2012 IEEE 28th International Conference on Data Engineering

# Efficient Top-$k$ Keyword Search in Graphs with Polynomial Delay

Mehdi Kargar and Aijun An

*Department of Computer Science and Engineering, York University*
*4700 Keele Street, Toronto, Canada*
{kargar,aan}@cse.yorku.ca

***Abstract*—A system for efficient keyword search in graphs is demonstrated. The system works in two steps: a search through only the nodes containing the input keywords for a set of nodes that are close to each other and together cover the input keywords and an exploration for finding how these nodes are related to each other. The system generates all or top-$k$ answers in polynomial delay. Answers are presented to the user according to a ranking criterion so that the answers with nodes closer to each other are presented before the ones with nodes farther away from each other. In addition, the set of answers produced by our system is duplication free. The system uses two methods for presenting the final answer to the user. The presentation methods reveal relationships among the nodes in an answer through a tree or a multi-center graph. We will show that each method has its own advantages and disadvantages. The system is demonstrated using two challenging datasets, very large DBLP and highly cyclic Mondial. Challenges and difficulties in implementing an efficient keyword search system are also demonstrated.**

## I. Introduction

Recently, keyword search has been extensively used for extracting information from structured data such as relational databases, which can be modeled as graphs. Users of such databases usually do not have sufficient knowledge about the structure of data, and are often not familiar with query languages such as SQL. Thus, they need a simple system that receives some keywords and returns a set of nodes or tuples that together cover all or part of the input keywords. A node that contains one or more keywords is called a *content node*.

Most of the existing methods for keyword search in graphs output either minimum connected trees or subgraphs that cover all or part of the input keywords. Both types of results show not only the content nodes that contain the input keywords but also some other nodes that connect the retrieved content nodes, which reveal some relationships among the content nodes. However, the tree-based methods have been criticized for producing too many trees with the same set of content nodes (albeit with different connections) [1]. Often, a user prefers to see different sets of content nodes that together cover the keywords and may not want to see the different relationships among the same set of content nodes. A graph-based method can reveal multiple relationships within a single search result, and thus reduces the duplicated results. However, as we will show in this demo, the graph-based method may produce results that contain more irrelevant nodes than tree-based results. In addition, in the current tree or graph based results, while some content nodes are close to each other, others may be far way from each other, meaning that weak relationships among content nodes may exist in the results. More importantly, current tree or graph based methods search through both content and non-content nodes in the graph for answers, which is very time-consuming when the input graph is large e.g., containing millions of nodes.

In [2], we proposed a keyword search method that finds $r$-cliques from an input graph. An $r$-clique is a set of content nodes that cover all of the input keywords and whose shortest distance between each pair of nodes is no larger than $r$. We developed two algorithms, an exact algorithm that finds all the $r$-cliques and an approximation algorithm that find top-$k$ answers with polynomial delay [2]. Our method has the following unique features. First, it only searches through the content nodes in the input graph to find $r$-cliques. This reduces the search space by orders of magnitude and thus greatly speeds up the search process. Second, by finding $r$-cliques, all the nodes in a search result are close to each other. The distance between each two nodes is bounded by $r$ for the exact algorithm and $2r$ for the approximation algorithm. Third, our method can display a search result in different ways. The method can show an $r$-clique to the user as a set of content nodes. But if the user wants to see relationships among the content nodes in an $r$-clique, our method further finds a minimum connected tree that connect all the nodes in the $r$-clique. Since a minimum connected tree is produced based on a retrieved $r$-clique, our method does not produce multiple trees with the same set of content nodes as long as the produced $r$-cliques are unique. In addition, the process of first finding an $r$-clique and then a minimum connected tree based on the $r$-clique is much faster than finding a tree from the input graph based on the input keywords because only content nodes are explored when finding $r$-cliques.

In this demo, we will showcase these features of our system. We also added to our system a procedure that displays an $r$-clique using a multi-centered graph [1], which can show multiple relationships among the content nodes in the $r$-clique. In the demo, we can show both trees and graphs as search results so that the user can see the advantages and disadvantages of each result-presenting method. Two challenging datasets will be used in our demonstration. One is DBLP[1] which has more

[1] http://dblp.uni-trier.de/xml/

1084-4627/12 $26.00 © 2012 IEEE
DOI 10.1109/ICDE.2012.124

than 7 million nodes, and the other is Mondial[2] which is very cyclic and has a complicated schema. We will show that our system efficiently finds answers on both datasets.

## II. Definitions and Problem Statement

A graph can model different types of data objects and their relationships. A node in a graph represents an entity such as a web page, an article, an author or a tuple in a relational database. An edge between two nodes represents a relationship between the two entities such as the author-article relationship and the foreign key relationship in relational databases. A graph can be directed or undirected. The edges or nodes may have weights associated with them. Since undirected graphs can be used to model different types of unstructured, semi-structured, and structured data, we consider undirected graphs with weights on the edges. An edge weight represents the distance between the two corresponding nodes.

Given a graph $G$ and a set of input keywords ($Q = \{k_1, k_2, \ldots, k_l\}$), an **$r$-clique** of $G$ with respect to $Q$ is a set of content nodes in $G$ that together cover all of the input keywords in $Q$ and in which the shortest distance between each pair of the content nodes is less than or equal to $r$. Suppose that the nodes of an $r$-clique are denoted as $\{v_1, v_2, \ldots, v_l\}$. The **weight of the $r$-clique** is defined as

$$weight = \sum_{i=1}^{l} \sum_{j=i+1}^{l} dist(v_i, v_j)$$

where $dist(v_i, v_j)$ is the shortest distance between $v_i$ and $v_j$, i.e., the sum of weights on the shortest path between $v_i$ and $v_j$ in $G$.

Given a graph $G$, a set $Q$ of input keywords and a distance threshold $r$, our problem is to find top-$k$ $r$-cliques in $G$ with respect to $Q$ in weight-increasing order. In [2], we proved that the problem of finding an $r$-clique with the minimum weight is NP-hard, and thus an approximation algorithm with 2-approximation ratio was proposed. The algorithm finds an answer whose weight is at most twice of that of an optimal $r$-clique. We further uses a procedure that systematically divides the search space and calls the approximation algorithm to find top-$k$ answers in polynomial delay.

To illustrate the advantages of our algorithm for finding $r$-cliques, we also implemented two other algorithms. One produces a multi-centered community with the minimum center distance [1] and the other produces distinct root trees [3]. The center distance of an answer is the sum of the distances from one node in the graph to each content node in the answer. We will show in this demo that our algorithm is faster and produces duplication-free answers.

## III. System Architecture

The architecture of our system for finding top-$k$ $r$-cliques is shown in Figure 1. It consists of five functional components and four data components. Below we first describe the data components and then briefly describe the functional components.

[2]http://www.dbis.informatik.uni-goettingen.de/Mondial/

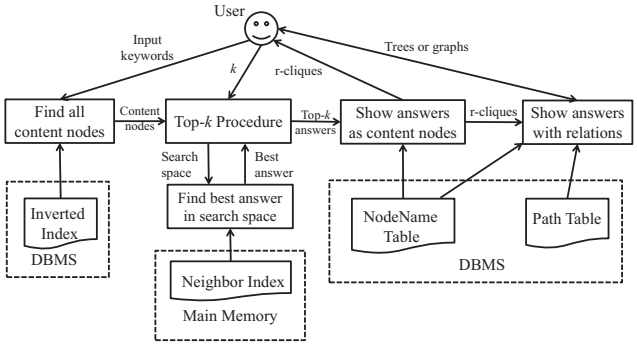


Fig. 1. System Architecture.

### A. Indexing and Storage of Data

For an input graph $G$, the following four data structures are pre-built to facilitate the search process:

*1) Inverted Index:* For each keyword in $G$, the IDs of the nodes that contain the keyword are stored in this index, which is maintained and accessed through a DBMS.

*2) Neighbor Index:* For each node $n$ in $G$, a *neighbor list* that contains the IDs of $n$'s neighbors and the shortest distances from $n$ to each neighbor is stored in this index. A node that is within $R$ distance from node $n$ is called a *neighbor* of $n$, where $R$ is a distance threshold. The purpose of building a neighbor index is to quickly obtain the shortest distance between two nodes during the search process, since computing the shortest distance on-the-fly is expensive. By using a distance threshold $R$, the shortest distances between nodes that are far way from each other are not stored in the index. Since our objective is to find $r$-cliques in which each pair of nodes are within $r$ distance from each other, $R$ should be bigger than $r$. The setting of $R$ value requires the estimation of possible $r$ values based on the graph structure and user preferences and may be estimated using the domain knowledge. At the same time, we should set it as small as possible to keep the index in a feasible size. The idea of indexing the graph using a distance threshold is also used in [4] [1] but with different methods.

*3) NodeName Table:* For each node in $G$, the textual information contained in the node is stored in this table. For example, in the DBLP dataset, the titles of the papers and the names of authors are stored in this table. In addition, the type of a node (e.g author or paper) is also stored. This table is maintained and accessed through a DBMS. It is used only when the search results are presented to the user.

*4) Path Table:* This table is used to retrieve the shortest path between a pair of nodes when finding the minimum connected tree for an $r$-clique. Similar to the neighbor index, for each node $n$ in $G$, a list of $n$'s neighbor IDs is stored in the path table. Also, for each neighbor $m$ of $n$, the pointer to the node right before $m$ on the shortest path from $n$ to $m$ is stored.

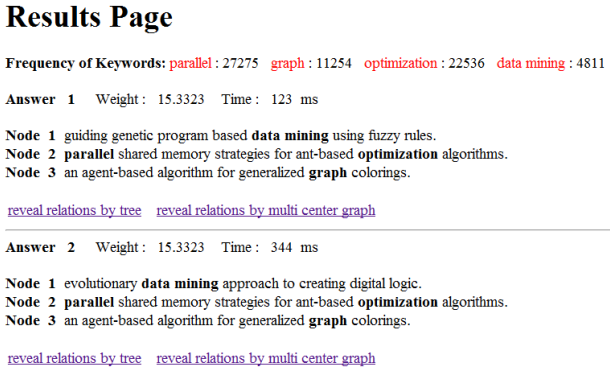


Fig. 2. Display of top-2 answers as sets of content nodes.

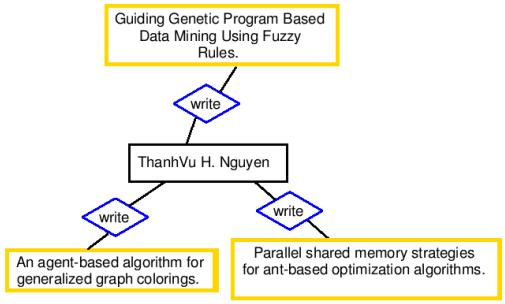


Fig. 3. Reveal relation by tree.

This node must be on $n$'s neighbor list. To find the shortest path between $n$ and $m$ from the path table, node $m$ is first located in $n$'s neighbor list. Following the pointer in $m$, the node $p$ right before $m$ on the shortest path from $n$ to $m$ is located. Following the pointer in $p$, the node right before $p$ on the path is located. This process continues until $n$ is reached. The path table is used only when presenting the search result using trees or graphs. It is maintained and accessed through a DBMS.

The only data that resides in main memory is the *neighbor index*. The rest of the information is managed in a DBMS, which is MySQL in our implementation. The advantage of managing most of the data in a DBMS is to reduce memory consumption by the search algorithm [5]. Also, note that the information stored in the path table can be put in the neighbor index so that the two structures can be combined into one. However, since the information in the path table is only used after the search for presenting the relationship among the nodes in a search result, separating them into two structures makes the neighbor index smaller so that the memory requirement for the search process is reduced.

### B. The Search Algorithm

After receiving the query keywords from the user, for each keyword our system finds the IDs of the nodes that contain the keyword using the inverted index. The found IDs are stored in a map in which the key is the keyword and the value is a list of node IDs containing the keyword. This map is passed to the procedure for finding top-$k$ answers in polynomial delay. Briefly, this top-$k$ procedure first calls a polynomial procedure for finding the best answer in the whole search space, then divides the search space into disjoint sub-spaces according to the best answer and repetitively calls the polynomial procedure to find the best answer in each sub-space. The answers from all the sub-spaces compete with each other for the next best answer. The procedure further divides the sub-search space from which the last best answer was found into sub-sub-spaces to find the best answer in each sub-sub-space. The process goes on until top-$k$ answers have been found. Clearly, the top-$k$ procedure only searches through the set of the content nodes, and the nodes that do not contain the input keywords are ignored in the search process. This is the main reason why our system is fast.

The details of our procedure for finding the best answer is described in [2]. It is a 2-approximation algorithm that finds an answer whose weight is at most twice that of the optimal $r$-clique in the given search space. The top-$k$ procedure used in this demo is an improved version of the top-$k$ procedure described in [2]. The top-$k$ procedure in [2] may produce duplicated answers when a node in the graph contains more than one input keyword although the answers are unique in terms of keyword-node coupling (which is considered to be duplication-free in [1]). The improved top-$k$ procedure used in this demo is completely duplication free (i.e., the set of content nodes in an answer is unique compared to other answers in the top-$k$ list). This is achieved by more wisely dividing the search space after a best answer is found so that the set of nodes in an already-generated answer is excluded for further consideration.

### C. Presenting the Answers to the User

Once an answer is found in the top-$k$ procedure, it is presented to the user as an $r$-clique containing a set of content nodes. The textual content of each content node is retrieved from $NodeName$ table based on the node ID, and is presented to the user. Figure 2 shows the top 2 answers for a query consisting of keywords $parallel$, $graph$, $optimization$ and *data mining* on the DBLP data set where each node corresponds to a paper or author and an edge between two nodes represents the citation or authorship relation[3].

If the user is interested in seeing the relations between the content nodes in an answer (i.e., how these nodes are connected in the input graph), he/she can click on "*reveal relations by tree*" or "*reveal relations by multi-center graph*" beneath the answer to obtain a minimal connected tree (called *Steiner tree*) or a multi-center graph (called *community*). A Steiner tree of an $r$-clique with respect to a graph $G$ is a subtree in $G$ that contains all the nodes in the $r$-clique with the minimum sum of edge weights. The procedure for generating

[3]In the data sets we used, the weight of the edge between two nodes $v$ and $u$ is $(\log_2(1+v_{deg}) + \log_2(1+u_{deg}))/2$, where $v_{deg}$ and $u_{deg}$ are the degrees of nodes $v$ and $u$ respectively [1], [6].

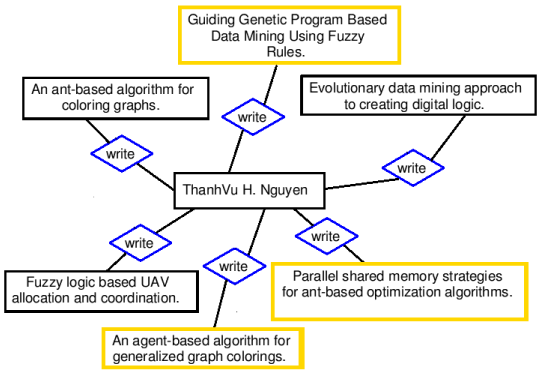


Fig. 4. Reveal relation by multi center sub-graph.

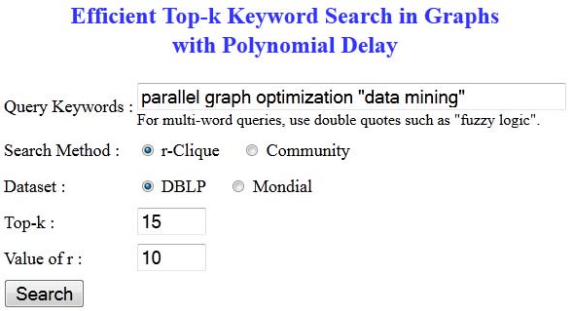


Fig. 5. First page of the system with parameters.

a Steiner tree given an $r$-clique and a graph $G$ is presented in [2]. The procedure for generating a multi-center graph (i.e., *community*) given an $r$-clique and an input graph $G$ is Algorithm 4 in [1] with the $r$-clique being the *core* of the community. A center in a community is a node in the input graph that is within $r$ distance from each of the content node in the $r$-clique. A multi-center graph contains all the centers that satisfy such a condition.

To generate a Steiner tree or a community, the system uses the $NodeName$ table for finding the details of each node and the $Path$ table for retrieving the shortest paths between nodes. Our core system of graphical presentation is based on $GoogleChart$[4]. A specific chart type is called GraphViz charts and is based on the GraphViz engine[5]. It is suitable for presenting different trees and graphs. Figure 3 and 4 show the Steiner tree and the community, respectively, for the first answer in Figure 2. Figure 3 reveals that the three papers in the answer share a common author *ThanhVu H. Nguyen*. Figure 4 further reveals that *ThanhVu H. Nguyen* also wrote another three papers. In this graph both the author node and the additional paper nodes are the centers of the community. As can be seen, both the tree and the multi-center graph reveal some relations among the content nodes in the answer with the community revealing more relations. In [2], we showed that although more relations can be revealed by a community, irrelevant nodes (such as the papers irrelevant to the search keywords) are often included in the community, which may make the answer harder to understand. In this demo, we will show the pros and cons of both tree and community presentation methods.

## IV. The Demonstration

We developed a web based browsing interface using Java to support interactive keyword search in graph data sets. The first page of the interface is shown in Fig. 5, where the user can specify query keywords and other parameters of the system (such as $k$ and $r$). The user can also choose the search method and the data set from this page. In addition to the $r$-clique based search method, we also implemented a graph-based method [1] that searches for communities directly from the input graph based on the input keywords. Note that this search method includes non-content nodes in the search space, while the $r$-clique method searches through only the content nodes for answers. The $r$-clique only looks at the non-content nodes for revealing the relations among the content nodes when presenting the answer as a tree or multi-center sub-graph. This search method is provided for the purpose of comparison to our $r$-clique based method.

In the demonstration, the user is able to compare the three search methods in the following aspects:

1) *Speed*. The $r$-clique method is faster than the graph based search methods.
2) *Duplication free*. All the answers in the top-$k$ list produced by the $r$-clique based method are unique. while the answers from the other method may not.
3) *Final answer presentation*. Two types of output for presenting an $r$-clique are provided for the users. One is a minimum connected tree and the other is a multi-center graph. Users are able to evaluate the advantages and disadvantages of each presentation method.

In addition to single-word keywords, we extended our system to allow a keyword to contain more than one word, such as ”data mining”, by extending the inverted index to include the positions of the words in text. This helps produce more meaningful and relevant answers.

[4]http://code.google.com/apis/chart/
[5]http://www.graphviz.org/

## References

[1] L. Qin, J. Yu, L. Chang, and Y. Tao, “Querying communities in relational databases,” in *Proc. of ICDE’09*, 2009, pp. 724–735.
[2] M. Kargar and A. An, “Keyword search in graphs: Finding rcliques,” in *Proc. of VLDB’11 (to appear )*, 2011.
[3] H. He, H. Wang, J. Yang, and P. Yu, “Blinks: ranked keyword searches on graphs,” in *Proc. of SIGMOD’07*, 2007, pp. 305–316.
[4] G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou, “Ease: Efficient and adaptive keyword search on unstructured, semi-structured and structured data,” in *Proc. of SIGMOD’08*, 2008, pp. 903–914.
[5] H. Achiezra, K. Golenberg, B. Kimelfeld, and Y. Sagiv, “Exploratory keyword search on data graphs,” in *Proc. of SIGMOD’10*, 2010, pp. 1163–1166.
[6] G. Bhalotia, C. Nakhe, A. Hulgeri, S. Chakrabarti, and S. Sudarshan, “Keyword searching and browsing in databases using banks,” in *Proc. of ICDE’02*, 2002, pp. 431–440.