Keyword-Aware Continuous *k*NN Query on Road Networks

Bolong Zheng[1], Kai Zheng[1,*], Xiaokui Xiao[2], Han Su[3], Hongzhi Yin[1], Xiaofang Zhou[1], Guohui Li[4]
[1]The University of Queensland, Australia
{b.zheng,h.yin1}@uq.edu.au, {kevinz,zxf}@itee.uq.edu.au
[2]Nanyang Technological University, Singapore
xkxiao@ntu.edu.sg
[3]University of Southern California, USA
Suan.sue@gmail.com
[4]Huazhong University of Science and Technology, China
guohuili@hust.edu.cn

***Abstract*—It is nowadays quite common for road networks to have textual contents on the vertices, which describe auxiliary information (e.g., business, traffic, etc.) associated with the vertex. In such road networks, which are modelled as weighted undirected graphs, each vertex is associated with one or more keywords, and each edge is assigned with a weight, which can be its physical length or travelling time. In this paper, we study the problem of *keyword-aware continuous k nearest neighbour* (𝕂C*k*NN) search on road networks, which computes the *k* nearest vertices that contain the query keywords issued by a moving object and maintains the results continuously as the object is moving on the road network. Reducing the query processing costs in terms of computation and communication has attracted considerable attention in the database community with interesting techniques proposed. This paper proposes a framework, called a Labelling AppRoach for Continuous *k*NN query (*LARC*), on road networks to cope with 𝕂C*k*NN query efficiently. First we build a pivot-based reverse label index and a keyword-based pivot tree index to improve the efficiency of *keyword-aware k nearest neighbour* (𝕂*k*NN) search by avoiding massive network traversals and sequential probe of keywords. To reduce the frequency of unnecessary result updates, we develop the concepts of dominance interval and region on road network, which share the similar intuition with safe region for processing continuous queries in Euclidean space but are more complicated and thus require more dedicated design. For high frequency keywords, we resolve the dominance interval when the query results changed. In addition, a path-based dominance updating approach is proposed to compute the dominance region efficiently when the query keywords are of low frequency. We conduct extensive experiments by comparing our algorithms with the state-of-the-art methods on real data sets. The empirical observations have verified the superiority of our proposed solution in all aspects of index size, communication cost and computation time.**

I. Introduction

With the rapid development of GPS-enabled smart mobile devices and location-based services, there is a clear trend that objects are increasingly being geo-tagged. To provide better user experience, these services maintain location-related information to answer user queries w.r.t. user-specified location. In addition to the spatial characteristics, a user may also have specific requirement on the description of the object such as "restaurant", "hotel", "petrol station", etc. For example, a person wants to find a restaurant within 10 minutes walking distance. Such queries, known as spatial keyword queries, which find the top-*k* objects of interest in terms of both spatial proximity and textual relevance to the query, have been extensively studied in recent years [6][13][15][20][21][25][26][27]. All these studies have focused on static query objects whose locations are fixed throughout the query lifetime. However, many real-world applications have the requirements to support the continuous *k* nearest neighbour (C*k*NN) queries, or also known as moving *k* nearest neighbour queries. For instance a Uber service provider looking for potential passengers is driving on the road, the *k* nearest potential passengers that have requested taxis should be reported to him through the application continuously. The C*k*NN query can also be used to report the *k* nearest petrol stations continuously while a car is running low of fuel.

Existing techniques for the static *k*NN query is not directly applicable for the C*k*NN query. Therefore, on-going efforts have been made to meliorate the user experience by improving the C*k*NN query processing efficiency [10][14][17][18][23]. Most of these works adopt the idea of "safe region" where all the inside points share the same *k*NN results, thus reducing the query processing cost in terms of both computation and communication. However, they assume objects are moving in free space, which might be inappropriate especially in urban areas where the movements of objects are constrained by the road network. Consider the example in Fig. 1; the current location of the query object is at v_6 and the query keyword is "b", thus v_3 that contains "b" would be the 1NN result in terms of Euclidean distance. However, v_5 is actually what we want instead of v_3 in terms of the network distance. Meanwhile, [9][23] study the problem of continuous top-*k* spatial keyword query on road networks by incorporating the spatial proximity and textual relevance to form a similarity function. However, this kind of similarity functions, which simply combine two unrelated dimensions together, usually cannot satisfy user's search intention well. As seen in more and more commercial location-based services, a more intuitive and practical query formulation for spatial keyword search is to find the objects that simply contain the query keywords. Motivated by these requirements and oversights of existing works, we study the

*Corresponding author: Kai Zheng

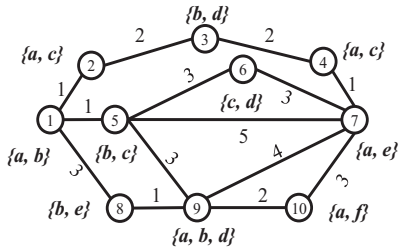


Fig. 1: Running example

keyword-aware continuous k nearest neighbour (IKC*k*NN) on road networks, which computes the *k*NN results that contain the query keywords and maintains the results in a continuous manner.

In order to process the IKC*k*NN query efficiently, we need to overcome several challenges. The first challenge is concerned with computing the network distances between vertices, as well as finding the *keyword-aware k nearest neighbour* (IK*k*NN) results efficiently. Unlike the Euclidean space, processing distance queries in scalable networks is a complicated problem. Given such a query and a network, an obvious solution is to apply Dijkstra's search [8] from the query location to hit the target vertices or find the *k*NN results that contain the keyword. However, this method becomes inefficient when dealing with large-scale road networks due to massive traversals. Inspired by the 2-hop label, which answers distance queries with small response time [1][2][3], we reorganize the structure of label index and build *pivot-based reverse label index* to fit our IK*k*NN search problem. For the keyword checking, we utilize a probabilistic data structure, i.e., the bloom filter [4], to skip sequential probe of all keywords. By combining this with the label index, we construct *keyword-based pivot tree*, by which means both the distance query and IK*k*NN query can be efficiently processed.

The second challenge is related to deriving a dominance interval or region as large as possible. Although some existing works adopt safe region technique to reduce the query processing cost [14][18], they fall short either in the region construction overhead or validation overhead. In addition, the safe region in Euclidean space is just surrounded by several bisect lines, and for each object pair, there exists only one bisect line. Nevertheless, in a road network, the dominance region is determined by bisect points, and each object pair may have several bisect points since they may be connected by multiple paths, which makes the construction of dominance region even more complex and time consuming. Moreover, the area of the dominance region is highly dependent on the frequency of the query keyword. Therefore, for high frequency keywords, we adopt a *window sliding approach* to build a dominance interval with low costs. For low frequency keywords, we propose a *path-based dominance updating approach* to resolve the dominance region on road network, which guarantees the validity of the current IK*k*NN results and significantly reduces the computation and communication costs.

The major contributions of this paper can be summarized as follows:

- By utilizing the labelling approach, we construct *keyword-based label index* that consists of *pivot-based reverse label* and *keyword-based pivot tree*. With such an index structure, we improve the IK*k*NN query efficiency by skipping the massive network traversals and sequential probe of keywords. For IKC*k*NN query, we propose a *LARC* algorithm to resolve the dominance intervals by a *window sliding approach* for the moving object when dealing with high frequency keywords.
- For low frequency keywords, we also develop a *LARC++* algorithm that employs a *path-based dominance updating approach* to construct an effective dominance region with low costs. This way, the frequency of communication between server and client is thoroughly reduced. In addition, a hybrid algorithm *LARC-C* that combines *LARC* and *LARC++* is introduced to cope with all cases of keyword frequency.
- Our experimental evaluation demonstrates the effectiveness and efficiency of our framework for processing the IKC*k*NN queries on real-world datasets. We show the superiority of our methods in answering IKC*k*NN queries efficiently, when compared with the state-of-the-art methods.

This paper is organized as follows. We give the related works in Section II, and describe the problem statement in Section III. Section IV presents the framework of our solution, i.e., *LARC* and analyses the algorithms in detail. In Section V, we introduce an enhanced algorithm *LARC++* to construct the dominance region. Our empirical observations are explained in Section VI. Section VII concludes this paper.

II. Related Work

In this section, we mainly introduce two aspects of related works and summarise the major characteristics of some existing *k*NN and C*k*NN queries in Table I.

Spatial Keyword Search. Searching geo-textual objects with query location and keywords has gained increasing attention recently due to the popularity of location-based services. In Euclidean space, *IR*^{2}*-tree* [7] integrates signature files and R-tree to answer boolean keyword queries. Also, *IR-tree* [6] is an R-tree augmented with inverted files that supports the ranking of objects based on a score function of spatial distance and text relevancy. On road network or general graph, *ROAD* [13] and *G-tree* [27] address the problem of object search on road network by partitioning the space into sub-graphs. For each subgraph, an object abstract is generated for keyword checking. By incremental network expansion, the subgraphs without intended objects are pruned out. However, its search space pruning is effective only if the objects are widely scattered. *OA-kNN* [21] studies keyword search on road networks in metric space, which combines keyword information and distance information to compute top-*k* answers, while in our work we emphasize on boolean keyword match. *SP-tree* [20] deals with the problem of keyword search on large graphs by introducing a shortest path tree, thus the network distances between results and query are approximated by tree distances. *FBS* [11] adopts 2-hop label for handling the distance query for *k*NN problem, and facilitates KT index to handle the performance issue of frequent keywords.

TABLE I: Comparison of existing kNN and CkNN techniques

Technique	Boolean keyword	Continuous query	Unknown path	Static data objects	Road network	Safe region
ROAD[13], G-tree[27], SP-tree[20], FBS[11]	✔	✖	–	✔	✔	–
OA-kNN[21]	✖	✖	–	✔	✔	–
YPK-CNN[24], CPM[16], GMA[17]	✖	✔	✔	✖	✖	✖
CkNN[22]	✖	✔	✖	✔	✖	✖
UNICONS[5]	✖	✔	✔	✔	✖	✖
V*-Diagram[18], MkSK[23], INS[14]	✖	✔	✔	✔	✖	✔
LARC	✔	✔	✔	✔	✔	✔

Continuous/Moving kNN Search. There are quite a number of studies on CkNN/MkNN queries. *YPK-CNN* [24], *CPM* [16] and *GMA* [17] study the problem of finding nearest moving objects (e.g., taxies) to a location and focus on dealing with frequent updates of moving objects. In our case, we focus on the efficiency of the $\mathbb{K}$CkNN queries on the static data objects (e.g., petrol station), of which the ideas and implementations are totally different. *CkNN* [22] finds the kNN for every single point on a predefined linear trajectory. This is achieved by identifying all influence points on the trajectory. However, it is limited that the query trajectory must be known at query time. *UNICONS* [5] precomputes and stores mNN results for each vertex in road network. For CkNN query, *UNICONS* computes the valid intervals of the query path. However, if k is large the massive network traversals still can not be avoid to obtain kNN results. Moreover, *UNICONS* is poor for handling sparse objects due to frequent recomputation of valid intervals. *V*-Diagram* [18], *MkSK* [23] and *INS* [14] adopt the concept of safe region which maintains a kNN set and an associated "safe region" where the query object can move freely without invalidating this kNN set. *MkSK* studies the problem similar to our work that considers both spatial locations and keywords and maintains a safe zone that guarantee the validity for CkNN query. However, *MkSK* is only limited to Euclidean space and cannot be well deployed for our problem due to the metric that combines spatial distance and text relevance together. Instead of computing a safe region, *INS* uses a small set of influential neighbour objects, which shares the similar functionality with safe region. As long as the current kNN results are closer to the query object than the influential neighbour objects, the current kNN results stay valid and no recomputation is required. In this paper, we use the algorithm *V*-Diagram* as one of the baseline algorithms for performance comparison detailed in Section VI.

III. Problem Statement

This section formally defines the $\mathbb{K}k$NN and $\mathbb{K}$CkNN queries. Table II summarizes the major notations of this paper.

We model a road network as a weighted undirected graph $G = (V, E)$, where V is the set of vertices and E is the set of edges. Each edge $(u, v) \in E$ has a positive weight, i.e., physical length or travelling time, denoted as $l(u, v)$. Each vertex $v \in V$ contains a set of keywords, denoted as $\Phi(v)$. Note that the algorithms proposed in this paper can be easily extended to the case that keywords locate on edges. Given a path between vertices u and v, denoted as $\mathcal{P}(u, \ldots, v)$, the length is the total weight of all edges along the path. For any two vertices u and v, the distance between u and v on G, denoted as $d_G(u, v)$, is the length of the shortest path $\mathcal{SP}(u, v)$ between u and v.

*Definition 1 (**Keyword-aware** k**NN**):* The Keyword-aware k Nearest Neighbour ($\mathbb{K}k$NN) query is defined as follows: Given a road network G, the query Q includes a query location l_q, a query keyword w_q, and a positive integer k, i.e., $Q = (l_q, w_q, k)$. Note that l_q can be either a vertex v_q or a point $\mathtt{p}$ on edge (u, v), denoted as $\mathtt{p_q}(u, v, d_s)$, where d_s is the distance along (u, v) between $\mathtt{p_q}$ and u. The query result consists of k vertices, denoted as $R = \{v_1, v_2, \ldots, v_k\} \subseteq V$, that are nearest to l_q among all vertices in V in terms of network distance, and each of which contains the query keyword w_q, i.e., $\forall v \in R, w_q \in \Phi(v)$.

TABLE II: Summary of notations

Notation	Definition
$G = (V, E)$	Road network with vertex V and edge E
$\mathcal{P}(u, \ldots, v)$	A path from u to v
$\Phi(v)$	The keywords associated with v
$d_G(u, v)$	Network distance between u and v in G
$\mathtt{p}(u, v, d_s)$	A point $\mathtt{p}$ on edge (u, v) with d_s to u
$L(v)$	2-hop label of v
$PR(o)$	Pivot-based reserve label of vertex o
$KP(o)$	Keyword-based pivot tree of vertex o
$\mathcal{DI}(R)$	The dominance interval of R
$\mathcal{DR}(R)$	The dominance region of R

*Definition 2 (**Keyword-aware** CkNN):* Given a road network G, and a moving query $Q = (l_q, w_q, k)$, a Keyword-aware Continuous k Nearest Neighbour ($\mathbb{K}$CkNN) query keeps returning the $\mathbb{K}k$NN results for every new location l_q of Q.

Example 1: As shown in Fig. 1, a $\mathbb{K}$C1NN query Q with keyword "d" is moving from v_1 to v_5. The result of Q reported at v_1 is $\{v_3\}$ and then changes to $\{v_6\}$ at $\mathtt{p}(v_1, v_5, 0.5)$. No update is needed elsewhere since the result does not change.

IV. Algorithm *LARC*

In this section, we introduce our labelling approach for $\mathbb{K}$CkNN query (*LARC*). Different from on-line search algorithms on graphs, e.g., Dijkstra, A* algorithms, etc, our method preprocesses the network G to construct an index structure and organizes the vertices with distance and keyword information to enhance the $\mathbb{K}k$NN search efficiency. With the help of such index, we can skip a large portion of disqualifying vertices, that either are far away from the query location or do not contain the query keyword, by looking up the information stored in the index entries. Similar to existing works on CkNN, we adopt the concept of dominance interval on road network to cope with $\mathbb{K}$CkNN query. As long as the moving object stays on the dominance interval, the $\mathbb{K}k$NN results maintain valid and no recomputation is required, thus both the computation and communication costs are reduced.

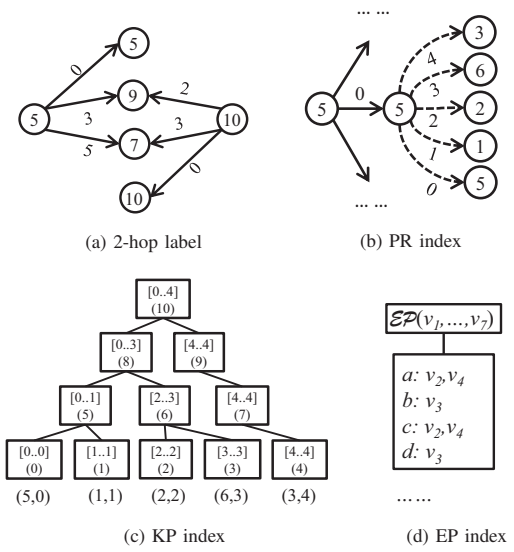


Fig. 2: Overview of keyword-based label index, with illustration of the four components: (i) 2-hop label; (ii) PR index; (iii) KP index; (iv) EP index.

A. Keyword-based Label Index

Inspired by recent development in shortest path computation and distance query over large graphs, we make use of the 2-hop labelling technique [1][2][3][12] as the base to construct our index for $\mathbb{K}$CkNN query processing.

2-hop Label Index. The 2-hop label, also known as 2-hop cover, constructs labels for vertices such that a distance query for any vertex pair u and v can be answered by only looking up the common labels of u and v. For each vertex v, we precompute a label, denoted as $L(v)$, which is a set of label entries and each label entry is a pair $(o, \eta_{v,o})$, where $o \in V$ and $\eta_{v,o} = d_G(v, o)$ is the distance between v and o. We say that o is a **pivot** in label entry if $(o, \eta_{v,o}) \in L(v)$. Given two vertices u and v, we can find a common pivot o that $(o, \eta_{u,o}) \in L(u)$ and $(o, \eta_{v,o}) \in L(v)$:

$$d_G(u, v) = \min\{\eta_{u,o} + \eta_{v,o}\} \quad (1)$$

We say that the pair (u, v) is covered by o and the distance query $d_G(u, v)$ is answered by o with smallest $\eta_{u,o} + \eta_{v,o}$. As shown in Fig. 2a, $L(v_5) = \{(v_5, 0), (v_9, 3), (v_7, 5)\}$ and $L(v_{10}) = \{(v_{10}, 0), (v_7, 3), (v_9, 2)\}$, then we have $d_G(v_5, v_{10}) = 3 + 2 = 5$.

2-hop label has been extensively studied in existing works, which can correctly answer the distance query between any two vertices in a graph, whilst keeping the size of the generated label index as small as possible. The problem of reducing label size is orthogonal to our work. We can fully utilize the state-of-the-art results to build a smaller index in our paper.

Keyword-based Pivot Index. As 2-hop label possesses the nature to process distance queries with fast response time, we modify the structure of original 2-hop label to construct a *pivot-based reverse index*, i.e., *PR* index, for $\mathbb{K}k$NN query processing. In 2-hop label, the distance between any vertex pair (u, v) can be computed correctly through their common pivot o, in other words, each vertex u can reach any other vertex v in graph through a pivot o. Therefore, we store all the label entries $(o, \eta_{v,o}) \in \bigcup_{v \in V} L(v)$ regarding vertex o as pivot into the *PR* label of vertex o, i.e., $(v, \eta_{v,o}) \in PR(o)$. In $PR(o)$, we assume that all the label entries $(v, \eta_{v,o})$ are sorted in non-decreasing order of distance. Generally, given vertex u, we first find its pivot o with smaller distance in $L(u)$, then we continue to search $PR(o)$ for target vertices v containing w_q incrementally until we obtain k results. As shown in Fig. 2b, given a $\mathbb{K}$1NN query $Q = (v_5, \text{“}a\text{”}, 1)$, we first search $PR(v_5)$ since $(v_5, 0) \in L(v_5)$. In sequential probe of $PR(v_5)$, we have “a”$\notin \Phi(v_5)$ and “a”$\in \Phi(v_1)$. Thus v_1 is reported as result.

However, we may still suffer a problem of inefficiency due to the keyword sparsity. Assume that the query keyword w_q is of low frequency, and only shows up in a few vertices, then we have to sequentially check the *PR* labels of many nearby pivots w.r.t. l_q until we reach vertices that contain w_q, which may incur significant traversal overheads. As we know, skipping some vertices that do not contain w_q allows for faster search. [11] proposes a forest index, which is a set of tree structures, to deal with kNN query on large graphs. As road networks are almost planar, the average number of entries in $PR(o)$, i.e., $\frac{|PR(V)|}{|V|}$, is usually small due to the limited number of vertices and edges. Therefore, we propose a *keyword-based pivot index*, i.e., *KP* index, to improve the search efficiency. For each pivot o, we take $PR(o)$ as input and simply construct a binary tree $KP(o)$. Specifically, we preserve each label entry $(v, \eta_{v,o})$ of $PR(o)$ as leaf node of $KP(o)$. For each non-leaf node of $KP(o)$, we denote the index range of leaf nodes that it covers as $[x \ldots y]$ and store the keyword information of all its sub-nodes so that the vertices containing w_q can be retrieved efficiently, as shown in Fig. 2c.

For each keyword w, we utilize a hash function $H(w)$ to generate a binary code for keyword inspection. Hence, the hash code of a vertex or non-leaf node $H(X)$ is the superimposition of $H(w)$ it covers, which is generated by the bitwise operation $\vee$. A non-zero value of $H(w) \wedge H(X)$ indicates that X may contain w. However, the general hash function is unable to control the false positive rate. By contrast, while risking some space, the bloom filter [4] has a strong false positive controlling advantage for membership checking. Therefore, we adopt the bloom filter to compress the keywords instead of a general hash function in the *KP* index.

Enclosed Path Index. Before introducing the *EP* index, we first give a formal definition of the enclosed path,

*Definition 3 (**Enclosed Path**):* We define a vertex whose degree is equal or greater than 3 as an intersection vertex. An enclosed path, denoted as $\mathcal{EP}(s, \ldots, e)$, is a path that only the starting and ending vertices are intersection ones among all vertices it passes. Formally, $\mathcal{P}(s, \ldots, e)$ is enclosed if and only if $s_{.deg} >= 3$, $e_{.deg} >= 3$ and $o_{.deg} < 3, \forall o \in \mathcal{P} \setminus \{s, e\}$.

For each $\mathcal{EP}(s, \ldots, e)$, we construct a keyword posting list for each keyword w contained by $o \in \mathcal{EP} \setminus \{s, e\}$, which is a list of the vertices that contain w, as shown in Fig. 2d. Given $\mathcal{EP}$ and query keyword w_q, the vertices in middle of $\mathcal{EP}$ that contain w can be quickly retrieved.

B. $\mathbb{K}k$NN Query Processing

With the index construction, given a query $Q = \{l_q, w_q, k\}$, we introduce the procedure of query processing for $\mathbb{K}k$NN. At

Algorithm 1: Keyword-aware kNN

Input: $Q = (l_q, w_q, k)$
Output: $\mathbb{K}k$NN results, i.e., $R = \{v_1, v_2, \ldots, v_k\}$
Candidate pivot queue PQ_q;
while $R.size < k$ **do**
  if $PQ_q.size = 0$ **then**
    Find the first pivot o with $H(w) \wedge H(root) \neq 0$;
    $(v, d) = FindNext(w, x, KP(o))$;
    $UpdatePivots()$;
    Find the 1NN result (v, d);
    $R.push(v, d)$;
  else
    for *pivot* o *in* PQ_q **do**
      Obtain the (v, d) with minimum d_{min};
    $FindNext(w, x, KP(o_{min}))$;
    $UpdatePivots()$;
    $R.push(v, d)$;
return R;

the beginning, we compute $H(w_q)$ for keyword matching. In order to retrieve the $\mathbb{K}k$NN results, two steps are considered. First, we need to update the candidate pivots o whose $KP(o)$ may contain target vertices, i.e., function *UpdatePivots*. Then, we incrementally search these $KP(o)$ to obtain the target vertices, i.e., function *FindNext*. For simplicity, we only present the case that l_q is a vertex $v_q \in V$, which can be easily extended to the general cases that l_p locates on edges.

Updating Candidate Pivots. In function *UpdatePivots*, we retain a priority queue PQ_p to restore the candidate pivots o whose $KP(o)$ may contain target vertices. For each $o_i \in PQ_p$, we keep track of a candidate vertex u_i in $KP(o_i)$ that u_i contains w_q and $d_G(v_q, u_i) = \eta_{v_q,o_i} + \eta_{o_i,u_i}$ is only greater than $d_G(v_q, v_f)$ where v_f is the furthest NN result in R obtained so far. Then we determine the minimum distance $\min\{d_G(v_q, u_i)\}$ on the top of PQ_q, and push more pivots o' into PQ_q. Note that only o' with $\eta_{v_q,o'} < \min\{d_G(v_q, u_i)\}$ are considered, since only the vertices in such $KP(o')$ are possibly to become results. After the updating, we pop u_i with $\min\{d_G(v_q, u_i)\}$ on the top of current PQ_q into R, and continue to keep track of next u_i' in $KP(o_i)$ that contains w_q by function *FindNext*. This process is repeated until we obtain k results.

Searching KP Index. We denote the j-th leaf node in $KP(o)$ that contains w_q as (u_j, d_j). In order to obtain the next leaf node u_j in $KP(o)$ later than u_{j-1} that also contains w_q, we use function *FindNext* to search $KP(o)$ in a depth-first manner. The *FindNext* function takes w_q and the index of u_{j-1} in $PR(o)$, say x_{j-1}, as input. The search starts from the root node, and for each iteration, we compute $H(w_q) \wedge H(\sigma)$ where σ is the tree node covering $[x_\sigma \ldots y_\sigma]$. If $H(w_q) \wedge H(\sigma) \neq 0$ and $y_\sigma > x_{j-1}$, we continue to search the sub-nodes of σ. If σ is a leaf node, we examine whether $w_q \in \Phi(\sigma)$. Otherwise, we backtrack its parent node. This process stops when we find the first vertex u_j with index $x_j > x_{j-1}$ that contains w_q.

Example 2: Given a $\mathbb{K}1$NN query $Q = (v_5, \text{“}a\text{”}, 1)$, we first search $KP(v_5)$ since $(v_5, 0) \in L(v_5)$. Then we compute $H(\text{“}a\text{”}) \wedge H(\sigma_{10}) \neq 0$, and $H(\text{“}a\text{”}) \wedge H(\sigma_8) \neq 0$. Finally, we have $H(\text{“}a\text{”}) \wedge H(\sigma_1) \neq 0$ and “a” $\in \Phi(v_1)$. Note that in this example, no pivot is updated. Thus v_1 is reported as result.

Handling Multiple Keywords. For keyword-aware query, we extend the single keyword search condition into multiple keywords. We consider the keyword-aware query in AND semantic, which aims to find the vertices that contain all these query keywords. Given the $\mathbb{K}$CkNN query $Q = (l_q, \Phi_q, k)$, for the keyword containment checking, we have $H(\Phi_q) = \vee_{w_q \in \Phi_q} H(w)$. For the candidate pivot o, we use $H(\Phi_q)$ to search $KP(o)$. The rest search procedure is just the same as single keyword case. Note that, the number of vertices that contain all the query keywords may be much smaller than the single keyword case, therefore there would be more false positives happening when searching the KP tree index.

Algorithm Analysis. We assume that the keywords are evenly distributed in road network. By using the bloom filter, the hash code of keyword w, i.e., $H(w)$, is a bit array of m bits. Given τ hash functions, each of which maps a keyword to one of the m array positions and set it to 1. Assume that the false positive probability is P_f. The average number of leaf nodes in $KP(o)$ is $\frac{|L(V)|}{|V|}$, and the average number of keywords that a vertex contains is $\frac{freq(V)}{|V|}$, so the average number of keywords in $KP(o)$ is $n_w = \frac{L(V) \cdot freq(V)}{|V|^2}$. From [4], we know given n_w and m, the value of τ that minimizes P_f is $\tau = \frac{m}{n_w} \cdot \ln 2 = \frac{m \cdot |V|^2 \cdot \ln 2}{L(V) \cdot freq(V)}$.

We define two cost functions $f(h)$ and $g(h)$ that $f(h)$ is the cost of searching $KP(o)$ that $KP(o)$ contains the query keyword w, and $g(h)$ is the cost of that $KP(o)$ does not contain w. We denote P_h as the probability that a tree node in $KP(o)$ with height h covers at least one query keyword w, and P as the probability that a vertex contain w. Thus, $P_h = 1 - (1-P)^{2^h}$. For $f(h)$, we know that the left subtree is searched with $g(h-1)$ if and only if the left subtree does not contain w and the left subnode is a false positive. Therefore, the probability we search the left subtree with the cost $g(h-1)$ is $P_g = (1 - P_{h-1}) \cdot P_f$; Otherwise, we search by cost $f(h-1)$. For $g(h)$, if the subnodes are false positives, we continue to examine the subtrees. For $h = 0$, we apply a binary search to check the containment of w in leaf node, i.e., $f(0) = g(0) = \ln \frac{freq(V)}{|V|}$. Hence, we have

$$\begin{cases} f(h) = 1 + P_g \cdot g(h-1) + (1 - P_g) \cdot f(h-1) \\ g(h) = 1 + 2P_f \cdot g(h-1) \end{cases} \quad (2)$$

As mentioned before, the P_f is usually small since the bloom filter is able to control the false positives well. Therefore, the Equation 2 can be simplified as $g(h) = O((2P_f)^h \ln \frac{freq(V)}{|V|})$ and $f(h) = O(h + \ln \frac{freq(V)}{|V|})$. The cost of *FindNext* is $O(\ln \frac{|L(V)|}{|V|} + \ln \frac{freq(V)}{|V|})$. Thus, the worst case of *UpdatePivots* is that we access each pivot $o \in L(v_q)$ for k times. Therefore, the time complexity of Algorithm 1 is $O(k \cdot \frac{|L(V)|}{|V|} \cdot \ln \frac{|L(V)| \cdot freq(V)}{|V|^2})$.

C. Dominance Interval for $\mathbb{K}$CkNN

The intuition to process $\mathbb{K}$CkNN query is that we find an interval on road network, as long as the moving object stays on such an interval, the $\mathbb{K}k$NN results maintain valid and no recomputation is required. In this paper, we call such interval as dominance interval w.r.t. its corresponding $\mathbb{K}k$NN results R, i.e., $\mathcal{DI}(R)$.

*Definition 4 (**Dominance Interval**):* Given a vertex v, a point $\mathtt{p}$ is dominated by v, i.e., $v \prec \mathtt{p}$, if and only if v is one

of the $\mathbb{K}k$NN results w.r.t. $\mathbf{p}$. The dominance interval $\mathcal{DI}(R)$ is a path where every point $\mathbf{p}$ locates on $\mathcal{DI}$ is dominated by $v \in R$, i.e., $R \prec \mathcal{DI}(R)$.

Therefore, the major task of $\mathbb{KC}k$NN is to determine such dominance intervals as long as possible in order that the communication cost between server and client ends, as well as the computation cost, are thoroughly reduced.

Lemma 1: Given $\mathcal{EP}(s, \ldots, e)$, let $R_{\mathcal{EP}}$ be the $\mathbb{K}k$NN query results of all points on $\mathcal{EP}$, then we have,

$$R_{\mathcal{EP}} = R_s \cup R_e \cup_{o \in \mathcal{EP} \setminus \{s,e\}} R_o \tag{3}$$

Proof: The proof is straightforward so we omit it here. ■

Given a moving query $Q = (l_q, w_q, k)$, we first identify the $\mathcal{EP}_q$ that l_q locates on. Based on Lemma 1, we compute the $\mathbb{K}k$NN results for both the vertices s and e of $\mathcal{EP}_q$ by Algorithm 1, denoted as R_s and R_e, and also obtain all the vertices on $\mathcal{EP}_q$ that contain w_q by accessing the *EP* index of $\mathcal{EP}_q$, denoted as R_m. Finally, we proceed to divide $\mathcal{EP}_q$ into dominance intervals by a **window sliding approach**. Normally, we will be faced with one of the three possible situations: (1) $R_s = R_e$; (2) $R_s \cap R_e = \phi$; (3) $R_s \cap R_e \neq \phi$.

Case 1. $R_s = R_e$. From Lemma 1, it is obvious to see that $R_m \subseteq R_s(R_e)$, which means all points on $\mathcal{EP}_q$ share the same dominance interval $\mathcal{DI}(R) = \mathcal{EP}_q$, and $R = R_s = R_e$.

Case 2. $R_s \cap R_e = \phi$. We denote the set $R_s^- = R_s \setminus R_s \cap R_m$ and $R_e^- = R_e \setminus R_e \cap R_m$, and construct a result array $\mathcal{A} = \{v_1, \ldots, v_{|\mathcal{A}_{\mathcal{EP}_q}|}\}$ by concatenating R_s^-, R_m and R_e^-. As no duplicate instances exist in $\mathcal{A}$, we have $|\mathcal{A}| = |R_s^-| + |R_m| + |R_e^-|$. Note that $v_i \in R_s^-$ is sorted in descending order of $d_G(v_i, s)$ while $v_j \in R_e^-$ is sorted in ascending order of $d_G(v_j, e)$. Then we employ a sliding window $\mathcal{W}$ with size k to form the $\mathbb{K}k$NN results by covering continuous vertices in $\mathcal{A}$ from v_1 to $v_{|\mathcal{A}|}$

Lemma 2: The adjacent dominance intervals $\mathcal{DI}(R_i)$ and $\mathcal{DI}(R_{i+1})$ are dominated by $R_i = \{v_i, \ldots, v_{i+k-1}\}$ and $R_{i+1} = \{v_{i+1}, \ldots, v_{i+k}\}$, respectively, $v_i \in \mathcal{A}$. The bisect point $\mathbf{p}_i$ between $\mathcal{DI}(R_i)$ and $\mathcal{DI}(R_{i+1})$ is determined by the median point of v_i and v_{i+k} on $\mathcal{EP}_q$.

Proof: Mapping all the NN results into $\mathcal{A}$ deduces this problem to an order-k voronoi diagram on one dimension. Thus, each dominance interval $\mathcal{DI}(R_i)$ is dominated by k continuous results in current $\mathcal{W}$, i.e., $R_i = \mathcal{W}$. When the moving object moves into $\mathcal{DI}(R_{i+1})$ from $\mathcal{DI}(R_i)$, $\mathcal{W}$ pushes v_{i+k} and pops v_i. Therefore, the bisect point $\mathbf{p}_i$ is the point on $\mathcal{EP}_q$ where $d_G(\mathbf{p}_i, v_i) = d_G(\mathbf{p}_i, v_{i+k})$. ■

From Lemma 2 we know the number of dominance intervals is $n = |\mathcal{A}| - k + 1$, and

$$\mathcal{DI}(R_i) = \begin{cases} \mathcal{P}(s, \ldots, \mathbf{p}_i), & i = 1 \\ \mathcal{P}(\mathbf{p}_{i-1}, \ldots, \mathbf{p}_i), & 1 < i < n \\ \mathcal{P}(\mathbf{p}_{i-1}, \ldots, e), & i = n \end{cases} \tag{4}$$

Case 3. $R_s \cap R_e \neq \phi$. We need to deliberate the cases $v \in R_s \cap R_e$. If v locates on $\mathcal{EP}_q$, i.e., $v \in R_m$, we only insert one instance of v into $\mathcal{A}$. Further, if s locates on $\mathcal{SP}(v, e)$ or e locates on $\mathcal{SP}(s, v)$, i.e., $d_G(v, e) = d_G(v, s) + d_G(s, e)$ or $d_G(v, s) = d_G(v, e) + d_G(s, e)$, we also keep one instance of v in R_s^- or R_e^-. For other cases, we retain two duplicate instances of v in $\mathcal{A}$ that one in R_s^- while the other in R_e^-.

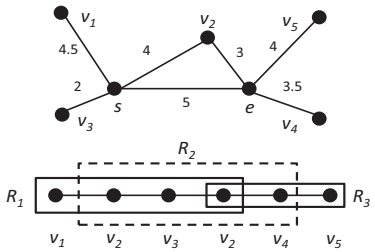


Fig. 3: Dominance interval

Given current sliding window $\mathcal{W}$ with k instances from $\mathcal{A}$, i.e., $\mathcal{W} = \{v_i, \ldots, v_{i+k-1}\}$, if there exists two instances $v_x = v_y$, $v_x, v_y \in \mathcal{W}$ or $v_x \in \mathcal{W}$ is same with v_{i+k}, we extend the sliding window $\mathcal{W}$ to include one more instance but $\mathcal{W}$ still contains k distinct vertices.

Lemma 3: Given sliding window $\mathcal{W} = \{v_i, \ldots, v_{v_{i+k}}\}$ that has been extended to $k+1$ instances. If v_i is the duplicate instance, the bisect point $\mathbf{p}_i$ between $\mathcal{DI}(R_i)$ and $\mathcal{DI}(R_{i+1})$ is the median point between v_{i+1} and v_{i+k+1} on $\mathcal{EP}_q$.

Proof: If v_i is duplicated, and $v_{i+1} = v_i$. Thus, the next sliding window $\mathcal{W}$ pops both v_i and v_{i+1} and pushes v_{i+k+1}. So the number of vertices in $\mathcal{W}$ is still kept k, and the bisect point $\mathbf{p}_i$ is determined by v_{i+1} and v_{i+k+1}. If v_i is duplicated, and $v_{i+1} \neq v_i$. Thus, the next sliding window $\mathcal{W}$ pops both v_i and v_{i+1} and pushes v_{i+k+1} as well. Note that $v_x \in \{v_{i+2}, \ldots, v_{v+k}\}$, $v_x = v_i$ is still covered by $\mathcal{W}$, so the number of vertices in $\mathcal{W}$ is still kept k. Therefore, $\mathbf{p}_i$ is determined by v_{i+1} and v_{i+k+1}. ■

For example in Fig. 3, $R_s = \{v_1, v_2, v_3\}$, $R_e = \{v_2, v_4, v_5\}$ and $\mathcal{A} = \{v_1, v_2, v_3, v_2, v_4, v_5\}$. Obviously, R_1 contains two duplicate instances of v_2, and $\mathbf{p}_1 = \mathbf{p}(s, e, 2)$ is the bisect point between v_1 and v_4 on $\mathcal{P}(s, \ldots, e)$. Thus, the dominance interval $\mathcal{DI}_1 = \mathcal{P}(s, \ldots, \mathbf{p}(s, e, 2))$. Consequently, $R_2 = \{v_2, v_3, v_4\}$. Note that v_2 is the duplicate instance, thus $\mathbf{p}_2 = \mathbf{p}(s, e, 3.5)$ is determined by v_3 and v_5 from Lemma 3. Therefore, $\mathcal{DI}_2 = \mathcal{P}(\mathbf{p}(s, e, 2), \ldots, \mathbf{p}(s, e, 3.5))$.

Following to the resolution of dominance intervals, on condition of that the moving object Q stays in $\mathcal{DI}(R_i)$, R_i is reported as $\mathbb{K}k$NN results and no communication takes place. Once Q moves out of $\mathcal{EP}_q$, a new round of dominance interval computation is issued. The pseudocode can be found in Algorithm 2.

Example 3: As shown in the running example Fig. 1, given $\mathbb{KC}k$NN query $Q = (v_2, \text{“}a\text{”}, 3)$. First we find the enclosed path that v_2 locates on, i.e., $\mathcal{EP}(v_1, \ldots, v_7)$. Then we compute $R_{v_1} = \{v_1, v_2, v_9\}$, $R_{v_7} = \{v_7, v_4, v_2\}$ and $R_m = \{v_2, v_4\}$. Thus we have $\mathcal{A} = \{v_9, v_1, v_2, v_4, v_7\}$. The resolved dominance intervals are $\mathcal{DI}(R_1) = \mathcal{P}(v_1, \ldots, \mathbf{p}(v_1, v_2, 0.5))$ w.r.t. $R_1 = \{v_9, v_1, v_2\}$, $\mathcal{DI}(R_2) = \mathcal{P}(\mathbf{p}(v_1, v_2, 0.5), \ldots, v_3)$ w.r.t. $R_2 = \{v_1, v_2, v_4\}$ and $\mathcal{DI}(R_3) = \mathcal{P}(v_3, \ldots, v_7)$ w.r.t. $R_3 = \{v_2, v_4, v_7\}$.

Algorithm Analysis. Algorithm 2 involves two phases of computation, the first phase is the $\mathbb{K}k$NN query processing and the other is dominance interval resolution.

Algorithm 2: Keyword-aware continuous kNN

Input: $Q = (l_q, w_q, k)$
Output: $R = \{v_1, v_2, \ldots, v_k\}$
Obtain $\mathcal{EP}_q$ and R_m;
Compute two $\mathbb{K}k$NN queries and obtain R_s and R_e;
if $R_s = R_e$ **then**
  $\mathcal{DI} = \mathcal{EP}_q$;
else
  Generate a result array $\mathcal{A}$;
  if $R_s \cap R_e \neq \phi$ **then**
    We have $n = |\mathcal{A}| - k + 1$ dominance intervals;
    Compute the $\{\mathcal{DI}\}$ and $\{R_i\}$;
  else
    Compute the $\{\mathcal{DI}\}$ and $\{R_i\}$ by Lemma 3;
while Q *is on* $\mathcal{EP}_q$ **do**
  Find the dominance interval $\mathcal{DI}(R_i)$ that Q locates on;
  return R_i;

Theorem 1: The expected time complexity of Algorithm 2 is $O(k \cdot \frac{|L(V)|}{|V|} \cdot \ln \frac{|L(V)| \cdot freq(V)}{|V|^2} + \ln|W| + n)$. The expected communication cost is $O(\frac{|E|}{\sum_{e \in E} l_e})$.

Proof: First, we issue two $\mathbb{K}k$NN queries for s and e which takes $O(k \cdot \frac{|L(V)|}{|V|} \cdot \ln \frac{|L(V)| \cdot freq(V)}{|V|^2})$. To obtain the vertices on $\mathcal{EP}_q$ that contain w_q, we apply a binary search to locate the keyword posting list for w_q which takes $O(\ln|W|)$ for the worst case. For the dominance interval resolution, we compute n bisect points which takes $O(n)$. Therefore, the time complexity of Algorithm 2 is $O(k \cdot \frac{|L(V)|}{|V|} \cdot \ln \frac{|L(V)| \cdot freq(V)}{|V|^2} + \ln|W| + n)$. The average length of dominance interval is $O(\frac{\sum_{e \in E} l_e}{|E|})$. As the communication cost is inversely proportional to the average length of dominance interval, thus we have the expected cost $O(\frac{|E|}{\sum_{e \in E} l_e})$. ■

V. Algorithm *LARC++*

Intuitively, the dominance intervals of frequent keywords are usually short so that *LARC* is able to handle the frequent cases well. As a contrast, infrequent keywords always hold a relative large region without the need of recomputation of $\mathbb{K}k$NN results. Accordingly, simply applying *LARC* on these cases may incur unnecessary communication and computation costs due to the limited length of dominance interval. Therefore, we introduce an enhanced algorithm *LARC++* to cope with infrequent cases in this section. Unlike the Euclidean space that the objects are randomly distributed without correlation, in road networks the objects are connected and organized by edges and paths between them due to the network properties. Thus, the *LARC++* adopts a **path-based dominance updating** approach to discover the paths that construct the dominance region by exploiting the properties of paths that connect objects in road networks.

A. Path-based Dominance Updating

Analogous to existing works on CkNN, we adopt the concept of dominance region where no recomputation is demanded on all inside paths, and aim to find such a region as large as possible with low cost.

Concepts and Notations. First, we give a formal definition of dominance region as follows,

*Definition 5 (**Dominance Region**):* The dominance region $\mathcal{DR}$ w.r.t. vertex v, i.e., $v \prec \mathcal{DR}^k(v)$, is a region where every inside point $\mathtt{p}$ is dominated by v. Likewise, given current $\mathbb{K}k$NN results $R = \{v_1, v_2, \ldots, v_k\}$, the dominance region w.r.t. R, i.e., $R \prec \mathcal{DR}^k(R)$, is a region where every inside point $\mathtt{p}$ is dominated by R. Thus, we have:

$$\mathcal{DR}^k(R) = \bigcap_{v \in R} \mathcal{DR}^k(v) \tag{5}$$

From Definition 5, we need to find the $\mathcal{DR}^k(v)$ where v is always one of the $\mathbb{K}k$NN results. However, it is unlikely to compute such a region directly since the $\mathbb{K}k$NN results that contain v may have many different combinations, which makes this problem inapplicable. Fortunately, inspired by the construction of oder-k Voronoi Diagram in Euclidean space, we only need to compute $\mathcal{DR}^1(v)$ where v is the $\mathbb{K}1$NN result, which is similar to the concept of order-1 Voronoi Diagram.

Lemma 4: Given $G = (V, E)$, the current $\mathbb{K}k$NN results $R = \{v_1, \ldots, v_n\}$. The dominance region w.r.t. R is computed as,

$$\mathcal{DR}^k(R) = \bigcap_{v \in R} \mathcal{DR}^1_{(V \setminus R \bigcup v)}(v) \tag{6}$$

Note that $\mathcal{DR}^1_{(V \setminus R \bigcup v)}(v)$ is the dominance region w.r.t. v that constructed in the sub-network $V \setminus R \bigcup v$.

Proof: From [19] in Euclidean space, we know that the order-k Voronoi Diagram w.r.t. R is the intersection of all the order-1 Voronoi Diagrams w.r.t. $v \in R$ that constructed by ignoring the rest results in R. Analogously, $\mathcal{DR}^1_{(V \setminus R \bigcup v)}(v)$ is same as the concept of order-1 Voronoi Diagram while the bisectors are defined by network distance. Obviously, all the points $\mathtt{p} \in \mathcal{DR}^1_{(V \setminus R \bigcup v)}(v)$ are closer to v than $u' \in V \setminus R$ in terms of network distance. Therefore, $\mathcal{DR}^k(R)$ is the intersections of all $\mathcal{DR}^1_{V \setminus R \bigcup v}(v)$. ■

In order to resolve the boundary of dominance region, we are obliged to find a set of vertices $u \in V \setminus R$ containing w_q, which are influential in determining whether the current $\mathbb{K}k$NN result R is valid. Therefore, we introduce the concept of potential neighbour as follows,

*Definition 6 (**Potential Neighbour**):* Given current $\mathbb{K}k$NN results $R = \{v_1, v_2, \ldots, v_k\}$, and a query keyword w_q. We assume that $u \in V \setminus R$ contains w_q. The vertex u is a potential neighbour w.r.t. v_i, i.e., $u \in \mathcal{PN}(v_i)$, if and only if there does not exist a vertex o on $\mathcal{SP}(u, v_i)$ that also contains w_q.

After obtaining the potential neighbour $\mathcal{PN}(R)$, we are able to resolve the dominance status of the vertices between $\mathcal{PN}(R)$ and R. Without loss of generality, we define such vertices inbetween as enrolled vertex as follows,

*Definition 7 (**Enrolled Vertex**):* Given current $\mathbb{K}k$NN results $R = \{v_1, v_2, \ldots, v_k\}$, and a potential neighbour $u \in \mathcal{PN}(R)$. An intersection vertex e is enrolled, $e \in \mathcal{EN}(v)$, if and only if e is on the path between $v_i \in R$ and u, and $d_G(v_i, e) < \max_{u \in PN(R)} d_G(v_i, u)$.

Dominance Region Construction. When a $\mathbb{K}$CkNN query is issued, i.e., $Q = (l_q, w_q, k)$, we compute an initial $\mathbb{K}k$NN set

R w.r.t. the start point l_q by Algorithm 1. For each $v \in R$, we first determine the potential neighbours $\mathcal{PN}(v)$, and in this process, we obtain a set of enrolled vertices in order that we can further validate the dominance status of the paths they locate on. After these paths are resolved, we merge them to form the final $\mathcal{DR}^k(R)$.

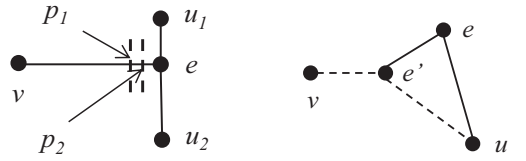
It is worth to notice that the step of determining $\mathcal{PN}(R)$ is critical since it to some extent defines the strength of $\mathcal{DR}^k(R)$. In other words, the more vertices are enrolled in the step, we are more possible to obtain a larger $\mathcal{DR}^k(R)$. However, it is really time consuming and unnecessary to discover all the potential neighbours for each $v \in R$. Instead we only need to find the nearest potential neighbour $u \in \mathcal{PN}(v)$ and $u \notin R$ for each v as tradeoff.

Assume that we obtain a set of enrolled vertices $\mathcal{EN}(R)$, each $e \in \mathcal{EN}(R)$ is sorted by their distances to $v \in R$, in the form of $(v, d_G(e, v))$. Next, we proceed to resolve the dominance status of these enrolled vertices and by which potential neighbour or result vertex they are dominated. Straightforwardly, we can compute the $\mathbb{K}$1NN for all enrolled vertices by Algorithm 1. Thus the dominance of each $e \in \mathcal{EN}(R)$ is easily determined. However, it is quite unoptimized that some $\mathbb{K}$1NN results of $e \in \mathcal{EN}(R)$ are repeatedly computed due to the observation that many enrolled vertices are actually dominated by the same potential neighbour or result vertex. Hence, we propose a *path-based dominance updating* method, i.e., *PathDom* (See Algorithm 3), to resolve the dominance of $e \in \mathcal{EN}(R)$ based on such an observation that if e is dominated by v, then all the other vertices e' on the path $\mathcal{P}(v, \ldots, e)$ are also dominated by v. In other words, we only need to find the furthest enrolled vertex e dominated by v.

We start the dominance status validation process from the enrolled vertex $e \in \mathcal{EN}(R)$ with the maximum value of $d_G(e, v)$, in the sense that more vertices are supposed to be included by a longer path $\mathcal{P}(v, \ldots, e)$. Then Algorithm 1 is applied from e to hit v, in this process, some intermediate enrolled vertices $e' \in \mathcal{EN}(v)$ are traversed with distance $d_G(e, e')$. If $v \in R$ is the first vertex containing w_q reached in this process, $\mathcal{SP}(e, \ldots, v)$ is inserted into $\mathcal{DR}$. Otherwise, if we hit another vertex u that also contains query keyword w_q before we hit v, then we have $u \in \mathcal{PN}(v)$. For all the intermediate vertices e', if $d_G(v, e') + d_G(e, e') = d_G(v, e)$, then e' is on the shortest path $\mathcal{SP}(v, \ldots, e)$, which is regarded as a candidate path for $\mathcal{DR}$. For these $e' \in \mathcal{SP}(v, \ldots, e)$, we store two entries, i.e., $\{(v, d_G(e', v)), (u, d(e', u))\}$ where $d(e', u) = d_G(e, e') + d_G(e, u)$, and remove these e' from $\mathcal{EN}(R)$. Note that $d(e', u)$ may not be the minimum distance between e' and u and requires further updating.

Then we repeat this process and start from $e \in \mathcal{EN}(R)$ with the maximum value of $d_G(e, v)$ in the rest $\mathcal{EN}(R)$. Here we introduce two updating conditions.

Path Updating Condition 1. Consider such a situation, if e hits a potential neighbour u', and encounters an intermediate enrolled vertex e' that is contained by a candidate path $\mathcal{SP}(e_o, \ldots, v)$ and has two entries $\{(v, d_G(e', v)), (u, d(e', u))\}$ w.r.t. v and u. Thus, we split the candidate path $\mathcal{SP}(e_o, \ldots, v)$ into two candidate paths $\mathcal{SP}(e_o, \ldots, e')$ and $\mathcal{SP}(e', \ldots, v)$ for further validation, and keep a new candidate path $\mathcal{SP}(e, \ldots, e')$. Then we compare the two distances $d(e', u)$ and $d(e', u')$.



(a) Updating condition 1 (b) Updating condition 2

Fig. 4: Potential Neighbour

Lemma 5: Given two potential neighbour $u_1, u_2 \in PN(v)$ w.r.t. v, the paths $\mathcal{P}(v, \ldots, e, \ldots, u_1)$ and $\mathcal{P}(v, \ldots, e, \ldots, u_2)$ share a common path $\mathcal{P}(v, \ldots, e)$. As shown in Fig. 4a, if $d_G(e, u_1) < d_G(e, u_2)$ and $d_G(e, u_1) < d_G(e, v)$, thus u_1 governs u_2 w.r.t. the path $\mathcal{P}(v, \ldots, e)$, denoted as $u_1 \curvearrowleft u_2$ on $\mathcal{P}(v, \ldots, e)$, which means the dominance status of $e' \in \mathcal{SP}(e, \ldots v)$ is only determined by v or u_1.

Proof: If $u_1 \curvearrowleft u_2$ on $\mathcal{P}(v, \ldots, e)$, we know that $d_G(e, u_1) < d_G(e, u_2)$ and $d_G(e, u_1) < d_G(e, v)$. For the dominance region between v and u_1, the bisect point $\mathtt{p}_1$ locates on $\mathcal{P}(v, \ldots, e)$ since $d_G(v, \mathtt{p}_1) = (d_G(v, e) + d_G(e, u_1))/2 < d_G(v, e)$. For u_2, the distance between bisect point $\mathtt{p}_2$ and v, i.e., $d_G(v, \mathtt{p}_2) = (d_G(v, e) + d_G(e, u_2))/2 > d_G(v, \mathtt{p}_1)$, therefore we know that $\mathtt{p}_1$ is closer to v than $\mathtt{p}_2$. ■

From Lemma 5, we know that if $d(e', u') < d(e', u)$, then we update the entries into $\{(v, d_G(e', v)), (u', d(e', u'))\}$ since it might be that $u' \curvearrowleft u$ on the path $\mathcal{P}(v, \ldots, e')$. Note that, we also update the entries of all the enrolled vertices on candidate path $\mathcal{SP}(e', \ldots, v)$.

Path Updating Condition 2. Consider another situation, if e hits a potential neighbour u, and encounters an intermediate enrolled vertex e' that already has two entries $\{(v, d_G(e', v)), (u, d(e', u))\}$ w.r.t. v and u, as shown in Fig. 4b. If $d_G(e, e') + d_G(e, u) < d(e', u)$, we know that the path $\mathcal{P}(e', \ldots, e, \ldots, u)$ is a shorter path than that in the previous iterations. Similarly, we update $d(e', u)$ with $d_G(e, e') + d_G(e, u)$ for all e' on the path $\mathcal{P}(v, \ldots, e')$. In addition, if this enrolled vertex e hits a potential neighbour u, and the first intermediate vertex e' that it encounters has two entries $\{(v, d_G(e', v)), (u, d(e', u))\}$ w.r.t. v and u. If $d_G(e, e') + d_G(e, u) > d(e', u)$, we know that the path $\mathcal{P}(e', \ldots, u)$ has already been visited by a shorter path in the previous iterations, thus we do not update the entries of vertices on the path $\mathcal{P}(v, \ldots, e')$. This process terminates when $\mathcal{EN}(R) = \emptyset$.

Lemma 6: When $\mathcal{EN}(R) = \emptyset$, for each enrolled vertex e, in the entries $\{(v, d_G(e, v)), (u, d(e, u))\}$, $d(e, u)$ is the minimum distance $d_G(e, u)$.

Proof: According to our algorithm, all the possible paths between e and u have been traversed, and there must exist a path that is the shortest one. Therefore,we have $d(e, u) = d_G(e, u)$. ■

Finally, the dominance status of each vertex e on candidate paths can be easily determined by comparing $d_G(e, v)$ and $d_G(e, u)$, as well as the dominance intervals on these candidate paths. Then we insert all the dominance intervals into $\mathcal{DR}$.

Algorithm 3: PathDom()

Input: v, $e \in EN(v)$
Output: For each e we have $(v, d_G(e,v)), (u, d_G(e,u))$

```
while EN(v) ≠ ∅ do
    e = EN(v).top;
    if e hits u ∈ PN(v) before v then
        for e′ on P(v, ..., e) do
            if The entry of e′ is ∅ then
                d(e′, u) = d_G(e, e′) + d_G(e, u);
                Store (v, d_G(e′, v)), (u, d(e′, u));
            else if The entry of e′ contains u′ and
            d(e′, u′) < d(e′, u) then
                Update into {(v, d_G(e′, v)), (u′, d(e′, u′))};
            else if The entry of e′ contains u and
            d_G(e, e′) + d_G(e, u) < d(e′, u) then
                Update d(e′, u) = d_G(e, e′) + d_G(e, u);
            else if The entry of e′ contains u and
            d_G(e, e′) + d_G(e, u) > d(e′, u) then
                Continue;
    return all entries;
```

Example 4: As shown in the running example Fig. 1, given $\mathbb{K}$C1NN query $Q = (v_1, \text{“}d\text{”}, 1)$, we first search for the $\mathbb{K}$1NN result of v_1, i.e., v_3. Then we continue to resolve the dominance region $\mathcal{DR}(\{v_3\})$. As the $\mathbb{K}$1NN result is v_6, thus the enrolled vertex set $\mathcal{EN}(\{v_6\}) = \{v_5, v_1, v_7\}$. Then we use v_5 to hit target vertices, and obtain $d_G(v_5, v_6) = 3$. Note that v_1 is traversed in this process, thus v_1 has two entries $\{(v_3, 3), (v_6, 4)\}$. Next we use v_7 to hit targets, and obtain $\{(v_3, 3), (v_6, 3)\}$. Therefore, the dominance region $\mathcal{DR}(\{v_6\}) = \{\mathcal{P}(\mathrm{p}(v_1, v_5, 0.5), \ldots, v_3), \mathcal{P}(v_7, \ldots, v_3)\}$.

B. Combination of LARC and LARC++

Generally, if the query keywords are densely distributed on road network, *LARC* is able to resolve the short dominance intervals efficiently. By contrast, if the query keywords are sparse in road network, *LARC++* is capable of determining the large dominance regions well. However, if we use *LARC* to deal with low frequency keywords or *LARC++* to cope with high frequency keywords, either redundant communication cost or computation cost will be incurred. As *LARC* and *LARC++* are sensitive to different keyword frequencies, we combine these two algorithms to develop a new algorithm *LARC-C* that when the query keywords are of high frequency, we use *LARC*; when the query keywords are of low frequency, then *LARC++* is utilized.

The key point in this combination algorithm is that we designate a threshold that half of the keyword occurrences are handled by *LARC* and the other half by *LARC++*. Motivated by [11], we assume that the keywords of road network are of Zipf's distribution [28]. In the experimental dataset, we use the keyword id *kid* to denote the its rank of frequency. We know that in Zipf's distribution, $freq(w) \propto \frac{1}{kid}$. For $G = (V, E)$, we have $|W|$ keywords and $|freq(W)|$ keyword occurrences. As we know, the sum of occurrences of top n frequent keywords is proportional to the Harmonic number n, i.e., $H_n = \sum_{kid=1}^{n} \frac{1}{kid} = \ln n$. Therefore, we have $H_{|W|} = \ln |W|$ and $\frac{H_{|W|^{1/2}}}{H_{|W|}} = \frac{\ln |W|^{1/2}}{\ln |W|} = 1/2$, which means the top $|W|^{1/2}$ keywords cover half the keyword occurrences. Therefore, in our real dataset of Beijing, we have $|W|^{1/2} = 298$ that we use *LARC* for top 298 keywords and use *LARC++* for the rest keywords.

VI. Experiments

In this section, we conduct extensive experiments on real road network datasets to study the performance of the proposed index structures and algorithms.

A. Experimental Settings

All these algorithms were implemented in GNU C++ on Linux and run on an Intel(R) CPU i7-4770@3.4GHz and 16G RAM.

Datasets. We use two real datasets, the road network datasets of Beijing and New York City from the 9th DIMACS Implementation Challenge[1]. Each dataset contains an undirected weighted graph that represents a part of the road network. Each edge in a graph represents the distance between two endpoints of the edge. We obtain the keywords of vertices from the OpenStreetMap[2]. As shown in Table III, for D1 in Beijing, we have 168,535 vertices and 196,307 edges. We also have 88,910 distinct keywords contained by vertices with the total occurrence 1,445,824. For D2 in New York, we have more vertices and edges than D1 in road network with almost twice the size of D1, the set of keywords contained are larger than D1 as well. For each experiment, we generate 50 $\mathbb{K}$CkNN queries, each of which is a sequence of locations in the form of (u, v, d_s). The query location can be either a vertex or a point locates on an edge.

TABLE III: Statistics of dataset

	Beijing	New York
$\#\|V\|$	168,535	264,346
$\#\|E\|$	196,307	733,846
$\#\|W\|$	88,910	102,450
$\#\|\Phi(V)\|$	1,445,824	3,086,166

Algorithms Evaluated. We introduce two baseline algorithms, Dijk-BF and V*-RN, for comparison. The first baseline algorithm is the brute-force approach (**Dijk-BF**), which computes the kNN results for every location reported by the moving object. If the object locates at vertex v_q, Dijkstra-algorithm is applied that expands from the query location and traverse the other vertices in the best-first way until reaching k vertices with the query keyword w_q. Additionally, if the object locates on edge (u, v), i.e., $l_q = \mathrm{p}(u, v, d_s)$, both the two ends u and v are regarded as the source of Dijkstra search with the initial distances d_s and $l(u, v) - d_s$, respectively. The second baseline algorithm is **V*-RN** that extended from V*-Diagram [18] since V*-Diagram is designed for the Euclidean space only. Generally, V*-RN keeps a safe region to reduce the communication cost. At the first step of retrieving $(k + \delta)$NN results, all the paths accessed by Dijkstra search are saved as the known region. To identify the safe region boundary

[1] http://www.dis.uniroma1.it/challenge9/download.shtml
[2] https://www.openstreetmap.org

of each path in the known region, we compute the network distances for both two endpoints of these paths, and finally obtain the safe region. For our algorithms proposed in this paper, we have exact algorithm *LARC* in Section IV, *LARC++* in Section V and the combination algorithm *LARC-C*. For the construction of label index, we adopt Pruned Landmark Labelling [3] and Hub-based Labelling [1] to generate the 2-hop label. For these four algorithms, we evaluate the CPU computation cost, and the communication cost between server and client. In the experiments, we vary the size of datasets, the length of the query objects, the number of results k, the speed of query object and the frequency of query keyword, to study the effects of these parameters.

Parameters. To evaluate the algorithms under various settings, we vary the value of some parameters. For the speed of object, we vary the report distance from 20 to 100 meters. For the number of the results, we vary the k from 5 to 50. For the length of query object, we vary the length from 200 to 1000. For the query keyword frequency, we vary the frequency from 10^0 to 10^4. We default choose the speed of object as 40, the k as 10, the length as 400, and the keyword frequency as 10^3. The parameters are summarized in Table IV.

TABLE IV: Summary of notations

Parameters	Values
Speed of object	20, **40**, 60, 80, 100
k number of results	5, **10**, 15, 20, 50
Length of query	200, **400**, 600, 800, 1000
Keyword frequency	10^0, 10^1, 10^2, $\mathbf{10^3}$, 10^4

B. Experiment results

Among all the algorithms discussed in this paper, we perform a comparative experimental study on Dijk-BF, V*-RN, *LARC*, *LARC++* and *LARC-C*. The next experiments compare these algorithms using different experimental parameters and study their effects on the performance. Most experiments presented in this subsection are using D1 dataset from Beijing.

Effect of dataset cardinality. In this set of experiments, we vary the datasets to study the effect of data cardinality for Beijing and New York. For the performance study, we compare the CPU time and communication cost of these three algorithms by varying the size of these two datasets. For Beijing dataset, we vary the size of vertices from 40K to 160K. For New York dataset, we vary and the size of vertices from 100K to 250K. As shown in Fig. 5, the CPU time of our proposed algorithm *LARC-C* outperforms these two baseline algorithms Dijk-BF and V*-RN, and the CPU time of each algorithm keeps relative stable with varying the size of datasets. In addition, the communication cost of Dijk-BF is constant since it communicates every time when the query location updates. For V*-RN and *LARC-C*, the communication costs are thoroughly reduced, and *LARC-C* incurs even less communication cost than V*-RN. This confirms the superiority of our proposed algorithm.

Effect of query length. In this set of experiments, we use the query lengths of 200, 400, 600, 800, 1000 to study its effect on these three algorithms. The query length is the number of locations the moving object reported. Intuitively,

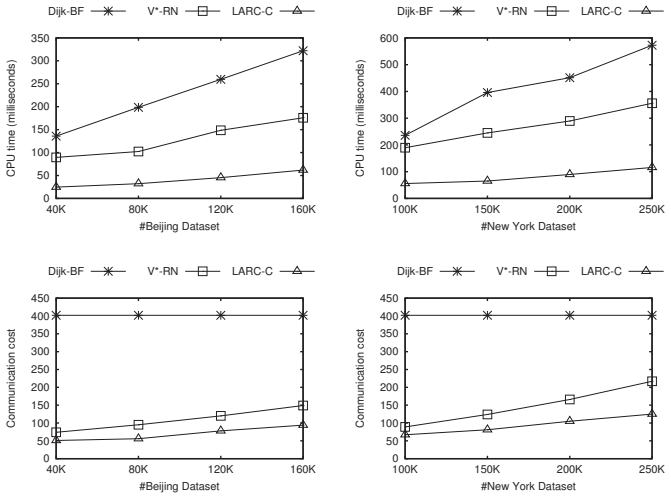


Fig. 5: Effect of dataset cardinality

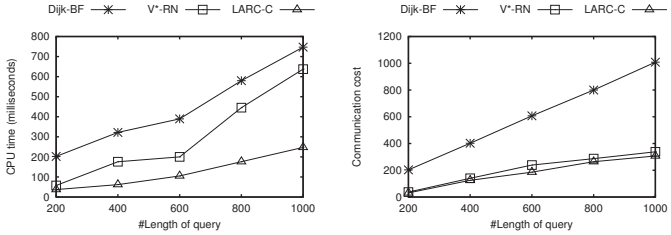


Fig. 6: Effect of query length

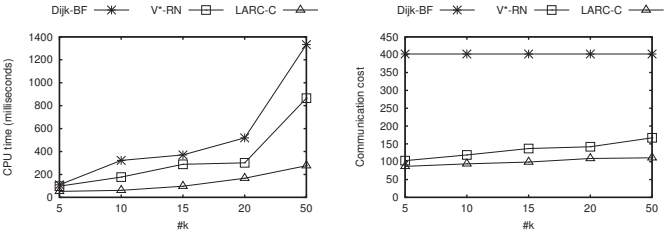


Fig. 7: Effect of k

as the number of locations increases, both the computation and communication costs increase. As shown in Figure 6, for the computation cost, Dijk-BF and V*-RN increase faster than *LARC-C*, since both of them evolve the nearest neighbour search by Dijkstra algorithm and keyword checking for every encountered vertex. For *LARC-C*, the increasing trend of computation cost is slow because they only need to search KP tree to obtain $\mathbb{K}k$NN results and in the meantime construct a dominance region to avoid recomputation of kNN. For the communication cost, as Dijk-BF does not construct the safe region, the communication cost is linear to the query length. For V*-RN, *LARC-C*, both of them adopt the concept of safe region or dominance region. As a result, we can see that the communication costs are highly reduced, and our proposed algorithm *LARC-C* is slightly better than V*-RN.

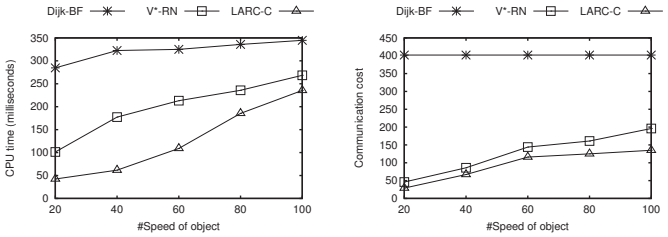


Fig. 8: Effect of speed

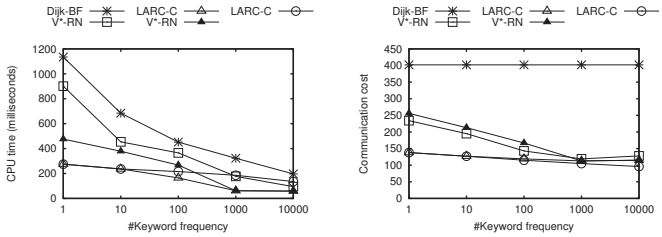


Fig. 9: Effect of keyword frequency

Effect of k. In this set of experiments, we vary the value of k by 5, 10, 15, 20 and 50 to study the effect on these three algorithms. As shown in Figure 7, we can see that our proposed algorithm *LARC-C* well outperforms the baseline algorithms Dijk-BF and V*-RN on both computation and communication costs. As we know that when we enlarge k, more vertices are included for consideration of keyword checking. When the value of k is small, say 5, we can see that the computation costs of these algorithms are close. However, the computation costs of Dijk-BF and V*-RN are increasing much faster than *LARC-C* due to the massive network traversals with enlarging k. For the communication cost, Dijk-BF gains a constant value since it reports the $\mathbb{K}k$NN results for every location. If the query length does not change, the communication cost is kept the same. For V*-RN, *LARC-C*, we have a similar observation to previous set of experiments that our proposed algorithm *LARC-C* is slightly better than V*-RN. Note that, even the communication cost of V*-RN is low, the computation overhead is incurred in the construction of safe region. This explains that V*-RN has a low communication cost but still has a high computation cost. As a result, this confirms the superiority of our proposed algorithm.

Effect of query object speed. In this set of experiments, we vary the speed of query object by 20, 40, 60, 80, 100 to study the effect on these three algorithms. The speed of query object is determined by the distance between two reported locations of query objects. Therefore, we vary this distance to simulate the speed of moving object. As shown in Figure 8, we can see the similar pattern that our proposed algorithm *LARC-C* well outperforms the baseline algorithms Dijk-BF and V*-RN on both computation and communication costs. For the computation cost, Dijk-BF has a steady trend because the speed of query object does not have an obvious effect on it. For V*-RN, *LARC-C*, if the distance is small, the query object is more possible to stay in the safe region or dominance region, thus we do not have more recomputation for $\mathbb{K}k$NN and region construction. If the distance is large, the query object is more possible to move out of the safe region or dominance region, thus the recomputation of $\mathbb{K}k$NN and region reconstruction are incurred more often. Therefore, the computation costs increase in all these three algorithms when the distance is enlarged. For the communication cost, Dijk-BF keeps the same cost, and our proposed algorithm *LARC-C* are slightly better than V*-RN.

Effect of keyword frequency. In this set of experiments, we vary the query keyword frequency by 10^0, 10^1, 10^2, 10^3, 10^4 to study the effect on these five algorithms. For each keyword frequency, we compute an average value by selecting some keywords with close frequencies to it. As we can see in Figure 9, when the keyword frequency is low, Dijk-BF and V*-RN have a bad computation performance, since they have to traverse large portion of the road network to obtain $\mathbb{K}k$NN results, and this process incurs large computation overheads on network traversal and keyword checking. We can see that that our proposed algorithms *LARC-C* well outperforms the baseline algorithms Dijk-BF and V*-RN because it adopts the $FindNext$ method to search the KP tree. For the communication cost, Dijk-BF gains a constant value just as the same to previous experiments. For V*-RN, *LARC-C*, both of them have a decrease in the communication cost when we enlarge the keyword frequency. We can also see that our proposed algorithms are slightly better than V*-RN. For *LARC*, *LARC++* and *LARC-C*, *LARC* outperforms *LARC++* when the query keywords are of high frequency in terms of CPU cost. But *LARC++* is slightly better than *LARC* in terms of communication cost, because the dominance region determined by *LARC++* is always larger than *LARC*.

Effect of m bits. In this set of experiments, we vary the number of m bits in hash code by 500,1000,1500,2000,2500 to study the effect on index size and false positive rate. As we can see in Figure 10, the index size increases and the false positive rate decreases when we enlarge the value of m. This is because when the value of m is large, more space will be used to construct the index structure especially the KP tree, and keywords are less possible to share a same hash code.

Effect of multiple keywords. In this set of experiments, we vary the number of query keywords by 1, 2, 3, 4, 5 to study the effect on these three algorithms. As we can see in Figure 11, the experiment results have a similar pattern that our proposed algorithm *LARC-C* well outperform these two baseline algorithms in terms of both the CPU cost and communication cost. Because when the number of query keywords increases, the number of target vertices decreases. Therefore, *LARC-C* has a better performance than Dijk-BF and V*-RN.

VII. Conclusions

In this paper, we study the problem of efficiently processing $\mathbb{KC}k$NN query on road networks with low computation and communication costs. By utilizing the 2-hop label technique on road networks, we modify the original index structure and co*LARC++*nstruct a keyword-based label index. Based on such index, we first introduce the $\mathbb{K}k$NN query processing, and then propose two efficient algorithms *LARC* and for processing the $\mathbb{KC}k$NN on road networks. For *LARC*, we introduce a *window sliding approach* to build a dominance interval to deal with

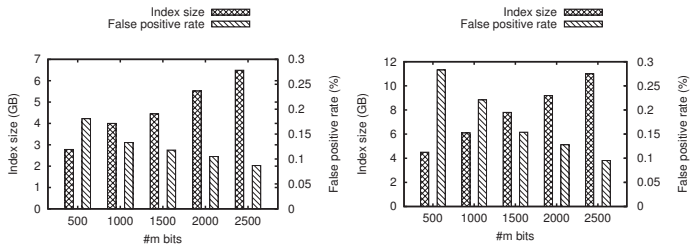


Fig. 10: Effect of m

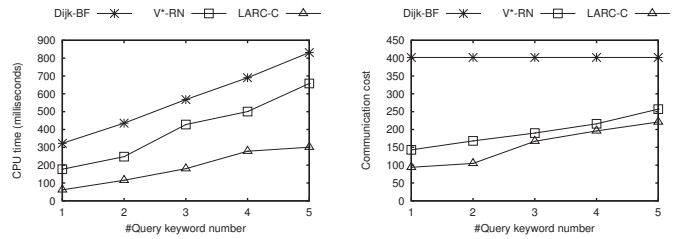


Fig. 11: Effect of multiple keywords

low frequency keywords. For *LARC++*, we propose a *path-based dominance updating approach* to resolve a dominance region for high frequency keywords. Our experimental evaluation demonstrates the effectiveness and efficiency of our solution for processing the $\mathbb{K}$CkNN queries on large real-world datasets, which outperforms the state-of-the-art method with almost 50% decrease on computation cost and almost 20% decrease on communication cost.

Acknowledgement

We wish to thank the anonymous reviewers for the insightful comments and suggestions that have improved the paper. This work is partially supported by the Natural Science Foundation of China (Grant No.61472263, Grant No.61502324 and Grant No. 61432018), the ARC grants DP120102829, DP140103171 and DE140100215.

References

[1] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. A hub-based labeling algorithm for shortest paths in road networks. In *SEA*, pages 230–241. Springer, 2011.

[2] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. Hierarchical hub labelings for shortest paths. In *ESA*, pages 24–35. Springer, 2012.

[3] T. Akiba, Y. Iwata, and Y. Yoshida. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *SIGMOD*, pages 349–360. ACM, 2013.

[4] B. Chazelle, J. Kilian, R. Rubinfeld, and A. Tal. The bloomier filter: an efficient data structure for static support lookup tables. In *SODA*, pages 30–39. SIAM, 2004.

[5] H.-J. Cho and C.-W. Chung. An efficient and scalable approach to cnn queries in a road network. In *VLDB*, pages 865–876. VLDB Endowment, 2005.

[6] G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. *PVLDB*, 2009.

[7] I. De Felipe, V. Hristidis, and N. Rishe. Keyword search on spatial databases. In *ICDE*, pages 656–665. IEEE, 2008.

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

[9] L. Guo, J. Shao, H. H. Aung, and K.-L. Tan. Efficient continuous top-k spatial keyword queries on road networks. *GeoInformatica*, 19(1):29–60, 2015.

[10] W. Huang, G. Li, K.-L. Tan, and J. Feng. Efficient safe-region construction for moving top-k spatial keyword queries. In *CIKM*, pages 932–941. ACM, 2012.

[11] M. Jiang, A. W.-C. Fu, and R. C.-W. Wong. Exact top-k nearest keyword search in large networks. In *SIGMOD*, pages 393–404. ACM, 2015.

[12] M. Jiang, A. W.-C. Fu, R. C.-W. Wong, and Y. Xu. Hop doubling label indexing for point-to-point distance querying on scale-free networks. *PVLDB*, 7(12):1203–1214, 2014.

[13] K. C. Lee, W.-C. Lee, and B. Zheng. Fast object search on road networks. In *EDBT*, pages 1018–1029. ACM, 2009.

[14] C. Li, Y. Gu, J. Qi, G. Yu, R. Zhang, and W. Yi. Processing moving k nn queries using influential neighbor sets. *PVLDB*, 8(2):113–124, 2014.

[15] G. Li, J. Feng, and J. Xu. Desks: Direction-aware spatial keyword search. In *ICDE*, 2012.

[16] K. Mouratidis, D. Papadias, and M. Hadjieleftheriou. Conceptual partitioning: an efficient method for continuous nearest neighbor monitoring. In *SIGMOD*, pages 634–645. ACM, 2005.

[17] K. Mouratidis, M. L. Yiu, D. Papadias, and N. Mamoulis. Continuous nearest neighbor monitoring in road networks. In *PVLDB*, pages 43–54. VLDB Endowment, 2006.

[18] S. Nutanong, R. Zhang, E. Tanin, and L. Kulik. The v*-diagram: a query-dependent approach to moving knn queries. *PVLDB*, 1(1):1095–1106, 2008.

[19] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. *Spatial tessellations: concepts and applications of Voronoi diagrams*, volume 501. John Wiley & Sons, 2009.

[20] M. Qiao, L. Qin, H. Cheng, J. X. Yu, and W. Tian. Top-k nearest keyword search on large graphs. *PVLDB*, 6(10):901–912, 2013.

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

[22] Y. Tao, D. Papadias, and Q. Shen. Continuous nearest neighbor search. In *VLDB*, pages 287–298. VLDB Endowment, 2002.

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

[24] X. Yu, K. Q. Pu, and N. Koudas. Monitoring k-nearest neighbor queries over moving objects. In *ICDE*, pages 631–642. IEEE, 2005.

[25] B. Zheng, N. J. Yuan, K. Zheng, X. Xie, S. Sadiq, and X. Zhou. Approximate keyword search in semantic trajectory database. In *ICDE*, pages 975–986. IEEE, 2015.

[26] K. Zheng, H. Su, B. Zheng, S. Shang, J. Xu, J. Liu, and X. Zhou. Interactive top-k spatial keyword queries. In *ICDE*, pages 423–434. IEEE, 2015.

[27] R. Zhong, G. Li, K.-L. Tan, and L. Zhou. G-tree: an efficient index for knn search on road networks. In *CIKM*, pages 39–48. ACM, 2013.

[28] G. K. Zipf. Human behavior and the principle of least effort. 1949.