Efficient Keyword Search on Graphs using MapReduce

Yifan Hao*, Huiping Cao*, Yan Qi†, Chuan Hu*, Sukumar Brahma*, Jingyu Han‡
*New Mexico State University, Las Cruces, NM
Email: yifan@nmsu.edu, hcao@cs.nmsu.edu, chu@cs.nmsu.edu, sbrahma@nmsu.edu
†Turn Inc., San Francisco, CA
Email: yan.qi@turn.com
‡Nanjing University of Posts and Telecommunications, Nanjing, China
Email: jyhan@njupt.edu.cn

***Abstract*—A solution of a keyword query over graphs is a Group Steiner tree, which is rooted at a node and whose nodes collectively satisfy the query (e.g. node keywords cover all the query keywords), and in which the sum of edge weights satisfies given conditions (e.g., need to be minimum or be the first K minimal among all possible sub-graphs satisfying the query). Most existing techniques for evaluating keyword queries over graphs run on a centralized computer. We propose a new approach, *SOverlapping*, to evaluate keyword queries over graphs on MapReduce framework by utilizing probabilistic theory to partition graphs. The new approach has shown to be effective and efficient when tested on real graph data sets.**

I. Introduction and Problem

Graphs have been widely used to represent objects and relationships among objects in many applications (e.g., social networks, smart grids). Keyword search, as an easy-to-use interface, over graphs has attracted increasing interest in the past decades (e.g., [1], [2]). Let G=$\langle V, E, A, W\rangle$ be a graph where one node $v \in V$ represents an object, an edge $e = (v_i, v_j) \in E(\subseteq V \times V)$ represents the connection between two objects, a node annotation $A(v_i)$=$\{k_{i1}, \cdots, k_{im}\}$ represents a set of distinct keywords associated with each node $v_i \in V$, and an edge weight $W(e)$ applies a numeric value w to an edge $e \in E$. Given a query Q=$\{k_1, k_2, \cdots, k_{|Q|}\}$ with $|Q|$ distinct keywords, a solution of the keyword query Q over a graph G is a Group Steiner tree T_{sol}=$\langle v_r, \{v_{k_1}, \cdots, v_{k_{|Q|}}\}, \omega\rangle$, which is rooted at a node $v_r \in V$, whose nodes collectively satisfy the query (e.g. node keywords cover all the query keywords, and in which the sum of edge weights satisfies given conditions (e.g., need to be minimum or be the first K minimal among all possible sub-graphs satisfying the query).

Our problem is to find the top K solutions to a keyword query Q over graphs G.

II. Proposed Solution

Many approaches were proposed to evaluate keyword queries over graph data such as the Web, XML documents, and RDF triples. Most of these existing techniques in evaluating keyword queries run on a centralized computer. The MapReduce framework has been introduced [3] as a data intensive framework. We have seen increasing interest in using the MapReduce framework to perform data mining, machine learning, online joins, and so on.

This work is supported by NSF Grants HRD-1345232 and CNS-1337884.

A. Baseline Approach

We first propose a baseline method to perform keyword queries using the MapReduce framework. To utilize MapReduce to find answers, the basic idea is to partition the problem from a larger size to a smaller size. Given this basic rationale, in the keyword search problem, we first partition (using Map functions) the large graph to multiple smaller sub-graphs and send each sub-graph to a reducer. Then, a reducer can run any existing keyword search algorithm (e.g., Best First Search) to find solutions from one small graph. However, running these basic Map and Reduce functions cannot find solutions which cross multiple partitions. To overcome this issue, we design an iterative scheme and call it *SIteration*.

SIteration first runs the above mentioned Map and Reduce functions to find solutions that can be found from one individual graph partition. At the same time, the Reduce functions also find partial solutions from each individual graph partition. Then, *SIteration* sends the partial solutions to other reducers to grow these partial solutions iteratively. A partial solution may become a full solution after the growing. In particular, if a solution crosses two graph partitions, it can grow to a full solution after one iteration; if a solution crosses multiple (e.g., l) graph partitions, it needs multiple (e.g., $l-1$) iterations to grow to a full solution. The *SIteration* generally needs multiple iterations to get the desired solutions.

B. SOverlapping: Search with MapReduce by Creating Overlapping Graph Partitions

The baseline *SIteration* is expensive due to the multiple iterations and communications caused by these iterations. To overcome this problem, we propose a new approach to perform keyword search over graphs using the MapReduce framework. The major contribution of the *SOverlapping* approach is that it creates overlapping graph partitions such that the solutions (even the cross-block solutions) of a query can be directly

978-1-4799-9926-2/15/$31.00 ©2015 IEEE

calculated from the overlapped graph partitions. We present the technical details of this approach in what follows.

SOverlapping consists of two steps, *offline graph partitioning* and *online searching*.

1) Offline graph partitioning: The offline graph partitioning step partitions the original graph to multiple blocks (i.e., subgraphs) and utilizes such partitions when running the search function. The overlapping of graph blocks introduces another problem that extra effort is needed to transfer the overlapped data. When the overlapping degree is higher, more data need to be dispatched from mappers to reducers. A fundamental issue in designing *SOverlaping* approach is to decide the degree of overlapping. We propose a heuristic approach to calculate the overlapping threshold based on the central limit theorem (CLT) and the three-sigma rule (or 68-95-99.7 rule) in probability theory. The CLT states that, given a certain condition, the mean of a sufficiently large number of independent random variables approximately follows normal distribution. The three-sigma rule states that nearly all values which follow a normal distribution $\mathcal{N}(\mu, \sigma^2)$ lie within $3 \cdot \sigma$ of the mean μ.

The procedure of calculating the overlapping threshold τ_Q for any query Q with m keywords is as follows. First, a large number (s) of sample queries are generated and are evaluated over the graphs (or a smaller partition of the graph) to get the top-K solutions for each query. Then, for each sample query, the score of the longest path in the K-th solution is calculated and recorded. The mean of these s values follows a normal distribution based on CLT. Second, from these s values, we calculate the mean μ and the standard deviation σ. We set the overlapping threshold τ_Q to be $\mu + 3 \cdot \sigma$. Based on the three-sigma rule, with 99.7% probability, the score of the longest path in the top-K solution for any given query Q is less than τ_Q. Such threshold is calculated only once for any graph.

In our experiments, we set the number of sample queries $s = 1000$. This s is chosen to make the sample queries statistically meaningful. For each $|Q|$, we generate 1000 sample queries with $|Q|$ keywords, calculate their top-K solutions, and extract the score for the longest path in the top-K solutions for each query. After getting the 1000 values for the sample queries, we discretize them into equi-width histograms and plot the ratio of the number of values in each bucket to the total number of values with a smooth curve. The results are shown in Figure 1, which figuratively illustrate that these values follow normal distributions.

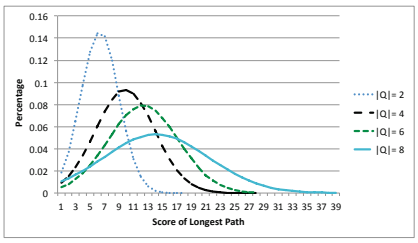


Fig. 1. Score of the longest path for the top-K solutions for sample queries with different number of distinct keywords

2) Online Searching: With the overlapping graph partitions, the second step, *online searching algorithm* just needs to start one MapReduce job which runs the Map (Fig. 1) and Reduce (Fig. 2) functions respectively. The mappers send the overlapping graph partitions to different reducers. One reducer then calculates the top-K solutions for individual graph blocks (which overlap with other blocks). The results from all the reducers are combined to a final MapReduce job.

Algorithm 1: Map(bid, $object$)

Data: $object$ can be a node or an edge
1 EMIT(bid, $object$) to send graph objects to reducers

Algorithm 2: Reduce(bid, $object$)

1 Get query Q from the configuration of the MapReduce job
2 Construct a graph g from the objects (nodes and edges) sent to this reducer `/* This g is a sub-graph which overlaps with other sub-graphs sent to other reducers. */`
3 $\mathbb{S}$ = BFS(Q, g) `/* Calculate all the solutions from g */`
4 **for** *every solution* $T_{sol} \in \mathbb{S}$ **do**
5 | EMIT(-1, T_{sol}) to send T_{sol} to the final MapReduce job
6 **end**

III. Experimental Results

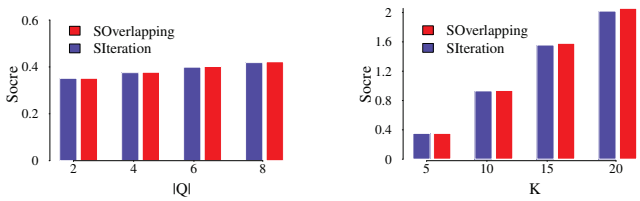
Experiment environment. We set up a Hadoop MapReduce cluster with eleven computers, each of which is configured with two Intel dual-core T9600 2.8GHz processors, 4GB of memory, 500GB SATA hard disk, and gigabit ethernet. The two approaches are implemented in Java.

Data. We got the DBLP XML file and extracted two graphs. *DBLP-ac* treats articles, authors, and publication venues (e.g., proceedings, journals) as nodes and their connections (crossref, authors write articles) as undirected edges. This graph contains about $4M$ nodes and $7M$ edges. *DBLP-finest* treats every element (e.g., authors, publication venues, articles) in the XML file as nodes and their connections (element embedding, crossref) as undirected edges. This graph contains about $10M$ nodes and $22M$ edges. For each graph, we generate edge weights, which are scalar values following uniform distribution in the range of $(0, 1.0]$.

Query. We generate random keyword queries with distinct number of keywords $|Q|$ to search graphs.

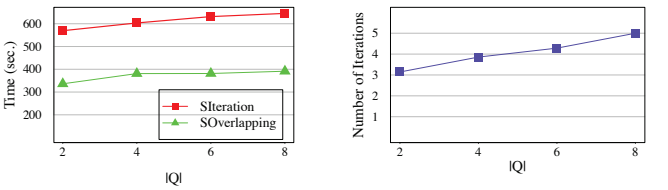
A. Effectiveness

As discussed in Section II, the *SIteration* approach calculates the correct top-K solutions since it is a natural extension of the best first search paradigm. However, *SOverlapping* may not get the exact top-K solutions due to the probabilistic nature of the generation of the overlapping sub-graphs. We show through experiments that the results of *SOverlapping* is comparable to the correct results generated by *SIteration*. For each query, after getting its top-K solutions, we calculate the summation scores of these top-K solutions and plot these scores for the two approaches *SIteration* and *SOverlapping*.

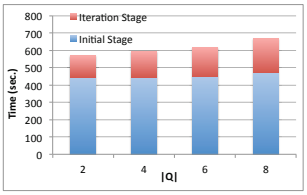


(a) Score vs. $|Q|$ (b) Score vs. K

Fig. 2. Compare *SIteration* and *SOverlapping* for *DBLP-ac*



(a) *DBLP-ac* (b) # of iterations in *SIteration*



(c) *SIteration* time

Fig. 3. Compare *SIteration* and *SOverlapping* (Time vs. $|Q|$): fix K to be 5

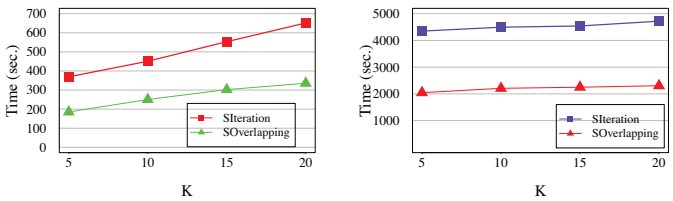
Figure 2(a) shows the results over the graph *DBLP-ac* when we vary the number of distinct keywords $|Q|$ from 2 to 8 (by fixing K to be 5). The results show that the average score of the top-K solutions are almost the same. We also plot the score values when varying the value of top-K for the fixed number of query keywords ($|Q| = 2$). Figure 2(b) shows the results over the graph *DBLP-ac*. We can observe that *SOverlapping* can find solutions which are comparable to those of *SIteration.*

B. Efficiency

We compare the execution time of the two approaches *SIteration* and *SOverlapping* by running different queries with different number of distinct keywords and the top-K values.

Figure 3 shows the comparison of the two approaches when we vary the size of queries $|Q|$ and fix the top-K value. First, as shown in Figure 3(a), *SOverlapping* runs much faster (half of the time) than the *SIteration*. This is because the *SIteration* approach gets the solution using multiple iterations. As shown in Figure 3(c) that most time are used in the initial stage which runs the basic best first search paradigm. The running time for the iteration stage grows slightly with the number of iterations used, which are shown in Figure 3(b).

Figure 4 shows how these approaches are affected by the value of K on both *DBLP-ac* and *DBLP-finest*. In the *DBLP-ac* graph, larger K needs more time to finish. This is because getting more solutions (for larger K), the program needs to explore a larger graph space. However, in *DBLP-finest*, the value of K does not affect the running time too much. This is because the time is more dominated by graph reading and



(a) *DBLP-ac* (b) *DBLP-finest*

Fig. 4. Compare *SIteration* and *SOverlapping* (Time vs. K): fix $|Q|$ be 2

information sending between mappers and reducers while the solution exploration time is not dominating any more.

IV. Related Works

Keyword search over relational databases (RDB) by viewing RDBs as graphs has gained much attention in the past decade. A RDB is viewed as a graph where each tuple is treated as a node and the directed edges capture the foreign key relationships among tuples or constraints. Along this direction, several works (e.g., BANKS [4], DBXplorer [5], DISCOVER [6]) and many others study the problem in a setting with heterogeneous relational databases. More recently, there are more keyword search variants, e.g., searching interval-weighted graphs [1], supporting taxonomy keyword search [7], performing k-nearest keyword query over graphs, etc.

The MapReduce framework has been introduced [3] as a data intensive framework. We have seen increasing interest in using the MapReduce framework to perform data mining, machine learning, online join operator evaluations (e.g., [8]–[10]), spatial keyword queries [11], and so on. The most related work is [12], which developed an index of graphs using MapReduce. However, the keyword query is not supported using the MapReduce framework. Our approaches are different in that we develop general frameworks that allow performing keyword queries over graphs using the MapReduce framework.

References

[1] H. Cao, K. S. Candan, and M. L. Sapino, "Skynets: searching for minimum trees in graphs with incomparable edge weights," in *CIKM*, 2011.
[2] H. He, H. Wang, J. Yang, and P. S. Yu, "BLINKS: ranked keyword searches on graphs," in *SIGMOD*, 2007.
[3] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *OSDI*, 2004.
[4] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan, "Keyword searching and browsing in databases using BANKS," in *ICDE*, 2002.
[5] S. Agrawal, S. Chaudhuri, and G. Das, "DBXplorer: A system for keyword-based search over relational databases," in *ICDE*, 2002.
[6] V. Hristidis and Y. Papakonstantinou, "DISCOVER: Keyword search in relational databases," in *VLDB*, 2002.
[7] B. Ding, H. Wang, R. Jin, J. Han, and Z. Wang, "Optimizing index for taxonomy keyword search," in *SIGMOD*, 2012.
[8] Y. N. Silva and J. M. Reed, "Exploiting mapreduce-based similarity joins," in *SIGMOD Conference*, 2012.
[9] A. Okcan and M. Riedewald, "Processing theta-joins using mapreduce," in *SIGMOD Conference*, 2011.
[10] W. Lu, Y. Shen, S. Chen, and B. C. Ooi, "Efficient processing of k nearest neighbor joins using mapreduce," *PVLDB*, vol. 5, no. 10, 2012.
[11] W. Li, W. Wang, and T. Jin, "Evaluating spatial keyword queries under the mapreduce framework," in *DASFAA Workshops*, 2012.
[12] M. Zhong and M. Liu, "A distributed index for efficient parallel top-k keyword search on massive graphs," in *International Workshop on Web Information and Data Management (WIDM)*, 2012.