DISKs: A System for Distributed Spatial Group Keyword Search on Road Networks

Siqiang Luo[§] Yifeng Luo[§] Shuigeng Zhou[§] Gao Cong[†] Jihong Guan[‡]
[§] School of Computer Science, Fudan University, China
[†] School of Computer Engineering, Nanyang Technological University, Singapore
[‡] Department of Computer Science & Technology, Tongji University, China
{sqluo, luoyf, sgzhou}@fudan.edu.cn, gaocong@ntu.edu.sg, jhguan@tongji.edu.cn

ABSTRACT

Query (*e.g.,* shortest path) on road networks has been extensively studied. Although most of the existing query processing approaches are designed for centralized environments, there is a growing need to handle queries on road networks in distributed environments due to the increasing query workload and the challenge of querying large networks. In this demonstration, we showcase a distributed system called *DISKs* (*DI*stributed *S*patial *K*eyword *s*earch) that is capable of efficiently supporting spatial group keyword search (SGKS) on road networks. Given a group of keywords X and a distance r, an SGKS returns locations on a road network, such that for each returned location p, there exists a set of nodes (on the road network), which are located within a network distance r from p and collectively contains X. We will demonstrate the innovative modules, performance and interactive user interfaces of DISKs.

1. INTRODUCTION

In a road network, each node represents a location[1], the edge between two nodes represents the path between them, and the length of the path is referred to as the edge weight. Each node can be tagged with textual information such as "restaurant" or "school". Query processing on road networks has many applications, such as online map services and mobile services.

In this demonstration, we present a new type of spatial keyword search query, called *spatial group keyword search* (SGKS), on road networks in a distributed environment. Given a set of keywords ψ, a specified distance r, and a road network, a location (*i.e.*, node) is retrieved as a result of the *spatial group keyword search* if for each keyword $t \in \psi$ there exists a node containing t and within a road network distance r to the location. An example application of SGKS is that a real estate company wants to select on the map a place whose distances to hospital, school and supermarket are all within 5 kilometers, where the query keywords are "hospital", "school" and "supermarket".

Most existing proposals on spatial keyword search [3, 7, 9] consider Euclidean distance of spatial objects rather than network distance. Recently, Li *et al.* [6] consider network distance for a type of spatial keyword query where each returned object nodes should contain a keyword similar to the query keyword. However, the technique [6] cannot be employed for processing SGKS. Furthermore, all of the proposals focus on the centralized setting. Most of the existing systems/methods of keyword search in graphs [2, 3, 4, 7, 9] are designed under a centralized setting, and cannot be used for SGKS.

In this demonstration, we present DISKs (*DI*stributed *S*patial *K*eyword *s*earch) for SGKS on road networks in a Hadoop [1] based distributed architecture. DISKs provides interactive user interfaces for composing queries and presenting the results. The main features of DISKs are as follows:

- DISKs employs an innovative distributed index called NPD-index (*N*ode *P*artition *D*istance index) for distributed query processing. Our experiments show that the index is compact in space. In theory, it can eliminate the communication cost between machines during distributed query processing.
- DISKs is implemented based on Hadoop, a popular cloud computing platform. We demonstrate how to build index on Hadoop and how to employ it to significantly boost the efficiency of distributed query processing on Hadoop.
- DISKs offers interactive user interfaces for composing queries as well as presenting, analyzing and comparing query results.

2. DISKS OVERVIEW

Figure 1 shows the architecture of DISKs, which consists of three major modules: Partitioner, Indexer and Distributed Query Processor. Partitioner and Indexer are preprocessing modules for constructing index. The index is constructed once, and works for all queries.

2.1 Partitioner and Indexer

The Partitioner takes a road network (a graph) as the input and outputs N small node-disjoint balanced subgraph partitions. This module employs METIS [5], which can minimize the number of cross-partition edges and balance the

[1]In the sequel we use interchangeably *node* and *location*.

Proceedings of the VLDB Endowment, Vol. 5, No. 12

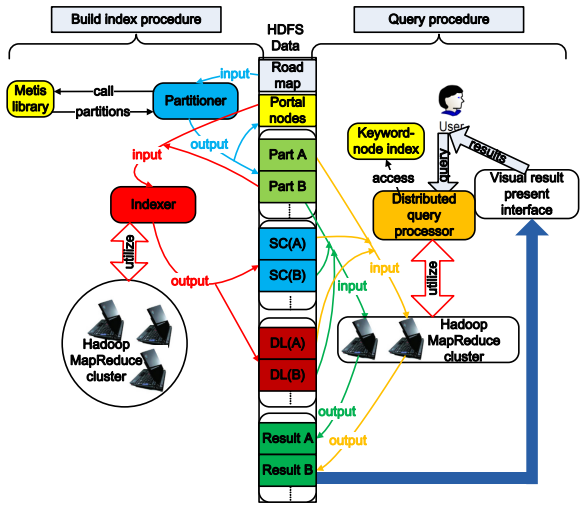

Figure 1: The architecture of DISKs.

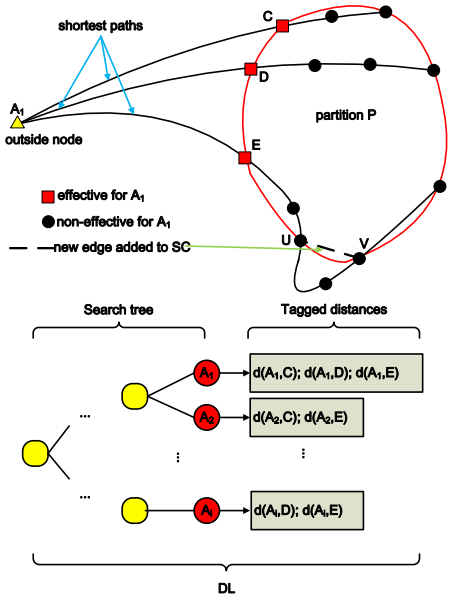

Figure 2: NPD-index illustration.

size of each partition. The cross-partition edges are only a small subset of the entire edge set in our experiments, as road networks are sparse graphs. We refer to the *end nodes* of cross-partition edges as *portal nodes*, which are also output by Partitioner.

The Indexer takes partitions and portal nodes as arguments, and outputs *NPD-index*. NPD-index is composed of N components, each of which corresponds to a partition. Each component contains two structures, which we call Distance-List (DL) and Short-Cut (SC), respectively. We proceed to describe DL and SC via Fig. 2.

DL is a search tree on the nodes of the road network, where each leaf node of the search tree is tagged with distances between this node and *part of* the portal nodes of partition P, as shown in Fig. 2. Here, A_1 is a leaf node of the DL search tree corresponding to partition P. We say that a portal node p in P is *effective* (*e.g.*, C, D and E in the figure) with respect to A_1, if any shortest path[2] between A_1 and p goes through only one node in P, which is p itself here. Our interesting finding is that only the distances between A_1 and its effective portal nodes need to be recorded in DL for query processing.

The other structure SC is a set of extra added edges between two portal nodes of P. If any shortest path between two portal nodes u and v covers only two nodes u and v (*e.g.*, U and V in Fig. 2) in P, then a new edge $(u, v, d(u, v))$ is added into SC, where $d(u, v)$ is the shortest distance between u and v.

With partition P and its corresponding DL and SC structures, we can compute the exact shortest distance from any node of the road network to any node in P. The rationale is that, for any node u outside P and any node p inside P, there exists a shortest path between them, which only intersects P by effective portal nodes with respect to u [8]. For example, in Fig. 2, the shortest distance from A_1 to V is computed by $d(A_1, E)$ (recorded in DL as E is an effective portal node with respect to A_1), $d(E, U)$ (computed inside partition P) and $d(U, V)$ (recorded in SC). If two nodes u and p are both inside P, then there exists a shortest path from u to p composed of original edges in P and newly added edges in SC. For example, the shortest distance from E to V is the sum of $d(E, U)$ and $d(U, V)$.

We define the spatial cost between a location u and a node set S (the set of nodes containing a keyword) as the minimum shortest path length (*a.k.a* shortest distance) between u and all nodes in S. For example, if set $S = \{p, q\}$, the spatial cost between u and S, $d(u, S)$, is the minimum of $d(u, p)$ and $d(u, q)$, where $d(u, p)$ (*resp.* $d(u, q)$) is the shortest distance between u and p (*resp.* u and q).

SGKS concerns the nodes whose spatial cost to each node set related to a keyword is no more than r. In practice, r is much smaller than the diameter of the spatial network. Therefore, we set a parameter $maxR$ as the maximal of r, instead of the diameter of the network. During index construction, we neglect the nodes whose shortest distances to all portal nodes are larger than $maxR$. Specifically, NPD-index can be constructed by performing a set of local *Dijkstra* algorithms on the whole network, with the portal nodes as sources. The *Dijkstra* search stops when the distance to the source is larger than $maxR$.

2.2 Distributed Query Processor

After the NPD-index is constructed, queries can be processed distributedly based on a machine pool (e.g., a cloud computing platform). Each partition together with its index is assigned to a machine for computing the results in that partition. Each machine may receive multiple partitions, depending on the number of machines and the number of partitions. The *distributed query processor* takes as input a parameter r and several keywords. For each keyword K, the node set X_K comprising the nodes containing K is extracted via a keyword-node inverted index (see Fig. 1). For partition P, the query processor performs the following steps:

[2]we use "any" as multiple shortest paths of equal length may exist.

Step 1: For each node set X_K (corresponding to a keyword K), the following 2 sub-steps are preformed:

Sub-step 1: Access the search tree of DL to find the leaf nodes contained in X_K, and retrieve their tagged portal nodes with distances at most r.

Sub-step 2: Beginning from the retrieved portal nodes, search over $P \cup SC$, and return the nodes in P whose distance to at least one node in the node set X_K is not larger than r.

Step 2: Intersect the returned nodes of all node sets.

Step 1 computes for each keyword K the set of nodes in P whose distance to the node set X_K (corresponding to K) is not larger than r. Step 2 does an intersection to find the result nodes of P whose distances to all node sets (each corresponding to a query keyword) are not larger than r. Note that the computation for different partitions is independent.

2.3 Hadoop-based DISKs Implementation

The proposed DISKs architecture and techniques are independent of any specific distributed computing platform. Our current system is implemented on Hadoop. Hadoop stores data on Hadoop Distributed File System (HDFS) and completes computation by *jobs*. A Hadoop job is composed of the *map* phase and the *reduce* phase. During the map phase, the workload is distributed to different machines (datanodes), while during the reduce phase the output of the map phase with the same key is sent to the same machine for further processing.

NPD-index construction can be reduced to a set of *Dijkstra* instances taking portal nodes as sources. The instances can be conveniently distributed to different machines in the Hadoop cluster for computing during the map phase. Key-value pairs in the form of $(key : \{u, P\}, value : \{s, d(u, s)\})$ are output at the map phase, indicating that node s is effective with respect to node u. As a result, in the reduce phase, the effective portal nodes and tagged distances (*e.g.*, $\{s, d(u, s)\}$) with respect to node u are aggregated, compromising all the tagged distances for the leaf node u in DL for partition P. DL is stored in Hadoop MapFile, which is an index file in Hadoop that stores the content as a series of key-value pairs, and the keys are accessed by a binary tree. Both DL and SC are stored distributively in HDFS.

The distributed query steps (Section 2.2) are implemented in a Hadoop job using only the map phase as the computation on different partitions can be separated. This is a desirable feature as the reduce phase will incur many network I/Os.

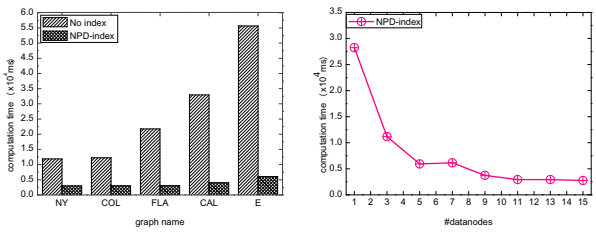

Figure 3: Left: scalability with graph size. Right: scalability with the number of datanodes.

.

Table 1: Datasets

name	nodes	edges	description
NY	264,346	733,846	New York
COL	435,666	1,057,066	Colorado
FLA	1,070,376	2,712,798	Florida
CAL	1,890,815	4,657,742	California,Nevada
E	3,598,623	8,778,114	Eastern USA

Table 2: The ratio of average index size (DL or SC) per partition to the road network size. The second row is the value of $maxR/10^5$. N = #partitions.

N	DL			SC		
	0.5	1	1.5	0.5	1	1.5
10	0.0196	0.0602	0.1107	0.0001	0.0002	0.0002
20	0.0198	0.0603	0.1126	0.0002	0.0004	0.0005
30	0.0118	0.0368	0.0692	0.0001	0.0002	0.0003

3. SYSTEM PERFORMANCE

We present a summary of experimental results on index size and query performance of the implemented system. The Hadoop cluster consists of 16 machines, installed with Hadoop 0.20.2 and Ubuntu Linux. The machines are interconnected with an 100MB TP-LINK switch. Statistics of the real road network datasets[3] used in the experiments are listed in Tab. 1. As the datasets have no textual information associated with the nodes, we randomly select keywords from a vocabulary for each node such that each keyword is contained by 100 nodes. Table 2 shows the index sizes for different numbers of partitions and different values of $maxR$, where the index size is measured by the number of effective portal nodes (for DL) and the number of newly added edges (for SC). We can see that DL size is relatively smaller than the size of road network (number of edges), while SC size is much smaller than the road network size. To demonstrate the effectiveness of the NPD-index, we compare the proposed approach with a baseline query method without employing index on Hadoop, which is marked as "No Index". The baseline simply distributes the entire road network to several machines, each of which is responsible for searching an r-range for a node set (corresponding to a keyword). The output node sets of all machines are intersected in the reduce phase as the final result. By default, the evaluation is performed on CAL, the number of query keywords is 4 and we report the average performance of 20 queries[4]. For index construction, the number of partitions is 10 and $maxR = 150000$ (150000 is a relatively large search range as the average edge length of CAL is 2650).

EXP 1: Effect of road network size. We evaluate the scalability of the methods with regard to graph size. Results are shown in the left of Fig. 3. The runtime of the index-based method only slightly increases with the increase of graph size, whereas the method without index is more sensitive to the network size.

EXP 2: Effect of the cluster size. We vary the number of datanodes (i.e., the machines handling computation) from 1 to 15 by a step size of 2. The results are shown in the

[3]http://www.dis.uniroma1.it/~challenge9/download.shtml

[4]Hadoop startup time is not included.

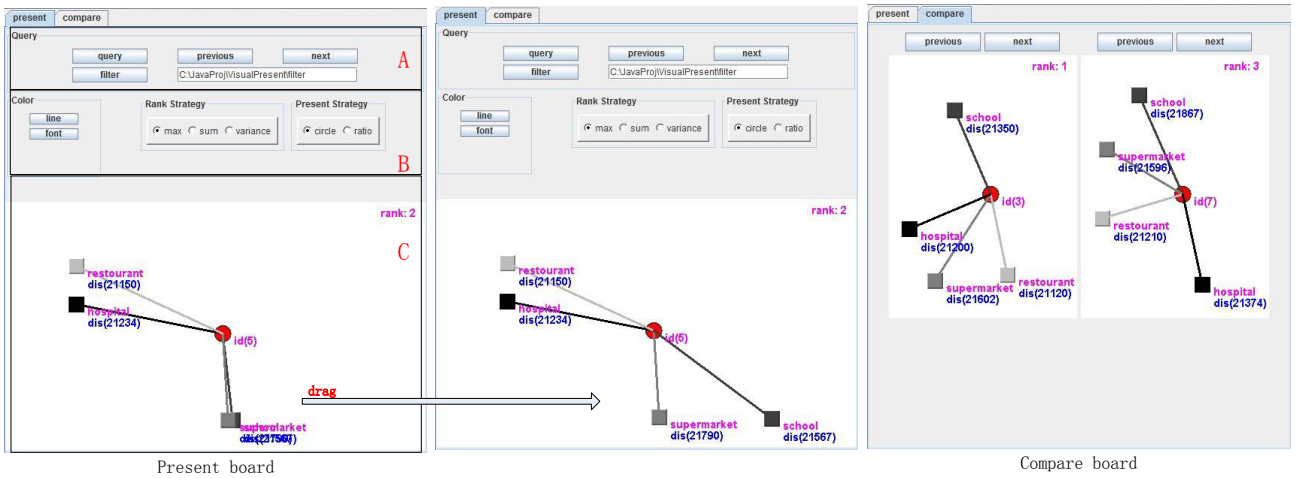

Figure 4: DISKs' visual presentation interface.

right of Fig. 3. We observe that the running time decreases as the number of datanodes increases.

4. DEMONSTRATION

The demonstration is the first system that is developed to distributedly process SGKS queries. The demonstration environment will consist of up to five laptop computers on site. Participants will be able to experience how DISKs can be used to issue queries, as well as selecting and comparing results via DISKs' interactive user interface.

4.1 Issuing and Evaluating SGKS Queries

The input is a text file containing the set of query keywords and a parameter r. It will be uploaded to a user-specified directory in HDFS of the Hadoop cluster. The index file locates in a specified directory in HDFS. The SGKS query can be performed when the input file and index file are ready. The result file will be stored under a user-specified HDFS directory. We intend to demonstrate the performance of DISKs with road networks of varying sizes and different numbers of machines, as illustrated in Fig. 3.

4.2 Visual Interactive Interface

Users can check the results via a visual interface of DISKs. The demonstration will show how DISKs interface facilitates users to examine the results. First, users can filter the results by specifying a filter file containing the set of nodes that users are interested in (the "filter" in Box A of Fig. 4) This functionality can be employed in a scenario where the users want to impose restriction on the returned locations, e.g., closing to restaurants. Second, users can also choose the ranking strategy to present the results (Box B of Fig. 4). For example, under the default ranking strategy "max", a result has a higher rank if the largest distance between the result node (in the center colored red) and the node sets is smaller. Other strategies include *sum* or *variance* of the spatial cost between the result node and the node sets. Finally, users can press "query" button (Box A) to inspect the result (Box C) satisfying the filtering condition and the selected ranking strategy.

The presentation of results highlights the spatial relationship between the result node (circle) and its closest keyword related node (rectangle) with respect to each keyword. The direction from the result node to such nodes reflects their real coordinates if the node coordinates are available. The results can be presented one by one according to their ranks from high to low by pressing "next" button in Box A. User can inspect the previous result by clicking the "previous" button. The corresponding distances and keywords are illustrated near the nodes. If the shown values are very close (left of Fig. 4), users can drag the node around for better presentation. DISKs also offers a convenient comparison board for comparing different results, and users can select any pair of results to compare, as shown in the right of Fig. 4.

5. REFERENCES

[1] Hadoop, http://hadoop.apache.org/.

[2] 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.

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

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

[5] G. Karypis and V. Kumar. A fast and highly quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998.

[6] F. Li, B. Yao, M. Tang, and M. Hadjieleftheriou. Spatial approximate string search. *IEEE Trans. Knowl. Data Eng.*, 99(PrePrints), 2012.

[7] Z. Li, K. C. K. Lee, B. Zheng, W.-C. Lee, D. L. Lee, and X. Wang. IR-Tree: An efficient index for geographic document search. *IEEE Trans. Knowl. Data Eng.*, 23(4):585–599, 2011.

[8] S. Luo, Y. Luo, and S. Zhou et al. Distributed index for keyword search. Technical Report #2011-218, School of Computer Science, Fudan University. 2011.

[9] B. Yao, F. Li, M. Hadjieleftheriou, and K. Hou. Approximate string search in spatial databases. In *ICDE*, pages 545–556, 2010.