Distributed Spatial Keyword Querying on Road Networks

Siqiang Luo[§] Yifeng Luo[§] Shuigeng Zhou[§] Gao Cong[†] Jihong Guan[‡] Zheng Yong[§]

[§] Shanghai Key Lab of Intelligent Information Processing, Fudan University, China
[†] School of Computer Engineering, Nanyang Technological University, Singapore
[‡] Department of Computer Science & Technology, Tongji University, China

{sqluo, luoyf, sgzhou, 09300240046}@fudan.edu.cn
gaocong@ntu.edu.sg, jhguan@tongji.edu.cn

ABSTRACT

Spatial-keyword queries on road networks are receiving increasing attention with the prominence of location-based services. There is a growing need to handle queries on road networks in distributed environments because a large network is typically distributed over multiple machines and it will improve query throughput. However, all the existing work on spatial keyword queries is based on a centralized setting. In this paper, we develop a distributed solution to answering spatial keyword queries on road networks. Example queries include "find locations near a supermarket and a hospital," and "find Chinese restaurants within 500 meters from my current location." We define an operation for answering such queries and reduce the problem of answering a query into computing a function of such operations. We propose a new distributed index that enables each machine to independently evaluate the operation on its network fragment in a distributed setting. We theoretically prove the space optimality of the proposed index technique. We conduct experiments with a distributed setting. Experimental results demonstrate the promising performance of our method.

Keywords

Distributed query processing, Spatial keyword query, Road network, Shortest path

1. INTRODUCTION

Keyword search on road networks is an indispensable function for many applications such as Google Earth and Yahoo! Maps. This motivates the extensive study of spatial keyword queries (e.g., [3, 9, 24, 2]). Most of the existing work on spatial keyword queries is based on the assumption that spatial objects are located in a Euclidean space and keywords (labels or categories) are associated with the objects. A representative type of spatial-keyword query is to find a set of objects that are within a specified spatial range while close (measured by Euclidean distances) to the query location. Although Euclidean distance assumption is made in most of the studies on spatial keyword querying, the Euclidean distance can be a bad approximation of the road network distance between two locations. For instance, around a lake, the shortest road network distance between two locations on the opposite banks of the lake will be very different from the Euclidean distance of the two locations. However, it is more challenging to compute the road network distance for the spatial-keyword queries. Very few proposals consider a spatial network [20] for spatial-keyword queries.

It is common to find distributed real-life network datasets, such as Facebook, Google Maps and Yahoo! Maps, that are stored at data centers, which typically host a cluster of machines. In this case, it is imperative to develop distributed techniques to support spatial-keyword queries. Moreover, many applications (e.g., a web-oriented architecture supporting spatial-keyword querying) may need to handle heavy query load and large road networks. To handle this, it is natural to develop distributed techniques to support spatial-keyword queries on road networks to improve the throughput of query processing. However, all the existing proposals on spatial-keyword querying focus on the centralized environment. None of the techniques in the existing work fits for a distributed setting on road networks. Actually it has been a challenging problem to develop distributed techniques for querying networks even if keywords are not considered, which often exhibits poor locality and incurs expensive network communication cost [17] (See Section 2.3).

In this paper, we aim to develop distributed techniques to handle a class of spatial-keyword queries on road networks, which consists of two types of queries, namely *spatial group keyword query* (SGKQ), and *Range Keyword Query* (RKQ). We next illustrate them with three example queries.

Q1 A real estate agent wants to locate sites that are close (e.g., within 1km) to daily facilities such as a supermarket, a gym and a hospital;

Q2 An investor wants to open a new pizza shop in a shopping mall that must be at least 1km far away from any of the existing pizza shops;

Q3 A tourist wants to find a restaurant offering both seafood and Chinese food within 500 meters from his hotel.

The first two queries are SGKQs and Q3 is an RKQ. SGKQ is to find locations that are close to or far away from locations each containing a query keyword. RKQ is to find

10.5441/002/edbt.2014.23

locations that are within a range of query location and contain the query keywords. We observe that a spatial keyword query typically contains a set of query keywords and we need to find the locations that contain the keywords or are close to (or far away from) locations containing all/some of the keywords. For instance, in Q1, the keywords are *supermarket*, *gym* and *hospital*.

One challenge of distributedly processing such queries is that nodes containing keywords can be in different fragments from a result node. Take Q1 as an example, we need to compute the network distance between a result node and nodes containing any of the three query keywords, which may be distributed in different fragments/machines. This will incur many rounds of communications among machines.

To this end, we develop a two-step framework and a new indexing technique. The two-step framework is based on an operation, called *keyword coverage*, that extracts the set of nodes within certain distance from a specific keyword. In the first step, we compute *keyword coverage* for each query keyword. For example, in Q1, we find the set of locations that are within 1km from any node containing keyword *supermarket*; we also do this for keywords *gym* and *hospital*, respectively. The second step is the aggregation on keyword coverages. For example, for Q1, the results can be obtained by the intersection operation on the keyword coverages of all keywords in Q1. The other example aggregation operators include *union* and *subtraction*. Hence, we reduce the problem of answering a query into the evaluation of a function of *keyword coverage* operations. It is easy to evaluate the second step distributedly. However, it is challenging to distributedly process the first step. To address the challenge, we develop a new distributed index technique for each fragment of a road network. The distributed index enables to perform the two-step framework **in a distributed manner without incurring communication between machines** for answering the two types of spatial-keyword queries. We also prove that the size of the index is optimal.

The main contributions of this work are listed as follows:

- We consider two types of spatial-keyword queries on road networks. We define the *keyword coverage* operation, and reduce the problem of answering these queries into evaluating a function of such operations. We design a novel distributed index to enable distributed query processing for such queries. The index enables us to eliminate the communication between machines. This is a very attractive feature for distributed graph query processing. We also prove the space optimality of the proposed distributed index.
- We propose a solution to answering querying utilizing the proposed index. The proposed methods can be implemented on a general coordinator-based share-nothing distributed computing platform.
- We analyze the time complexity and the load balance of our approach.
- We conduct experiments on a distributed coordinator-based cluster to demonstrate the superior efficiency of our method.

The rest of this paper is organized as follows. Section 2 describes the problem definition. Section 3 and Section 4 introduce the index-based approach, which is the core part of this paper. Section 5 discusses the extension of the approach. Section 6 is the performance evaluation. Section 7 surveys the related work. Section 8 concludes the paper.

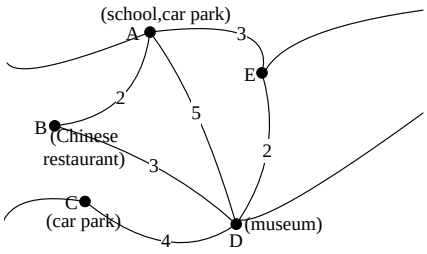


Figure 1: Road network illustration.

2. PROBLEM STATEMENT

2.1 Road Networks

We consider a road network graph, denoted as $\mathcal{G}$, which is an edge-weighted undirected graph. Our method can be easily adapted for the directed graph. There are two types of nodes in $\mathcal{G}$. One type represents road junctions and the other represents objects (*e.g.*, points of interest). For the latter type of nodes, each of them is associated with text description. Edges represent the road segments. The weight of an edge (A, B) represents the length of the path from node A to node B, and (A, B, d) refers to edge (A, B) with weight d. A road network segment is illustrated in Fig. 1. The words in the brackets near the nodes are keywords. Nodes A, B, C and D correspond to objects and each contains some keywords; Node E is a junction node and does not contain any keyword. The formal definition of a road network is as follows.

DEFINITION 1 (ROAD NETWORK). *A road network $\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathcal{W}, \mathcal{K}, \mathcal{L})$ is a weighted graph, where $\mathcal{V}$, $\mathcal{E}$ are the node set and edge set, respectively; $\mathcal{W}$ is a mapping from $\mathcal{E}$ to real values, denoting the edge weights; $\mathcal{K}$ is a vocabulary of keywords; $\mathcal{L}$ is a mapping from a node in $\mathcal{V}$ to a set of keywords (or an empty set) in $\mathcal{K}$.*

The *length* of a path is the total weight of all the edges on the path. The *shortest path* between two nodes A and B, denoted as $A \leftrightsquigarrow B$, is the path with the minimum length between A and B. The shortest path between node A and node set S, denoted as $A \leftrightsquigarrow S$, is defined as the path with the minimum length among all the shortest paths, each corresponding to a shortest path from A to a node in S. The shortest path between node A and keyword ω, denoted by $A \leftrightsquigarrow \omega$ with a bit abuse of notation, is defined as the shortest path between A and the node set that comprises all nodes containing keyword ω. The *distance* between two nodes A and B, denoted as $d(A, B)$, is the sum of weights in path $A \leftrightsquigarrow B$. Similarly, the distance between node A and node set S, denoted as $d(A, S)$, is the sum of weights in $A \leftrightsquigarrow S$. The distance between node A and keyword ω, denoted as $d(A, \omega)$, is the sum of weights in $A \leftrightsquigarrow \omega$.

2.2 Problem Definition

We introduce two types of spatial-keyword queries on road networks, namely *spatial group keyword query*, and *Range Keyword Query*, and then define our problem.

DEFINITION 2 (**Spatial Group Keyword Query**). *Given query keywords $\omega_1, \ldots, \omega_k$ and a distance r, for a spatial group keyword query $SGKQ(\omega_1, \ldots, \omega_k, r)$, node A is one of its results iff $d(A, \omega_i) < r$, $\forall 1 \leq i \leq k$.*

The query Q1 in Introduction is an SGKQ. Next, we give another example SGKQ based on Fig. 1, which is to find locations, each of which is within 3km of both a museum and a school.

EXAMPLE 1 (SGKQ: Q4). *In Fig. 1, given keywords, "*museum*" and "*school*", and a radius 3, query SGKQ ({"*museum*", "*school*"}, 3) returns $\{B, E\}$ because B and E are within a distance of 3 to both keyword "*museum*" (node D) and "*school*" (node A).*

We further extend the definition of the SGKQ for a more general form of SGKQ in two ways: 1) The distance between a result and a query keyword is larger than a distance threshold, i.e., $d(A, \omega_i) > r$. An example of the extended form of query is Q2 in Introduction. In Q2, a result should be faraway from pizza shop (at least 1km) and it should contain keyword "shopping mall". 2) The distance between a result and one of query keywords is smaller than a distance threshold, i.e., $d(A, \omega_i) \leq r$, $\exists 1 \leq i \leq k$. An example query is

Q5: "finding the locations that are within 0.5km of either a university or a park."

Next, we define the second type of query we handle in this work, called *Range Keyword Query*.

DEFINITION 3 (**Range Keyword Query**). *Given a query location l, query keywords $\{\omega_1, \ldots, \omega_k\}$ and a distance r, for a range keyword query $RKQ(l, \omega_1, \ldots, \omega_k, r)$, node A is one of its results iff $d(l, A) \leq r$ and A contains ω_i, $\forall 1 \leq i \leq k$.*

The query Q3 in Introduction is a Range Keyword Query, where the query location is "his hotel" and the query keywords are "restaurant, seafood and Chinese food." We next introduce another example based on Fig. 1.

EXAMPLE 2 (RKQ: Q6). *In Fig. 1, an example RKQ query is to find museum of within 4km of location B, where the query location is "B" and the query keyword is "museum," which is denoted by RKQ(B,{"*museum*"},4). It returns node D because node D contains the keyword "*museum*" while the distance from node B to D is less than 4.*

To the best of our knowledge, no study considers SGKQ; The RKQ has been studied in Euclidean distance space in several existing studies, but has not been studied in a road network setting.

Problem Statement Given N partitions of a graph $\mathcal{G}$ distributed in N separate machines, and an SGKQ or RKQ query Q, our problem is to return results for query Q in the distributed setting. Here, we consider a general coordinator-based distributed setting, which comprises N computing resources (*e.g.*, machines). One of the N resources acts as a coordinator C. The query is submitted to the coordinator C. The communication cost of assigning tasks from coordinator to each machine and the cost of returning results from each machine to the coordinator are unavoidable. Apart from this, *we aim to eliminate communication cost between machines* at query time.

2.3 Remark

It is observed in [17] that graph algorithms often exhibit poor locality and hence, may incur prohibitive overhead on network traffic. In the SGKQ or RKQ query, we need to compute the road network distance between two nodes. If the two nodes are not in the same machine (subgraph) or any node in shortest path between the two nodes is not in the same machine as the two nodes, we may need communications between machines. One naive way is to ship relevant subgraphs to the same machine. This, however, incurs expensive communication cost and cannot exploit parallel computation.

The general graph processing engine, Pregel [17] is based on the general bulk synchronous parallel (BSP) model. In each iteration of the BSP execution, Pregel applies a user-defined function on each vertex in parallel. The communications between vertexes are performed with message passing interfaces. There also exist distributed algorithms for specific problems, such as shortest path algorithm. Some implementations divide the Dijkstra algorithm into a number of phases for parallel computation [16] and a recent implementation [23] resorts to graph partitioning and runs the Dijkstra in each partition. However, they also need multiple rounds of communications between machines. The general graph processing engine [17] and the distributed algorithms for the shortest path query [16, 23] are not suitable for answering the SGKQ or RKQ query that have keyword restrictions. Moreover, they still need many rounds of communications.

In contrast, we aim to evaluate the SGKQ or RKQ query in one round with 0 communication cost except for sending results. This is challenging for a graph query. To achieve this goal, we next show how do we reduce the evaluation of our SGKQ or RKQ queries into the evaluation of an operation and propose an index structure to enable each machine to evaluate the operation independently.

3. NPD-INDEX STRUCTURE

We define an operation and present how to reduce query evaluation into the operation evaluation in Section 3.1. Then we present the proposed index structure for evaluating the operation in Sections 3.2–3.4. The index structure enables each machine to be able to **independently** compute the operation and answer queries. We also show that the proposed distributed index is optimal in both size and communication cost in Section 3.5.

3.1 Primitive Operation

We first define an operation called Keyword Coverage. Based on the operation, we define a type of function and show that the function can be used to answer SGKQ and RKQ queries in a distributed setting.

DEFINITION 4 (**Keyword Coverage**). *Given a keyword ω and a radius r, the keyword coverage $R(\omega, r)$ is defined as a node set, such that a node $A \in R(\omega, r)$ iff $d(A, \omega) \leq r$.*

EXAMPLE 3 (KEYWORD COVERAGE). *In Fig. 1, $R($"*school*", 3) $= \{A, B, E\}$ due to $d(A,$"*school*"$) \leq 3$, $d(B,$"*school*"$) \leq 3$ and $d(E,$"*school*"$) \leq 3$. Note that node A is also in $R($"*school*"$, 3)$ as it contains the keyword "*school*".*

With the keyword coverage operation, an SGKQ $(\omega_1, \ldots, \omega_k, r)$ can be answered by evaluating $\bigcap_{i=1}^{k}(R(\omega_i, r))$, i.e., the

intersection of results of keyword coverage operations. The query in example 1 can be computed by $R(\text{“}museum\text{”}, 3) \cap R(\text{“}school\text{”}, 3)$.

The two extended SGKQ given in Section 2.2 can be computed by utilizing the keyword coverage operation. For Query Q2 in Introduction, we can use subtraction operator in evaluating the result to the query, which can be depicted as $R(\text{“}shopping\ mall\text{”},0) - R(\text{“}pizza\ shop\text{”},1\text{km})$. For the other extended SGKQ “finding the locations that are within 0.5km of either a university or a park”, it can be computed by the union operation of two keyword coverage operations, i.e., $R(\text{“}university\text{”},0.5\text{km}) \bigcup R(\text{“}park\text{”},0.5\text{km})$

Next, we show that *Range Keyword Query* can also be evaluated with the keyword coverage operation using Example 2. We treat each node id as a keyword, and then we set the distance radius to 4 for the keyword “B”. For the keyword “*museum*”, we set its distance radius to 0 to make sure a result object contains the keyword. Finally, we can apply the intersection operator on the variables of the D-function to get the final results. Therefore, in this example, RKQ Q6 is evaluated by $R(\text{“}B\text{”}, 4) \cap R(\text{“}museum\text{”}, 0)$.

Let $\mathcal{X}$ denote a keyword coverage set. The aforementioned examples show that we can reduce the evaluation of an SGKQ or RKQ into a function $\mathcal{F}(\mathcal{X}_1, \ldots, \mathcal{X}_k) = \mathcal{X}_1\theta_1, \ldots, \theta_{k-1}\mathcal{X}_k$, where θ_i $(1 \leq i \leq k-1)$ is one of $\{\cup, \cap, -\}$ (*i.e.*, union, intersection, and subtraction). We call the function $\mathcal{F}$ distributable-function (D-function). We next show that D-function $\mathcal{F}$ can be evaluated distributedly with an example.

Example 4. *Let $\mathcal{U}$ be the node set in Fig. 1, which is $\{A, B, C, D, E\}$; $\mathcal{U}_1 = \{A, B\}$ and $\mathcal{U}_2 = \{C, D, E\}$ are two fragments of $\mathcal{U}$. Let $\mathcal{F}(\mathcal{X}_1, \mathcal{X}_2) = \mathcal{X}_1 \cap \mathcal{X}_2$. Given two sets $\mathcal{X}_1 = \{A, B, C, D\}$ and $\mathcal{X}_2 = \{B, C, D, E\}$ and the function $\mathcal{F}(\mathcal{X}_1, \mathcal{X}_2)$, clearly we can compute $\mathcal{F}(\mathcal{X}_1, \mathcal{X}_2) = \{A, B, C, D\} \cap \{B, C, D, E\} = \{B, C, D\}$. On the other hand, we can compute $\mathcal{F}(\mathcal{X}_1, \mathcal{X}_2)$ in a distributed way by $\mathcal{F}(\mathcal{X}_1, \mathcal{X}_2) = \mathcal{F}(\mathcal{X}_1 \cap \mathcal{U}_1, \mathcal{X}_2 \cap \mathcal{U}_1) \cup \mathcal{F}(\mathcal{X}_1 \cap \mathcal{U}_2, \mathcal{X}_2 \cap \mathcal{U}_2) = \mathcal{F}(\{A, B\}, \{B\}) \cup \mathcal{F}(\{C, D\}, \{C, D, E\}) = \{B\} \cup \{C, D\} = \{B, C, D\}$.*

Consider a graph whose node set is $\mathcal{U}$ and its m fragments, each of which has a node set $\mathcal{U}_i$ $(i = 1, ..., m)$. We have the following lemma that the evaluation of D-function $\mathcal{F}$ can be done distributedly.

Lemma 1. *For any $\mathcal{X}_j \subseteq \mathcal{U}$, $1 \leq j \leq t$, we have:* $\mathcal{F}(\mathcal{X}_1, \ldots, \mathcal{X}_t) = \bigcup_{1 \leq i \leq m} \mathcal{F}(\mathcal{X}_1 \cap \mathcal{U}_i, \ldots, \mathcal{X}_t \cap \mathcal{U}_i)$

proof: By Set's generalized distributive law,

$$\mathcal{F}(\mathcal{X}_1 \cap \mathcal{U}_i, \ldots, \mathcal{X}_t \cap \mathcal{U}_i) = (\mathcal{X}_1 \cap \mathcal{U}_i)\theta \ldots \theta(\mathcal{X}_t \cap \mathcal{U}_i)$$
$$= (\mathcal{X}_1\theta_1 \ldots \theta_{t-1}\mathcal{X}_t) \cap \mathcal{U}_i.$$

So

$$\bigcup_{1 \leq i \leq m} \mathcal{F}(\mathcal{X}_1 \cap \mathcal{U}_i, \ldots, \mathcal{X}_t \cap \mathcal{U}_i)$$
$$= \bigcup_{1 \leq i \leq m} ((\mathcal{X}_1\theta_1 \ldots \theta_{t-1}\mathcal{X}_t) \cap \mathcal{U}_i)$$
$$= (\mathcal{X}_1\theta_1 \ldots \theta_{t-1}\mathcal{X}_t) \cap (\bigcup_{1 \leq i \leq m} \mathcal{U}_i) = (\mathcal{X}_1\theta_1 \ldots \theta_{t-1}\mathcal{X}_t) \cap \mathcal{U}$$
$$= \mathcal{X}_1\theta_1 \ldots \theta_{t-1}\mathcal{X}_t = \mathcal{F}(\mathcal{X}_1, \ldots, \mathcal{X}_t) \quad \square$$

In Lemma 1, D-function $\mathcal{F}$ takes t sets $\mathcal{X}_1, \ldots, \mathcal{X}_t$ as variables (t is also a variable). It says that the function $\mathcal{F}$ can be evaluated in each fragment (on a machine). The lemma is based on the assumption that we have an approach to computing the set variables $\mathcal{X}_j$ $(1 \leq j \leq t)$ with respect to each fragment $\mathcal{U}_i$ $(1 \leq i \leq m)$ *i.e.*, $\mathcal{X}_j \cap U_i$ $(1 \leq i \leq m)$. We will study this in the next subsection.

In what follows, we will take SGKQ as the default query to present our method. We present how to extend the method to handle the other types of queries in Section 5.4.

In summary, answering an SGKQ query involves two steps: 1) evaluating the keyword coverage for each query keyword for each machine; 2) intersecting the keyword coverages for each machine. The second step is trivial compared to the first step. Hence, we concentrate on the first step.

3.2 Index Overview

In this subsection, we present indexing techniques that enable each machine to compute the keyword coverage for its fragment **independently**

For distributed algorithms, one way to maximize parallelization is to explore what can be computed locally and reduce the communication cost [18]. Following the principle, we aim to develop techniques such that for each fragment $\mathcal{P}$ we are able to evaluate the distance between any node associated with certain query keyword and any node inside $\mathcal{P}$, **without incurring communication with other machines**. However, this objective is challenging due to the following reasons: to compute the keyword coverage for nodes in a fragment, we need to compute their network distance from the nodes associated with the query keyword, which can be in any fragment distributed in the whole network $\mathcal{G}$; thus it seems unavoidable to communicate with other machines to compute the keyword coverage for a fragment.

To this end, we propose an index, called *N*ode-*P*artition-*D*istance (NPD) index, to store useful distances so that each fragment can **independently** compute the exact distance between any node associated with certain query keyword and any node inside $\mathcal{P}$.

We first define some notations. An edge belongs to a fragment if its end nodes are in that fragment. $part(A)$ denotes the fragment containing node A. If the end nodes of an edge belong to different fragments, then they are *portal nodes*. We denote the set of portal nodes in a fragment $\mathcal{P}$ as $port(\mathcal{P})$. We build an index file for each fragment. We denote the index for a fragment $\mathcal{P}$ as IND($\mathcal{P}$). Fragment $\mathcal{P}$ and its index IND($\mathcal{P}$) are combined to evaluate the keyword coverage on fragment $\mathcal{P}$, for query keywords.

The index IND($\mathcal{P}$) contains two components: SC($\mathcal{P}$) (short for *S*hort*C*ut) and DL($\mathcal{P}$) (short for *D*istance *L*ist). SC($\mathcal{P}$) includes the useful distances between nodes within $\mathcal{P}$, while DL($\mathcal{P}$) includes the useful distances between nodes outside $\mathcal{P}$ and the nodes inside $\mathcal{P}$.

For presentation convenience, we assume the uniqueness of the shortest path between two nodes in this section. We discuss how to remove the assumption in Section 5.3. In the next two subsections, we present the two components of NPD-index.

3.3 SC Component

Before presenting SC, we introduce three notions, namely, *shortcut edge*, *shortcut path* and *complete fragment*. A *shortcut edge* is a manually added edge $(X, Y, d(X, Y))$ that short-

cuts $X \leftrightsquigarrow Y$, where $(X, Y, d(X, Y))$ is not an original edge in $\mathcal{G}$. For example, in Fig. 2, $(C, F, d(C, F))$ is a shortcut edge; it is not an original edge and it shortcuts $C \leftrightsquigarrow F = ((C, D), (D, E), (E, F))$. We say the shortcut edge $(X, Y, d(X, Y))$ corresponds to $X \leftrightsquigarrow Y$.

Given a shortest path $X \leftrightsquigarrow Y$, if we replace some edge-disjoint sub-paths with their corresponding shortcut edges, we get a *shortcut path* of $X \leftrightsquigarrow Y$. For example, in Fig. 2, $((A, C), (C, F))$ is a *shortcut path* of $A \leftrightsquigarrow F$ because it contains a shortcut edge (C, F) that shortcuts a sub-path $C \leftrightsquigarrow F$. We define *complete fragment* as follows.

DEFINITION 5 (COMPLETE FRAGMENT). *A complete fragment of $\mathcal{P}$, denoted by $\mathcal{P}'$, comprises $\mathcal{P}$ and some shortcut edges whose end nodes are in $\mathcal{P}$, such that any shortest path within $\mathcal{P}'$ is either exactly the shortest path in $\mathcal{G}$ or the shortcut path of the shortest path in $\mathcal{G}$.*

Hence, with a complete fragment $\mathcal{P}'$, the distance between every pair of nodes in $\mathcal{P}$ can be computed.

Now, we are ready to present the SC component. SC($\mathcal{P}$) comprises a set of shortcut edges whose end nodes are in $\mathcal{P}$, such that $\mathcal{P}\cup$SC($\mathcal{P}$) will be a complete fragment of $\mathcal{P}$. If all the shortcut edges whose end nodes belonging to $\mathcal{P}$ are put into SC($\mathcal{P}$), $\mathcal{P}\cup$SC($\mathcal{P}$) certainly forms a complete fragment. However, this will result in a large SC($\mathcal{P}$) and many shortcuts are unnecessary to be put into SC($\mathcal{P}$). For example, if every edge of a shortest path is within $\mathcal{P}$, then $\mathcal{P}$ itself is sufficient for computing the distance between the start node and the end node, and hence there is no need to put into SC($\mathcal{P}$) the shortcut edge corresponding to the shortest path. In addition, as will be theoretically analyzed in Section 5.1, minimizing the size of SC($\mathcal{P}$) benefits the time complexity of our method.

We proceed to propose a rule of constructing SC($\mathcal{P}$), Theorem 1 shows that the construction will result in a complete fragment. We also prove that the construction is optimal in term of the size of the generated complete fragments.

RULE 1. *We add a shortcut edge (A, B) with weight $d(A, B)$ to SC($\mathcal{P}$) iff: (1) $A \in \mathcal{P}$ and $B \in \mathcal{P}$; (2) $(A, B, d(A, B))$ is not an edge in the road network graph $\mathcal{G}$; (3) The shortest path from A to B does not contain any other node of $\mathcal{P}$;*

THEOREM 1. *$\mathcal{P}\cup$SC($\mathcal{P}$) is a complete fragment.*

proof: For any two nodes $A \in \mathcal{P}$ and $B \in \mathcal{P}$, the shortest path $A \leftrightsquigarrow B$ is divided into *inner part* and *outer part*, where inner part refers to the sub-paths consisting of edges which belong to $\mathcal{P}$ and outer part refers to the sub-paths consisting of edges that are not in $\mathcal{P}$ (See Fig. 2). According to the condition (3) of Rule 1, any outer part corresponds to (*i.e.*, is shortcut by) a shortcut edge in SC($\mathcal{P}$). In addition, any inner part can be directly accessed within $\mathcal{P}$. It follows that any shortest path within $\mathcal{P}\cup$SC($\mathcal{P}$) is a shortcut path of the shortest path in $\mathcal{G}$. □

To prove the optimality of the construction rule in terms of the size of generated SC($\mathcal{P}$), we introduce the notion of *standard shortcut set*, to depict the difference between an original fragment and its complete fragment.

A standard shortcut set describes a subset $\mathcal{S}$ of the set $\{(X, Y, d(X, Y)) | X \in \mathcal{P}, Y \in \mathcal{P}\}$ such that $\mathcal{P} \cup \mathcal{S}$ is a complete fragment of $\mathcal{P}$. We show that Rule 1 is optimal (in terms of the number of added edges) with Theorem 2, among all the standard shortcut sets.

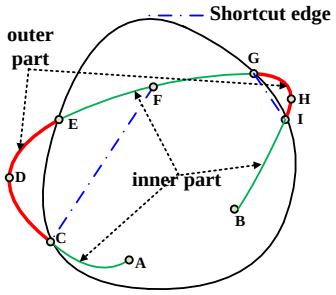


Figure 2: Illustration of inner part and outer part in $A \leftrightsquigarrow B$.

DEFINITION 6 (STANDARD SHORTCUT SET). *A standard shortcut set with respect to fragment $\mathcal{P}$ is a shortcut edge set $\mathcal{S} \subseteq \{(X, Y, d(X, Y)) | X \in \mathcal{P}, Y \in \mathcal{P}\}$, such that $\mathcal{P} \cup \mathcal{S}$ is a complete fragment.*

According to Theorem 1, SC($\mathcal{P}$) is a standard shortcut set. Furthermore, we prove that, among all the standard shortcut sets for $\mathcal{P}$, SC($\mathcal{P}$) is space optimal by Theorem 2.

THEOREM 2. *Among all standard shortcut sets for fragment $\mathcal{P}$, the size of SC($\mathcal{P}$) (constructed following Rule 1) is the minimum.*

proof: For any standard shortcut set $\mathcal{S}$ with respect to $\mathcal{P}$, it is sufficient to show that SC($\mathcal{P}$)$\subseteq \mathcal{S}$. Therefore we need to show that for any $(A, B, d(A, B)) \in$ SC($\mathcal{P}$), $(A, B, d(A, B))$ also belongs to $\mathcal{S}$. We assume by contradiction that $(A, B, d(A, B))$
$\notin \mathcal{S}$. Since $\mathcal{P} \cup \mathcal{S}$ is a complete fragment, the shortest path between A and B in $\mathcal{P} \cup \mathcal{S}$ must be $A \leftrightsquigarrow B$ or one of its shortcut path. Furthermore, $(A, B, d(A, B)) \in$SC($\mathcal{P}$) $\Rightarrow$ $(A, B, d(A, B)) \notin \mathcal{P} \Rightarrow (A, B, d(A, B)) \notin \mathcal{P} \cup S$ (the last step is due to the assumption). Then, the shortest path between A and B in $\mathcal{P} \cup S$ contains at least one internal node, which belongs to $\mathcal{P}$. On the other hand, the internal nodes in $A \leftrightsquigarrow B$ are all outside of $\mathcal{P}$, causing a contradiction to the fact that the shortest path between A and B in $\mathcal{P} \cup \mathcal{S}$ must be $A \leftrightsquigarrow B$ or one of its shortcut path. Hence, the assumption is incorrect and follows the theorem. □

3.4 DL Component

The component SC($\mathcal{P}$) is used to compute the distance between two nodes within fragment $\mathcal{P}$. In contrast, DL($\mathcal{P}$) guarantees the correct evaluation of the distance between any node $A \notin \mathcal{P}$ and any node $B \in \mathcal{P}$. Intuitively, $A \leftrightsquigarrow B$ must intersect $\mathcal{P}$ by a portal node. Hence an straightforward idea is to store all the distances between any node outside of $\mathcal{P}$ and every node in *port*($\mathcal{P}$). However, this approach shall include many unnecessary distances. For instance, in Fig. 3, the shortest path from node A to C passes node B, while B and C are both portal nodes of $\mathcal{P}$. In this case, the distance $d(A, C)$ is unnecessary to be recorded if $d(A, B)$ has been recorded, as $d(B, C)$ can be computed with the edges in $\mathcal{P}\cup$SC($\mathcal{P}$). Therefore, we need only to record the length of any shortest path, which does not contain an internal node (on the path) of $\mathcal{P}$.

We organize DL($\mathcal{P}$) into an entry-value form so that it can be indexed by the entry. In particular, for all $A \in \mathcal{P}$, it has the following (entry$\rightarrow$ value) mapping format:
$(A, \mathcal{P}) \rightarrow \{(N_1, d_1), (N_2, d_2), \ldots, (N_s, d_s)\}$.

$(A, \mathcal{P})$ is the index entry. $\{(N_1, d_1), (N_2, d_2), \ldots, (N_s, d_s)\}$ is a list of node-distance pairs where N_i $(1 < i < s)$ are portal nodes of $\mathcal{P}$, and $d_i = d(A, N_i)$. Recall that our objective is to guarantee the correct evaluation of $d(A, B)$ $(\forall A \notin \mathcal{P}, \forall B \in \mathcal{P})$, with $\{(N_1, d_1), (N_2, d_2), \ldots, (N_s, d_s)\}$ and $\mathcal{P} \cup \text{SC}(\mathcal{P})$ (denoted by $\mathcal{P}'$). Note that $A \rightsquigarrow B$ must pass certain portal nodes, it suffices to record only the distance between A and its nearest node in $\mathcal{P}$ along $A \rightsquigarrow B$, as the remaining part can be computed within $\mathcal{P}'$. An example is shown in Fig. 3 where the red curve represents a fragment $\mathcal{P}$. In the figure, only the distances between A and red square nodes are recorded in $\text{DL}(\mathcal{P})$ as they are the first intersection nodes with $\mathcal{P}$ and the shortest paths started from node A. Specifically, $(B, d(A, B))$ is in the list mapped to entry $(A, \mathcal{P})$ while $(C, d(A, C))$ is not. We summarize Rule 2 for constructing $\text{DL}(\mathcal{P})$ as follows:

RULE 2. *For entry* $(A, \mathcal{P})$, $\{(N_1, d_1), (N_2, d_2), \ldots, (N_s, d_s)\}$ *conforms to: (1)* $N_i \in port(\mathcal{P})$, $\forall 1 \leq i \leq s$; *(2) The shortest path from* A *to* N_i *only intersects* $\mathcal{P}$ *by* N_i *itself; (3)* $d_i \leq d_{i+1}$ $(1 \leq i \leq s-1)$.

3.5 Optimality for Communication Cost and Size of NPD-index

Optimality for Communication Cost Theorem 3 establishes that the NPD-index of each fragment is sufficient for computing distances between nodes in the fragment and any other nodes in graph $\mathcal{G}$. In other words, Theorem 3 says that each fragment is visited once only and fragments run independently without communication between each other.

THEOREM 3. *For any fragment* $\mathcal{P}$, *the distance from any node* $A \in \mathcal{G}$ *to any node* $B \in \mathcal{P}$ *can be computed with the edges in* $\mathcal{P} \cup SC(\mathcal{P}) \cup DL(\mathcal{P})$.

proof: We consider two cases $A \in \mathcal{P}$ and $A \notin \mathcal{P}$. For the first case, by Theorem 1, $\mathcal{P} \cup \text{SC}(\mathcal{P})$ is a complete fragment, then the distance from A to any node in $\mathcal{P}$ can be computed by $\mathcal{P} \cup \text{SC}(\mathcal{P})$. For the second case, we denote the nearest node from A that are both in $A \rightsquigarrow B$ and in $\mathcal{P}$ as node C. According to Rule 2, $d(A, C)$ is recorded in $\text{DL}(\mathcal{P})$. In addition, $d(C, B)$ can be computed by $\mathcal{P} \cup \text{SC}(\mathcal{P})$ as $\mathcal{P} \cup \text{SC}(\mathcal{P})$ is a complete fragment. Therefore, the distance $d(A, B)$ can also be computed in this case, due to $d(A, B) = d(A, C) + d(C, B)$. □

Optimality for Index Size We proceed to prove that the NPD-index is optimal in terms of size among the indices that are able to guarantee the correctness of distance computation. We first define a *standard fragment index* that suffices to compute network distances.

DEFINITION 7 (STANDARD FRAGMENT INDEX). *A standard fragment index with respect to fragment* $\mathcal{P}$ *is a shortcut edge set* $\mathcal{I} \subseteq \{(X, Y, d(X, Y)) | X \in \mathcal{G}, Y \in \mathcal{P}\}$ *such that every distance between a node* $A \in \mathcal{G}$ *and a node* $B \in \mathcal{P}$ *can be evaluated by* $\mathcal{P} \cup \mathcal{I}$, *where "d(A,B) can be evaluated by* $\mathcal{P} \cup \mathcal{I}$*" is either of the following two cases:*

i $(A, B, d(A, B)) \in \mathcal{P} \cup \mathcal{I}$.

ii There exists an internal node $C \in A \rightsquigarrow B$ *such that both* $d(A, C)$ *and* $d(C, B)$ *can be evaluated.*

By definition, a standard fragment index guarantees the correct computation of $d(A, B)$ $(\forall A \in \mathcal{G}, \forall B \in \mathcal{P})$, and thus meets the query need of SGKQ. In addition, the standard fragment index characterizes *all* the indices that are a subset of a broad set $\{(X, Y, d(X, Y)) | X \in \mathcal{G}, Y \in \mathcal{P}\}$ and are able to guarantee the correct computation.

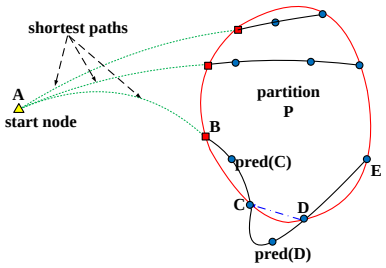


Figure 3: Rule 2 illustration: the DL component records only red nodes (square) that are nearest to node A on the shortest paths started from A.

By Theorem 3, we know $\text{IND}(\mathcal{P}) = \text{SC}(\mathcal{P}) \cup \text{DL}(\mathcal{P})$ is a standard fragment index. Theorem 4 guarantees $\text{IND}(\mathcal{P})$ achieves the minimum size among all standard fragment indexes. We introduce Lemma 2 to prove Theorem 4.

LEMMA 2. *If* $d(A, B)$ *can be evaluated by a standard fragment index with respect to fragment* $\mathcal{P}$, *then* $B \in \mathcal{P}$.

proof: By the condition (ii) in the definition of *standard fragment index*, $d(A, B)$ can be recursively divided into $d(A, u_1)$, $d(u_1, u_2)$, …, $d(u_i, B)$, where each of them is recorded in $\mathcal{P} \cup \mathcal{I}$. Since the last (shortcut) edge, $(u_i, B, d(u_i, B))$, belongs to $\mathcal{P} \cup \mathcal{I}$, then node B must belong to $\mathcal{P}$, noting that the second node in $\mathcal{P} \cup \mathcal{I}$ is in $\mathcal{P}$. □

THEOREM 4. *Among all the standard fragment indexes with respect to* $\mathcal{P}$, *IND(*$\mathcal{P}$*) achieves the minimum size where the size is measured by the number of distances being recorded.*

proof: For any standard fragment index $\mathcal{I}$ with respect to $\mathcal{P}$, we want to show that $\text{IND}(\mathcal{P}) \subseteq \mathcal{I}$. Therefore, we need to show that for any $(A, B, d(A, B)) \in \text{IND}(\mathcal{P})$, $(A, B, d(A, B))$ also belongs to $\mathcal{I}$. We assume by contradiction that $(A, B, d(A, B)) \notin \mathcal{I}$. Since $\text{IND}(\mathcal{P}) \cap \mathcal{P} = \emptyset$, then $(A, B, d(A, B)) \notin \mathcal{P}$. Considering the assumption, we have $(A, B, d(A, B) \notin \mathcal{P} \cup \mathcal{I}$. Moreover, as $(A, B, d(A, B)) \in \text{IND}(\mathcal{P})$, the second end node B must belong to $\mathcal{P}$ according to the construction rules. So by the definition of standard fragment index, $d(A, B)$ can be evaluated by $\mathcal{P} \cup \mathcal{I}$. Due to $(A, B, d(A, B)) \notin \text{IND}(\mathcal{P})$ (condition i not satisfied), it follows that condition ii should be satisfied, *i.e.*, there exists an internal node C in $A \rightsquigarrow B$ such that $d(A, C)$ and $d(C, B)$ can both be evaluated by $\mathcal{P} \cup \mathcal{I}$. Now we can generate a contradiction. On one hand, by Lemma 2, node C should belong to $\mathcal{P}$ as $d(A, C)$ can be evaluated by $\mathcal{P} \cup \text{IND}(\mathcal{P})$. On the other hand, node C does not belong to $\mathcal{P}$ by Rule 1 and Rule 2, which causes a contradiction. So the assumption is incorrect and $(A, B, d(A, B))$ should also belong to $\mathcal{I}$, which follows the theorem. □

3.6 Remark

The precomputed distances are also employed in existing work. However, their objectives and approaches are significantly different compared with NPD-index. First, some existing algorithms utilizes partition-based methods [11, 10] to speed up the centralized computation. They first partition

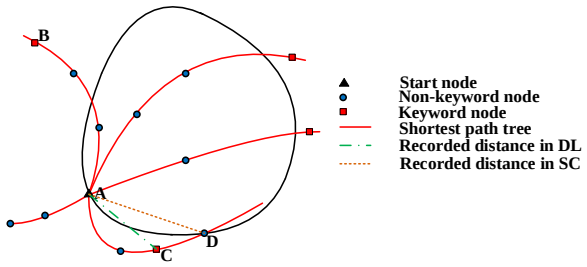


Figure 4: Index construction illustration: the Dijkstra search starts from node A, records distances according to Rule 1 / Rule 2 in SC / DL.

graphs and then record (1) the distances between the boundary nodes from different partitions and (2) the distances between a node and all the boundary nodes of the partition containing the node. In these methods, the distance between two nodes is computed via the boundary nodes of the both partitions containing them. These approaches need extensive interactions between partitions—They [11, 10] are designed for the centralized setting, and are not suitable for the distributed setting. In contrast, In the NPD-index, the distance between a source node and a target node can be computed by only the boundary nodes of the fragment containing the target node. In other words, the distance can be computed within the fragment containing the target node, and no communication is required between fragments.

Second, precomputed distances are also employed in shortcut based shortest path algorithms [21, 22]. However, those algorithms are not based on fragments, and their ideas are totally different from NPD-index.

3.7 Additional Pruning

In practice, the r value in SGKQ will not be too large, as the nodes that are too far away from the keyword nodes are of no interest in keyword search. Therefore, we set a parameter $maxR = \lambda \times e$, where e is the average edge length and λ is a factor. We assume that all r values should be at most $maxR$. Under this assumption, we only retain the distances that are at most $maxR$ in SC($\mathcal{P}$) and DL($\mathcal{P}$). For example, a node N_i in list $\{N_1, N_2, \ldots, N_s\}$ for index entry $(A, \mathcal{P})$ is retained only if $d(A, N_i) \leq maxR$.

An SGKQ mainly involves computing keyword coverages. Thus, we can do further optimizations by reformatting the graph slightly: assigning each keyword ω a virtual keyword node W, and connecting node W with any node that contains a keyword ω with the edge weight as 0. The direction of the virtual edges are from node W to the nodes containing ω. By doing so, focus on keyword nodes, instead of every node in the graph. Specifically, we prune all the entries in DL such that the node in the entry is not a keyword node.

4. INDEX CONSTRUCTION AND QUERYING

4.1 Parallel Fragment-wise Index Construction

A straightforward method to construct the DL component is to run *Dijkstra* algorithm starting from each keyword node, to search their first intersection nodes with every fragment along the shortest paths. Such method has two major drawbacks:

- the computation of the DL component and the SC component cannot be computed simultaneously, as the SC component is not only related to keywords nodes.
- the construction process is not fragment-wise.

Ideally, the index construction can be fragment-wise. In other words, one machine only takes charge of one fragment, and thus the index for the other fragments does not need to store in this machine. To this end, we propose a "backward search" scheme to construct the index. In particular, *Dijkstra* algorithm is run only with *portal nodes* as the source nodes. For a source portal node B, if the shortest path $A \leftrightsquigarrow B$ does not contain any other node in $\mathcal{P}$, then $d(A, B)$ will be recorded at entry $(A,\mathcal{P})$ of DL($\mathcal{P}$) (if $A \notin \mathcal{P}$ and A is a keyword node) or SC($\mathcal{P}$) (if $A \in \mathcal{P}$). Figure 4 shows an example that $d(A, C)$ is recorded in DL mapped by entry $(A, \mathcal{P})$ and $d(A, D)$ is recorded in SC($\mathcal{P}$). Alg. 1 presents the pseudo code for index construction for fragment $\mathcal{P}$ and returns SC and DL. To chase the visited fragments for a shortest path, we define variable $visitedParts$. The $visitedParts$ value changes when the shortest path tree from the source node grows. In pseudo code, each portal node is processed one by one (line 1). For the processing of each portal node, line 2–3 initialize the variables. Line 4–16 combines a *Dijkstra* search and our rules. In particular, line 7–9 describes Rule 2, line 10–11 describes Rule 1, and line 12–16 describes the edge relax for *Dijkstra* search.

Algorithm 1 NPD-Index Construction

```
    function Index_Construct({n_1, ..., n_s})
    variables:
    {n_1, ..., n_s} - portal nodes of P;
    pred[i] - the shortest path predecessor of node i;
    visitedParts[i] - visited fragments along the shortest path
    (end nodes excluded) from the source node to node i;
    part[i] - the fragment containing node i;
 1:   for each n_i
 2:     initialize visitedParts to be empty
 3:     insert n_i with weight 0 into priorityqueue H
 4:     while H not empty and top weight of H ≤ maxR
 5:       p = pop(H)
 6:       merge visitedParts[pred[p]] into visitedParts[p]
 7:       if p is a keyword node
 8:         if P ≠ part[p] and part[p] ∉ visitedParts[p]
 9:           add pair (p, d(n_i, p)) into entry (n_i, part[p]) in DL
10:       if p ∈ port(P) and part[p] ∉ visitedParts[p]
11:         add edge (p, n_i, d(n_i, p)) into SC
12:       for each q ∈ neighbor(p)
13:         if d(n_i, q) > d(n_i, p) + d(p, q)
14:           d(n_i, q) = d(n_i, p) + d(p, q)
15:           update pred[q] = p
16:           insert (q, d(n_i, q)) into H
```

When the list of node-distance pairs for an entry in DL has been computed, the pairs are sorted in an increasing order of distance. We do not include this operation in the pseudo code as it is very simple. The process makes the construction naturally parallel. For example, the construction of the index on different fragments can be handled by different machines, based on the application requirements.

4.2 Distributed SGKQ Querying

According to Theorem 3, the evaluation of SGKQ query can be performed in a fragment-wise manner, using the index at each fragment alone without incurring communication between fragments. This lays the foundation of our

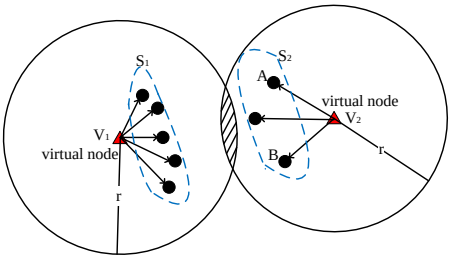


Figure 5: Illustration of the method for one fragment. A virtual node (red triangle) connects to every node (black circle) containing certain query keyword, by a virtual distance 0. The keyword coverage is computed by the *Dijkstra* algorithm with the virtual node as the source. The intersection of the coverages is the final result (shade).

method that can be naturally performed in the distributed setting. In the sequel, we only specify how the evaluation is done in a specific fragment $\mathcal{P}$ as the computation process is the same in other fragments. We call the computation on a fragment a *task*. Every task proceeds by steps as follows:

Step 1 **Read complete fragment.** Each task reads the edges of $\mathcal{P}' = \mathcal{P} \cup \text{SC}(\mathcal{P})$.

Step 2 **Search from index.** Searches the list of node-distance pairs for entry $(\omega_i, \mathcal{P})$ in DL($\mathcal{P}$), where ω_i is the i-th query keyword. Among the returned node-distance pairs, we retain the pairs with distance values at most radius r.

Step 3 **Extend $\mathcal{P}'$.** Add directed shortcut edges to fragment $\mathcal{P}$, where each of those shortcut edges corresponds to a node-distance pair in the retaining list output by Step 2. The direction of the added shortcut edges is from keyword nodes to other nodes.

We denote the extended fragment generated by the three steps as $\mathcal{P}_0$. Now, to evaluate the result nodes in $\mathcal{P}$ is equivalent to evaluating the result nodes in $\mathcal{P}_0$ (By theorem 3).

We present a simple approach to answer an SGKQ Q in a fragment. For each query keyword ω_i $(1 \leq i \leq k)$, we denote the set of nodes containing the keyword ω_i as node set S_i. To facilitate the distance computation between a node and S_i. We add a virtual node V_i with a distance of 0 connected to every node in S_i. The direction of the virtual edges is from node V_i to nodes in S_i, and thus we have $d(A, S_i) = d(A, V_i)$. Note that, we let the virtual edges be directed to avoid traveling back from the nodes in S_i to the virtual node V_i. For example, in Fig. 5, the path $A \to V_2 \to B$ should not be considered in *Dijkstra* search, otherwise the distance between node A and B will be set to 0, which is incorrect.

Next, for each keyword ω_i in query Q, we employ the *Dijkstra* algorithm [7] with V_i as the source to compute the keyword coverage $R(\omega_i, r)$. Finally, the query result can be obtained by intersecting all $\mathcal{R}(\omega_i, r)$ $(1 \leq i \leq k)$. In Fig. 5, red nodes (triangle) are virtual nodes of node set $\mathcal{S}_1$ and $\mathcal{S}_2$, they act as the source nodes of the *Dijkstra* algorithm.

Based on the aforementioned method, the pseudocode of NPD-index-based query computation is shown in Alg. 2. A query example is shown in Fig. 6.

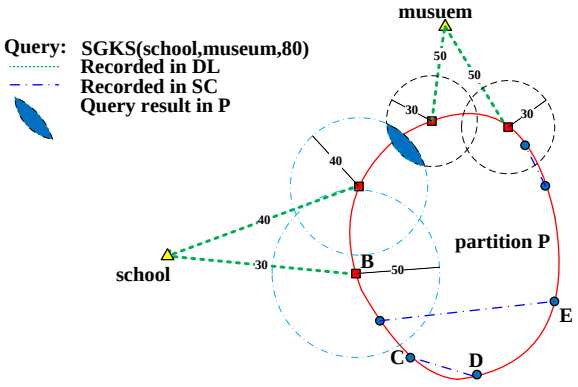


Figure 6: An SGKQ query is performed. First, complete fragment $\mathcal{P} \cup \text{SC}(\mathcal{P})$ are extended with additional edges from DL($\mathcal{P}$) (bold green dot lines); Second, keyword coverages from query keywords *museum* and *school* are searched on the extended fragment. The result is the intersection of the coverages.

Algorithm 2 NPD-index based SGKQ query processing

function *query*($\mathcal{P}$)
variables: $\mathcal{P}$ - a graph fragment
1: read edges in $\mathcal{P} \cup \text{SC}(\mathcal{P})$
2: **for each** keyword node ω_i
3: *List*=search(entry $(\omega_i, \mathcal{P})$) in index file $IND(\mathcal{P})$
4: **for each** node-distance pair (q, d_q) in *List*
5: **if** $d_q \leq r$
6: add directed edge (p, q, d_q) to fragment $\mathcal{P}$
7: search keyword coverage $R_i = R(\omega_i, r)$ in the new fragment using *Dijkstra* algorithm
8: **return** $\cap R_i$

5. ANALYSIS AND EXTENSIONS

5.1 Complexity

Since the computation cost in every fragment is identical, we take a fragment $\mathcal{P}$ for the cost analysis. We divide the cost analysis in $\mathcal{P}$ into 2 parts.

First, for each query keyword ω, the keyword coverage $R(\omega, r)$ in $\mathcal{P}$ is figured out with $\mathcal{P} \cup \text{IND}(\mathcal{P})$. Hence the computation cost is no more than that of executing a *Dijkstra* algorithm from the virtual keyword node W (for ω) on $\mathcal{P} \cup \text{IND}(\mathcal{P})$. We denote the number of node-distance pairs in DL($\mathcal{P}$) mapped from the entry $(\omega, \mathcal{P})$ is α and the number of edges in SC($\mathcal{P}$) is β. Hence the size of additional edges added to $\mathcal{P}$ is $\alpha + \beta$, and the number of explored nodes during executing *Dijkstra* algorithm is $|\mathcal{P} \cap R(\omega, r)|$. It follows that the complexity of computing keyword coverage $R(\omega, r)$ in $\mathcal{P}$ is $\mathcal{O}(\alpha + \beta + |\mathcal{P} \cap R(\omega, r)| log(|\mathcal{P} \cap R(\omega, r)|))$.

Second, computing the intersection among $(R(\omega, r) \cap \mathcal{P})$ (for every query keyword ω) can be finished by visiting each $(R(\omega, r) \cap \mathcal{P})$ once. Therefore, the computation cost of part 1 is dominant.

By these analysis, we have the following result:

THEOREM 5. *The time complexity of Alg. 2 for answering the SGKQ query in fragment $\mathcal{P}$ is $\mathcal{O}(\sum_{1 \leq j \leq k} (\alpha_j + \beta + |\mathcal{P} \cap R(\omega_j, r)| log(|\mathcal{P} \cap R(\omega_j, r)|)))$, where $\beta = |SC(\mathcal{P})|$ and α_j is the number of node-distance pairs in entry $(\omega_j, \mathcal{P})$ in DL($\mathcal{P}$); ω_j $(1 \leq j \leq k)$ is a query keyword.*

Analysis of Theorem 5. In distributed computing, the response time is determined by the slowest task, which is given in Theorem 5 for our problem. Observe that part of the time complexity, $\mathcal{O}(\sum_{1\leq j\leq m} |\mathcal{P}_i \cap R(k_j, r)|(log(|\mathcal{P}_i \cap R(k_j, r)|)))$, will become smaller as the number of fragments become larger. The remaining part, *i.e.*, $\sum_{1\leq j\leq k} (\alpha_j + \beta) \leq k\beta + \sum_{1\leq j\leq k} \alpha_j \leq k|\text{IND}(\mathcal{P})|$, is bounded by the index size. Our experiments show that the index size on every fragment is rather small (See EXP. 1).

5.2 Load Balance

To discuss the load balance of our approach, we assume a simple task assignment strategy: an un-assigned task must be assigned to certain idle machine if there are idle machines. This strategy, although simple, is considered to be one of the most general strategies in distributed computing. Note that, when more sophisticated load balance strategy is employed, the correctness of the following discussion still holds. We define the unbalance factor to be $U = \max_{1\leq i\neq j\leq c} \frac{cost(M_i)}{cost(M_j)}$, where M_i is the i-th machine and c is the number of machines that have been assigned with tasks. Intuitively, U is the maximal workload ratio between two machines M_i and M_j. $U = 1$ means perfect balance while $U >> 1$ means unbalance. We give the following theorem to describe the relationship between U and the task costs.

THEOREM 6. *$U \leq 1 + \frac{\max_{1\leq k\leq N} cost(\mathcal{P}_k)}{\min_{1\leq k\leq N} cost(\mathcal{P}_k)}$, where $cost(\mathcal{P}_k)$ refers to the cost of evaluating query results in fragment $\mathcal{P}_k$.*

Theorem 6 shows that if the costs of tasks are similar, then the algorithm will achieve a satisfactory load balance. Note that the fragment technique we use guarantees a balance fragment of the road network, thus an acceptable load balance is achieved by our approach.

5.3 Multiple Shortest Paths Scenario

Previously, we assumed that the shortest path between any two nodes is unique. Although this assumption often holds in practice, the proposed index can correctly handle the multiple shortest paths scenario. In fact, we can change the algorithm slightly to handle this scenario. In particular, Rule 1 is changed to Rule 3:

RULE 3. *We add a shortcut edge (A, B) with weight $d(A, B))$ to SC($\mathcal{P}$) iff: (1) $A \in P$ and $B \in P$; (2) $(A, B, d(A, B))$ is not an edge in the road network graph $\mathcal{G}$; (3) **Any** shortest path from A to B does not contain any other node of $\mathcal{P}$;*

Rule 2 is changed to Rule 4:

RULE 4. *Value $\{(N_1, d_1), (N_2, d_2), \ldots, (N_s, d_s)\}$ for entry $(A, \mathcal{P})$ conforms to: (1) $N_i \in port(\mathcal{P})$, $\forall 1 \leq i \leq s$; (2) **Any** shortest path from A to N_i only intersects $\mathcal{P}$ by N_i itself; (3) $d_i \leq d_{i+1}$ $(1 \leq i \leq s-1)$.*

With the techniques in the proofs of Theorems 1 to 4, it is easy to show that the theorems still hold with above rules in the multiple shortest paths scenario.

5.4 From SGKQ to Other Queries

We generalize *Spatial Group Keyword Query* (SGKQ) and *Range Keyword Query* (RKQ) into Q-class queries.

DEFINITION 8 (Q-CLASS). *Given a weighted graph $\mathcal{G}$, a set of radiuses $\{r_1, \ldots, r_k\}$ and a set of keywords $\{\omega_1, \ldots, \omega_k\}$, we consider the class of spatial keyword queries Q such that Q can be answered by evaluating a D-function $\mathcal{F}(R(\omega_1, r_1), \ldots, R(\omega_k, r_k))$. We refer to the class of spatial keyword queries as Q-class.*

We proceed to discuss why our approach is feasible for all the queries in Q-class in two aspects.

- (**Property i: Implementation**) It is trivial to extend Alg. 2 to handle other specific queries in Q-class.
- (**Property ii: Time complexity**) Any D-function $\mathcal{F}$ will not hurt the advantages of our approach, compared to a centralized algorithm.

Satisfying Property i. Note that, varying radiuses for query keywords causes only trivial parameter changes by Alg. 2. When the radiuses r_i $(1 \leq i \leq k)$ are fixed, for any two query types drawn from Q-class, only the final functions $\mathcal{F}$ are different. By the definition of D-function, as long as there is a technique that can evaluate the specific D-function in the centralized setting, this technique can be directly employed in the distributed setting.

Satisfying Property ii. Irrespective of r_i $(1 \leq i \leq k)$ and the D-function $\mathcal{F}$, the cost of evaluating $\mathcal{F}$ is linear to the size of input set, and takes a small portion of cost compared to evaluating the keyword coverage (super linear to the size of input set), no matter on the whole network or on a fragment. Therefore, the correctness of Theorem 5 holds for any Q-class query. By the analysis for Theorem 5 in Section 5.1, the superiority of the distributed computing in time complexity still holds.

5.5 When $r > maxR$

The selection of $maxR$ affects the index sizes and index construction cost, as well as the applicable range of the query factor r. As we discussed earlier in most real-life queries r will not be very large, and thus we can set a $maxR$ to be sufficient large. In the rare cases where the query factor r is larger than the selected $maxR$, we can address this by building a bi-level index such that each machine holds two sets of indexes. One index is generated by fixing $maxR$ as a manually selected value (according to the applications) and the other is built without the $maxR$ restriction. With a properly selected value of $maxR$, most queries can be handled by the first index. Meanwhile, the queries with a value of r larger than the $maxR$ value are handled by the second index. In fact, even when we remove the $maxR$ (or equally letting $maxR = \infty$), the query time will only slightly goes up (Fig. 9).

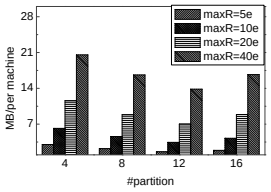
6. PERFORMANCE EVALUATION

We study the performance of SGKQ in detail in EXP. 1 – 6. We evaluate the effect of different D-functions in EXP. 7 and the performance of RKQ in EXP. 8.

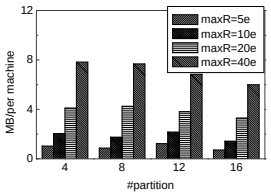
Experimental Settings. The experiments are conducted in a cluster consisting of 16 machines, each machine with 4G memory and installed with Ubuntu Linux. The machines are interconnected by a 100MB TP-LINK switch.

Datasets. We use 2 real road networks for evaluation, where BRI (British) and AUS (Australia) are extracted from openstreetmap [1]. Both datasets contain a road network,

[1]http://www.openstreetmap.org/



(b) BRI, Index size



(c) AUS, Index size

Figure 7: The index size with different $maxR$ values and the number of fragments.

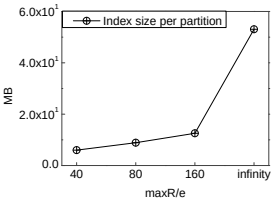


Figure 8: Index size vs $maxR$ on AUS

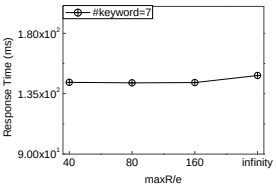


Figure 9: Query time vs $maxR$ on AUS.

Table 1: Datasets

name	nodes	objects	edges	keywords
BRI	3,760,213	300,891	9,730,188	57,600
AUS	1,223,171	70,064	3,364,364	18,750

Table 2: Parameters

maxR/avg edge	5, 10, 20, **40**
#keywords	3,5,**7**,9,11
#fragments	2,4,8,12,**16**
r	**40e**,maxR,maxR/2,maxR/3,maxR/4

together with a number of objects on the road network. Each object is tagged with some keywords. In preprocessing, we take each object as a node and let it connect to its nearest network node. Summary statistics of the datasets are listed in Tab. 1. Tab. 2 gives the possible parameter settings, where the bold ones are the default values.

Each graph is fragmented to N node-disjoint fragments, aiming at minimizing cross-partition edges for parallel computing. We use distributed graph partitioning algorithms ParMetis [13] for a balanced fragmenting. We let each machine (except for the coordinator) handle one fragment.

Generating queries. We select query keywords in a manner of considering both keyword closeness and frequency. We first select a circle range centered by a random node. Then, within the range we choose the keywords according to their frequency. Keywords with higher frequency have a larger chance to be chosen. The selection method is reasonable as keywords are correlated in a range and keywords with higher frequency are considered to have a higher probability to be issued by a user.

EXP 1: Storage cost. We measure the average storage cost for index files on each machine. The storage cost in each machine is incurred by storing the SC file and DL file. From Fig. 7 (a) and (b), we can see that the average storage cost in each machine is within 21MB for BRI, and below 8MB for AUS. Even to set $maxR$ to infinity, the index size is still acceptable, below 60MB for AUS on each machine (Fig. 8); and the index size for BRI is below 170M (the figure is ignored due to the space limitation).

The cost is very small compared to the size of hard disk of a modern PC. As expected, the average storage cost in each machine increases when $maxR$ becomes larger. We note $maxR = 40e$ (*i.e.*, e is the average edge length) is a relatively large radius. For example, the average edge length of AUS is more than 1.2 kilometer, therefore, an index with $maxR = 40e$ means it can handle all queries with radiuses within 0–50 kilometers. We do not observe a regular tendency for the change of the average storage cost in each machine normally as the number of machines varies.

Table 3: Indexing time per fragment (in minutes)

maxR/avg edge	10	20	40
#fragments=4	19	25.3	25.8
#fragments=8	10.8	14.1	16.6
#fragments=12	8.7	11.9	14.6
#fragments=16	6.2	8.9	11

EXP 2: Indexing time. Indexing time is the time for constructing indexes. We show the per-fragment indexing time (AUS) with varying fragments and varying $maxR$ in Tab. 3. In a nutshell, for a normal network size (such as AUS) with ordinary parameter settings, the time constructing the index on a fragment is within 30 minutes, which is surely affordable as the indexing process is done offline.

Next, we evaluate the query performance of our method and summarize how the performance is affected by different factors. In particular, we study the effect of the following factors: the number of fragments, the number of query keywords, index factor $maxR$ and query parameter r. Note that, unless stated, except the factor/parameter examined, the other factors/parameters (resp. varying factors/parameters) is set by default as in Tab. 2.

EXP 3: Effect of the number of keywords. From Fig. 10 and Fig. 11, we can see that the run time of both methods increases with the increase of the number of keywords. This is reasonable as the more query keywords, the more keyword coverages should be evaluated and thus the more computational cost will be incurred. In Fig. 10 and Fig. 11, we also give the runtime of 1 fragment for reference. Note that the Q-class spatial-keyword query has not been studied and there exist no centralized algorithm for it. However, the distributed method has a much better scalability with the number of query keywords than that the algorithm running one 1 machine. The superiority attributes to the NPD-index that enables our method to evaluate each fragment independently. With the NPD-index, the cost incurred by spatial keyword query is distributed to different fragments (machines), resulting in a better scalability.

EXP 4: Effect of r. Query performance is also affected by r. Figures 14 and 15 show the results when r varies among $[\frac{1}{4}maxR, maxR]$. When r is larger, the query response time is longer as a larger r means evaluating a larger keyword coverage. The value of r has much less effect on the performance of our method than on the centralized method. This reflects the robustness of our method.

EXP 5: Effect of $maxR$. The factor $maxR$ affects the indexing time and size as shown in EXP 1. This experiment is to evaluate its effect on the query performance. We sum-

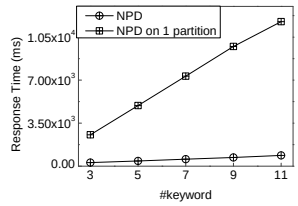


Figure 10: BRI, varying keywords.

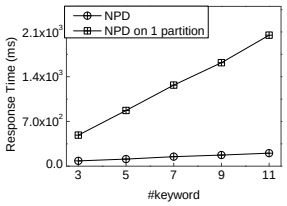


Figure 11: AUS, varying keywords.

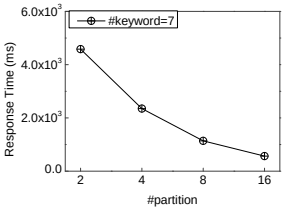


Figure 12: BRI, varying fragments

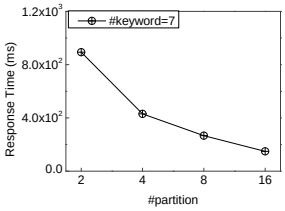


Figure 13: AUS, varying fragments

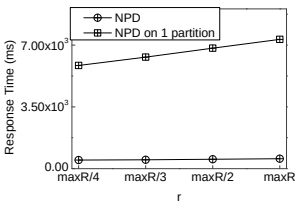


Figure 14: Varying r on BRI.

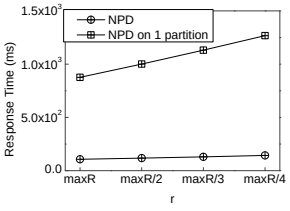


Figure 15: Varying r on AUS.

marize the results in Fig. 9. The results demonstrate that the $maxR$ value has a very limited effect on the query performance, even when $maxR$ is set to positive infinity. The result on AUS is qualitatively similarly and is ignored due to the space limitation.

EXP 6: Effect of the number of fragments. This experiment aims to evaluate the effect of the number of fragments on the performance of the NPD-index based method. We vary the number of fragments from 2 to 16. Each fragment is handled by one machine. The results are shown in Fig. 12 and Fig. 13, which indicate that the response time is approximately cut by half when the fragments are doubled, demonstrating a good scalability.

EXP 7: Effect of different D-functions. We evaluate the effect of different D-functions on performance. We fix the number of query keywords to 7, and the D-function in the format of $\mathcal{X}_1\theta_1\mathcal{X}_2\theta_2\ldots\theta_6\mathcal{X}_7$ is generated by taking θ_i $(1 \leq i \leq 6)$ from $\{\cap, -\}$. We vary the ratio of the two set operators, *i.e.*, intersection $\cap$ and subtraction $-$, to generate 6 cases. Figure 16 illustrates the performance of different cases evaluated on AUS, where the x-axis is the number of used subtraction operators. We can conclude from the figure that different D-functions have minor effect on the overall performance, since in all cases, the cost of evaluating keyword coverage takes more than 95% of the total cost.

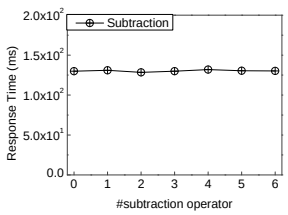


Figure 16: AUS, different D-function

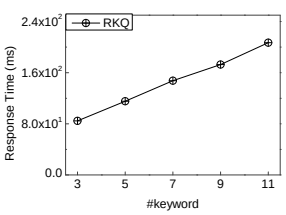


Figure 17: AUS, RKQ query

EXP 8: Range keyword query. The range keyword query (`RKQ`) is a type of Q-class query and can be handled by our method. The evaluation results on AUS are illustrated in Fig. 17. We conclude from the figure that, the performance of RKQ with our NPD-index based method scales well with the number of keywords.

7. RELATED WORK

Spatial-keyword queries in Euclidean Space. A type of popular spatial keyword query is to retrieve all objects whose text descriptions contain a given set of keywords and whose locations are within a specified distance of the query location [4, 5]. This is similar to the RKQ query in this paper except that the RKQ query considers road network distance rather than Euclidean distance. Another type of spatial keyword queries is to find the k objects with the highest ranking scores, measured as a combination of their distances to the query location and the relevances of their text descriptions to the query keywords [6], or find the k most nearest objects whose descriptions contain query keywords [9].

Spatial-keyword queries in Road Networks. Most of the existing work on spatial-keyword querying focuses on Euclidian space. Recently, road network distance is also considered in spatial-keyword querying [20, 14]. Li *et al.* [14] addressed a spatial query that returns nodes satisfying both the spatial constraints (*i.e.*, within a spatial range from a specific location) and keyword similarity constraints. The top-k spatial-keyword query [20] was proposed to find the k objects with the highest ranking scores, measured as a combination of their road network distances to the query location and the relevances of their text descriptions to the query keywords. However, none of these proposals considers the SGKQ and they cannot be used to answer the Q-class query. In addition, a demonstration of the preliminary version of this work was presented in [15].

Keyword search over relational databases. Our work is also related to the research on keyword search over relational databases. By transforming a database to a graph with tuples as nodes and foreign-key references as edges, many approaches [1, 8, 10, 12] have been proposed for keyword search on the database graphs. Ding *et al.* [8] employed dynamic programming to optimize the min-cost group Steiner trees searching. BANKS[1, 12] and BLINKS [10] search rooted trees with backward search or bidirectional search techniques to approximate group Steiner tree searching. BLINKS [10] indexes the distances between keywords and nodes and the distances between nodes and portal nodes in each fragment of a relational graph. Note that the index used in BLINKS is significantly different from the proposed NPD-index—BLINKS index records the intra-fragment information while the NPD-index focuses on the information between a fragment and the nodes outside the fragment.

Moreover, BLINKS's index is designed for a centralized setting and cannot be used for distributed computing while the NPD-index is proposed for a distributed setting.

Parallel keyword search over graphs. Our work is related to parallel keyword search over graphs [19]. Qin *et al* focused on the parallel potential on a multi-core platform for Candidate Network evaluation, and proposed an approach to distributing SQLs to different cores when considering minimizing workload skew, minimizing inter-core sharing and maximizing intra-core sharing. The problem and approach are different from ours.

Parallel graph query processing. We have discussed closely related research on parallel graph query processing in Section 2.3.

8. CONCLUSIONS

This paper proposes a new distributed index scheme called NPD-index to answer two types of spatial-keyword queries on road networks in a distributed setting. We analyze the optimality and complexity of the proposed method, and show the rationality of the proposed scheme and algorithms. To the best of our knowledge, the NPD-index is the first technique for answering spatial-keyword queries on road networks in a general distributed environment. The experimental studies conducted under a general distributed environment demonstrate the efficiency of the proposed methods based on the the NPD-index.

In the future, it would be interesting to extend our method to handle other types of graphs such as relational database graphs and social networks. Also, it remains open whether other types of queries can benefit from NPD-index.

9. ACKNOWLEDGEMENT

This work was supported in part by NSFC (61373036), the Research Innovation Program of Shanghai Municipal Education Commission, a grant awarded by a Singapore MOE AcRF Tier 2 Grant (ARC30/12) and a Singapore MOE AcRF Tier 1 Grant (RG66/12).

10. REFERENCES

[1] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *ICDE*, pages 431–440, 2002.

[2] X. Cao, L. Chen, G. Cong, C. S. Jensen, Q. Qu, A. Skovsgaard, D. Wu, and M. L. Yiu. Spatial keyword querying. In *ER*, pages 16–29, 2012.

[3] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi. Collective spatial keyword querying. In *SIGMOD*, pages 373–384, 2011.

[4] L. Chen, G. Cong, C. S. Jensen, and D. Wu. Spatial keyword query processing: An experimental evaluation. *PVLDB*, 6(3):217–228, 2013.

[5] M. Christoforaki, J. He, C. Dimopoulos, A. Markowetz, and T. Suel. Text vs. space: efficient geo-search query processing. In *CIKM*, pages 423–432, 2011.

[6] G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. *PVLDB*, 2(1):337–348, 2009.

[7] E. W. Dijkstra. A note on two problems in connexion with graphs. In *Numerische Mathematik*, pages 269–271, 1959.

[8] B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin. Finding top-k min-cost connected trees in databases. In *ICDE*, pages 836–845, 2007.

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

[10] H. He, H. Wang, J. Yang, and P. S. Yu. BLINKS: Ranked keyword searches on graphs. In *SIGMOD*, pages 305–316, 2007.

[11] S. Jung and S. Pramanik. An efficient path computation model for hierarchically structured topographical road maps. *IEEE Trans. Knowl. Data Eng.*, 14(5):1029–1046, 2002.

[12] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB*, pages 505–516, 2005.

[13] G. Karypis and V. Kumar. Metis [online] http://glaros.dtc.umn.edu/gkhome/views/metis/.

[14] F. Li, B. Yao, M. Tang, and M. Hadjieleftheriou. Spatial approximate string search. *IEEE TKDE*, 99(PrePrints), 2012.

[15] S. Luo, Y. Luo, S. Zhou, G. Cong, and J. Guan. Disks: A system for distributed spatial group keyword search on road networks. *PVLDB*, 5(12):1966–1969, 2012.

[16] K. Madduri, D. Bader, J. Berry, and J. Crobak. Parallel shortest path algorithms for solving large-scale instances. In *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, volume 74, pages 249–290. 2009.

[17] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *SIGMOD*, pages 135–146, 2010.

[18] M. Naor and L. J. Stockmeyer. What can be computed locally? *SIAM J. Comput.*, 24(6):1259–1277, 1995.

[19] L. Qin, J. X. Yu, and L. Chang. Ten thousand SQLs: parallel keyword queries computing. In *VLDB*, volume 3, pages 58–69, 2010.

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

[21] P. Sanders and D. Schultes. Highway hierarchies hasten exact shortest path queries. In *Algorithms–Esa 2005*, pages 568–579. Springer, 2005.

[22] D. Schultes and P. Sanders. Dynamic highway-node routing. In *Experimental Algorithms*, pages 66–79. Springer, 2007.

[23] Y. Tang, Y. Zhang, and H. Chen. A parallel shortest path algorithm based on graph-partitioning and iterative correcting. In *HPCC*, pages 155–161, 2008.

[24] D. Zhang, Y. M. Chee, A. Mondal, A. K. H. Tung, and M. Kitsuregawa. Keyword search in spatial databases: Towards searching by document. In *ICDE*, pages 688–699, 2009.