ISSN 1330-3651 (Print), ISSN 1848-6339 (Online)

https://doi.org/10.17559/TV-20160419053402
Original scientific paper

Keyword Search in Large-Scale Databases with Topic Cluster Units

Yingqi WANG, Nianbin WANG, Lianke ZHOU

Abstract: To solve the inefficiency of the existing keyword search methods in large databases, this paper proposes TCU-based query, an offline query method based on topic cluster units. First, topic cluster units (TCUs) are constructed through vertical grouping and horizontal grouping on tables and tuples. In contrast to traditional keyword query methods, this offline method cannot only reduce the query response time, but also return results comprising richer and more complete semantic information. In order to further improve the efficiency of data preprocessing, an optimized solution for table join ordering based on the genetic algorithm is presented. Second, we select index terms using the association rule, and then we build an index on every topic cluster; by doing so we can improve the query speed significantly. Finally, we conduct extensive experiments to demonstrate that our approach greatly improves the performance of keyword search.

Keywords: clustering; keyword search; relational databases; subject indexes; topic cluster units

1 INTRODUCTION

Recently, keyword search has been widely used in information retrieval [1]. Because it is simple and easy to use, more and more people have accepted and applied the technology. Since documents and large amounts of data are normally stored in relational databases, it is necessary to provide a simple and effective query method targeting relational databases [2, 3]. For non-expert users, it is difficult to exploit related information by means of traditional structured query methods such as *SQL*, because these query methods require users to have good knowledge of the underlying schema and the skill of using complex query languages. Obviously, these requirements are complicated and unfriendly for ordinary users. Thus keyword search over relational databases has been paid widespread attention.

Existing studies on keyword query over relational databases can be broadly classified into two types of methods: online query and offline query [4]. The main idea of online query is that database is modeled as a schema graph or data graph. At search time, the system traverses the graph to return one or more sub-graphs (*Candidate Networks* or *Steinertrees*) as query results [5-11]. Due to the frequent table joins in the process of queries, these methods can be extremely costly. In contrast, offline query methods have well solved the above problems by using *virtual documents* or *tuple units* [12, 13]. These offline query methods connect tables using width-first traversal method before a keyword query is issued. Still, the offline query methods do not consider the query efficiency and thus are not practical in large-scale databases. The data preprocessing is expensive and time-consuming using above methods if the database contains hundreds of tables. In addition, since the scale of obtained tables at the preprocessing phase is large, it costs a lot of time to get the query results, even if indexes are constructed.

In order to address the above problems, we emphasize the keyword search over large-scale databases, and the contributions of this paper are summarized as follows:

(1) We propose a TCU-based query method for keyword search over large-scale databases.
(2) We construct a new data structure — topic cluster units. Based on an improved spectral clustering strategy and an association graph of topic cluster tuples, tables and tuples in a database are divided into several clusters. Generating a set of TCUs as search results offline cannot only significantly reduce the query response time, but also return results with richer and more complete semantics.
(3) We design an optimized solution for table join ordering which can reduce the cost of data preprocessing.
(4) We select terms according to the association rule algorithm, then we construct a subject index for each topic cluster, which improves the query speed remarkably.
(5) We implement our approach over the real dataset *Freebase* [14]. The results demonstrate that the TCU-based query method achieves high efficiency and effectiveness, and outperforms state-of-the-art approaches significantly.

2 RELATED WORK

2.1 Online Query

DBXplorer, *DISCOVER*, *BANKS,* and *BLINKS* are online query methods in a relational database. These methods share the similar idea: queries are processed by a graph traversal that searches for connected tuples containing the query keywords.

DBXplorer and *DISCOVER* view the database as a schema graph with tables as nodes, and relationships as edges [5, 6]. These methods construct and evaluate a set of *CNs* (*Candidate Networks*) for a given query, then identify potential answers that are composed of relevant tuples based on the *CNs* and schema graph of the database.

BANKS and *BLINKS* first covert the whole database into a data graph, where each node denotes a tuple, and edges connect tuples which can be joined together. The methods identify the *Steiner trees* from the whole graph, which is inefficient and proved to be an *NP-Complete* problem [8, 10, 11]. *BANKS* identifies the *Steiner trees* by employing a backward search strategy. A *Steiner tree* is a connected tree where every leaf node represents a tuple containing at least one query keyword, and internal nodes correspond to tuples that connect the leaf records. But when there are some nodes with big in-degree, the performance of *BANKS* will be decreased remarkably. So *BLINKS* proposed new techniques — a bidirectional expansion and a *BLINKS* index that can improve search efficiency significantly.

2.2 Offline Query

A drawback of the online query methods is the high cost of the online table joins. In general, the above problems can be solved through the pretreatment of tables and tuples in databases [12, 13, 15, 16]. In recent years, the problem of off-line query has been discussed in-depth, and some solutions are put forward primarily. Su and Widom first propose the concept of *text objects* and *virtual documents* [12]. Multi-joining of tables is completed offline and the query efficiency is improved significantly. Improving and expanding the *text objects*, Feng et al. group the tuples with same attribute values to construct a more complete data structure — *tuple units* [13]. It integrates multiple tuple units to response to a keyword query and achieves a high performance. Because above methods only consider the simple table joins and use *SQL-based* methods to create the *virtual documents* or *tuple units*, they are not suitable for large-scale databases with complex schemas. Based on the offline query methods mentioned above, we define a more reasonable data structure — topic cluster units and put forward aTCU-based query method. We consider the table join order at the same time, so this method can obtain good performance in terms of retrieval efficiency and effectiveness.

3 OVERVIEW OF TCU-BASED QUERY METHOD

The main idea of this paper is presented in Fig. 1.

(1) **Query.** After users submit queries to the query processor, the query processor returns a set of topic cluster units containing one or more keywords as results to the users by scanning the subject indexes.

(2) **Offline data preprocessing.** The offline data preprocessing includes mainly four modules: Vertical grouping, Table join order optimization, Horizontal grouping and Construction of subject indexes.

(a) **Vertical grouping.** This module takes the database and the query logs as input. It uses a graph partition strategy (i.e., an improved spectral clustering algorithm) and combines the characteristics of the database with the user feedback to vertically group the data tables. This module outputs a set of topic clusters, where every topic cluster contains a set of data tables with same topic and high co-occurrence frequency in query logs.

(b) **Table join order optimization.** For every topic cluster obtained from the Vertical grouping, it is necessary to connect tables in the topic cluster according to primary-foreign-key relationships. When the database contains a large number of tables, this operation will be quite time-consuming. Consequently we should improve the performance of joining as well as possible. A selection scheme of table join order based on the genetic algorithm is presented in this module.

(c) **Horizontal grouping.** This module constructs a tuple association graph for each topic cluster by calculating the comprehensive similarity between topic cluster tuples, and then it horizontally groups every topic cluster by using hierarchical clustering method. As a result, sets of topic cluster units are formed.

(d) **Construction of subject indexes.** This module takes the sets of topic cluster units obtained from the Horizontal grouping module as input. It selects terms according to the association rule algorithm, then it constructs a subject index for each topic cluster.

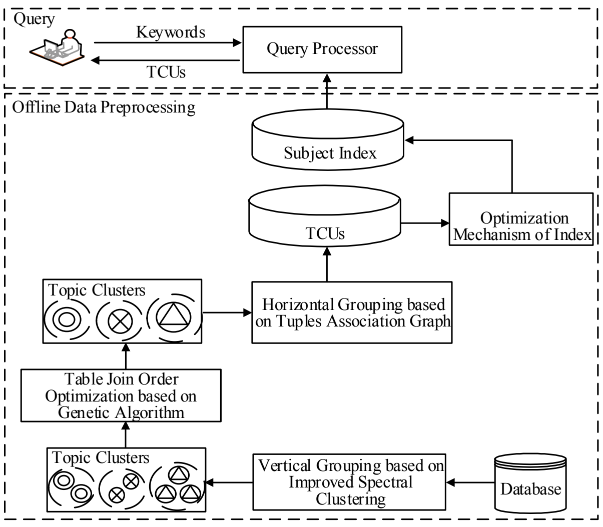

Query
Keywords
TCUs
Query Processor
Offline Data Preprocessing
Subject Index
TCUs
Optimization Mechanism of Index
Topic Clusters
Horizontal Grouping based on Tuples Association Graph
Table Join Order Optimization based on Genetic Algorithm
Topic Clusters
Vertical Grouping based on Improved Spectral Clustering
Database

Figure 1 Architecture of the TCU-based query method

4 QUERY METHOD BASED ON TOPIC CLUSTER UNITS

4.1 Construction of Topic Cluster Units

4.1.1 Vertical Grouping Based on Characteristics of Databases and Query Logs

This section proposes a vertical grouping strategy based on an improved spectral clustering algorithm. In order to make the results of vertical grouping more accurate, we design a novel construction method of similarity matrix between tables. In particular, we construct the similarity matrix from two aspects of table characteristics (i.e., topology compactness and content similarity) and query logs. Thus, this method is comprehensive and reflects the user preferences. Fig. 2 shows that the Vertical grouping is divided into 3 modules: Input module, Similarity matrix construction module and Output module. It takes the relational database and schema graph as input to describe the contents and structures of databases respectively. Another input is the query logs which reflect the distribution characteristics of information in databases from a different aspect. In the Similarity matrix construction module, the topology compactness and content similarities between tables are calculated by analyzing the schema graph and database in the Input module. We then construct a similarity matrix which is made up of topology compactness matrix and content similarity matrix. Finally, we conduct a statistical analysis on query logs to adjust the above results. A set of topic clusters is produced as output.

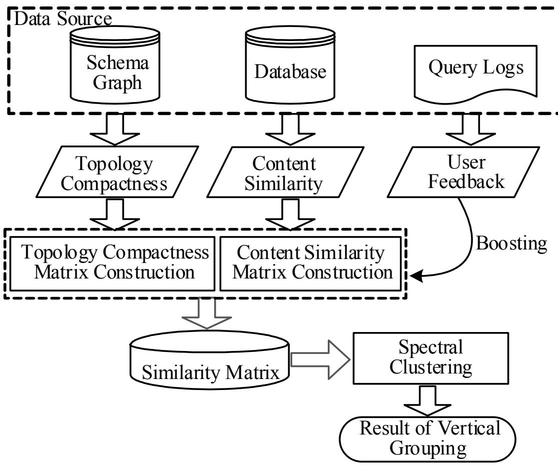


Figure 2 Architecture of vertical grouping

(1) **Topology compactness**

In the process of measuring the topology compactness, this paper introduces the concept of topology potential in data fields and proposes a novel calculation method of topology compactness. In a database schema graph, we use Gaussian function to describe the interaction between nodes. Gaussian function has good mathematical characters and it can describe the short-range field (the interaction between nodes has localization character, the influence of a node tends to attenuate significantly as the logical distance between nodes increases). We suppose each node can generate a force field along the direction of each edge. Nodes in the schema graph associate with each other and the interaction force is closely related to the size of nodes and the distance between them. Obviously, the topology potential of a node contains rich structural information, it can be used to measure the topology compactness between nodes.

For a database schema graph $G=(V,E)$, the topology compactness between v_i and v_j is defined as follows:

$$c'_{ij}=\begin{cases} |v_i|\cdot|v_j|\cdot\exp(-(ld_{ij}/\sigma)^2) & ld_{ij}\le\left[3\sigma/\sqrt{2}\right] \quad i\ne j \\ 0 & ld_{ij}>\left[3\sigma/\sqrt{2}\right] \quad i\ne j \\ 0 & i=j \end{cases} \tag{1}$$

where $|v_i|$ ($|v_j|$) denotes the number of tuples contained in the table T_i (T_j). σ is an influence factor, which determines the influence scope of nodes in the schema graph. When σ is small, the interaction between two nodes is weak; conversely, when σ is big, the interaction between nodes becomes strong. So we need to choose a proper value for σ. This paper employs the topology potential entropy to select the optimal σ. The specific method is as follows: we suppose there are n nodes $v_1, v_2, \dots, v_n$ in the database schema graph, where the topology potential of each node is $\varphi(v_i)=\sum_{l=1}^{n}|v_i|\cdot|v_l|\cdot\exp(-(ld_{ij}/\sigma)^2)$. The topology potential entropy is defined as

$$H=-\sum_{i=1}^{n}(\varphi(v_i)/Z)\cdot\log(\varphi(v_i)/Z)\ ,$$

where n is the number of nodes in the schema graph, $Z=\sum_{i=1}^{n}\varphi(v_i)$ is the normalization factor. When H reaches the minimum, the value of σ is optimal. That is, the optimal value of σ varies with the underlying schema and size of databases.

ld_{ij} denotes the logical distance between v_i and v_j, namely, the path length between them in the schema graph. According to the mathematical properties of Gaussian function, for a given σ, if node v_i is out of the influence scope ($\left[3\sigma/\sqrt{2}\right]$) of node v_j, the topology compactness between them attenuates to zero rapidly [17, 18].

Note:the basic form of Gaussian function is $f(x)=(1/\sigma\sqrt{2\pi})\exp(-(x-\mu)^2/2\sigma^2)$. According to the rule of thum b for Gaussian function, we may know that about 99.7% of the values are within 3 standard deviations of the mean. That is, if $x>3\sigma$ then $f(x)=0$. Eq. (1) is a specific form of Gaussian function, where the mean μ is 0. Compared with the basic form of Gaussian function, the independent variable ld_{ij} in Eq. (1) equals $x/\sqrt{2}$. Therefore, when x equals 3σ, ld_{ij} equals $3\sigma/\sqrt{2}$. That is, when ld_{ij} is greater than $\left[3\sigma/\sqrt{2}\right]$, c'_{ij} equals 0.

Based on Eq. (1), we can get the topology compactness between two nodes, and then we use the normalized formula to transform the results and construct a matrix of topology compactness.

$$\boldsymbol{c}=\begin{bmatrix} 0 & c_{12} & \cdots & c_{1n} \\ c_{21} & 0 & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & 0 \end{bmatrix}$$

Example 4.1:

Fig. 3 shows a database schema graph, which contains six nodes. Here, we take the figure as an example to show the calculation of topology compactness and the construction of topology compactness matrix $\boldsymbol{c}$. For this schema graph, the optimal influence factor σ is 1.29 based on the minimal potential entropy; thus, the influence scope $\left\lfloor 3\sigma/\sqrt{2}\right\rfloor$ equals 2. The logical distance ld_{36} between Tab. 3 and table 6 is 2, the sizes of the above tables are $|v_3|=300$ and $|v_6|=200$, respectively. By Eq. (1) and normalized formula $f(c'_{ij})=\begin{cases}1-1/(1+\log_{10}c'_{ij}) & c'_{ij}>0\\ 0 & c'_{ij}=0\end{cases}$ we can get the topology compactness between two tables.

$$c_{36}=1-1/\left(1+\log_{10}c'_{36}\right)=$$
$$=1-1/\left(1+\log_{10}\left(300\cdot200\cdot\exp\left(-(2/1.29)^2\right)\right)\right)=0.79$$

In the same way, we can calculate the topology compactness among the remaining tables. Thus, the topology compactness matrix $\boldsymbol{C}$ is obtained.

$$\boldsymbol{C}=\begin{bmatrix} 0 & 0.77 & 0.77 & 0.82 & 0 & 0.76 \\ 0.77 & 0 & 0.76 & 0.81 & 0 & 0.75 \\ 0.77 & 0.76 & 0 & 0.83 & 0 & 0.79 \\ 0.82 & 0.81 & 0.83 & 0 & 0.81 & 0.83 \\ 0 & 0 & 0 & 0.81 & 0 & 0.82 \\ 0.76 & 0.75 & 0.79 & 0.83 & 0.82 & 0 \end{bmatrix}$$

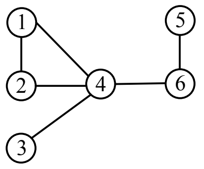


Figure 3 Database schema graph

(2) Content similarity

Furthermore, the information stored in tuples and attributes of tables can also affect the similarity between tables and play an important role in the process of Vertical grouping. Obviously, the more similar the contents of both tables, the higher probability that they will be assigned into the same cluster. Therefore, this section will discuss the content similarity between any two tables, which provides the theoretical basis for the graph partition.

The analysis of the content similarity between tables should be divided into two aspects of name similarity and value similarity.

Name similarity imposes an important influence on the content similarity between tables. Specifically, it includes two parts, table name similarity and attribute name similarity. This paper employs *Vector Space Model* to calculate the name similarity [19]. First, we extract the keywords from table names T and attribute names A. Then we construct the vector V_i for every table T_i. At last we calculate the name similarity using Eq. (2).

$$Sim_1(T_i,T_j)=\begin{cases} Sim(V_i,V_j)=V_i\cdot V_j/(|V_i|\cdot|V_j|) & i\neq j \\ 0 & i=j \end{cases} \tag{2}$$

The other factor influencing the content similarity is the attribute value similarity between two tables. The key steps of calculating attribute value similarity are as follows:

① Calculate the content similarity between attributes using *Jaccard* function.

$$J(T_iA_m,T_jA_n)=|T_iA_m\cap T_jA_n|/|T_iA_m\cup T_jA_n| \tag{3}$$

② Test the set Z of matching attribute pairs based on a greedy-matching strategy.

a. Set Z to ϕ, it is used to store the matched attribute pairs, T_iA and T_jA contain all attribute-columns in the tables T_i and T_j, respectively.

b. Find the attribute-column pair (T_iA_m,T_jA_n), which has the maximum $J(T_iA_m,T_jA_n)$, where $T_iA_m\in T_iA$ and $T_jA_n\in T_jA$.

c. $Z\leftarrow Z\cup\{(T_iA_m,T_jA_n)\}$, remove T_iA_m and T_jA_n from T_iA and T_jA, respectively.

d. Repeat steps b and c until there are not attribute-column pairs with *Jaccard* similarity larger than zero.

③ Use the method of weighted average values to get the attribute value similarity between tables.

$$Sim_2(T_i,T_j)=\begin{cases} \dfrac{1}{|Z|}\sum_{(T_iA_m,T_jA_n)} max\left\{\widehat{T_iA_m},\widehat{T_jA_n}\right\}\cdot J(T_iA_m,T_jA_n) & i\neq j \\ 0 & i=j \end{cases} \tag{4}$$

where $|Z|$ is the number of elements in Z. $\widehat{T_iA_m}=S_m/\overline{A_m}$ ($\widehat{T_jA_n}=S_n/\overline{A_n}$) is the variation coefficient of attribute A_m (A_n) in the table $T_i(T_j)$. It is a statistic measuring the extent of variation. The smaller the variation coefficient, the less the richness of attributes, and vice versa. We map the attribute values to integers in ascending order. $S_m(S_n)$ and $\overline{A_m}$ ($\overline{A_n}$) are standard deviation and average of the mapped attribute values, respectively.

Through the above analysis, we can get the content similarity matrix.

$$\boldsymbol{S}=\begin{bmatrix} 0 & s_{12} & \cdots & s_{1n} \\ s_{21} & 0 & \cdots & s_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ s_{n1} & s_{n2} & \cdots & 0 \end{bmatrix}$$

where $s_{ij}=\alpha\cdot Sim_1(T_i,T_j)+(1-\alpha)\cdot Sim_2(T_i,T_j)$.

Finally, the similarity matrix $\boldsymbol{A}_{DB}=\beta\cdot\boldsymbol{C}+(1-\beta)\cdot\boldsymbol{S}$ is obtained by synthesizing the structural compactness matrix and content similarity matrix where α and β are balancing factors which are used to adjust the impacts of influence factors on similarity between tables. In reality, the values of α and β are usually obtained according to a number of experiments. In this paper we simplify the problems by assigning the same weight (1/2) to them based on maximum entropy model.

Example 4.2:

Table 1 Football team

ID	Team name	Head coach	Founded time
1	Tornado	Jesse Cannon	1996
2	Victory	Braden Shield	1989
3	Battle	Harlin Christian	1990
4	Treasures	Adrian Mckinney	2000
5	Nighthawk	Victor Hammer	1986

Table 2 Football coach

ID	Coach name	Football team	Nationality
1	Harlin Christian	Treasures	Netherlands
2	Adrian Mckinney	Battle	Russia
3	Braden Shield	Tornado	America
4	Adrian Mckinney	Tornado	England
5	Lyle Aiken	Victory	France

Suppose tables Football team and Football coach are any two tables in a database. In order to present the details of methods more clearly, we assume they have the above composing in this example as shown in Tab. 1 and Tab. 2.

Now we will explain how to calculate the name similarity between two tables by Eq. (2). It includes three steps: a) extract all of the terms (i.e., Football, team, ID, name, Head, coach, Founded, time, Nationality) from the table names and attribute names, b) generate the vectors (i.e., V_1=[1,2,1,1,1,1,1,1,0] and V_2=[2,1,1,1,0,2,0,0,1]) by calculating the term frequency, c) calculate the name similarity between tables 1 and 2 by Eq. (2).

$$Sim_1(T_1,T_2)=Sim(V_1,V_2)=V_1\cdot V_2\big/(|V_1|\cdot|V_2|)=$$
$$=\frac{1\times 2+2\times 1+1\times 1+1\times 1+1\times 2}{\sqrt{1^2+2^2+1^2+1^2+1^2+1^2+1^2+1^2}\cdot\sqrt{2^2+1^2+1^2+1^2+2^2+1^2}}=0.69$$

Similarly, we can get the name similarity among the remaining tables with the method described above. Then, we will continue to illustrate the calculating procedure of attribute value similarity between tables.

First, we calculate the content similarity between any two attributes of two tables using Eq. (3) in step ①. For example, attribute columns Team name in table 1 and Football team in Tab. 2 are denoted as T_1A_2 and T_2A_3, respectively. The content similarity between them is as follows:

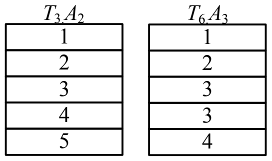
$$J(T_1A_2,T_2A_3)=|T_1A_2\cap T_2A_3|\big/|T_1A_2\cup T_2A_3|=4/5=0.8$$

Second, we can get the set Z of matching attribute pairs between tables 1 and 2 according to the greedy-matching strategy in step ②.

$$Z=\{(T_1A_2,T_2A_3),\ (T_1A_3,T_2A_2)\}$$

Before calculating the attribute value similarity between tables, we need to calculate the variation coefficient of each attribute column in tables. Thus we can assign a proper weight to each matching attribute pair. Next, we will discuss the calculation method of the variation coefficient with attributes T_1A_2 and T_2A_3 as an example.

We map the attribute values to integers in ascending order. The mapped attributes T_1A_2 and T_2A_3 are illustrated in Fig. 4. We will calculate the coefficient of variation of two attributes to compare their richness.



T_3A_2	T_6A_3
1	1
2	2
3	3
4	3
5	4

Figure 4 Comparison of attribute columns in two tables

The coefficient of variation of attributes T_1A_2 and T_2A_3 can be calculated as follows.

$$\widehat{T_1A_2}=S_2\big/\overline{A_2}\times 100\%=\sqrt{\sum_{i=1}^{n}(A_i-\overline{A_2})^2/n}\Big/\overline{A_2}=$$
$$=\sqrt{((1-3)^2+(2-3)^2+(3-3)^2+(4-3)^2+(5-3)^2)/5}\Big/3=47\%$$

$$\widehat{T_2A_3}=S_3\big/\overline{A_3}\times 100\%=\sqrt{\sum_{i=1}^{n}(A_i-\overline{A_3})^2/n}\Big/\overline{A_3}=$$
$$=\sqrt{((1-2.6)^2+(2-2.6)^2+(3-2.6)^2+(3-2.6)^2+(4-2.6)^2)/5}\Big/2.6=39\%$$

$\widehat{T_1A_2}>\widehat{T_2A_3}$ shows that the degree of variation of attribute T_1A_2 is bigger than that of attribute T_2A_3, then the richness of attribute T_1A_2 is greater than that of attribute T_2A_3.

Finally, we use Eq. (4) to calculate the attribute value similarity between the tables. The specific calculation process is as follows:

$$Sim_2(T_1,T_2)=\frac{1}{|Z|}\sum_{(T_1A_m,T_2A_n)\in Z}max\left\{\widehat{T_1A_m},\widehat{T_2A_n}\right\}\cdot J(T_1A_m,T_2A_n)=$$
$$=\frac{1}{2}(0.47\times 0.8+0.47\times 0.5)=0.31$$

In conclusion, the content similarity between Tabs. 1 and 2 is as follows:

$$s_{12}=(Sim_1(T_1,T_2)+Sim_2(T_1,T_2))/2=(0.69+0.31)/2=0.5$$

Likewise, we can calculate the content similarity among the remaining tables. Thus, the content similarity matrix $\mathbf{S}$ is obtained.

$$\boldsymbol{S}=\begin{bmatrix} 0 & 0.50 & 0.37 & 0.64 & 0.28 & 0.30 \\ 0.50 & 0 & 0.56 & 0.66 & 0.41 & 0.49 \\ 0.37 & 0.56 & 0 & 0.71 & 0.33 & 0.42 \\ 0.64 & 0.66 & 0.71 & 0 & 0.54 & 0.72 \\ 0.28 & 0.41 & 0.33 & 0.54 & 0 & 0.87 \\ 0.30 & 0.49 & 0.42 & 0.72 & 0.87 & 0 \end{bmatrix}$$

(3) Adjustment of similarity matrix

Query logs record a large number of query results and contain three fields: user ID, user queries and tables which contain query results. All of the information can reflect user preferences. The basic idea of the vertical grouping approach with user feedback is to perform a statistical analysis on query logs, and to use Eq. (5) adjust the result.

$$boost_{lg}(T_i,T_j)=exp(lg(count(T_i,T_j))/lg(max(count))) \qquad (5)$$

where $count(T_i,T_j)$ is the number of co-occurrence of tables T_i and T_j in the same log record, and $max(count)$ is the maximum value of $count(T_i,T_j)$.

Example 4.3:

To illustrate the enhanced approach for similarity between tables in Eq. (5), we simplify the constitution of query logs as shown in Tab. 3. In this query log, tables 2 and 4 have the highest co-occurrence and the $max(count)$ is 8. The number of co-occurrence of table 3 and table 6 is 2. The enhanced coefficient of $A_{DB}(T_3,T_6)$ can be calculated by Eq. (5) as follows:

$$boost_{lg}(T_3,T_6) = exp(lg(count(T_3,T_6)) / lg(max(count)))$$
$$=exp(lg2 / lg8)=1.39$$

Table 3 Query log

ID	Queries	Involved Tables
237060360	Q_1	T_1 T_2 T_4
237060361	Q_2	T_3 T_6
237060362	Q_3	T_2 T_4
237060363	Q_4	T_3 T_4 T_6
237060364	Q_5	T_2 T_3 T_4
237060365	Q_6	T_2 T_4 T_6
237060366	Q_7	T_2 T_4
237060367	Q_8	T_2 T_4 T_5
237060368	Q_9	T_2 T_4
237060369	Q_{10}	T_2 T_4 T_5 T_6

We use the information contained in the query logs to strengthen the result and get the final similarity matrix shown below.

$$A_{\text{Final}} = \left[A_{DB}(T_i,T_j) \cdot boost_{lg}(T_i,T_j) \right]_{i,j=1}^{n} \quad (6)$$

(4) Vertical grouping

Algorithm 1: Vertical grouping based on improved spectral clustering algorithm [20]

Input: database schema graph G = (V, E), where $V = \{v_1,...,v_n\}$, $E = \{e_{v_i v_j} \mid v_i, v_j \in V\}$ and influence factor σ;

Output: topic cluster set $C = \{C_1, C_2, ..., C_k\}$;

Algorithm Description:

① Construct the matrix $A_{\text{Final}}(v_i, v_j)$ with Eq. (6);

② Calculate the first k eigenvectors and eigenvalues of $A_{\text{Final}}(v_i, v_j)$;

③ Map all nodes in V to R^k, assign the nodes in R^k into clusters $C_1, C_2, ..., C_k$ with the *k-means* algorithm.

4.1.2 Join Order Optimization

In order to avoid the complex table join during query processing, the data preprocessing must be done by connecting the tables $T_{i1}, T_{i2}, ..., T_{ij}$ of topic cluster $C_i = (T_{i1}, T_{i2}, ..., T_{ij})$ into a big table T'_i. The existing methods connect the tables by doing breadth-first traversal on the schema graph of database. For large-scale relational databases with thousands of tables, breadth-first traversal can greatly affect the efficiency of data preprocessing. To solve this problem, this section presents a table join order optimization based on the genetic algorithm. The basic flow chart is illustrated in Fig. 5.

To the best of our knowledge, the key step of the join order optimization is encoding data tables. We use a join tree to store and express join orders. In order to preserve more detailed information of the join tree, we encode the join order by the preorder traverse of the join tree, as shown in Fig. 6.

After the above encoding operation, we randomly select poplength join trees as the initial population and execute a genetic operation on the initial population to produce genlength new individuals. Crossover operator and mutation operator are used in the above genetic operation. The calculation rules of the operators are as follow:

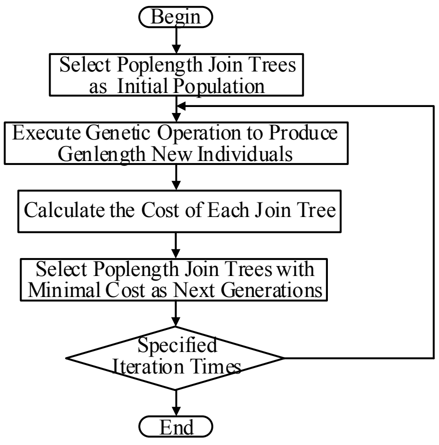

Begin
Select Poplength Join Trees as Initial Population
Execute Genetic Operation to Produce Genlength New Individuals
Calculate the Cost of Each Join Tree
Select Poplength Join Trees with Minimal Cost as Next Generations
Specified Iteration Times
End

Figure 5 Flow chat of join order optimization

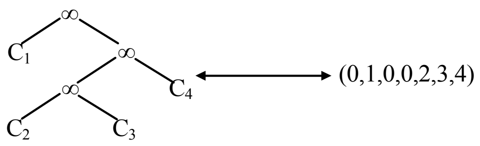


Figure 6 Corresponding relationships between join tree and integer sequence

Crossover operator: exchange the sub-trees with the same size randomly. Replace the duplicate tables in the join tree with the table which does not appear in the tree.

Mutation operator: exchange tables with nonzero values in the join tree.

Then we calculate the cost of each join tree based on the existing cost function, select poplength join trees with minimal cost as the next generations. Repeat the above processes of genetics and selection until it reaches the specified iteration times. The reasonable iteration times can be acquired through many experiments. The join order represented by the minimum cost tree in the last generation of the genetic algorithm is the optimal join order.

4.1.3 Horizontal Grouping Based on Association Graph of Topic Cluster Tuples

After the above sections, we can get a topic cluster set $C = (C_1, C_2, ..., C_k)$, where each C_i contains a big table T'_i. In many cases, multiple topic cluster tuples should be integrated to answer keyword queries. Because of that, we need to further group the topic cluster tuples using a reasonable horizontal grouping method to improve the query speed. The existing methods use *SQL-based* methods like group by operator to group the tuples in the process of horizontal grouping. This approach is limited to equality of the key attributes, which causes the related tuples containing discrepant values be assigned to different groups. In this section, we design a horizontal grouping method based on the association graph of topic cluster tuples. It can improve the efficiency of queries while making the results of grouping meet the users' demand. The calculation method of comprehensive similarity is put forward to make the results more accurate.

(1) Calculation of similarities between topic cluster tuples

One important data model in the horizontal grouping is the association graph $G = (V, E)$ of topic cluster tuples. It is a weighted undirected graph that uses the similarity between topic cluster tuples as weight values of edges. This section proposes a novel calculation method of similarity.

① Define different distance functions, such as *Euclidean distance*, *Edit distance* and *Hamming distance*;

② Map the topic cluster tuples to an n-dimensional space and calculate the distance $d_k(t'_i, t'_j)$ between two tuples using the distance functions;

③ Calculate the comprehensive similarity between two tuples according to the following formula:

$$Sim_k(t_i', t_j') = 1 - d_k(t_i', t_j') / \max_{k\in\{1,2,3\}} \{d_k(t_i', t_j')\}$$
$$Sim(t_i', t_j') = 1 / (1 + exp(-\sum_{k=1}^{3} Sim_k(t_i', t_j'))) \tag{7}$$

Example 4.4:

Suppose there are the following two topic cluster tuples:

t'_i: 2015 Conference Data Integration

t'_j: Beijing Conference 2015 Data Integration

Next, we calculate the *Euclidean distance*, *Edit distance* and *Hamming distance* between t'_i and t'_j.

We extract all of the terms (i.e., 2015, Conference, Beijing, Data, Integration) from the tuples and construct the vectors V_i=[1,1,0,1,1] and V_j=[1,1,1,1,1]. Then we calculate the comprehensive similarity between t'_i and t'_j using the above two vectors.

***Euclidean distance*:**

$$d_1(t'_i, t'_j) = \sqrt{(1-1)^2 + (1-1)^2 + (0-1)^2 + (1-1)^2 + (1-1)^2} = 1$$

***Edit distance*:**

We need at least edit 3 times to convert t'_i to t'_j: insert "Beijing" before the word "2015" in t'_i; remove the word "2015" from t'_i; insert the "2015" between the words "Conference" and "Data". So the Edit distance between t'_i and t'_j is $d_2(t_i', t_j')=3$.

***Hamming distance*:**

Topic cluster tuples t'_i and t'_j are encoded to the corresponding codewords $M_i = (1,1,0,1,1)$ and $M_j = (1,1,1,1,1)$. To transform M_i to M_j, we need to replace a character. Thus, the Hamming distance between t'_i and t'_j is $d_3(t'_i, t'_j)=1$.

After calculating these three distances between t'_i and t'_j, we can get the similarity between them using Eq. (7).

$$Sim_1(t'_i, t'_j) = 1 - d_1(t'_i, t'_j) / \max_{k\in\{1,2,3\}} \{d_k(t'_i, t'_j)\} = 1 - 1/3 = 0.67$$
$$Sim_2(t'_i, t'_j) = 1 - d_2(t'_i, t'_j) / \max_{k\in\{1,2,3\}} \{d_k(t'_i, t'_j)\} = 0$$
$$Sim_3(t'_i, t'_j) = 1 - d_3(t'_i, t'_j) / \max_{k\in\{1,2,3\}} \{d_k(t'_i, t'_j)\} = 1 - 1/3 = 0.67$$
$$Sim(t'_i, t'_j) = 1 / \left[1 + exp\left(-\sum_{k=1}^{3} Sim_k(t'_i, t'_j)\right)\right] = 1/(1+e^{-1.34}) = 0.79$$

(2) Strategy of horizontal grouping

Algorithm 2: Horizontal grouping based on the association graph of topic cluster tuples

① Construct the association graph of topic cluster tuples by calculating the comprehensive similarity between tuples in each topic cluster;

② The association graph is divided into several topic cluster units (connected components) with the similarity threshold ϕ;

③ Find out the topic cluster units which contain more than *Minsize* topic cluster tuples. Calculate their degrees of cohesion and select the topic cluster with minimum values. Remove the edge with the smallest weight;

④ Repeat step 3 until the number of tuples in each topic cluster unit is less than *Minsize*;

⑤ Calculate the separation between topic cluster units and merge the units with the minimum;

⑥ Repeat step 5 until the number of topic cluster tuples reaches r ;

The cohesion and separation of the topic cluster units in step 3 and 5 can be calculated by the following formulas:

$$I(TCU_m) = \sum_{t'_i\in TCU_m, t'_j\in TCU_m} Sim(t'_i, t'_j) \tag{8}$$

$$S(TCU_m, TCU_n) = \frac{1}{\sum_{t'_i\in TCU_m, t'_j\in TCU_n} Sim(t'_i, t'_j)} \tag{9}$$

All parameters involved in the algorithm include the similarity threshold ϕ, the number of topic cluster units r in each topic cluster and *Minsize*. The first two parameters are given by the users, *Minsize* is generally set to 1% ~ 3% of the number of tuples in each topic cluster.

4.2 Optimization Mechanism of Index Based on Association Rule Algorithm

4.2.1 Construction of Subject Index

In order to speed up queries, we need to construct indexes for the database. Traditional indexing methods create indexes for every single word. Their efficiency is greatly affected in multi-keyword query. To solve the above problem, this section presents an index mechanism for the topic cluster units that uses the association rule algorithm to select frequent item sets as index terms. Each frequent item corresponds to an index entry. The structure of inverted index is as follows:

$$Keyword(s) \rightarrow (TCU_1, TCU_2 \ldots)$$

We use *Lucene toolkit* (a full-text search engine which provides a complete query engine and indexing engine) to generate the index over the topic cluster units, its construction algorithm is presented as follows:

Algorithm 3: Index Construction

```
Input: $C = \{C_1, C_2, ..., C_k\}$, minimum support $min\_sup$;
Output: Index $L$;
① for $each\ C_i \in C$
② $FrequentItemsList \leftarrow Apriori(C_i, min\_sup)$
③ $F = FrequentItemsList - 1\text{-}itemsets$
④ for $each\ TCU_j \in C_i$
for $each\ token_k \in F$
    if $(token_k\ exists\ in\ TCU_j)$ then
 $L[token_k, TCU_j]$;
 endif
endfor
for $each\ token_l \in TCU_j$
    $L[token_k, TCU_j]$;
endfor
endfor
        endfor
⑤ $return\ L$;
```

4.2.2 Query Processing

Using above indexes over topic cluster units, the topic cluster units containing the search keywords can be returned to users as the query results. Given a set of query keywords $K = \{k_1, k_2, ..., k_n\}$, the system scans multiple indexes in parallel, and it then calculates the score of each relevant topic cluster units based on an existing ranking function. Finally, we rank the results in descending order and return *top-k* results to users.

5 EXPERIMENTAL RESULTS
5.1 Dataset and Environment Setup

We have designed and performed a comprehensive set of experiments on the real dataset *Freebase*. *Freebase* is a shared database with a large size and complex schema. It contains approximately 2000 tables and 300 million entities. Due to the limitation in our laboratory equipment, we simplify the dataset without affecting the experimental results. We extract a small portion of data (about 400 MB) from the *Freebase* database for our experiments, meanwhile maintaining the database schema and the relationships between tables unchanged. To validate the performance of theTCU-based query method proposed in the paper, the following three groups of experiments are conducted. Experiment 1 examines the effectiveness of the optimization scheme of table join order by evaluating the preprocessing time of the method TCU-based and *Naive* (that is, a straightforward solution without optimizing the table join order). In Experiment 2, we compare our approach against the baseline methods of *DBXplorer*, *BLANKS* and *SAINT* [5, 11, 13], the results indicate that the efficiency and effectiveness of the method TCU-based have been further improved. In Experiment 3, comparative experimental results on different datasets show that the TCU-based method has a good scalability.

Our experimental platform is a computer running the Windows 7, with Intel®Core (TM) 2.5 GHz CPU, a 4 GB of RAM and 500G Disk. All the algorithms are implemented in Java.

5.2 Experimental Evaluation
5.2.1 Optimization Scheme Evaluation

Experimental comparison between TCU-based query and *Naive* approach is conducted on the databases with the different number of the tables. Fig. 7 reports the preprocessing time for TCU-based query and *Naive* approach by varying the number of the tables. From the results, we observe that the TCU-based method beats the method *Naive* at the preprocessing time. The main reason is that TCU-based query method optimizes the join order among the tables. Additionally, the bigger the number of tables is, the more obvious the advantage of the scheme is.

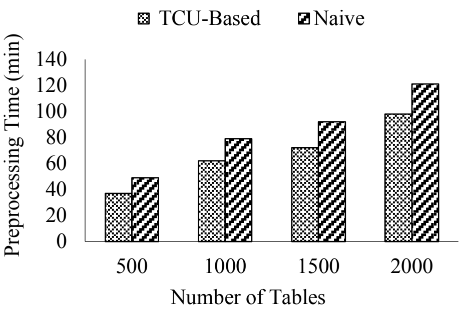


Figure 7 Comparison of preprocessing time

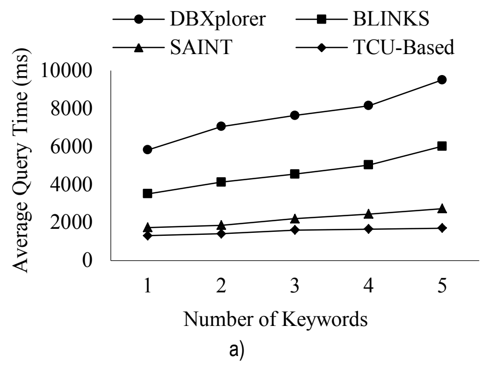
5.2.2 Comparison with State-of-the-Art Approaches

This section focuses on the performance of the TCU-based method. We randomly select 100 keyword queries with different numbers of keywords from query logs, then we evaluate the efficiency and the effectiveness of our method by comparing it with the baseline methods of *DBXplorer*, *BLANKS* and *SAINT*.

(1) Query efficiency

We vary the different numbers of input keywords to identify all the answers and compare the corresponding average *top-2* response time. Query response time is the duration from the instance that a user issues a query Q, until the time it produces the *top-2* results, excluding the offline data preprocessing time. As expected, the query length affects the response time significantly. Longer queries result in larger response time. The figure also demonstrates that the offline query methods *SAINT* and TCU-based achieve much higher efficiency than the online query methods *DBXplorer* and *BLANKS*.

The main reason is that the latter two methods require to perform complex join operations of tables during the query phases. Furthermore, it is not surprising that it takes more time for above table joins to perform keyword search over the database with a complex schema than over that with a simple schema. In contrast, the tables and tuples in the database are preprocessed in offline query methods, thereby a lot of time can be saved for real search and the query speed is greatly improved. Moreover, compared to the method *SAINT*, the method TCU-based proposed in this paper has a great improvement in query performance. More specifically, the TCU-based method constructs a subject index for every topic cluster, and the indexes can be scanned in parallel. So it has a superior performance compared to *SAINT*.



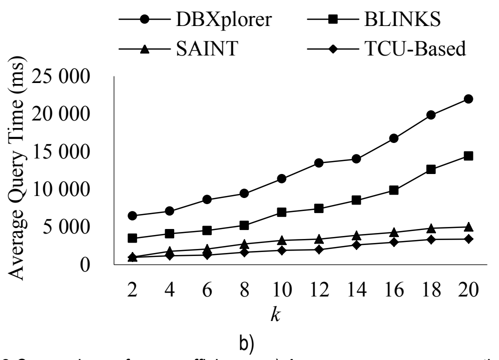


Figure 8 Comparison of query efficiency: a) Average query response time with different numbers of keywords, b) Average query response time with different values of *k*

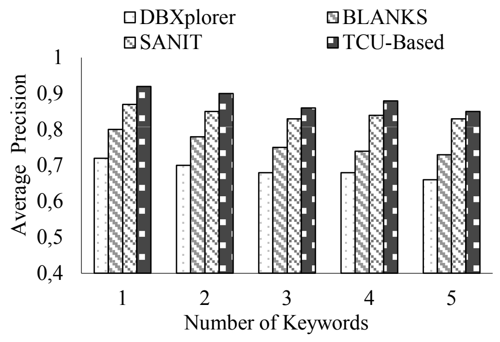
Fig. 8a plots that the performance of our method is improved slightly when the number of keywords is greater than 3. It is mainly because our method selects index terms using the association rule. It can directly scan the index items containing all query keywords, which does not need join index.

In order to better evaluate the performance of our query method, we identify the *top-k* answers with different values of *k* and compare the corresponding average response time. In Fig. 8b the different *k* is taken as *x*-axis and the corresponding average response time is taken as *y*-axis. Obviously, the method proposed in this paper significantly outperforms the baseline methods, suggesting that using topic cluster units is crucial to obtain good performance. For example, considering finding *top-12* results of the queries, our method costs about 2014 ms while *DBXplorer*, *BLANKS*, *SAINT* consume 13500 ms, 7452 ms and 3420 ms respectively. As *k* grows, the response time of our method increases much more slowly than the other three methods. The reason is very similar to the one described in Fig. 8a, we will not repeat it here. This comparison further demonstrates the high efficiency of our method.

(2) Query effectiveness

In this section, we report the effectiveness of the TCU-based query method in comparison with the other three approaches. The query effectiveness of all methods, namely, the quality of returned answers, is measured by the average precision and average recall metrics. Given a set of keyword queries, we employ the corresponding *SQL* queries to generate our baseline query results which are accurate and complete, so that they can be used to compare with the results of keyword query methods. The average precision and average recall of different methods mentioned above are illustrated in Fig. 9.

An interesting observation is that the accuracy of the query methods is gradually reduced when we increase the number of keywords. Specifically, the accuracy of 1-keyword and 2-keyword queries is generally higher than that of 3-keyword, 4-keyword and 5-keyword. This is because the relationships between keywords become more complex when the number of query keywords increases. The TCU-based method achieves roughly 90 percent precision, which leads to about 3~7 percent over the alternative method *SAINT*, and it is even higher when compared with the other methods such as *DBXplorer* and *BLANKS*. Fig. 9b shows that the TCU-based method outperforms the other three kinds of methods in terms of the recall, and the recall of our approach is about 15 percent higher than that of *SAINT*.



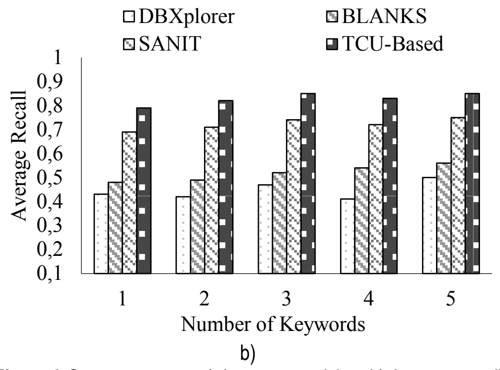


Figure 9 Query accuracy: a) Average precision, b) Average recall

The reason is that our method utilizes the additional pretreatment and then integrates multiple topic cluster units to answer a keyword query, which can improve both precision and recall. The information in database is preprocessed, which includes the Vertical grouping and Horizontal grouping. The Vertical grouping takes into account both the topology compactness and content similarity. Besides, combining the query logs, it can use the history queries to adjust the result of the Vertical grouping. This makes the data tables with a higher topic similarity be classified into the same topic cluster and the result of Vertical grouping is of higher classification accuracy. Horizontal grouping classifies the tuples with high similarity into the same topic cluster units by using a hybrid similarity measurement method. Through the above preprocessing operations, the subsequent query results can contain more comprehensive and accurate information. Comparing with other methods, the TCU-based method can discover answers that indirectly contain query keywords and consider the semantic correlation. Consequently, theTCU-based method is effective enough to provide high-quality query service.

5.2.3 Scalability of TCU-Based Query

This part of the experiment investigates the scalability of TCU-based query method. We study how the size of database affects the performance of the query method. The measurement of *top-5* average query response time is described in Experiment 3.

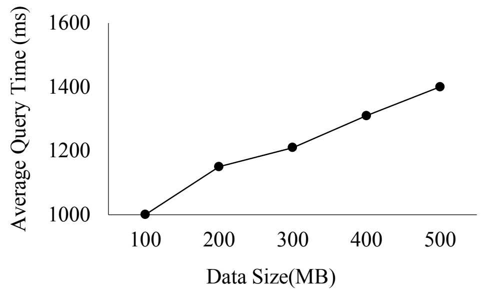


Figure 10 Average query response time with different databases

In Fig. 10, the *x-axis* represents the size of databases while the *y-axis* represents *top-5* average query response time. In this experiment, datasets DB_1, DB_2, DB_3, DB_4 and DB_5 are used, and their sizes vary from 100 MB to 500 MB. As the size of dataset grows, the response time of our method increases slowly. This is because the size of dataset only exerts great influence on the preprocessing time but not the online indexing. This experiment shows that our method is extensible on databases with different size.

6 CONCLUSION

This paper has studied the keyword query over relational databases. We put forward TCU-based query method — an offline query method based on the topic cluster units, which is suitable for large-scale databases with complex schemas and enormous data. Firstly, we define the concept of topic cluster units. In order to construct topic cluster units, we use an improved spectral clustering algorithm to make a vertical grouping on tables and use an association graph of topic cluster tuples to make a horizontal grouping on tuples. Tuples with high topology compactness, content similarities and co-occurrence in query logs are clustered into the same topic cluster through the above options. We use the topic cluster units as results which contain richer and more complete semantic information. Secondly, this paper designs an optimized solution for table join ordering based on genetic algorithm to reduce the cost of data preprocessing. Finally, we select index terms using the association rule to construct indexes for topic clusters, and thus the query speed is improved significantly.

In future research, we will continue to study the parallel query in distributed databases and the dynamic construction of indexes. They will also have positive effects on the query efficiency.

ACKNOWLEDGEMENT

This work is sponsored by the National Natural Science Foundation of China under Grant No. 61272185, and the Natural Science Foundation of Heilongjiang Province of China under Grant No. F020510 and F201238 and the Basic Research Project(No. JCKY2016206B001 and JCKY2015206C002).

7 REFERENCES

[1] Zuze, H. & Weideman, M. (2013). Keyword stuffing and the big three search engines. *Online Information Review, 37*(2), 268-286. https://doi.org/10.1108/OIR-11-2011-0193

[2] Chen, J., Chen, Y., Du, X. et al. (2013). Big data challenge: A data management perspective. *Frontiers of Computer Science, 7*(2), 157-164. https://doi.org/10.1007/s11704-013-3903-7

[3] Brown, P., Haas, P., Myllymaki, J. et al. (2005). Toward automated large-scale information integration and discovery. *Proceeding of Data Management in a Connected World* / Heidelberg, 161-180. https://doi.org/10.1007/11499923_9

[4] Yu, J. X., Qin, L., & Chang, L. (2010). Keyword Search in Relational Databases: A Survey. *IEEE Data Eng Bull, 33*(1), 67-78.

[5] Agrawal, S., Chaudhuri, S., & Das, G. (2002). DBXplorer: A system for keyword-based search over relational databases. *Proceeding of 18th International Conference on Data Engineering* / San Jose, CA, US, 5-16. https://doi.org/10.1109/ICDE.2002.994693

[6] Hristidis, V. & Papakonstantinou, Y. (2002). Discover: keyword search in relational databases. *Proceeding of the 28th international conference on Very Large Data Bases* / Hong Kong, China, 670-681. https://doi.org/10.1016/B978-155860869-6/50065-2

[7] Luo, Y., Lin, X., Wang, W. et al. (2007). Spark: top-k keyword query in relational databases. *Proceeding of the 2007 ACM SIGMOD International Conference on Management of Data* / Beijing, China, 115-126. https://doi.org/10.1145/1247480.1247495

[8] Bhalotia, G., Hulgeri, A., Nakhe, C. et al. (2002). Keyword searching and browsing in databases using BANKS. *Proceeding of the 18th International Conference on Data Engineering* / San Jose, CA, US, 431-440. https://doi.org/10.1109/ICDE.2002.994756

[9] Hristidis, V., Gravano, L., & Papakonstantinou, Y. (2003). Efficient IR-style keyword search over relational databases. *Proceeding of the 29th international conference on Very large data bases* / Berlin, Germany, 850-861. https://doi.org/10.1016/B978-012722442-8/50080-X

[10] Kacholia, V., Pandit, S., Chakrabarti, S. et al. (2005). Bidirectional expansion for keyword search on graph databases. *Proceeding of the 31st international conference on Very large data bases* / Trondheim, Norway, 505-516.

[11] He, H., Wang, H., Yang, J. et al. (2007). BLINKS: ranked keyword searches on graphs. *Proceeding of SIGMOD 2007: ACM SIGMOD International Conference on Management of Data* / Beijing, China, 305-316. https://doi.org/10.1145/1247480.1247516

[12] Su, Q. & Widom, J. (2005). Indexing relational database content offline for efficient keyword-based search. *Proceeding of 9th International Database Engineering and Application Symposium* / Montreal, QC, Canada, 297-306.

[13] Feng, J., Li, G., & Wang, J. (2011). Finding top-k answers in keyword search over relational databases using tuple units. *IEEE Trans Knowl Data Eng. 23*(12), 1781-1794. https://doi.org/10.1109/TKDE.2011.61

[14] Freebase [Online]. http://www.freebase.com

[15] Li, G., Feng, J., & Zhou, L. (2008). Retune: Retrieving and materializing tuple units for effective keyword search over relational databases. *Proceeding of 27th International Conference on Conceptual Modeling* / Barcelona, Spain, 469-483. https://doi.org/10.1007/978-3-540-87877-3_34

[16] Li, G., Feng, J., & Wang, J. (2009). Structure-aware indexing for keyword search in databases. *Proceeding of ACM 18th International Conference on Information and Knowledge Management* / Hong Kong, China, 1453-1456. https://doi.org/10.1145/1645953.1646143

[17] Carlsson, G. (2009). Topology and Data. *Bulletin of the American Mathematical Society, 46*(2), 255-308. https://doi.org/10.1090/S0273-0979-09-01249-X

[18] Gan, W.-Y., He, N., Li, D.-Y. et al. (2009). Community discovery method in networks based on topological potential. Ruan Jian Xue Bao/Journal of Software, 20(8), 2241-2254. https://doi.org/10.3724/SP.J.1001.2009.03318

[19] Turney, P. D. & Pantel, P. (2010). From Frequency to Meaning: Vector Space Models of Semantics. *Journal of Artificial Intelligence Research, 37*(4), 141-188.

[20] Wang, Z., Chen, Z., Zhao, Y. et al. (2014). A Community Detection Algorithm Based on Topology Potential and Spectral Clustering. *The Scientific World Journal*, 2, 325-333. https://doi.org/10.1155/2014/329325

Contact information:

Yingqi WANG, PhD candidate
Corresponding author
School of Computer Science and Technology,
Harbin Engineering University
Harbin, 150001, China
E-mail: hgcwyq@hrbeu.edu.cn

Nianbin WANG, Prof. PhD
School of Computer Science and Technology,
Harbin Engineering University
Harbin, 150001, China
E-mail: wangnianbin@hrbeu.edu.cn

Lianke ZHOU, PhD
School of Computer Science and Technology,
Harbin Engineering University
Harbin, 150001, China
E-mail: zhoulianke@hrbeu.edu.cn