Available online at www.ijpe-online.com

International Journal of Performability Engineering

vol. 14, no. 4, April 2018, pp. 656-664
DOI: 10.23940/ijpe.18.04.p8.656664

Keyword Query based on Hypergraph in Relational Database

Yingqi Wang, Lianke Zhou*, and Nianbin Wang

School of Computer Science and Technology, Harbin Engineering University, Harbin, 150001, China

Abstract

Recently, the keyword query in relational databases has received widespread attention. The traditional methods typically traverse the entire database for the final results. With the database, structure becomes more complex and its size increases quickly; the efficiency of above methods cannot be ensured. To solve this issue, we propose a hypergraph-based keyword query method. First, the concept of hypergraph is formally defined to model the relational database. Second, the strategy of multi-granular index construction is presented to prune the irrelevant supernodes. Then, a filtering-validating query method is put forward based on the above index. Finally, experiments are taken on the dataset DBLP to verify the validity of the proposed method.

Keywords: relational database; keyword query; hypergraph; multi-granular index

(Submitted on December 21, 2017; Revised on January 30, 2018; Accepted on March 3, 2018)

1. Introduction

In information retrieval, the keyword query method has been widely applied according to its simplicity and ease of use [7,15,19]. In recent years, it has been introduced into the relational database and become mature [11,16]. Thanks to the method, users can get desired results by submitting several keywords to the query system without knowing the underlying schema of the database and structured query language, which reduces the threshold to use relational databases. However, the keyword query as a fuzzy query. If it is directly applied to the relational database, there will be many problems. Therefore, research on the keyword query algorithm in relational databases is significant.

Existing query methods can be divided into two categories: query based on schema graph and query based on data graph [3,4,14,18,20]. The former models the database by using data tables as nodes and primary-foreign key relationships between the tables as edges. Based on the above schema graph, we can enumerate all possible candidate networks to instantiate a set of SQL queries. While the latter models the database by using tuples as nodes and primary-foreign key relationships between the tuples as edges, it finds the minimum join tree that contains the keywords in the data graph. Both of the above methods need to traverse the entire database for each query. When the size of the database increases, the query efficiency of these methods will be greatly affected. In order to solve the above problem, this paper proposes a keyword query method HG-B (**h**yper**g**raph-**b**ased) based on the hypergraph model in relational databases. First, we formally define the concept of hypergraph and use it to model the relational database. Second, we present the construction strategy of multi-granular index based on the hypergraph model. Finally, we use the filtering-validating query method to improve the query efficiency.

The main contributions of the paper are reflected as follows:

(1) We propose the concept of hypergraph based on the schema graph and data graph to model the relational database.
(2) We present the construction strategy of multi-granular index based on the hypergraph model then further decrease the query scope and refine results through the coarse-granular index and fine-granular index.
(3) We put forward the filtering-validating query method based on the hypergraph and multi-granular index.

* Corresponding author.
E-mail address: zhoulianke@hrbeu.edu.cn

(4) We perform the experiments on the real data set DBLP [5] to verify the effectiveness of the method.

Section 2 of the paper introduces the relevant research on the keyword query in relational databases. Sections 3 and 4 describe the construction strategy of the multi-granular index and the filtering-validating query method in detail. Section 5 shows the experimental results. Section 6 summarizes the whole paper and points out the future research directions.

2. Related work

In recent years, many research institutes and researchers have conducted in-depth research into the keyword query methods in relational databases. Existing methods can be divided into two types according to the modeling approach: query methods based on schema graph and query methods based on data graph.

- **Keyword query based on schema graph**

References [1,8,13] use the schema graph to query, which mainly includes the following steps: 1. Enumerate the candidate networks using the database schema. 2. Transform the keywords into SQL statements according to the candidate networks obtained from Step 1. 3. Query databases using SQL directly and return the results to users. In 2002, Agrawal et al. proposed a search system DBXplorer. This system finds the keywords in databases through the symbol table. It enumerates all the tuple connection trees using the database schema. Hristidis et al. put forward the DISCOVER system which enumerated candidate networks using the breadth-first traversal strategy and built SQL statements to return query results. In 2007, Luo Yi et al. presented the algorithms Skyline Sweeping and Block Pipeline based on the concept of virtual document, which decreased the time accessing database. Such methods store the abstract structural information in memory and cost little memory space, but the generation of candidate networks requires more time overhead. In addition, when the database contains a large number of data tables and its structure is complex, many candidate networks eventually generate empty results, which also lead to a loss of query efficiency.

- **Keyword query based on data graph.**

References [2,6,9,12,17] conduct the keyword query based on the data graph to enumerate the Steiner trees. It mainly includes the following two steps: 1. Search for the nodes containing keywords; 2. Traverse the data graph to find the Steiner trees containing the keyword nodes and sort the results. In 2002, Aditya et al. proposed the BANKS system using the reverse graph lookup. It traverses the data graph by running the Dijkstra shortest path algorithm multiple times. In 2005, Kacholia et al. further presented a bidirectional search system BANKS-II improving the query performance. In 2007, Ding Bolin et al. put forward a dynamic programming method to return the top-*k* tuple connection trees. In 2008, Li Guoliang et al. came up with the concept of *r*-radius Steiner trees, which returned the query results containing semantic information. In 2009, Jeffrey Xu Yu et al. defined the concept of community based on the Steiner tree; this structure fully reflects the relationships between tuples so as to express the semantic information of results more completely and comprehensively. In 2012, Mehdi Kargar et al. presented the *r*-clique method that used polycentric graphs to represent the relationships between content nodes [10]. However, these methods need to traverse the entire data graph. When the size of the database is large, the query efficiency will also be affected greatly.

In order to solve the above problems, the paper proposes the concept of hypergraph to model the database combining the schema graph with the data graph. In the meantime, it presents the construction strategy of the multi-granular index and the filtering-validating query algorithm to reduce the query scope and improve the query efficiency.

3. Construction strategy of index

3.1. Database modeling based on hypergraph

Before introducing the construction strategy of the index, we model the database using the hypergraph. The formal definition of hypergraphs is as follows.

Define 1 Hypergraph. A hypergraph $H = (V, E)$ comprises a set of supernodes and superedges, as shown in Figure 1.

Supernode: The original data graph can be divided into multiple clusters through a Hub-based breadth-first search method HBFS(Hub Breadth First Search). Each cluster is represented by a node called supernode.

Each supernode contains a set of original nodes called internal nodes.

Superedge: The edge between supernodes is called superedge. It can be constructed by the following rules: If there is at least one internal node in the supernode SN_1 connected to an internal node in the supernode SN_2, then there is one superedge between them.

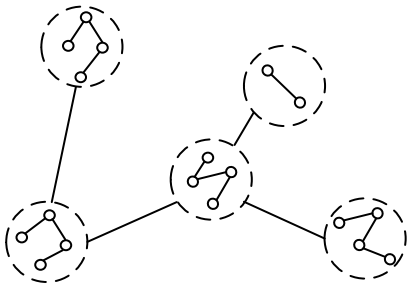

Figure 1. Hypergraph model

We use a Hub-based breadth-first search method HBFS to build the hypergraph model. The method is based on the breadth-first traversal and combines with the concept of Hubs. There are several nodes with higher degrees (Indegree and Outdegree) in the data graph. They are located in the core of the graph and have higher utilization; they are called Hubs. Centered with Hubs, we build a cluster around them to partition the graph, which can provide more sufficient location information for the query. The specific implementation process is shown in the algorithm 1. Define the degree $d(v) = |K(v)|$ ($K(v)=\{w|(v,w) \in E\}$) for the node v.

Algorithm 1 HBFS
Input: G, k, t, Set of Hubs HS
Out: $SN_1, SN_2, \cdots, SN_k$
Method:
1. $SN_i \leftarrow$ A cluster
2. $S_i \leftarrow$ A list of cluster candidates
3. for all SN_i do
4. Add k Hubs in HS as root Hubs to SN_i
5. Add the neighbors of the root Hubs to SN_i
6. while SN_i not full do
7. for all $v \in SN_i$ do
8. for all $u \in K(v) \wedge u$ not visited do
9. Set u visited
10. $d_{SN_i}(u) = |K(u) \bigcap SN_i|$
11. $S \leftarrow S \bigcup \{u\}$
12. end for
13. end for
14. Order S by $d_{SN_i}(u)/d(u)$
15. Add top-t of S to SN_i and remove from S
16. Set $S \setminus SN_i$ unvisited
17. end while
18. end for
19. Add unreachable nodes to the smallest SN
20. Return $SN_1, SN_2, \cdots, SN_k$

Steps 4 and 5 put the k Hubs in HS and the neighbors of them into each supernode SN_i. Perform the breadth-first search algorithm taking the above k Hubs as starting points. Steps 7~15 take the nodes ranked top-t in $d_{SN_i}(u)/d(u)$ as the latest traversal nodes and add them to the supernode SN_i. Then remove them from the set S. Step 16 marks the nodes in

the set $S \setminus SN_i$ as unvisited. The nodes that have not been traversed are added into the smallest supernode in Step 19. And ultimately return k supernodes. With $d_{SN_i}(u)/d(u)$, the algorithm can measure the distance between the node u and current supernode SN_i more accurately.

Below is an example to illustrate the specific implementation of the algorithm 1. The original data graph is as shown in Figure 2a. Suppose k equals 2, nodes 1 and 6 with the highest degree in the graph are selected as Hubs. Then, start from the node 1 and add its direct neighbors to the supernode SN_1. If the number of nodes in supernode SN_1 reaches the upper limit, the breadth-first search algorithm terminates. Otherwise, the breadth-first search continues from the neighboring nodes of the selected nodes. The same operation is performed on the central node 6. Finally, the result is shown in Figure 2b.

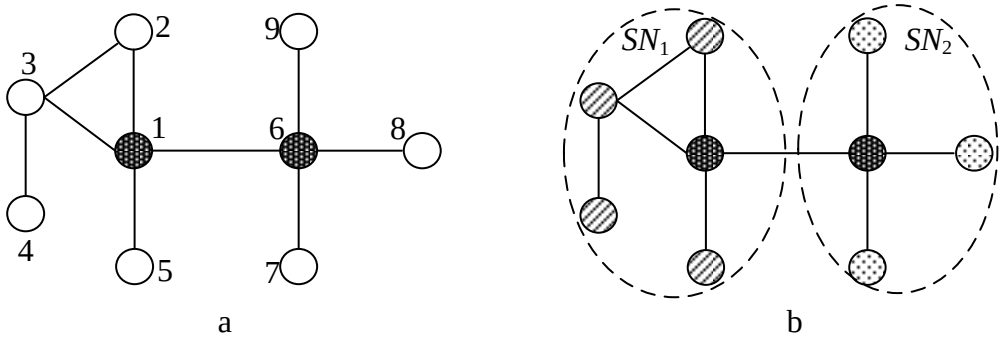

Figure 2. Comparison of MAP on DBLP dataset

3.2. Construction of multi-granular index

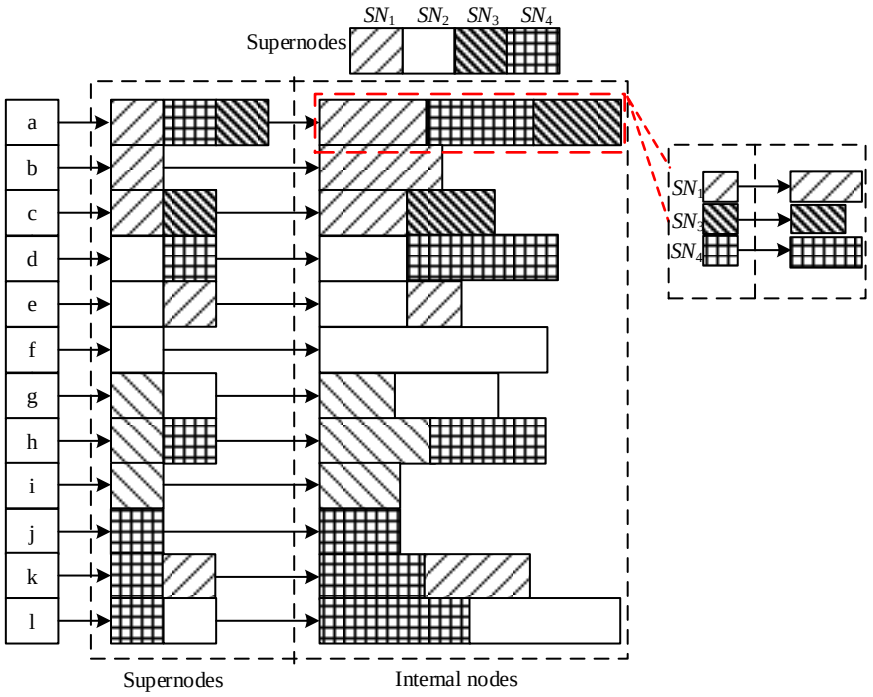

Figure 3. The structure of multi-granular index

In order to reduce the query scope and improve the query efficiency, we propose the multi-granular index. Its basic structure is shown in Figure 3, which includes: 1) Terms: they contain all the words that appear in the database 2) Supernode list: each term corresponds to a set of supernodes 3) Internal-node list: the coarse-granular index stores the summary information on supernode level to guide the query. If you need more detailed information, you should continue indexing and querying within the corresponding supernodes. Through the coarse-granular index, we can retrieve all supernodes containing token terms, and then a large number of irrelevant supernodes are pruned. Thus, the fine-granular index only needs to scan a small

number of supernodes, which include token terms. The second level index further refines the index results against the original nodes on the data graph. For instance, for the given query Q={d} only the supernodes SN_2 and SN_4 contain the keyword, so the supernodes SN_1, SN_3 and their internal nodes are pruned by the first level coarse-granular index. The fine-granular index only needs to scan in the supernodes SN_2 and SN_4. The index is built using an offline method. For example, for the token term a in the database, the supernodes SN_1, SN_3 and SN_4 will be returned through the search on the hypergraph. Thus, the above supernodes are put into the supernode list corresponding to the token term a. Repeat the same operation on the data graph to get all the data graph nodes containing the token a and put them into the corresponding internal-node set.

4. Filtering-validating query algorithm

This section proposes a filtering-validating query algorithm based on the hypergraph and multi-granular index. The algorithm has two stages: query filtering and query verification. The query filtering stage is based on the coarse–granular index and performs the algorithm 2 on the hypergraph H. It prunes the supernodes with low relevance to the query and returns the r-radius hyper-subgraph containing the keywords. In order to get more accurate query results, we further perform the query verification. The same query operation is performed in the r-radius super-subgraphs returned in Stage 1 to get the final results. Since query filtering and query verification have the same query idea, we only introduce the query on the hypergraph in the algorithm 2.

Algorithm 2 Hypergraph Query Algorithm
Input: hypergraph H ; the query $Q\{k_1, k_2, \cdots, k_d\}$ and r
Out: the set of all r-subgraphs
Method:
1. for $i \leftarrow 1$ to d do
2. $S_i \leftarrow$ the set of supernodes in H containing k_i
3. *rList* $\leftarrow$ empty
4. for $i \leftarrow 1$ to size (S_1) do
5. *rList*.add(S_{1i})
6. for $i \leftarrow 2$ to d do
7. *newRList* $\leftarrow$ empty
8. for $j \leftarrow 1$ to size (S_i) do
9. for $k \leftarrow 1$ to size (*rList*) do
10. if $\forall$supnode $\in$ *rList* dist(supnode,S_{ij}) $\le r$
11. then *newCandidate* $\leftarrow$ S_{ij}.*concatenate*(supnode)
12. *newRList*$_k$.add(*newCandidate*)
13. *rList* $\leftarrow$ *newRList*
14. Return *rList*

The algorithm takes the hypergraphs H, query $Q\{k_1, k_2, \cdots, k_d\}$ and distance limits r as input, a set of r-radius subgraphs as output. The nodes containing the keyword k_i are stored in the set S_i. S_{ij} represents the first j elements in the set S_i. Steps 1~2 of the algorithm extract the supernodes containing the keywords based on the coarse–granular index. Steps 4~5 add the supernodes containing the first keyword to the *rList*. Steps 6~13 calculate the shortest distance between each element in the set S_2 and the ones in the *rList*. If the distance does not exceed r, then S_2 and the corresponding supernodes in *rList* are added to the candidate list newRList. After the comparison, the *newRList* is used to replace the list *rList*. Repeat the above process for the remaining keywords, eventually, the list *rList* is returned. The output of the algorithm 2 is a set of r-radius super-subgraphs containing the keywords in which the shortest distance between any two content nodes is less than/equal to r, where the content nodes are supernodes containing one or more keywords. Repeat the above algorithm on each subgraph to get the final query results.

5. Experiment

In this section, we verify the performance of the HG-B query method by comparing it with the benchmark experiment r-clique method [10]. The method r-clique is a query method based on data graph. In this section, the method r-clique and the method proposed in the paper are denoted as r-clique and HG-B, respectively. Factors that affect the performance of the

query algorithm include the number of keywords and the number of returned results *k*, which will be analyzed in Section 5.2. We use the query response time and Mean Average Precision (MAP) as the metrics. The MAP of queries is calculated as Equations (1) and (2):

$$P = \sum_{i=1}^{k} (P(i) \times rel(i)) \Big/ N \tag{1}$$

$$MAP = \sum_{q \in Q} P(q) \Big/ |Q| \tag{2}$$

where i is the ranking of query results, k denotes the number of results already retrieved, $P(i)$ is the precision when i results are retrieved. $rel(i)$ is a binary function, if the ith result is the relevant result then $rel(i)=1$, otherwise, $rel(i)=0$. N is the number of the relevant results.

5.1. Experimental setup

- **Dataset**

We use the 2014 release version of DBLP as the experimental dataset; its download address is http://dblp.uni-trier.de/xml. This dataset is a standard dataset for evaluating keyword queries and includes information like articles, authors, and meetings. We use the method in [9] to build a data graph for the database and construct a hypergraph model using the methods in Section 3.1. Table 1 records the details of the dataset and Table 2 records part of the query examples.

- **Query set**

200 keywords are randomly selected from the DBLP dataset and divided into 5 subsets. Each subset contains a certain number of keywords. For example, subsets 1, 2, 3, 4 and 5 contain 20, 30, 40, 50 and 60 keywords respectively. Select keywords from the above five subsets to form queries with the number of keywords of 2, 3, 4, 5 and 6, respectively. Through this method, 100 queries are generated and used as the query set for this experiment.

Table 1. DBLP dataset

Database	Size of graph	Nodes(tuples)	Edges
DBLP	110MB	1.83MB	8.9MB

Database	Size of hypergraph	Supernodes	Superedges
DBLP	17.3MB	18024	1.41MB

Table 2. Query examples.

	Query examples
Q_1	Deep learning Jeffery Yu Xu
Q_2	Machine learning algorithm
Q_3	Graph mining Big data Xin Dong
Q_4	Database supervised learning ACM
Q_5	Data mining VLDB
Q_6	Image content retrieval video
Q_7	Sub-graph similarity matching SIGIR
Q_8	Incremental computational social network analysis
Q_9	Efficient sub-graph similarity matching database

- **Experimental environment**

The experiments have been performed on a computer with an Intel Pentium 3.2GHz CPU and 8 GB of RAM on Windows XP platform. All of the methods in this paper are implemented by JAVA.

5.2. Performance study

This section uses the query response time and Mean Average Precision to evaluate the performance of the query method.

- **Query efficiency**

The experiment evaluates the effect of the query complexity and the number of returned results on the query response time. It verifies the query efficiency of our method by comparing the query response time of r-clique and HG-B. In Figure

4a, the query complexity is changed by changing the number of query keywords. It measures the query response time of two methods for the top-5 query results under different number of keywords. The *x*-axis represents the number of keywords and the *y*-axis represents the query response time to return the top-5 results. It can be seen that the query efficiency of HG-B is obviously better than the r-clique method. When the number of keywords is 3, the query response time of r-clique method is 6000ms, while the one of HG-B is 3900ms, which is 35% less than the former. This is because we propose a multi-granular index in the HG-B method, which can effectively avoid unnecessary graph traversal and prune the irrelevant supernodes in advance by the first-level coarse-granular index. In addition, as the number of keywords increases, the query response time of both methods is on the rise. As the number of keywords increases, the iterations of both query algorithms increase. Figure 4b sets the number of keywords as 2 and remains unchanged. It compares the query response time under different *k* values. The *x*-axis represents the number of returned results *k*, and the *y*-axis represents the query response time corresponding to the different number of results.

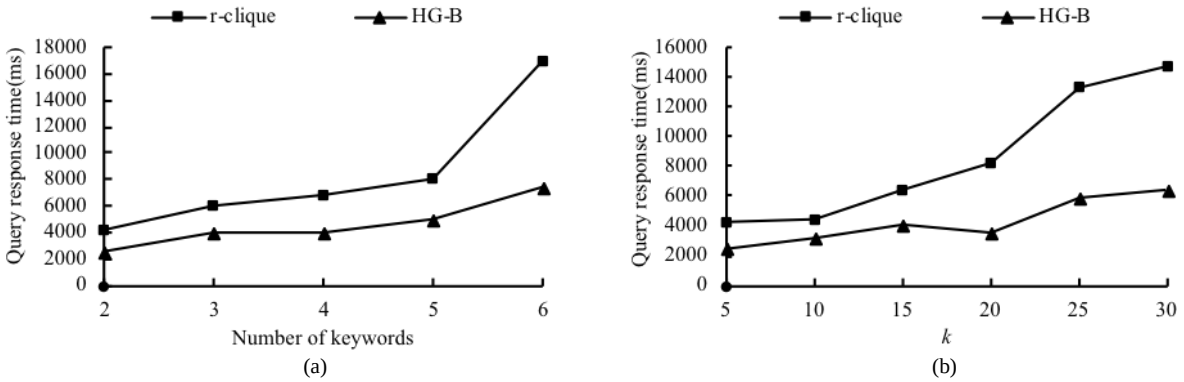

Figure 4. Query performance on DBLP dataset

As seen from Figure 4b, the query efficiency of HG-B is significantly better than the method r-clique. When k=10, the query response time of the method r-clique is 4500ms, while the one of HG-B is 3100ms, which is 31.1% less than the former. In addition, as the number of return results k increases, the query response time of both methods increase significantly.

- **Query effectiveness**

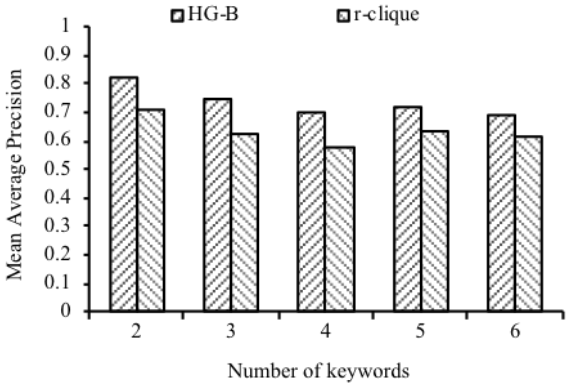

Figure 5. Comparison of MAP on DBLP dataset

In this section, the Mean Average Precision of HG-B method is further evaluated by comparing with the method r-clique. Figure 5 shows the Mean Average Precision of keyword queries with different lengths. It can be found that for queries with different length, the Mean Average Precision of HG-B method is better than the method r-clique. When the number of query keywords is 5, the Mean Average Precision of methods HG-B and r-clique are 0.72 and 0.63, respectively. The former increases by 14.3% compared with the latter. Because the HG-B method uses a filtering-validating query algorithm, the query results are more accurate after the two-level query on the hypergraph and the data graph. The results show that the HG-B query method can produce more relevant query results.

6. Conclusions

In this paper, a new and effective method HG-B is proposed for keyword query in relational databases. First, the concept of hypergraph is proposed and used to model the relational database. Second, the multi-granular index is built, and the filtering-validating query is performed based on it. Because the multi-granular index prunes the irrelevant supernodes in advance, unnecessary traversal of graphs is avoided and the efficiency of the query algorithm is improved. The filtering-validating query method makes the results more accurate through the graph traversal on the hypergraph and the original data graph. Experiments are performed on the public dataset DBLP to verify the performance of this method.

In the future, we will continue to study the dynamic query method in distributed databases. The research will solve query problems when the structure and content constantly change in the large-scale data environment.

Acknowledgements

This work is sponsored by the National Natural Science Foundation of China under Grant No. 61772152 and 61502037, and the Basic Research Project (No. JCKY2016206B001, JCKY2014206C002 and JCKY2017604C010).

References

1. S. Agrawal, S. Chaudhuri, G. Das, "Dbxplorer: A System for Keyword-Based Search over Relational Databases," in *Proceedings of 18th International Conference on Data Engineering*, pp. 5-16, San Jose, CA, United States, 2002
2. G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, S. Sudarshan, "Keyword Searching and Browsing in Databases Using Banks," in *Proceedings of 18th International Conference on Data Engineering*, pp. 431-440, San Jose, CA, United States, 2002
3. Y. Chen, W. Wang, Z. Liu, "Keyword-Based Search and Exploration on Databases," in *Proceedings of 2011 IEEE 27th International Conference on Data Engineering*, pp. 1380-1383, Hannover, Germany, 2011
4. Y. Chen, W. Wang, Z. Liu, X. Lin, "Keyword Search on Structured and Semi-Structured Data," in *Proceedings of International Conference on Management of Data and 28th Symposium on Principles of Database Systems*, pp. 1005-1010, Providence, United States, 2009
5. "Dblp," Available at http://dblp.uni-trier.de/, Last accessed on September 15, 2017
6. B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, X. Lin, "Finding Top-K Min-Cost Connected Trees in Databases," in *Proceedings of 23rd International Conference on Data Engineering*, pp. 836-845, Istanbul, Turkey, 2007
7. M. Fernandez, I. Cantador, V. Lopez, D. Vallet, P. Castells, E. Motta, "Semantically Enhanced Information Retrieval: An Ontology-Based Approach," *Journal of Web Semantics*, vol. 9, no. 4, pp. 434-452, 2011
8. V. Hristidis, Y. Papakonstantinou, "Discover: Keyword Search in Relational Databases," in *Proceedings of the 28th international conference on Very Large Data Bases*, pp. 670-681, Hong Kong, China, 2002
9. V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, H. Karambelkar, "Bidirectional Expansion for Keyword Search on Graph Databases," in *Proceedings of the 31st international conference on Very large data bases*, pp. 505-516, Trondheim, Norway, 2005
10. M. Kargar, A. An, "Efficient Top-K Keyword Search in Graphs with Polynomial Delay," in *Proceedings of IEEE 28th International Conference on Data Engineering, ICDE 2012*, pp. 1269-1272, Arlington, VA, United States, 2012
11. A. Kothawade, M. Harak, J. Bagul, B. Patil, "Ranking Based Prediction of Keyword over Big Databases," in *Proceedings of 1st International Conference on Green Computing and Internet of Things, ICGCIoT 2015*, pp. 899-903, Greater Noida, Delhi, India, 2015
12. G. Li, B. C. Ooi, J. Feng, J. Wang, L. Zhou, "Ease: An Effective 3-in-1 Keyword Search Method for Unstructured, Semi-Structured and Structured Data," in *Proceedings of 2008 ACM SIGMOD International Conference on Management of Data*, pp. 903-914, Vancouver, Canada, 2008
13. Y. Luo, X. Lin, W. Wang, X. Zhou, "Spark: Top-K Keyword Query in Relational Databases," in *Proceedings of SIGMOD 2007: ACM SIGMOD International Conference on Management of Data*, pp. 115-126, Beijing, China, 2007
14. J. Park, S. G. Lee, "Keyword Search in Relational Databases," *Knowledge and Information Systems*, vol. 26, no. 2, pp. 175-193, 2011
15. S. S. Pawar, A. Manepatil, A. Kadam, P. Jagtap, "Keyword Search in Information Retrieval and Relational Database System: Two Class View," in *Proceedings of International Conference on Electrical, Electronics, and Optimization Techniques, ICEEOT 2016*, pp. 4534-4540, Palnchur, Chennai, Tamilnadu, India, 2016
16. P. Pujari, R. Ade, "Enhancing Performance of Keyword Query over Structured Data," in *Proceedings of 2nd International Conference on Computing, Communication, Control and Automation*, pp. 25-31, Pune, India, 2016
17. L. Qin, J. X. Yu, L. Chang, "Keyword Search in Databases: The Power of Rdbms," in *Proceedings of International Conference on Management of Data and 28th Symposium on Principles of Database Systems*, pp. 681-693, Providence, United States, 2009
18. Y. Wang, N. Wang, L. Zhou, "Keyword Query Expansion Paradigm Based on Recommendation and Interpretation in Relational Databases," *Scientific Programming*, vol. 2017, no. 2017, pp. 26-37, 2017
19. M. L. Wilson, B. Kules, M. C. Schraefel, B. Shneiderman, "From Keyword Search to Exploration: Designing Future Search Interfaces for the Web," *Foundations and Trends in Web Science*, vol. 2, no. 1, pp. 1-97, 2010
20. J. X. Yu, L. Qin, L. Chang, "Keyword Search in Relational Databases: A Survey," *IEEE Data Eng Bull*, vol. 33, no. 1, pp. 67-78, 2010

Yingqi Wang is currently pursuing the Ph.D. degree in computer science and technology, Harbin Engineering University, Harbin, China. Her interests include: keyword query, dataspace and data mining.

Lianke Zhou received the PhD degree in computer science and technology from Harbin Institute of Technology, Harbin, China, in 2011. He is member of CCF. His interests include: Data Visualization, Dataspace, Distributed Computing.

Nianbin Wang received the PhD degree in computer science and technology from Harbin Institute of Technology, Harbin, China, in 2001. He is member of CCF, he has been a Professor with the Department of Computer Science and Technology, Harbin Engineering University. His interests include: Dataspace, Deep Learning, Data Integration.