path from query point to all qualified objects. Then, the up-to-date results are obtained by adjusting the range tree accordingly. To alleviate the effect the frequent object updates, Wang et al. [24] develop the Shortest-Distance-based Tree (SD-Tree), which supports incremental computation of answer sets using retrieved paths. Literature [25–28] explore the range queries with the assumption that objects move on the Euclidean space. Specifically, [26, 28] consider an uncertainty model, in which the motion of objects is uncertainty. Chung et al. [28] propose to transform moving objects into points by hough transform, and then index them with R-tree. In [26], by transforming the irregular motion areas to polygon regions, the initial problem can be reduced to a simplified version. Wu et al. [27] study the continuous range queries over moving objects but stationary queries. They propose the containment-encoded squares (CESs), which decompose query regions and store precomputed search results. Zhang et al. [25] study the predictive range query and the predictive k nearest neighbor query. In fact, they distinguish from our work by two major aspects. On one hand, they model the motion of objects as a linear function of time but not the road network. On the other hand, they retrieve one answer set for whole querying period, while we keep searching the up-to-date answer set. Transformed minkowski sum (TMS) are used to determine whether the moving objects intersect the moving query point.

Given a query path, [20] investigates the problem of continuous nearest neighbor queries. To enable efficient query processing, they precompute k NN sets for a fraction of intersection points. Nevertheless, in practice, one may require different k NN sets due to different needs when considering keywords. As a result, computing all the k NN sets for different query keywords is infeasible. Literature [1, 6] try to answer the k NN queries by safe region technique, which is motivated by V-Diagram [27]. To avoid prohibitive computation cost, Li et al. [1] utilize influential sets to restrict the safe zone. Note that, it can achieve the same large safe zone as order-k Voronoi cell [29] while minimizing the computation overhead. In [3], Mouratidis et al. study the continuous k NN queries on road networks. Different from our setting, they consider both the moving objects and queries without making assumption on the motion patterns. Two methods are developed, namely incremental monitoring algorithm (IMA) and group monitoring algorithm (GMA). The former monitors the answer set of individual query by maintaining an expansion tree from query point. With this tree, only object updating that may affect the answer set is considered. GMA groups all queries lying on the same edge, and then gets the answer sets for them based on the answer sets of end points of corresponding edge and the objects lying on this edge. [4] studies the continuous k nearest neighbor queries on the context of both Euclidean space and spatial network. They propose to construct the safe zone based on the objects, query points and the search space. However, this method has to recompute the safe regions more frequently and has higher validation overhead as pointed by [1]. Zheng et al. [9] utilize the 2-hop label to facilitate the distance computation. But this work suffers from extensive computation overhead and storage overhead. More importantly, the retrieved safe regions are too large and inaccurate when applied to our problem. Mouratidis et al. [2] develop the conceptual partitioning (CPM) technique, which handling location updates only from objects that fall in the vicinity of some query (and

Table 1: Summary of the related works.

Techniques	Network	Path	Moving	Keyword	Safe zone	KNN	Range
DBRQ [22]	**Y**	**N**	**Y**	**N**	**Y**	**N**	**Y**
CMRSK [23]	**Y**	**N**	**Y**	**Y**	**N**	**N**	**Y**
C-MNDR [24]	**Y**	**N**	**Y**	**N**	**N**	**N**	**Y**
TMS [25]	**N**	**N**	**Y**	**N**	**N**	**Y**	**Y**
CSPRQ [26]	**N**	**N**	**Y**	**N**	**N**	**N**	**Y**
UNICONS [20]	**Y**	**Y**	**Y**	**N**	**Y**	**Y**	**N**
M k NN [1]	**N**	**N**	**Y**	**N**	**Y**	**Y**	**N**
GMA [3]	**Y**	**N**	**Y**	**N**	**N**	**Y**	**N**
V*-digram [4]	**Y**	**N**	**Y**	**N**	**Y**	**Y**	**N**
MkSK [6]	**N**	**N**	**Y**	**Y**	**Y**	**Y**	**N**
LARC [9]	**Y**	**N**	**Y**	**Y**	**Y**	**Y**	**N**
CPM [2],YPK-CNN [7], DKNN [8]	**N**	**N**	**Y**	**N**	**N**	**Y**	**N**

ignoring the rest). In [7], to support various scenarios, Object-Indexing and Query-Indexing are developed. Specifically, both of them are based on the grid index. Specifically, [8, 11, 30] retrieve the k NN set in the context of distributed environment. To be adaptable to dynamic environment, [8] develops the dynamic strip index (DSI), which enables to merge or split objects dynamically. Besides, [31, 32] consider querying trajectories/routes that a composed of a sequence of geo-locations associated with text descriptions, Cong et al. [31] proposes the B^{ck}-tree index integrating spatial information and text information for the top-k spatial keyword query on trajectories. Feng et al. [32] focus on the indoor top-k keyword-aware routing query.

2.2. Road Index Structure. Distance computation brings great challenge in the presence of large scale road network. To address this problem, various index structures have been studied. In general, previous works on the road network index structure can be roughly classified into two categories: (1) some maintain the precomputed distances in flat structure [14, 15]. Literature [14] utilizes the lab technique to facilitate the distance computation. In particular, the distance between each pair of vertices can be computed easily by looking up the precomputed results. Hu et al. [15] propose the distance signature structure. Each vertex in the road network maintains a distance signature. Intuitively, distance signature needs to store all the distances between other vertices to current vertex. To minimize the storage cost, it maps the distances between objects and network nodes into categories and then encodes these categories. (2) others consider the hierarchical structure [16–18, 33, 34]. CH [33] provides a method to generate a hierarchical structure by iteratively contracting the least important node. That is, replacing these nodes with shortest cuts. This strategy relies heavily on the weight of edges, thus the whole index structure is needed to be recomputed when the weight of edges changes. In [17], they pre-compute all pairs of distance and organize the computed results in different subsets. This is motivated by the fact that the shortest paths from vertex u to all of the remaining vertices can be decomposed into subsets based on the first edges on the shortest paths to them from u. Literature [16, 18] devise the hierarchical structures to organize the road network by partitioning the road network into multiple sub-networks. In [18], authors build a multi-level index structure called G-tree. It is constructed by iteratively partitioning the road networks into equal-size sub networks until the size of each leaf node is within the given threshold. Specifically, each node in G-tree is associated with a distance matrix that records the distance between borders (for non-leaf nodes) or the distance between borders and vertices (for leaf nodes). With G-tree, the distance between each pair of vertices can be computed by assembly-based method.

Summary of the related works can be found in Table 1.

Table 2: Summary of the notations.

Notation	Explanation
$G(V, E)$	Road network with node set V and edge set E
v	a vertex of the road network
$e_{i,j}$	An edge of the road network
$\|v_i, v_j\|$	Network distance between v_i and v_j
P	a path on the road network
Q_r	Path based range query
Q_k	Path based k NN query
BNI	Backbone network index structure

3. Preliminary

3.1. Problem Statement. In this work, we model a road network as an undirected and edge weighted graph $\mathcal{G} = <V, E>$, where $\mathcal{V}$ refers to the set of vertices and $\mathcal{E}$ refers to the set of edges. Each vertex $v \in \mathcal{V}$ is comprised of a location $v.\ell$ and a set of keywords $v.\psi$. Each edge $e_{i,j}$ connecting the vertices v_i and v_j is associated with a real-valued number $w_{i,j}$ $(w_{i,j} > 0)$ to capture the weight, e.g., the distance or travel time, of this edge. For ease of discussion, we refer to the weight as distance hereafter. Given two vertices v_i, $v_j \in \mathcal{V}$, we denote by $\|v_i, v_j\|$ the shortest distance among all paths between them. Without loss of generality, we denote the path by a sequence of adjacent vertices on the road network, e.g., $P = (v_{i_1}, v_{i_2}, \ldots, v_{i_k})$. Table 2: lists the frequently used notations. Next, we formally define our problem.

Definition 1. Path Based Continuous Range (PCR) Queries. Given a road network $\mathcal{G}$, a path based continuous range

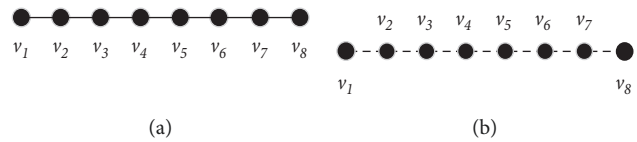


Figure 2: Illustration of observation 1. (a) Original edge. (b) Virtual edge.

query is of the form $Q_r = <P, \mathcal{T}, \psi>$, where P is the query path, $\mathcal{T}$ is the distance threshold, and ψ is the set of query keywords. The query asks for a set of vertices $S_q^r = \{v | \psi \subseteq v.\psi \wedge \|v, q\| \leqslant \mathcal{T}\}$ for each query point q on P.

Definition 2. Path Based Continuous k **NN (PC** k NN) Queries. Given a road network $\mathcal{G}$, a path based continuous k NN query is of the form $Q_k = <P, k, \psi>$, where P is the query path, k is the retrieved size, and ψ is the set of query keywords. The query asks for a set of vertices S_q^k for each query point q on P. Specifically, S_q^k satisfies the following three conditions:

(i) $|S_q^k| = k$.
(ii) $\forall v \in S_q^k$, $\forall v' \notin S_q^k$, $\|v, q\| \leq \|v', q\|$.
(iii) for each $v \in S_q^k$, it covers the query keywords set, i.e. $\psi \subseteq v.\psi$.

Intuitively, there are infinite number of points on the query path, which renders issuing query at each point infeasible. We will show how to transform our queries to an equivalent problem in Section 5.

3.2. Backbone Network. We notice that all the proposed road network index structures are built over the original road network. When the road network is large, the index construction overhead and the query evaluation overhead are expensive. To address this challenge, we propose to build the index over the backbone network based on the following observation.

3.2.1. Observation 1. For any vertex v with two adjacent edges, we can treat v as an internal vertex on a virtual edge.

Figure 2(a) depicts the instance of an original road network with 8 vertices, i.e., $v_1, v_2, \ldots, v_8$. Since $v_2, v_3, \ldots, v_7$ only have two adjacent edges, we can introduce a virtual edge $e_{1,8}$ and take them as the internal vertices as shown in Figure 2(b).

3.2.2. Simplifying Strategy. Observation 1 offers a simple but powerful insight for us to simplify the road network. That is, if a vertex only has two adjacent edges, we can introduce a virtual edge and take it as an internal vertex on the virtual edge. This strategy is similar to CH [33]. We observe that CH is computationally expensive and relies heavily on the weight of edges. It needs to be rebuilt when the weight of edges changes. Hereafter, we denote by backbone network the simplified network. We call the remaining vertices on the backbone network as backbone vertex (e.g.), and vertices lying on the virtual edges as internal vertex.

Figure 3 presents an instance of the backbone network. We denote by the black (white) circle the backbone (internal) vertex in the original network (Figure 3(a)). We present the retrieved backbone network in Figure 3(b). Furthermore, we denote by the dotted line the virtual edge. Clearly, the backbone network achieves a compact structure compared with the original road network (Experimental results in Section 7 show that the size of road network can be cut off from 50% to 95%). It is obvious that there may be more than one edge between two vertices. For instance, there are two paths, namely (v_1, v_{10}) and $(v_1, v_{15}, v_4, v_{10})$, between v_1 and v_{10} (one is the virtual edge and the other is the original edge). In this case, we take the one with smallest distance as the edge in the backbone network, whist maintaining all edges in our index structure (to be shown later). We denote by $e_{i,j}^B$ the edge that connects the vertices v_i, v_j. Note that, $e_{i,j}^B$ is either an original edge or a virtual edge comprising of multiple original edges. For two internal vertices v_i, v_j, we denote by $[v_i, v_j]$ the direct distance between them on the corresponding virtual edge.

Theorem 1. *$\forall v_m, v_n \in V$, the shortest distance between them is defined as:*

if v_m, v_n are on the same virtual edge $e_{i,j}^B$, then:

$$\|v_m, v_n\| = \min\left\{[v_m, v_n], \|v_i, v_j\| + [v_i, v_m] + [v_n, v_j]\right\}. \quad (1)$$

if v_m, v_n are not on the same virtual edge $e_{i,j}^B$, then:

$$\|v_m, v_n\| = \min\left\{\|v_m, v_i\| + [v_i, v_n], \|v_m, v_j\| + [v_n, v_j]\right\}. \quad (2)$$

Proof. This theorem is self-evident, we omit the proof here due to the space consideration, the illustration of this theorem can be found in Figure 4. □

4. Backbone Network Index Structure

In this section, we introduce the backbone network index (BNI) structure, which is a dual-index. Specifically, it maintains the global insights of network by G-tree (in memory) and the detailed information of network by the adjacent list (on disk).

4.1. G-Tree Index Structure. Though there are various partition strategies for road network, we adopts [18] in this work. This is because G-tree has the following properties: (1) enables efficient distance computations for large network; (2) offers global sight of the road network. We proceed to introduce some basic notions that lay the foundation of G-tree.

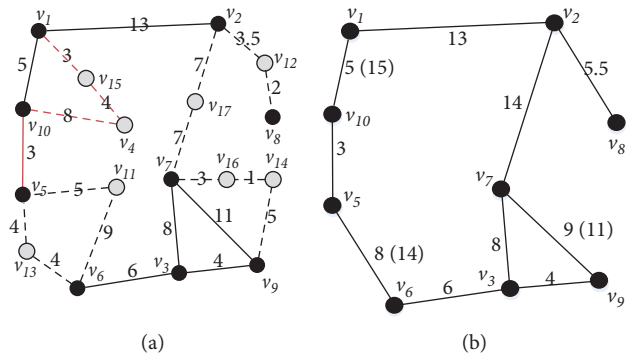


Figure 3: Backbone network. (a) Original network. (b) Backbone network

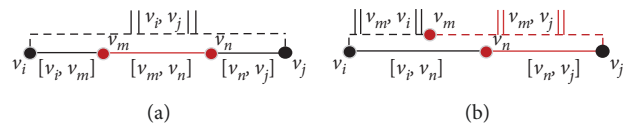
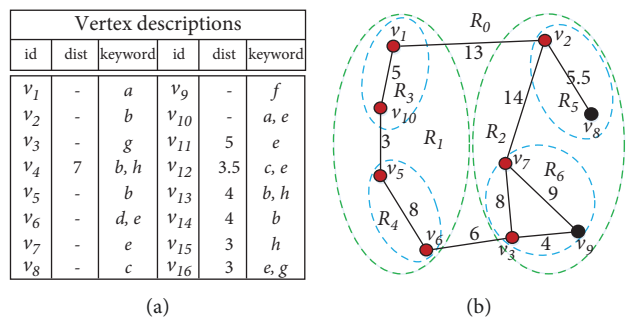


Figure 4: Illustration of Theorem 1. (a) Internal distance computation. (b) External distance computation.



Vertex descriptions

id	dist	keyword	id	dist	keyword
v_1	-	a	v_9	-	f
v_2	-	b	v_{10}	-	a, e
v_3	-	g	v_{11}	5	e
v_4	7	b, h	v_{12}	3.5	c, e
v_5	-	b	v_{13}	4	b, h
v_6	-	d, e	v_{14}	4	b
v_7	-	e	v_{15}	3	h
v_8	-	c	v_{16}	3	e, g

Figure 5: Backbone network structure. (a) Vertex descriptions. (b) Network partition.

Definition 3. (Backbone Network Partition). Given a road network $\mathcal{G} = <V, E>$, we denote by $\mathcal{G}^B = <\mathcal{V}^B, \mathcal{E}^B>$ the corresponding backbone network. The partition of $\mathcal{G}^B$ is a set of regions, i.e., $\mathcal{R}_1 = <\mathcal{V}_1^B, \mathcal{E}_1^B>$,..., $\mathcal{R}_m = <\mathcal{V}_m^B, \mathcal{E}_m^B>$ such that:

(i) $\bigcup_{1 \le i \le m} \mathcal{V}_i^B = \mathcal{V}^B$,
(ii) For $i \neq j, \mathcal{V}_i^B \cap \mathcal{V}_j^B = \emptyset$, and
(iii) $\forall u, v \in \mathcal{V}_i^B$, if $(u, v) \in \mathcal{E}^B$, then $(u, v) \in \mathcal{E}_i^B$.

In the partitions, there are some portal vertices that connect to the vertices of other regions. To illustrate this, we define the concept of borders as follows.

Definition 4 (Backbone Network Border). Given a backbone network $\mathcal{G}^B = <\mathcal{V}^B, \mathcal{E}^B>$, a region $\mathcal{R}_i$ of $\mathcal{G}^B$. We say that $u \in \mathcal{V}_i^B$ is a border if $\exists e_{u,v} \in \mathcal{E}^B$ and $v \notin \mathcal{V}_i^B$. Hereafter, we denote by $\mathcal{B}(\mathcal{R}_i)$ all the borders of $\mathcal{R}_i$.

As shown in Figure 5, we present an instance of the partition of the backbone road network. Note that, all borders are marked with red. Correspondingly, we show the G-tree in Figure 6, which is built over the road network in Figure 5(b). Specifically, each node in G-tree corresponds to one subregion in Figure 5(b).

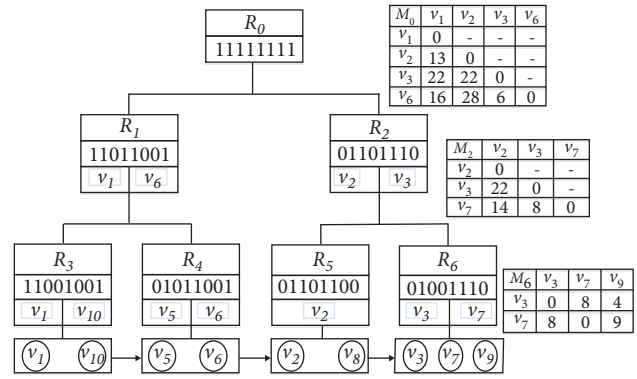


Figure 6: G-tree index structure.

Each node in G-tree is of the form $\langle i\ d, \psi_H, \mathcal{B}(\mathcal{R}), \mathcal{M} \rangle$, where $i\ d$ is the identifer of the node, ψ_H contains some keyword information of the corresponding node (will be discussed later). We denote by $\mathcal{B}(\mathcal{R})$ the borders of $\mathcal{R}$, and $\mathcal{M}$ refers to the distance matrix. Specifically, for leaf node, it records the distance

between borders and backbone vertices (e.g. $\mathcal{M}_6$). For non-leaf node, it record the distance between borders (e.g. $\mathcal{M}_2$).

As with [35], we map vertices and keywords to a h bits number by a hush function H. Specifically, for each keyword ω, $H(\omega)$ sets exactly one of the h bits to 1. $H(v)$ is the superimposition of $H(w)$ for each $w \in v.\psi$, and ψ_H is the superimposition of $H(v)$ for each relevant vertex v. We say v is relevant to a node $\mathcal{R}$ if v lies on an edge $e^B_{i,j}$ such that $v_i \in \mathcal{R}.\mathcal{V}^B$ or $v_j \in \mathcal{R}.\mathcal{V}^B$. That is, relevant vertices contain all the vertices in $\mathcal{R}$ and the vertices on the adjacent edges to $\mathcal{R}$.

For a vertex v and a node $\mathcal{R}$, we denote by $\text{minDist}(v, \mathcal{R})$ the minimum possible distance between v and the vertices in $\mathcal{R}$. Specifically, if $v \in \mathcal{R}.V^B$, then we set $\text{minDist}(v, \mathcal{R})$ to 0. Otherwise, we set $\text{minDist}(v, \mathcal{R})$ as the minimum distance between v and the borders of $\mathcal{R}$, i.e., $\text{minDist}(v, \mathcal{R}) = \min_{v' \in \mathcal{B}(R)} \|v, v'\|$.

4.2. Generalized Adjacent List File. Adjacent list has been used widely in the context of road network [19, 36] for maintaining detailed information of edges. In this work, we generalize the adjacent list by the following two aspects. First, we maintain all edges (including both the real edge and virtual edges) for two adjacent vertices v and u in the backbone network. Second, the road network is stored on disk based on locality principle rather than the vertex id. The partition strategy of G-tree suggests that the vertices in the same subregion has higher possibility to be accessed together. Inspired by this, we propose to maintain the road network sequently from left to right leaf nodes, as shown by the arrow line in Figure 6.

As shown in Figure 7, adjacent list is comprised of three components, namely hash table, adjacent file and point file. We proceed to introduce them. By hash table, each backbone vertex v_i is mapped to a pair $<\mathcal{R}_i, \mathcal{A}_i>$, where $\mathcal{R}_i$ refers to the leaf node containing v_i and $\mathcal{A}_i$ records the start address of the item for v_i in adjacent file. For each backbone vertex v_i, we maintain an item for it in adjacent file. The item first records the number of adjacent vertices, and the associated keywords for v_i. Then, each adjacent vertex corresponds to an entry, and each edge is associated with a record file, the superimposition of keywords associated with the internal vertices, and the start address of internal vertices on this edge.The structure of point file is simple compared with adjacent file. In the point file, all internal vertices lying on edges between two vertices are placed in one group. More details about the generalized adjacent list file can be found in our previous work [21].

The structure of point file is simple compared with adjacent file. In the point file, all internal vertices lying on edges between two vertices are placed in one group. The group first records the corresponding backbone vertices, and the total number of internal vertices. Then, we maintain the id, the direct distance to smaller vertex, and the associated keywords for each internal vertex.

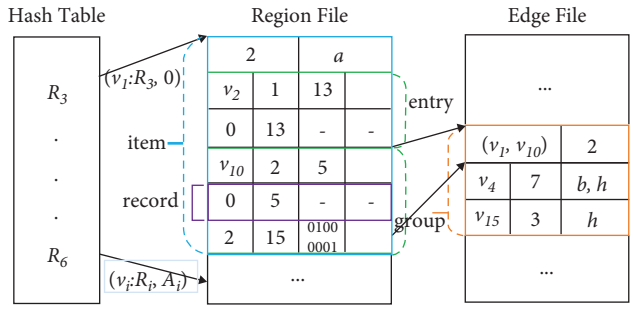


Figure 7: Generalized adjacent list file.

5. Algorithms

In this section, we first explore the query framework at a high level, and then present the algorithms for PCR and PC k NN queries.

5.1. Overview of the Framework. Considering there are infinite number of query points on the query path, it is infeasible to issue the query at each point. We proceed to show how to transform our queries to a simple but equivalent problem.

Theorem 2. *Given a path based range query Q_r, and an edge $e_{i,j}$ on $Q_r.P$. For any point q on $e_{i,j}$, it holds that $S^r_q \subseteq S^r_{v_i} \cup S^r_{v_j} \cup \Omega(e_{i,j})$, where $\Omega(e_{i,j})$ is the set of internal vertices on $e_{i,j}$ that cover $Q_r.\psi$.*

Proof. The proof of this theorem can be found in the conference version in [21]. And the illustration of this theorem can be found in Figure 8. □

Theorem 3. *Given a path based k NN query Q_k, and an edge $e_{i,j}$ on $Q_k.P$. For any point q on $e_{i,j}$, it holds that $S^k_q \subseteq S^k_{v_i} \cup S^k_{v_j} \cup \Omega(e_{i,j})$, where $\Omega(e_{i,j})$ is the set of internal vertices on $e_{i,j}$ that cover $Q_k.\psi$.*

Proof. The proof of this theorem can be derived as with Theorem 2. We omit the proof here.

Inspired by the safe zone technique [4] we know that the answer set might not change when query point moves within some intervals. This intuition offers us a direction to transform our queries to a simple but equivalent problem. □

Definition 5. Safe Interval. Given a query path P, we define the safe interval as a disclose segment on P, denoted by $\mathcal{SI}(x, y)$, where x and y denote the start point and end point of this interval. When q moves within $\mathcal{SI}(x, y)$, the answer set is unchanged. We denote the safe interval of Q_r and Q_k as $\mathcal{SI}_r(x, y)$ and $\mathcal{SI}_k(x, y)$ respectively.

In practice, there are multiple safe intervals on P, which are split by some special points, i.e., the start and end points of safe intervals. For ease of presentation, we call these points as event point hereafter.

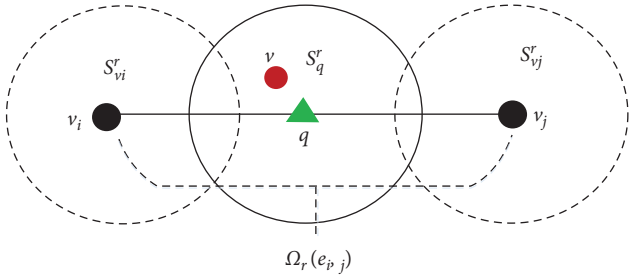


Figure 8: Illustration of Theorems 2 and 3.

Definition 6. Backbone Path. Given a query path P in $\mathcal{G} = <V, E>$, we define the corresponding backbone path P^B as a path in $\mathcal{G}^B = <V^B, E^B>$ such that:

(i) $\forall v \in P, \exists v_i, v_j \in P^B$ such that v locates on $e_{i,j}$, and
(ii) P^B is minimum.

In the above definition, we say P^B is minimum if there does not exist another backbone path P' that contains less number of vertices than P^B and satisfies the first condition.

We answer the path based queries by a two phases query framework. The task of the first phase is to compute the answer set for each vertex in P^B. Note that, this can be achieved by only issuing the query once. We maintain the retrieved answer sets for the next phase. We call this phase Issuing query. In the second phase, we mainly focus on finding the event point on the P^B. As discussed in Theorems 2 and 3, the answer sets for these points can be identified easily without issuing the query at these points. This phase is called Identifying event point.

Clearly, by these two phases, we can answer the queries easily. Meanwhile, the computation cost can be cut off significantly. The Two-Phase Progressive (TPP) framework is described in Algorithm 1. The details of the algorithm can be found in the conference version in [21].

5.2. PCR Query Processing. We answer the PCR query by two steps. Firstly, we get the answer sets for backbone vertices by issuing the range query at the first backbone vertex in P^B. Then, we identify event points by these sets.

5.2.1. Issuing Range Query. Intuitively, the answer sets for these backbone vertices are overlapped. This motivates us to compute the answer sets progressively. The following theorem offers us the insight for progressive computation.

Theorem 4. *Given two query point q, q' on the query path, and a distance threshold $\mathcal{T}$. We denote by $S^r_{q'}$ the range query results of q'. For any $v \in S^r_{q'}$, it holds that $\|v, q\| \leq \mathcal{T} + \|q, q'\|$.*

Proof. This theorem is obvious. We omit the proof here.

Considering the fact that the distance between backbone vertices is not large, we propose to achieve the progressive computation by a filter-and-verification model. At a high level, we expand the network from the first vertex v_f in P^B progressively (for filter). We know that, for each backbone vertex v, the candidates must within the range of $\mathcal{T} + \|v, v_f\|$. That is, we only need to verify the vertices within this range for other backbone vertices (for verification).

As presented in Algorithm 2, we retrieve the answer sets for backbone vertices in P^B by a filter-and-verification framework. Particularly, we utilize curtDist ≤ stopDist as the filter condition for the current visited vertex v (line 7). From Theorem 4, we know that the range query results for v are within the range of stopDist to v_f. In the process of tree traversal, if $\mathcal{R}_i$ is a non-leaf node, we need to add its child nodes that cover the query keywords into the priority queue PQ for further exploration (lines 9–13). Otherwise, that is $\mathcal{R}_i$ is a leaf node. We first compute the distance between v_f and the backbone vertices in $\mathcal{R}_i$ (lines 15–16). Then, for each backbone vertex v', we find the results by checking the vertices lying on the adjacent edges (lines 17–19). Note that, we also maintain the candidates, which cover the query keywords, by $\mathcal{C}$ for achieving quick verification. When the filter condition is broken, we proceed to build S^r_v by verifying candidates in $\mathcal{C}$ (lines 20–23).

Optimizing Strategies: To facilitate the query processing, we investigate the optimizing strategies. We observe that there are much repeated computation when computing the minDist(·) and performing verifications, thus we can maintain the computed distances for the following distance computation. □

5.2.2. Identifying Event Point. We mainly focus on identifying event points on edges. Specifically, we distinguish event points by two types, namely *IE* and *OE*. We denote by *IE* the in event point, and *OE* the out event point. We refer to in/out from the direction of the backbone vertex that is near to v_f. As suggested in Theorem 2, the event points on $e_{i,j}$ are determined by $S^r_{v_i}$, $S^r_{v_j}$ and $Q_r.\mathcal{T}$. Specifically, there are seven cases, as follows:

Theorem 5. *Given two backbone vertices v_i, v_j on $\overline{R}$, and a vertex covering the keywords. Then, the event point in the edge has the following seven cases (we only describe the first three cases according to Figure 9, and referring our previous work [21] for more details.):*

(i) If v on one edge between v_i, v_j, and $[v_i, v] > Q_r.T \wedge [v, v_j] > Q_r.T$, then there are two event point ;
(ii) If v on one edge between v_i, v_j, and $[v_i, v] > Q_r.T \wedge [v, v_j] > Q_r.T$, then there are two event point .
(iii) If v on one edge between v_i, v_j, and $[v_i, v] < Q_r.T \wedge [v, v_j] > Q_r.T$, then there are two event point.

Algorithm 3 presents the pseudo-code for identifying the event points. For each edge $e_{i,j}$ on P^B, we denote by $S^r_{v_i}$ and

Input : $Q_r = <P, \mathcal{T}, \psi>$
Output: S
(1) V_Q←Construct the query vertices from $Q_r.P$
(2) S_Q←IssuingSpatialKeywordQuery(Q_r)
(3) Ξ←IdentifyingEventPoint(S_Q, V_Q);
(4) S←Compute answer sets for event points in Ξ progressively
(5) Return S.

Algorithm 1: TPP Framework.

Input : $Q_r = <P, \mathcal{T}, \psi>$
Output: P^B, S
(1) build P^B from P
(2) PQ.push$(\mathcal{R}_0, 0)$;
(3) curtDist← 0; stopDist←0
(4) v_f←The first vertex in P^B;
(5) **for** each P^B **do**
(6) stopDist←$Q_r.\mathcal{T} + \|v, v_f\|$
(7) **while** (curtDist ≤ stopDist) **do**
(8) $\mathcal{R}_i$, curtDist←PQ.pop;
(9) **if** $\mathcal{R}_i$ is a non-leaf node **then**
(10) **for** each child node $\mathcal{R}_c$ of $\mathcal{R}_i$ **do**
(11) **if** $Q_r.\psi \subseteq \mathcal{R}_c.\psi_H$ **then**
(12) compute the minDist(v_f, R_c);
(13) PQ.push$(R_c$, minDist$(v_f, R_c))$;
(14) **else**
(15) **if** v_f is in $\mathcal{R}_i$ **then**
(16) InternalDist $(v_f, \mathcal{R}_i)$;
(17) **else** ExternalDist $(v_f, \mathcal{R}_i)$;
(18) ;
(19) **for** each $v' \in \mathcal{R}_i$ **do**
(20) put candidates on adjacent edges into $\mathcal{C}[\mathcal{R}_i]$;
(21) put qualified candidates into $S^r_{v_f}$;
(22) **for** each $\mathcal{R}_i$ in $\mathcal{C}$ **do**
(23) **if** v is in $\mathcal{R}_i$ **then** InternalDist $(v, \mathcal{R}_i)$;
(24) ;
(25) **else** ExternalDist $(v, \mathcal{R}_i)$;
(26) ;
(27) put all qualified candidates of $\mathcal{C}[\mathcal{R}_i]$ into S^r_v
(28) Return P^B and S;

Algorithm 2: Issuing Range Query.

$S^r_{v_j}$ the answer sets for v_i and v_j. We first identify the event points for internal vertices on $e_{i,j}$ (lines 4–5). Then, for each vertex $v \in S^r_{v_i}$, if it is on the edge, we know that it has already been processed. Otherwise, we find out whether it is in $S^r_{v_j}$. If yes, there might exist both in and out event points (lines 8–9). If no, we only need to compute the out event points for v (lines 10–11). Then, for the vertices in $S^r_{v_j}$, we only need to process ones that are neither on $e_{i,j}$ nor in $S^r_{v_i}$. For these vertices, we need to compute the in event point for them (lines 12–15).

5.3. PC k NN Query Processing. Similarly, to answer the PC k NN queries, we first retrieve the k NN sets for backbone vertices in P^B. Then, we identify the event points.

5.4. Issuing k NN Query

Theorem 6. *Given two query point q, q' on the query path. We denote by $S^k_{q'}$ the k nearest neighbor query results of q'. For any $v \in S^k_{q'}$, it holds that $\|v, q\| \le \mathcal{K}_q + \|q, q'\|$. Here, $\mathcal{K}_q$ refers to the distance between q and the k th nearest neighbor.*

Proof. This theorem is obvious. We omit the proof here.

Theorem 6 enables to find the k nearest neighbor by only issuing the k nearest neighbor search once. Algorithm 4 shows the pseudo-code for finding k NN sets for backbone vertices. Generally, it is similar to Algorithm 1. The major difference comes from the filter condition (line 7). We denoted by $kth[v_f]$ the distance between v_f and the k th nearest neighbor. Besides, we need to update $kth[v_f]$ accordingly when new result is inserted into $S^r_{v_f}$ (lines 17–20). □

Input : S, P^B
Output: The sequence of event points.
(1) **for** $l \leftarrow 1$ to m **do**
(2) $v_i \leftarrow P^B[l-1]$; $v_j \leftarrow P^B[l]$;
(3) $\Omega(e_{i,j}) \leftarrow$ the qualified vertices on $e_{i,j}$;
(4) **for** each $v \in \Omega(e_{i,j})$ **do**
(5) $\Xi_{v_i} \leftarrow$event points computed by (1–4)
(6) **for** each $v \in S^r_{v_i}$ **do**
(7) **if** v is not on $e_{i,j}$ **then**
(8) **if** v is not in $S^r_{v_j}$ **then**
(9) $\Xi_{v_i} \leftarrow$event points computed by (5);
(10) **else**
(11) $\Xi_{v_i} \leftarrow$event points computed by (7);
(12) **for** each $v \in S^r_{v_j}$ **do**
(13) **if** v is not on $e_{i,j}$ **then**
(14) **if** v is not in $S^r_{v_i}$ **then**
(15) $\Xi_{v_i} \leftarrow$event points computed by (6)
(16) Return Ξ

Algorithm 3: PCR – IdentifyingEventPoint.

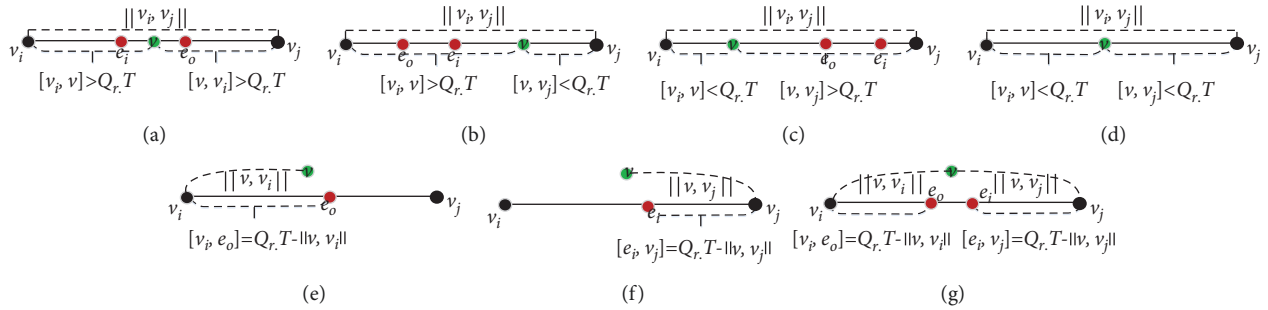


Figure 9: Illustration of range event point computation. (a) Event point 1. (b) Event point 2. (c) Event point 3. (d) Event point 4. (e) Event point 5. (f) Event point 6. (g) Event point 7.

5.4.1. Identifying Event Point. Compared with the range query, the k NN query is more complicated. Whether a vertex v is the k NN result for the query is not only determined by its distance to query point but also by other vertices. This leads to the difficulty in identifying the event points for k NN queries. To address this problem, we introduce the concept of distance curve, which captures the change of distance between query point and the candidate when query point moves along the edge.

Figure 10(a) illustrates the distance curve for different cases. Given the candidate vertex v, for c_1, c_2, we know that the shortest path from v to v_j or v_i must go through v_i or v_j. For c_3, c_4, it is clear that v is on $e_{i,j}$. Different from c_3, c_4, we know that c_5, c_6 present cases as there are more changes. Based on the distance curve, we identify the event points as illustrated in Figure 10(b). We know that the event points always appear at the intersections and turn points. To find out the event points, we can first compute all the intersections and turn points, and then compute the k NN at each turn points. This naive method is time consuming, thus we propose the enhanced method. We maintain the k NN curves that has the lowest distance. Then, we compute the intersections between these curves and other curves. We select the first intersection, and compute the k NN. This procedure continues until this edge is explored.

We present the details of how to compute the event points for k NN in Algorithm 5. We first compute the distance curves for vertices in $S^k_{v_i}$ and $S^k_{v_j}$ (line 3). Then, we compute the event points by accessing the next explored intersection. At this point, we update the k NN by comparing the distance value of start and end points. If k NN is changed, we add the event point into Ξ_{v_i} (lines 8–9).

6. Analysis

We proceed to offer some insights of our index structure and algorithms.

6.1. Analysis on Index Structure. As discussed in Section 3, the key point of compact strategy is to remove some vertices from the original network, and then takes them as internal vertices on introduced virtual edges. Suppose the compression ratio is Υ (As shown by experimental studies, this value varies from 5% to 75%). That is, $\mathscr{V}^B = V\Upsilon$. Since an

Input : $Q_r = <P, \mathcal{T}, \psi>$
Output: P^B, S
(1) build P^B from P;
(2) $P\mathcal{Q}$.push$(R_0, 0)$;
(3) curtDist← 0; stopDist←0;
(4) **for** each v in P^B **do**
(5) $kth[v]$← *INFINITY* ;
(6) stopDist←$\|v, v_f\|$;
(7) **while** (curtDist < (stopDist + kth$[v_f]$)) **do**
(8) R_i, curtDist←$P\mathcal{Q}$.pop
(9) **if** $\mathcal{R}_i$ is a non-leaf node **then**
(10) **for** each child node $\mathcal{R}_c$ of $\mathcal{R}_i$ **do**
(11) **if** $Q_r.\psi \subseteq \mathcal{R}_c.\psi_H$ **then**
(12) compute the minDist(v_f, R_c)
(13) $P\mathcal{Q}$.push$(R_c$, minDist$(v_f, R_c))$;
(14) **else**
(15) **if** v_f is in $\mathcal{R}_i$ **then**
(16) InternalDist $(v_f, \mathcal{R}_i)$
(17) **else** ExternalDist $(v_f, \mathcal{R}_i)$
(18) ;
(19) **for** each $v' \in \mathcal{R}_i$ **do**
(20) put candidates on adjacent edges into $\mathcal{C}[\mathcal{R}_i]$;
(21) put qualified candidates into $S^r_{v_f}$;
(22) update $kth[v_f]$;
(23) **for** each $\mathcal{R}_i$ in $\mathcal{C}$ **do**
(24) **if** v is in $\mathcal{R}_i$ **then** InternalDist $(v, \mathcal{R}_i)$
(25) ;
(26) **else** ExternalDist $(v, \mathcal{R}_i)$
(27) ;
(28) put all qualified candidates of $\mathcal{C}[\mathcal{R}_i]$ into S^r_v
(29) Return P^B and S

Algorithm 4: IssuingkNNQuery.

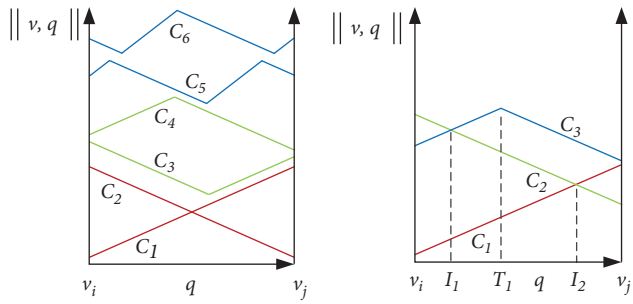


Figure 10: Illustration of distance function. (a) Distance curve. (b) Event point.

edge will be removed from the original network when a vertex is removed. We know that, it holds that $\mathcal{E}^B = EY$.

The whole space cost of BNI is comprised of three parts, namely hash table, G-tree and generalized adjacent list. By hash table, we need to maintain the corresponding leaf node and address for each backbone vertex in $\mathcal{V}^B$. Thus, hash table costs $\mathcal{O}(|\mathcal{V}^B|)$. Motivated by the analysis about G-tree in [18], we know that the space cost of G-tree is $\mathcal{O}(|\mathcal{V}^B|/\tau + \log_2 f/\sqrt{\tau} \cdot |\mathcal{V}^B| + \log_2 f \cdot \sqrt{\tau}\,|\mathcal{V}^B| + \log_2^2 f \cdot \log_f |\mathcal{V}^B|/\tau \cdot |\mathcal{V}^B|)$. Please refer to [18] for more details. Furthermore, we also maintain the detailed information about road network in the generalized adjacent list, which costs $O(|\mathcal{V}^B| + |\mathcal{E}^B|)$. We denote by f and τ the number of partition regions for each partition and the upper bound of the number of vertices in leaf nodes. Thus, the total cost is $\mathcal{O}(\log_2 f \cdot \sqrt{\tau}\,|\mathcal{V}^B| + \log_2^2 f \cdot \log_f |\mathcal{V}^B|/\tau \cdot |\mathcal{V}^B| + |\mathcal{E}^B|)$.

6.2. Analysis on Algorithms

6.2.1. Analysis on Range Queries

(1) Time Complexity. Clearly, the time complexity of path based continuous range queries comes from two components, namely issuing range queries and identifying event

Input : S, P^B
Output: The sequence of event point.
(1) **for** $l \leftarrow 1$ to m **do**
(2) $v_i \leftarrow P^B[l-1]$; $v_j \leftarrow P^B[l]$
(3) compute the distance curve for vertices in $S^k_{v_i}, S^k_{v_j}$;
(4) k NN $\leftarrow pres$; $curtPos \leftarrow 0$;
(5) **while** curtPos is on this edge **do**
(6) curtPos$\leftarrow$ minimum intersection position between k NN and other curves
(7) k NN $\leftarrow$ recompute the k NN at $curtPos$
(8) **if** k NN changes **then**
(9) add event point into Ξ_{v_i}
(10) Return Ξ

Algorithm 5: PCkNNIdentifyingEventPoint.

points. Suppose the length of query path is $\mathscr{L}$. That is, $\mathscr{L} = |P|$. As noted earlier, there are $\mathscr{L}^B = \Upsilon\mathscr{L}$ backbone vertices in P^B. We need to issue the range query once and perform $(\mathscr{L}^B - 1)$ times verification for other backbone vertices. Without loss of generality, we assume the range query needs to access $\mathscr{N}_r$ leaf nodes (regions) before retrieving all results. In the worst case, that is, all backbone vertices in P^B belong to different regions. Thus, each backbone vertex should explore at most $\mathscr{N}_r$ regions. Then, ExternalDist$(\cdot)$ should be invoked at most $\mathscr{L}^B\mathscr{N}_r$. Since the matrix size of each node at level i is $O(\log_2^2 f \cdot |\mathscr{V}^B|/f^i)$. The height of the G-tree is $\mathscr{H} = \log_f |\mathscr{V}^B|/\tau + 1$. Thus, the cost for *External Di st*$(\cdot)$ is $\mathcal{O}(\log_2^2 f \cdot |\mathscr{V}^B|/f) + \mathcal{O}(\log_2^2 f \cdot |\mathscr{V}^B|/f^2) + \ldots + \mathcal{O}(\log_2^2 f \cdot |\mathscr{V}^B|/f^{\mathscr{H}}) = \mathcal{O}(\log_2^2 f \cdot |\mathscr{V}^B|/f)$. Suppose the average number of vertices maintained for each backbone vertex is $\mathscr{N}^r_v$. The second part of cost is $\mathcal{O}(\mathscr{L}^B \cdot \mathscr{N}^r_v)$. The overall cost is $\mathcal{O}(\mathscr{L}^B\mathscr{N}_r \cdot \log_2^2 f \cdot |\mathscr{V}^B|/f + \mathscr{L}^B \cdot \mathscr{N}^r_v)$. In practice, the complexity is much smaller than the worst-case complexity.

(2) Space Complexity. As discussed earlier, we need to maintain the range queries answer sets for the backbone vertices in P^B. The space complexity is $\mathcal{O}(\mathscr{L}^B \cdot \mathscr{N}^r_v)$. Compared with distance matrices of G-tree, this space cost is negligible.

6.2.2. Analysis on k NN Queries

(1) Time Complexity. As presented by Algorithms 1 and 3, we know that the only difference between them is the filter condition. That is, we need to issue the k NN query once and perform $(\mathscr{L}^B - 1)$ times verification for other backbone vertices. Without loss of generality, we assume the k NN query needs to access $\mathscr{N}_k$ leaf nodes (regions) before retrieving all results. Similarly, in the worst case, each backbone vertex should explore at most $\mathscr{N}_k$ regions. Then, ExternalDist$(\cdot)$ should be invoked at most $\mathscr{L}^B\mathscr{N}_k$. Thus, the total cost for ExternalDist$(\cdot)$ is $\mathcal{O}(\mathscr{L}^B\mathscr{N}_k \cdot \log_2^2 f \cdot |\mathscr{V}^B|/f)$. Besides, the cost for computing the event points is $\mathcal{O}(\mathscr{L}^B \cdot k^2)$ in worst case. Thus, the total cost is $\mathcal{O}(\mathscr{L}^B\mathscr{N}_k \cdot \log_2^2 f \cdot |\mathscr{V}^B|/f) + \mathscr{L}^B \cdot k^2)$.

Table 3: The property of real life datasets.

Property	#Vertex	#Edge	$\lvert\mathscr{V}^B\rvert/\lvert\mathscr{V}\rvert$
SF	174,956	21,693	0.0648518
NA	175,813	179,179	0.0519074
LA	408,161	494,953	0.74266
NY	708,520	891,990	0.718204
CA	1,595,577	1,975,026	0.754325

Table 4: The query parameters.

Parameters	Instance value	Query
AK	3,4,5,**6**,7,8	Both
PL	30,40,**60**,70,80	Both
QK	2,3,4,**5**,6,7	Both
RT	1,2,4,**8**,16,32	Range
K	50,100,150,**200**,250,300	k NN

(2) Space Complexity. We need to maintain the range queries answer sets and the distance curves for the backbone vertices in P^B. Thus, the space complexity is $\mathcal{O}(\mathscr{L}^B \cdot k)$.

7. Experiments

We evaluate the performance of our proposed algorithms on both synthetic and real life data sets. We introduce the experimental settings in Section 7.1, and present the overall performance of the algorithms in Section 7.2. Section 7.3 studies the impact of different parameters on the performance of the algorithms.

7.1. Experimental Setup

7.1.1. Comparison Algorithms. Previous studies on the continuous spatial keyword queries usually exploit the modified version of Grid index [2, 8, 22] to organize the spatial objects. It is mainly because the index structure is easy to construct and can be applied to dynamic environment. As a result, we modify the algorithms in [22] for comparison, which is developed on the top of Grid index. To be fair, we partition the generated backbone network into $2^n * 2^n$ grid cells, such that the number of grid cells is as close to the leaf

Index	SF	NA	LA	NY	CA
Grid	485	535	48, 141	92, 870	460, 557
BNI	185	1, 849	64, 385	219, 222	1, 007, 403

Figure 11: Index construction time (second).

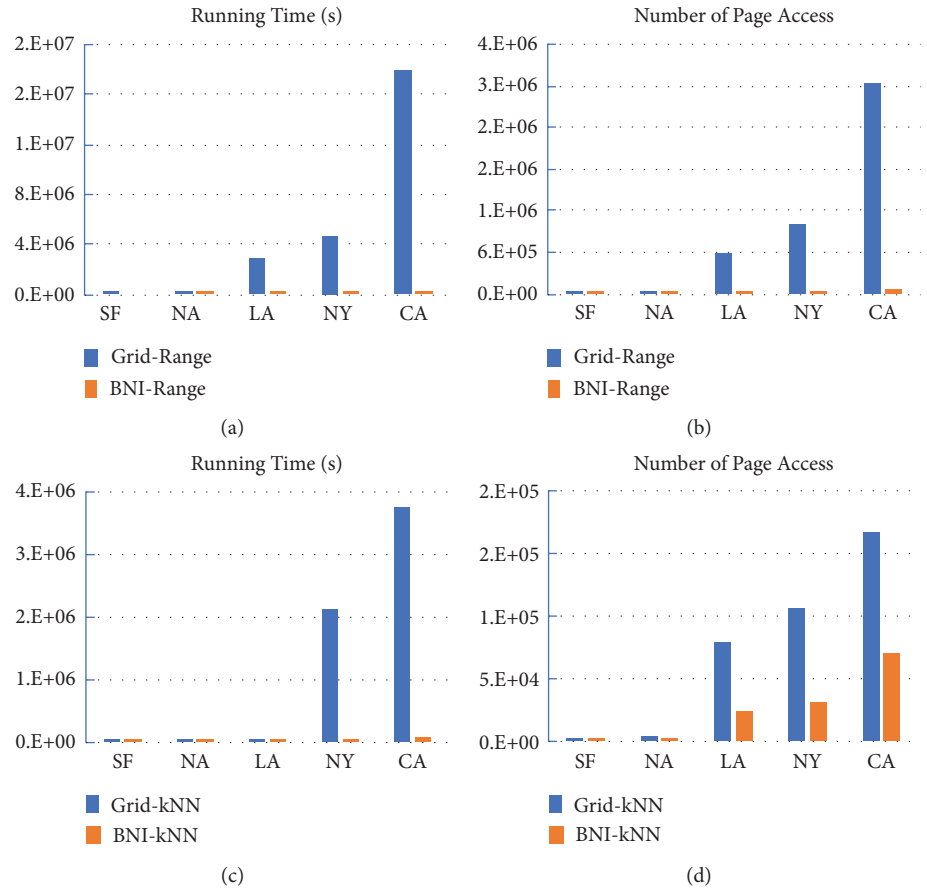


Figure 12: Overall performance on different data sets.

nodes of G-tree as possible. Similarly, we maintain borders (connecting vertices), distance matrix (distance table) and keyword bitmap for each cell. The range query algorithm first finds all relevant cells that satisfy the given distance threshold constraint. Then, the vertices in these cells are checked by query conditions to find out the final answer set. The k NN query algorithm explores the closest unvisited cell progressively using the modified version of Dijkstra algorithm until k qualified vertices are retrieved. During the query processing, distance matrix are exploited to facilitate the distance computation. For ease of discussion, we term our algorithms and competitors as BNI-Range, BNI-k NN, Grid-Range and Grid-k NN respectively.

7.1.2. Data and Queries. We test the algorithms on five real road networks, namely SF, NA, LA, NY, CA. We obtain SF and NA from [37], and LA, NY and CA are retrieved from TIGER/Line. Since the original road networks in LA, NY and CA are unconnected, we take the corresponding maximal connected subgraph as the road network. In Table 3, we reveal some statistics properties for real road networks. In particular, we denote by $|\mathcal{V}^B|/|\mathcal{V}|$ the compression ratio for the road network. Clearly, the road network can be reduced by 25%–95%. We generate the associated keywords for each vertex as shown in [36] based on the experimental setting. We mainly evaluate the performance on five parameters: (1) the average number of keywords associated with each vertex (AK); (2) the length of query path, i.e., the number of vertices on the path (PL); (3) the number of query keywords (QK); (4) the ratio of query scope, that is, the distance threshold for range query (RT); (5) the retrieved size for k NN query (K). To study the effect of these parameters, we conduct experiments on data set NA. Table 4 offers detailed information about the parameters. We mark their default values in bold. For each experiment, we randomly generate 20

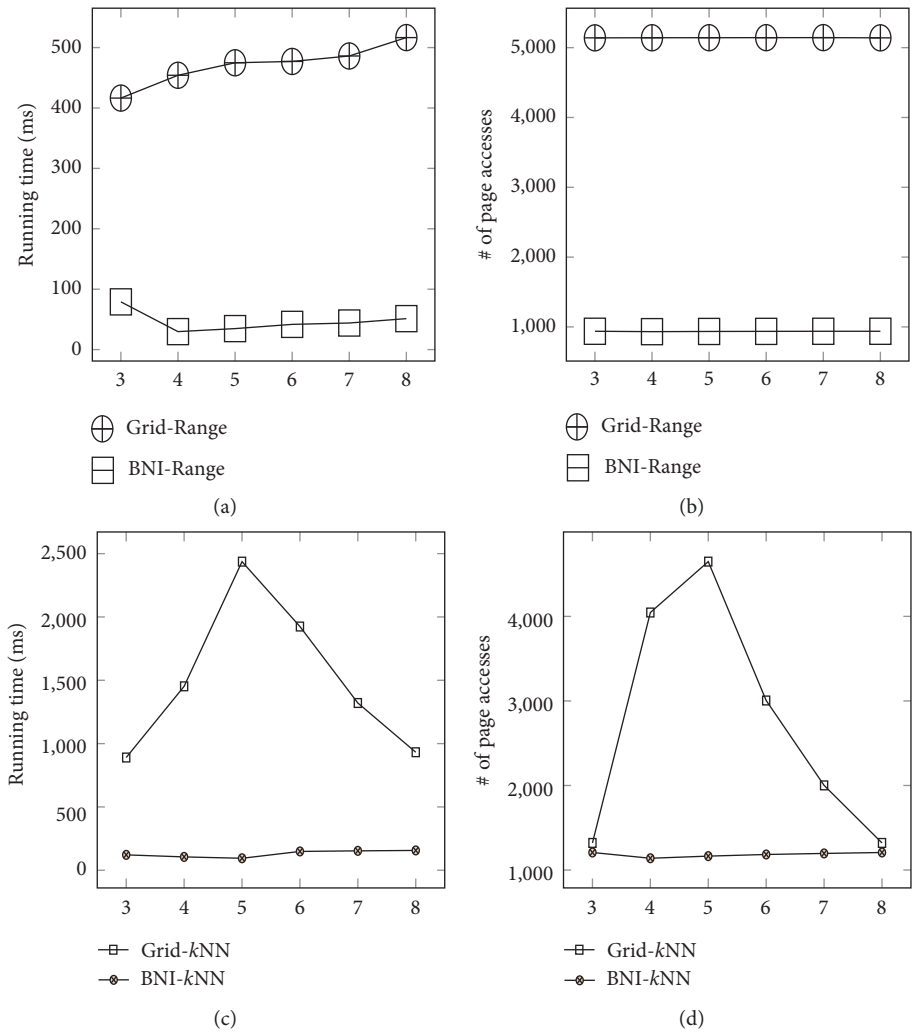


Figure 13: Effect of parameter AK.

queries based on the experimental setting in Table 4. We report the averaged performance.

All algorithms are implemented in C/C++ and run in Windows 7 System on an Intel(R) Core(TM) i5-4590 CPU@ 3.30 GHz with 8 GB RAM. The index KHT and LIR-tree are disk-resident and the buffer size is set to 4 MB.

7.2. Overall Performance of Algorithms. We proceed to offer the global insights about the index structures and algorithms. We illustrate the construct time for grid index structure and backbone network index (BNI) structure over real road networks in Figure 11. We observe that the construction time for two index structures does not vary much for moderate road networks, such as NA, LA and NY. As for the large network, i.e., CA, much more time is spent for building BNI than that for building grid index. The reason behind that is BNI exploits much more complicated partition strategy compared with the grid index.

As shown in Figure 12, we evaluate the overall performance of algorithms on several real road networks, namely SF, NA, LA, NY and CA. Experimental results in Figure 12 reveal that our algorithms (both the range query algorithm and k NN query algorithm) outperform competitors by several orders of magnitude in terms of number of page access and running time. The strength of our algorithms are obvious, especially on very large road network, such as CA.

7.3. Effect of Parameters

7.3.1. Effect of Parameter AK. We first study the effect of AK. In particular, we generate six synthetic data sets based on NA by varying AK from 3 to 8. Figure 13 shows experimental results as AK increases. As expected, the running time for range query algorithms increases when AK increases (see Figure 13(a)). This is because more qualified vertices are to be verified. However, AK has little effect on the number of page access(see Figure 13(b)), since it mainly affected by the search range. We notice some new findings from k NN query algorithms, as shown in Figures 13(c) and 13(d). Both the running time and the number of page access achieve the worst performance when $AK = 5$. Recall that, we set the

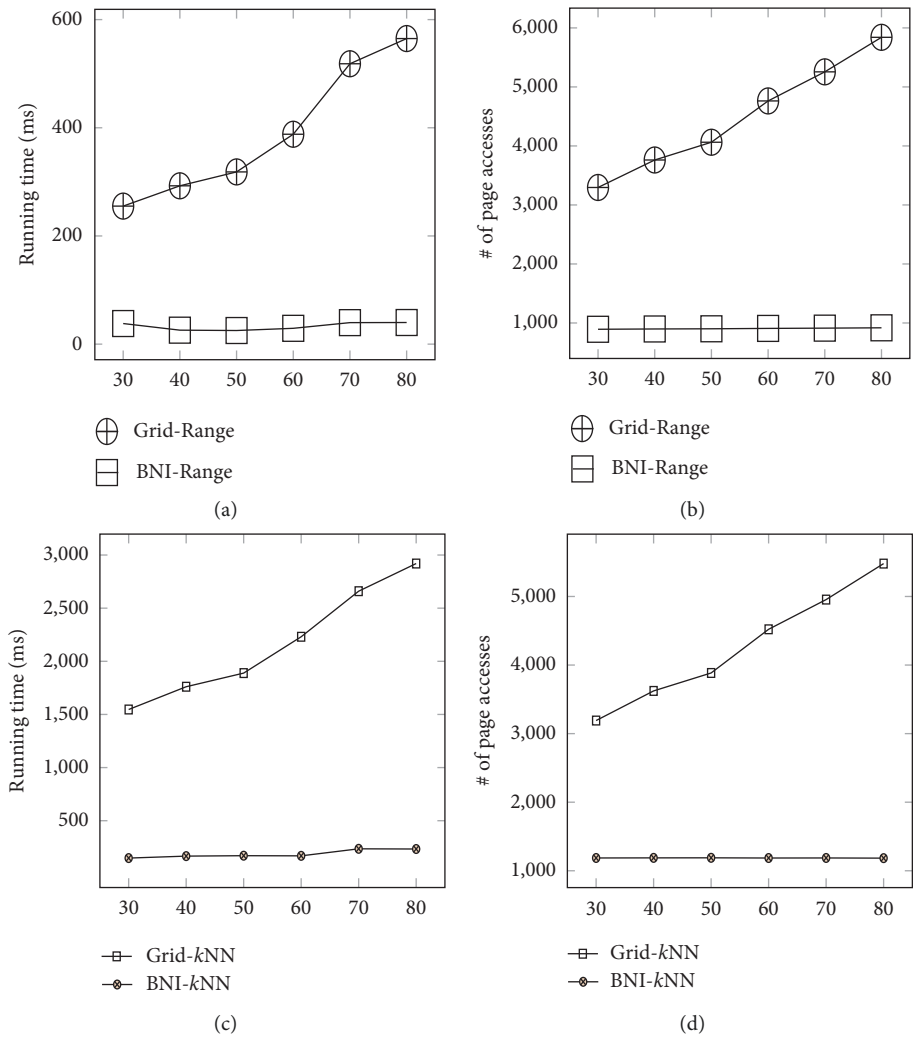


Figure 14: Effect of parameter PL.

number of query keywords as 5 by default. That is, the false positive rate achieves the highest value in that case. Thus, much more cost is spent for checking whether or not the current visited vertex satisfies the query condition for competitors. On the other hand, BNI-k NN performs well, which thanks to the global insights provides by our hierarchical index structure.

7.3.2. Effect of Parameter PL. In this experiment, we evaluate the effect of the length of the query path, i.e., the number of vertices on P^B. Figures 14(a) and 14(b) show the results for range query algorithms. When the length of query path increases, the number of corresponding backbone vertices on P^B increases accordingly. To answer the queries, much more iterations are required for building the answer sets for these backbone vertices. This leads to the increase of running time and the number of page access for Grid-Range. With the maintained answer sets, BNI-Range can retrieve answer sets for backbone vertices incrementally, rather than retrieving from scratch as Grid-Range. This explains the difference in performance. Figures 14(c) and 14(d) show the same findings, and thus can be explained with the same reason.

7.3.3. Effect of Parameter QK. In Figure 15, we present the experimental results achieved by varying QK. Figures 15(a) and 15(b) reveal that the running time decreases for both Grid-Range and BNI-Range When QK increases, we know that the number of qualified vertices decreases within the query range, which incurs less verification overhead. Figure 15(b) shows that QK has little effect on the number of page access for both algorithms. In contrast, we notice the running time and the number of page access increases greatly for Grid-k NN. This is not as strange as it seems. Note that, Grid-k NN requests for k qualified vertices. When QK becomes large, it is much more difficult for Grid-k NN to obtain the answer set. That is, Grid-k NN has to explore a larger proportion of network before finding the k NN results. We omit the experimental results when QK is greater than 5, because the answer set cannot be found within

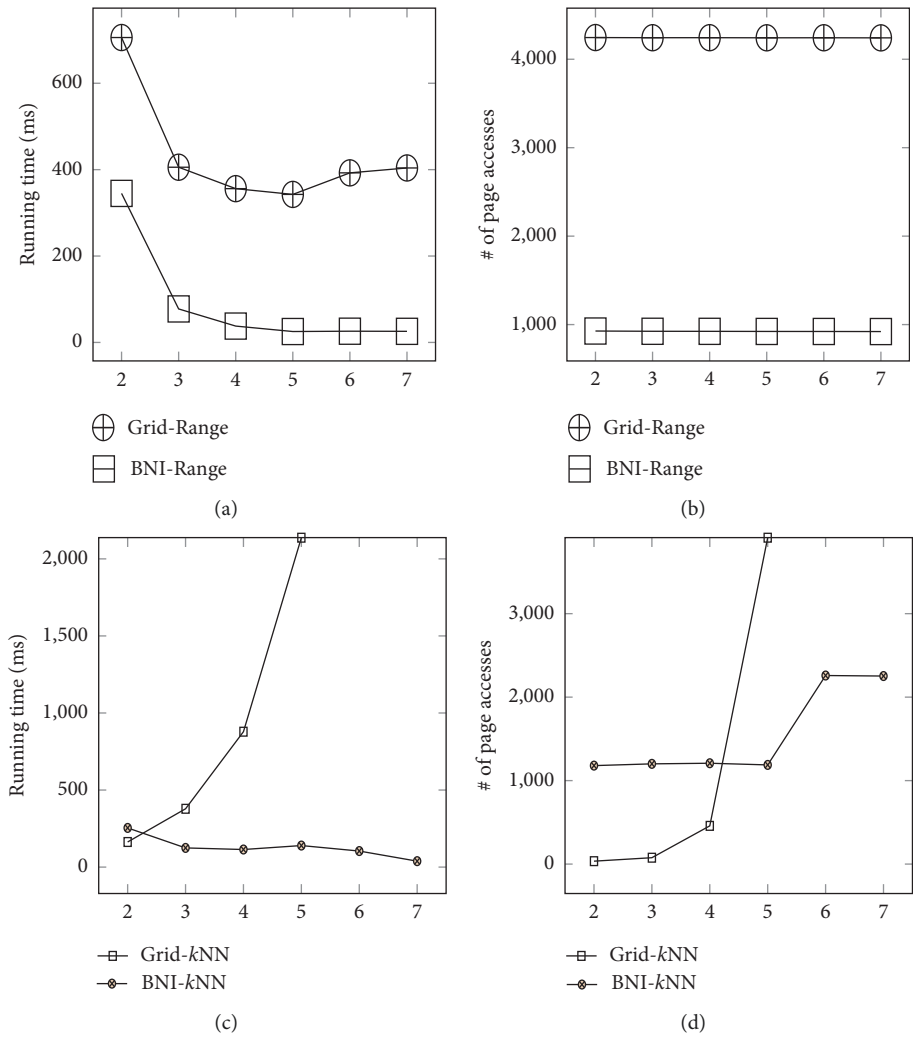


Figure 15: Effect of parameter QK.

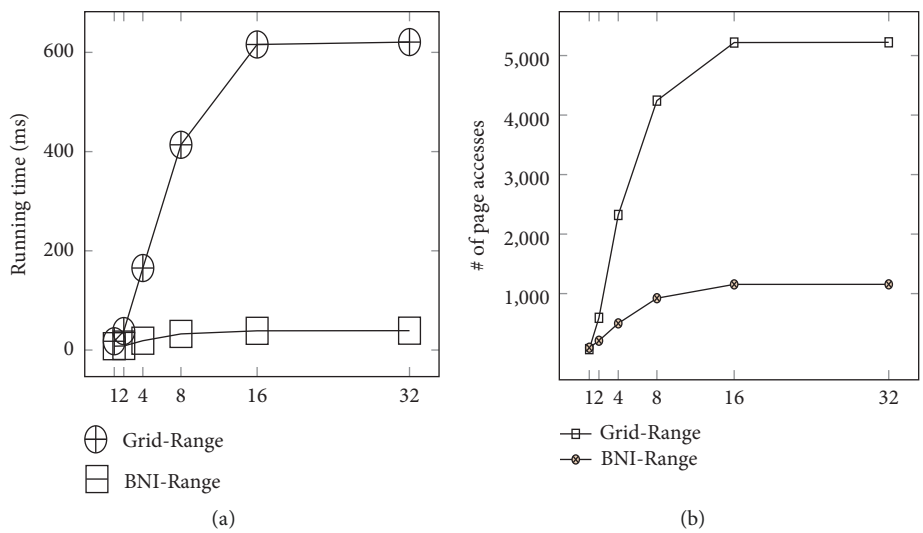


Figure 16: Effect of parameter RT.

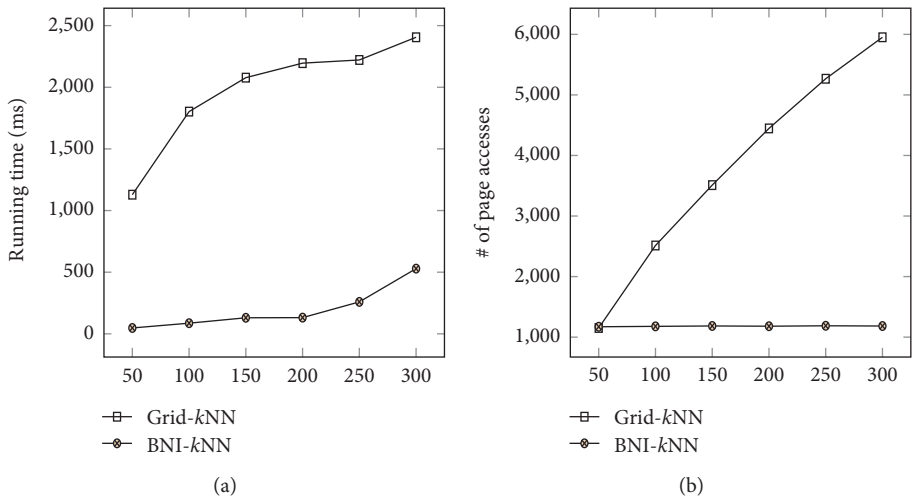


Figure 17: Effect of parameter K.

one day. On the other hand, BNI-k NN is affected slightly by QK. This is because we utilize the hierarchical structure to guide the search process.

7.3.4. Effect of Parameter RT. To study the effect of distance threshold on range query algorithms, we tune RT from 1% to 32%. To set RT, we first get the whole length of the road network, denoted by $\mathscr{L}_W$, and then RT can be set by some specific ratio. For instance, when RT is set to 8%, the corresponding real value is $0.08 * \mathscr{L}_W$. Figures 16(a) and 16(b) show the running time and the number of page access for range query algorithms respectively. As expected, both the running time and the number of page access increase as RT increases. This is because the range query algorithms would explore a larger proportion of the networks. BNI-Range achieves much better performance than Grid-Range, which is largely due to the effective of our proposed index structure.

7.3.5. Effect of Parameter K. As shown in Figures 17(a) and 17(b), we observe that K has much effect on k NN query algorithms. Clearly, when K increases, that is, much more vertices are required for satisfying the query. This renders the algorithms to explore much more vertices before meeting the query conditions. Figure 17(b) shows that the number of page access of Grid-k NN increases almost liner to K, whereas BNI-k NN is affected by K slightly, this demonstrates the effectiveness of our proposed index structure.

8. Conclusions and Future Work

In this paper, we introduce a novel query type, called path based continuous spatial keyword query. Different from previous studies, in this paper, we study the spatial keyword queries on a given query path. With this model, we propose to study the range queries and k NN queries. We propose a progressive framework to address these queries. Besides, to support our queries efficiently, we propose a backbone network index structure. This dual-index takes the advantage of G-tree and adjacent list. As verified by the experiments, our algorithms outperform the competitors by several orders of magnitude.

There are several interesting directions to be studied in the future. First, we can study the path based continuous spatial keyword query in a dynamic environment with the moving or mobile objects. Then, it is also interesting to study the problem in the distributed environment.

Data Availability

The five real road networks data used to support this study, which have been cited. And all these data are available at: https://www.cs.utah.edu/~lifeifei/SpatialDataset.htm.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work is supported by National Natural Science Foundation of China (Grant no. 62002216), the Key Disciplines of Computer Science and Technology of Shanghai Polytechnic University, the Construction of Electronic Information Master Degree of Shanghai Polytechnic University, the Construction of University Enterprise Cooperation Automotive Electronics Engineering Technology Center (No. A11NH190704), the Cultural Relic Protection Science and Technology project of Zhejiang Province (No. 2018007), the Shanghai Sailing Program (No. 20YF1414400), the Shanghai Pujiang Program (No.18PJ1433400), the Collaborative Innovation Platform of Electronic Information Master of Shanghai Polytechnic University (No.A10-GY21F015), the National Natural Science Foundation of China (No.61103213), the Innovation Program of Shanghai Municipal Education Commission (No.14ZZ167), and the Open Foundation of Big Data Management System Engineering Research Center.

References

[1] C. Li, Y. Gu, J. Qi, G. Yu, R. Zhang, and W. Yi, "Processing moving k nn queries using influential neighbor sets," *Proceedings of the VLDB Endowment*, vol. 8, no. 2, pp. 113–124, 2014.

[2] K. Mouratidis, D. Papadias, and M. Hadjieleftheriou, "Conceptual partitioning: an efficient method for continuous nearest neighbor monitoring," in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pp. 634–645, ACM, Baltimore, MD, USA, June 2005.

[3] K. Mouratidis, M. L. Yiu, D. Papadias, and N. Mamoulis, "Continuous nearest neighbor monitoring in road networks," in *Proceedings of the 32nd international conference on Very large data bases*, vol. 43–54, VLDB Endowment, Seoul, Korea, September 2006.

[4] S. Nutanong, R. Zhang, E. Tanin, and L. Kulik, "The V*-Diagram," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 1095–1106, 2008.

[5] Y. Wu, R. Jin, and X. Zhang, "Fast and unified local search for random walk based k-nearest-neighbor query in large graphs," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pp. 1139–1150, ACM, Snowbird, UT, USA, June 2014.

[6] D. Wu, M. L. Yiu, C. S. Jensen, and G. Cong, "Efficient continuously moving top-k spatial keyword query processing," in *Proceedings of the Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pp. 541–552, IEEE, Washington, DC, USA, April 2011.

[7] X. Yu, K. Q. Pu, and N. Koudas, "Monitoring k-nearest neighbor queries over moving objects," in *Proceedings of the Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, pp. 631–642, IEEE, Washington, DC, USA, April 2005.

[8] Z. Yu, Y. Liu, X. Yu, and K. Q. Pu, "Scalable distributed processing of k nearest neighbor queries over moving objects," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 5, pp. 1383–1396, 2015.

[9] B. Zheng, K. Zheng, X. Xiao et al., "Keyword-aware continuous knn query on road networks," in *Proceedings of the Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*, pp. 871–882, IEEE, Helsinki, Finland, May 2016.

[10] K. Zheng, G. Trajcevski, X. Zhou, and P. Scheuermann, "Probabilistic range queries for uncertain trajectories on road networks," in *Proceedings of the 14th International Conference on Extending Database Technology*, pp. 283–294, ACM, Uppsala, Sweden, March 2011.

[11] C. Li, Y. Gu, J. Qi, J. He, Q. Deng, and G. Yu, "A gpu accelerated update efficient index for knn queries in road networks," in *Proceedings of the IEEE International Conference on Data Engineering*, Paris, France, April 2018.

[12] C. Zhang, K. Cheng, L. Zhu, R. Chen, Z. Zhang, and F. Huang, "Efficient continuous top-*k* geo-image search on road network," *Multimedia Tools and Applications*, vol. 78, 2018.

[13] T. Guo, X. Cao, and G. Cong, "Efficient algorithms for answering the m-closest keywords query," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pp. 405–418, ACM, Victoria, Australia, May 2015.

[14] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick, "Reachability and distance queries via 2-hop labels," *SIAM Journal on Computing*, vol. 32, no. 5, pp. 1338–1355, 2003.

[15] H. Hu, D. L. Lee, and V. Lee, "Distance indexing on road networks," in *Proceedings of the 32nd international conference on Very large data bases*, pp. 894–905, VLDB Endowment, Seoul, Korea, September 2006.

[16] K. C. K. Lee, W.-C. Lee, B. Zheng, and Y. Tian, "ROAD: a new spatial object search framework for road networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 3, pp. 547–560, 2012.

[17] H. Samet, J. Sankaranarayanan, and H. Alborzi, "Scalable network distance browsing in spatial databases," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 43–54, ACM, Vancouver, BC, Canada, June 2008.

[18] R. Zhong, G. Li, K.-L. Tan, L. Zhou, and Z. Gong, "G-tree: an efficient and scalable index for spatial search on road networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 8, pp. 2175–2189, 2015.

[19] S. Shekhar and D.-R. L. Duen-Ren Liu, "CCAM: a connectivity-clustered access method for networks and network computations," *IEEE Transactions on Knowledge and Data Engineering*, vol. 9, no. 1, pp. 102–119, 1997.

[20] H.-J. Cho and C.-W. Chung, "An efficient and scalable approach to cnn queries in a road network," in *Proceedings of the 31st international conference on Very large data bases*, pp. 865–876, VLDB Endowment, Trondheim, Norway, August 2005.

[21] F. Chen, P. Zhang, H. Lin, and S. Tang, "Continuous path-based range keyword queries on road networks," in *Proceedings of the 2019 IEEE International Conference on Big Knowledge, ICBK 2019*, pp. 42–49, IEEE, Beijing, China, November 2019.

[22] M. A. Cheema, L. Brankovic, X. Lin, W. Zhang, and W. Wang, "Continuous monitoring of distance-based range queries," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 8, pp. 1182–1199, 2011.

[23] X. Du, Y. Li, Q. Huang, and L. Shu, "Search continuous spatial keyword range queries over moving objects in road networks," *Journal of Computational Information Systems*, vol. 11, no. 2, pp. 759–767, 2015.

[24] H. Wang and R. Zimmermann, "Processing of continuous location-based range queries on moving objects in road networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 7, pp. 1065–1078, 2011.

[25] R. Zhang, H. V. Jagadish, B. T. Dai, and K. Ramamohanarao, "Optimized algorithms for predictive range and knn queries on moving objects," *Information Systems*, vol. 35, no. 8, pp. 911–932, 2010.

[26] Z.-J. Zhi-Jie Wang, D.-H. Dong-Hua Wang, B. Bin Yao, and M. Minyi Guo, "Probabilistic range query over uncertain moving objects in constrained two-dimensional space," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 3, pp. 866–879, 2015.

[27] K. L. Kun-Lung Wu, S. K. Shyh-Kwei Chen, and P. S. Yu, "Incremental processing of continual range queries over moving objects," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 11, pp. 1560–1575, 2006.

[28] B. S. E. Chung, W. C. Lee, and A. L. P. Chen, "Processing Probabilistic Spatio-Temporal Range Queries over Moving Objects with Uncertainty," in *Proceedings of the EDBT '09: Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, pp. 1399-1400, Saint-Petersburg, Russia, March 2009.

[29] A. Okabe, B. Boots, and K. Sugihara, "Spatial tessellations. concepts and applications of voronoi diagrams," *The College Mathematics Journal*, vol. 43, 1992.

[30] R. Jin, G. Chen, A. K. H. Tung, L. Shou, and Y. Gu, "Dim: a distributed air index based on mapreduce for spatial query processing in road networks," *EURASIP Journal on Wireless Communications and Networking*, vol. 2018, no. 1, 2018.

[31] G. Cong, H. Lu, B. C. Ooi, D. Zhang, and M. Zhang, "Efficient spatial keyword search in trajectory databases," 2012, https://arxiv.org/abs/1205.2880.

[32] Z. Feng, T. Liu, H. Li, H. Lu, L. Shou, and J. Xu, "Indoor top-k keyword-aware routing query," in *Proceedings of the 36th IEEE International Conference on Data Engineering, ICDE 2020*, pp. 1213–1224, IEEE, Dallas, TX, USA, April 2020.

[33] R. Geisberger, P. Sanders, D. Schultes, and D. Delling, "Contraction hierarchies: faster and simpler hierarchical routing in road networks," in *Proceedings of the International Workshop on Experimental and Efficient Algorithms*, pp. 319–333, Springer, Provincetown, MA, USA, May 2008.

[34] L. Zhu, W. Yu, C. Zhang, Z. Zhang, and H. Yu, "Efficient spatial visual similarity join for geo-multimedia," *IEEE Access*, vol. 7, no. 99, p. 1, 2019.

[35] M. Jiang, A. W.-C. Fu, and R. C.-W. Wong, "Exact top-k nearest keyword search in large networks," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pp. 393–404, ACM, Victoria, Australia, May 2015.

[36] Y. Gao, X. Qin, B. Zheng, and G. Chen, "Efficient reverse top-k boolean spatial keyword queries on road networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 5, pp. 1205–1218, 2015.

[37] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S. H. Teng, *On Trip Planning Queries in Spatial Databases*, Springer, Berlin, Germany, 2005.