Received January 26, 2020, accepted February 6, 2020, date of publication February 11, 2020, date of current version February 19, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2973217

An Efficient Relational Database Keyword Search Scheme Based on Combined Candidate Network Evaluation

GUOHUI DING[1], HAOHAN SUN[1], JIAJIA LI[1], CHENYANG LI[1], RU WEI[1], AND YUNFENG FEI[2]
[1]School of Computer Science of Shenyang Aerospace University, Shenyang 110136, China
[2]Beijing Huayu Information Technology Company Ltd., Dalian 116036, China

Corresponding author: Jiajia Li (lijiajia@sau.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61303016 and Grant 61502317, and in part by the Natural Science Foundation of Liaoning Province under Grant 2019-MS-253.

ABSTRACT Relational keyword search (R-KWS) systems provide users with a convenient means in relational database queries. There exist two main types of R-KWS: those based on Data Graphs and those based on Schema Graphs. In this paper, we focus on the latter, R-KWS based on Schema Graphs. Most existing methods are typically inefficient due to the large number of repetitive operation caused by overlapping candidate networks and the execution of lots of complex queries. We present the R-KWS approach based on combined candidate network evaluation to improve the query efficiency. The proposed Combined Candidate Network (CCN) can efficiently share the overlapping part between candidate networks, and then avoid the repetitive operation during the evaluation of candidate networks. Meanwhile, CCN possesses another important characteristic that candidate networks within a CCN are still identifiable after candidate networks being compressed into a CCN. We design an algorithm based on this characteristic to evaluate CCN for the generation of final query results. This algorithm is able to eliminate the execution of a large number of complex queries required by most existing approaches, and thus significantly improve the efficiency of keyword search. Experiments on real datasets show that our approach can improve query efficiency without any loss of the quality of query results with respect to existing approaches.

INDEX TERMS Relational database, keyword query, schema graphs, combined candidate network.

I. INTRODUCTION

Structured Query Language (SQL) is the main means to access data from a relational database, which requires users to understand the complex SQL syntax and schema information of database [4], [27], [29], [31]. Compared with SQL, relational database keyword search (R-KWS) is simpler, which enables users to query information from a relational database by the way of search engines.

Recently, lots of works about R-KWS are proposed, which can be classified into two main types: R-KWS based on Data Graphs [4]–[7], [28], [30] and R-KWS based on Schema Graphs [8]–[24]. R-KWS based on Data Graphs typically needs to preload the Data Graphs into memory, which cannot sometimes be completed at a time because of the consumption of a large amount of memory [7]. R-KWS based on Schema Graphs also represents a relational database as a graph whose nodes represent tuple sets each of which includes tuples from the same relation, and edges represent join between the corresponding relations. The final query results can be generated by a large number of complex SQL queries derived from Candidate Networks (CNs).

This paper mainly focuses on the research of R-KWS based on Schema Graphs, which makes full use of the function of relational database to retrieve the query results. It has a general problem of low query efficiency [8]–[15] because of the execution of a large number of complex queries. In addition, the overlapping part between CNs in R-KWS causes the same operation to be executed by different CN evaluation plans for many times, which further exacerbates the efficiency problem of traditional methods. Most existing approaches [9]–[20], [22] return the final search results based on the execution of top-*k* queries with complex joins among multiple relations, which are derived from CNs. These complex queries

The associate editor coordinating the review of this manuscript and approving it for publication was Yongqiang Zhao.

significantly decrease the efficiency of keyword search in traditional approaches.

Consequently, we propose an approach based on CCN evaluation to solve the problem of low query efficiency of R-KWS based on Schema Graphs. The novel CCN (actually it is a tree) is proposed to efficiently share the overlapping structure between CNs, and thus saves the large amount of redundant operation during the process of generation of query results. Meanwhile, CCN possesses another important characteristic that CNs within a CCN are still identifiable after CNs being combined. We design an algorithm based on this characteristic to evaluate CCN for the generation of final query results. The CCN evaluation is actually the restricted traversal of tuple sets in CCN and does not require the conversion from CNs to SQL queries as well as the following execution of complex queries in database, which are typically required by most existing approaches. Consequently, the proposed algorithm significantly improves the efficiency of keyword search. We conduct extensive experiments based on real datasets, and experimental results show that the proposed approach is more efficient than existing methods.

The main contributions of this paper are as follows:

- The novel CCN is proposed to efficiently share the repetitive part between CNs, and thus eliminates redundant operation causing extra time cost. Meanwhile, we propose an algorithm to generate CCNs.
- We propose an efficient algorithm based on CCN evaluation to generate the final query results, which greatly reduces the complex database query operation required by traditional methods.
- Extensive experiments based on real datasets are conducted to compare the proposed approach with existing approaches, and the experimental results show that our approach can significantly improve query efficiency of R-KWS.

The related work is introduced in Section II. Definitions are described in Section III. Section IV presents the proposed method. The experimental results are analyzed in Section V. We conclude in Section VI.

II. RELATED WORK

Recently, R-KWS [25], [32]–[38] has become a hot topic in database community. The R-KWS based on Schema Graphs has query flexibility but with high execution cost. A large amount of relevant literature has been devoted to the improvement of query efficiency of R-KWS based on Schema Graphs. However, the evaluation of CNs by most approaches [15]–[18] results in lower query efficiency. In order to solve this problem, DISCOVER-II [19] introduces an IR ordering strategy which employs the sparse algorithm and the global pipeline algorithm to calculate relevance for CNs and evaluate CNs according to the relevance. DISCOVER-II will not evaluate any other CNs after generating top-k query result, so as to improve query efficiency. Lu *et al.* [1] propose the multiple keyword queries on graph data while most traditional approaches focus on a single query setting. SPARK [20] proposes skyline scan algorithm which can achieve the minimum amount of data exploration. In addition, SPARK also uses a non-monotonic upper limit function to further limit unnecessary data access. Li *et al.* [3] propose an efficient and progressive group steiner tree algorithm to overcome the limitation of traditional parameterized dynamic programming algorithm for keyword search in relational databases. CNRank [21] proposes a CN ranking algorithm based on the Bayesian network model which scores CNs according to the probability of generating the result and the relevance of CNs. The few CNs with the highest scores in CNRank contain most of the query results. As a result, CNRank only evaluates a small number of CNs to improve search efficiency. DISCOVER proposes a plan generator for redundant operation during the evaluation of CNs. The plan generator can eliminate redundant operation by establishing intermediate relationships which also need a high evaluation cost. Lu *et al.* [2] present the parallel computing keyword queries on a multicore architecture. They study the query level parallelism, operation level parallelism and data level parallelism. However, their approach is still based on the traditional thought evaluating CNs into SQLs, which is the main difference in contrast to our approach.

Although the existing methods improve the query efficiency of R-KWS from different perspectives, most of them cannot avoid the CN evaluation which is really time-consuming. Consequently, the CCN evaluation is proposed to improve the query efficiency by sharing the repetitive part between CNs and avoiding the execution of complex queries in database.

III. PRELIMINARIES

In this section, some classic concepts in R-KWS research area are introduced from work [12]–[14], and some modifications are made to them. Schema Graph G represents a relational schema in which nodes represent relations and edges represent foreign key constraint between relations.

Definition 1: A joining network of tuples j is a tree with tuples as nodes. For any pair of adjacent tuples t_p and t_q in j, where t_p and t_q belong to the relations R_p and R_q respectively, then there must be an edge $\langle R_p,R_q\rangle$ in G and $\langle t_p,t_q \rangle \in \langle R_p, R_q \rangle$.

Definition 2: Given a keyword query $Q = \{k_1, \cdots, k_n\}$, a joining network of tuples j is Minimal Total Joining Network of Tuples (MTJNT), when it is both total, that is, j contains all the keywords in Q, and minimal, that is, j is no longer a total joining network of tuples if any node is removed in j.

Definition 3: Given a keyword query $Q = \{k_1, \cdots, k_n\}$, a Tuple Set R_p^K is a collection of tuples that contains all the keywords in K which is a subset of Q but does not contain any keywords in $Q - K$ from the relation R_p. If $K = \emptyset$, the tuple set is called free tuple set, otherwise the tuple set is called key tuple set. For example, $P^{\{k_1,k_2,\ldots,k_{n-1}\}}$ represents a key tuple set including keywords $\{k_1, \ldots, k_{n-1}\}$ but excluding k_n from table *PAPER*.

Definition 4: Given a keyword query $Q = \{k_1, \cdots, k_n\}$, a Candidate Network C is a tree with tuple sets as nodes. For any pair of adjacent tuple sets R_p^K and R_q^M in C, there must exist an edge $\langle R_p, R_q \rangle$ in G. Furthermore, C is both total, that is, C contains all the keywords in Q, and minimal, that is, C is no longer a complete tree if any node is removed.

CN is a key component of R-KWS based on Schema Graph, which is used to generate the final query results. However, CNs generated by traditional algorithms typically have overlap. Thus, the CCN is introduced to solve the problem of repetitive operation caused by overlap. As shown in the Definition 2, MTJNT is actually the final query results returned to users. MTJNT shows users which tuples in database include their interested keywords and how these tuples are organized by the foreign key references between them.

IV. COMBINED CANDIDATE NETWORK EVALUATION

In this section, we introduce our approach in details. The overall framework of our approach is presented in subsection IV-A, while the corresponding CCN generation and the MTJNT generation are described in the following subsections respectively.

A. FRAMEWORK OF COMBINED CANDIDATE NETWORK EVALUATION

The framework of CCN evaluation is shown in Figure 1. A user enters the keywords at the interactive interface to start the whole process (step 1). The system firstly queries the database for tuples containing the keywords, and generates tuple sets according to the return results (step 2). Then we use the traditional methods [12], [19] to generate CNs based on the tuple sets and schema information (step 3). CCNs are generated based on the CNs in step 4, which is detailed in subsection IV-B. Next, MTJNTs are generated by traversing CCNs (step 5, namely subsection IV-C). Finally, the system measures MTJNTs, and returns the top-k results with the highest score to the user (step 6). Initialization phase creates a set of the foreign key constraint to support the execution of the MTJNT generation algorithm (step 7).

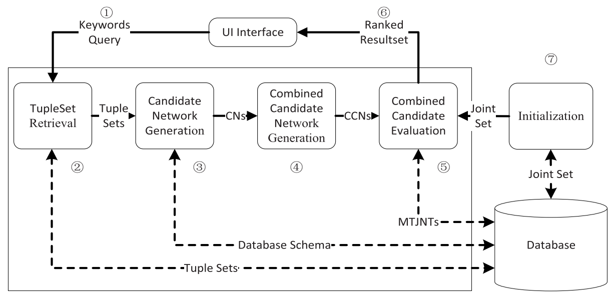


FIGURE 1. Framework of CCN evaluation.

B. ALGORITHM OF CCN GENERATION

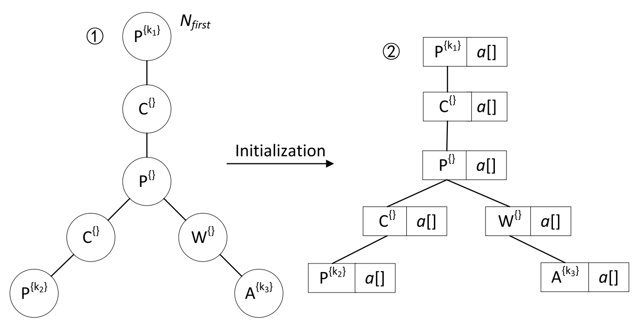
R-KWS based on Schema Graphs usually deals with each CN separately, which results in the overlapping part between CNs to be handled for many times. To solve this problem, we propose the novel concept CCN which compresses multiple CNs sharing repetitive part into a tree whose nodes are still tuple sets. CCN is able to eliminate redundant part causing unnecessary operation by sharing repetitive structure between CNs. In order to generate MTJNT from each CN in a CCN, we design CCN with an important feature that enables each CN to be identifiable within a CCN. However, it is difficult to fulfill this requirement due to the repetitive structure between CNs. To achieve this feature, we propose a concept duplicate layer in Definition 5.

Definition 5: Let ccn be a CCN which is formed by combining a set of CNs $M = \{cn_1, cn_2, \ldots, cn_n\}$, and $\hat{M} = \{S_1, S_2, \ldots, S_n\}$ be a set where $S_i \in \hat{M}$ is the node set of $cn_i \in M$. Let $U_d = \{S_1^d, S_2^d, \ldots, S_k^d\}_{1 \leq k \leq n}$ be a set where $U_d \subseteq \hat{M}$. If U_d satisfies $|U_d| = \max(|U_x|)_{\forall U_x \subseteq \hat{M}}$ and $D = S_1^d \cap S_2^d \cap \ldots \cap S_k^d \neq \emptyset$, then D is defined as a duplicate layer. For any $S_i \in \hat{M}$, if the difference $\tilde{D} = S_i - S_1 \cup \ldots S_{i-1} \cup S_{i+1} \ldots \cup S_n$ is not empty, then $\tilde{D}$ is also defined as a duplicate layer.

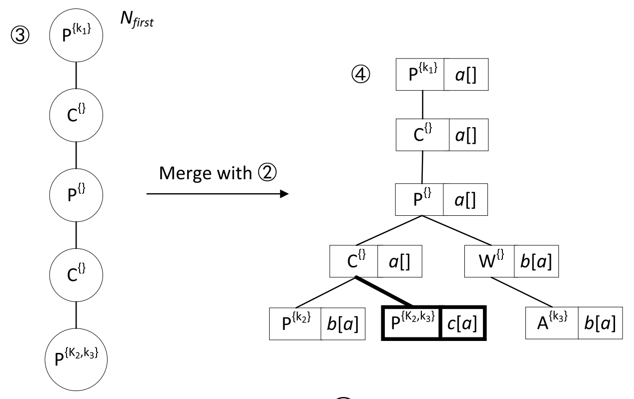
The node set of a CCN can be divided as duplicate layers without intersection. CCN can also be considered as a tree with duplicate layers as nodes. The duplicate layer with children is owned by multiple CNs while the leaf duplicate layer belongs to only one CN. Actually, given a CCN, its CN combined can be identified by a unique path from the root to the leaf in the corresponding duplicate layer tree. In order to maintain a duplicate layer tree, we design codes to each node in a CCN. It contains two dimensions, *name* field and *parent* field. Given a node n in a CCN, *name* represents the identifier of the duplicate layer where n exists while *parent* is the identifier of its parent duplicate layer.

CCN generation along with the duplicate layer division process is shown in Figure 2. The capital letters P, A, C, W represent the database tables *PAPER*, *AUTHOR*, *CITE* (relationship between *PAPER*s) and *WRITE* (relationship between *AUTHOR* and *PAPER*) respectively. The figures in the left side are examples of CNs. Each node in a CN is a tuple set. For example, $P^{\{k_1\}}$ represents a key tuple set including keyword k_1 but excluding all the rest keywords from table *PAPER* while the capital letter without keywords (like $C^{\{\}}$) is free tuple set [12]. At the beginning, any CN is chosen as an initialization to form the first CCN. For example, the CN with label ① in Figure 2(a) is used as the initial CCN whose nodes' *name* and *parent* fields are set to the code a and null (donated by $a[]$) respectively. Then, all the CNs with the same first node as the CCN are added to the CCN ②. Note that only CN and CCN that have the same first node are able to have repetitive structure.

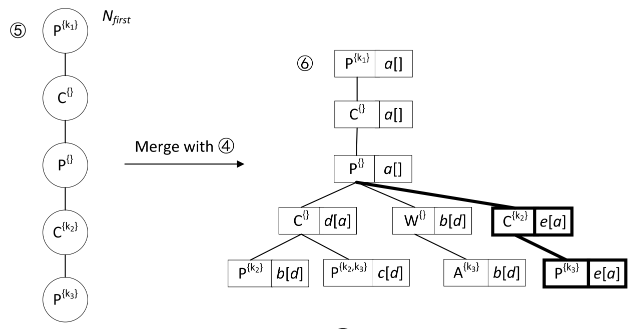
The process of how to combine a CN into a CCN is shown in the following. Given a CN cn_1 and a CCN ccn_1, we firstly need to find the repetitive structure. It is determined whether the first duplicate layer dl_1 of ccn_1 exists in cn_1. If dl_1 exists in cn_1, then we check whether any child of dl_1 exists in cn_1. This process is repeated until no duplicate layer exists in cn_1



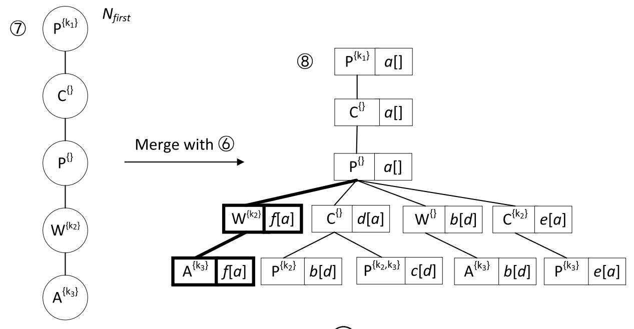
(a) Initial CCN



(b) Add CN ③ to CCN



(c) Add CN ⑤ to CCN



(d) Add CN ⑦ to CCN

FIGURE 2. CCN generation process. (a) Initial CCN.

or arriving at the leaf node, which is an iterative process like DFS in a graph. During each of iteration, we will find a child sdl_1 of a given duplicate layer, satisfying that sdl_1 exists in cn_1. The repetitive structure is determined by the iterative process.

There exist two result cases from the perspective of duplicate layer comparison when the algorithm terminates. The first case is that only a part of the duplicate layer currently being compared exists in the CN to be combined. For this case, the non-repetitive part (the bold part in Figure 2(b)) of the CN is added as a new duplicate layer to the given CCN, and the current duplicate layer becomes its parent. Meanwhile, the repetitive part is separated from the current duplicate layer and becomes a new one with the current duplicate layer as its parent. In addition, the parent of all the original children of the current duplicate layer is changed to the duplicate layer newly created and separated from the current duplicate layer; that is, the *parent* fields of all children are modified to the name of the new duplicate layer.

For example, both the first duplicate layers of CCN ④ and CN ⑤ have the overlapping structure $P^{\{k_1\}}$-$C^{\{\}}$-$P^{\{\}}$ in Figure 2. Therefore, the non-repetitive structure $C^{\{k_2\}}$-$P^{\{k_3\}}$ of CN ⑤ is added to CCN ④, and then becomes a new duplicate layer whose corresponding *name* field and *parent* field are set to e and a, respectively. The remaining, non-repetitive structure $C^{\{\}}$ of the current duplicate layer of CCN ④ is separated as a new duplicate layer whose *name* and *parent* are set as d and a, respectively. In addition, the *parent* fields of all the original children of the current duplicate layer are modified to d which is the new duplicate layer separated from the current duplicate layer. The process in Figure 2 (b) also falls into this category. The second case is that the whole duplicate layer currently being compared exists in the CN to be combined and there is not any overlap between each of its children and the CN. For this case, the non-repetitive part of the CN is added as a new duplicate layer to the CCN (shown in bold in Figure 2) and its parent is set to the current duplicate layer. For example, CN ⑦ contains the first duplicate layer of CCN ⑧ whose all children are completely absent in CN ⑦ and no part of each child appears in CN ⑦. Thus, we just need to add the non-repetitive structure $W^{\{k_2\}}$-$A^{\{k_3\}}$ of CN ⑦ to CCN ⑧ as a new duplicate layer and set the current duplicate layer a to be its parent.

The CCN generation process is shown in Algorithm 1. First of all, a CN which is not added into any CCN is used as the initial TempCCN (Line 4) and the *name* field is set for its nodes (Line 5). Then, the CNs that have the same first node with the TempCCN are added to the TempCCN in turn (Lines 6-19). When a CN is added to the CCN, the first duplicate layer of the CCN is compared with the CN (Line 7). If the first duplicate layer exists in the CN, then the algorithm checks whether one of its children exists in the CN (Lines 8-10). This process is repeated until there is only a part of the current duplicate layer existing in the CN or all its children do not exist in the CN. If the current duplicate layer exists in the CN and all its children do not exist in the CN (Line 11), the algorithm adds the non-repetitive structure of the CN as a new duplicate layer to the CCN (Line 12) and set *name* field and *parent* field for its nodes (Line 13). If only a part of the current duplicate layer exists in the CN (Line 14), add a non-repetitive structure of the CN as a

Algorithm 1 CCN Generation

```
Input: A set of CN CNs; Threshold of CN CNThreshold
Output: A set of CCN CCNs
1:  function CCNGeneration
2:      Initialize queue TempCCN // stores intermediate CCN
3:      for CNa∈CNs∧IsNew(CNa) do
4:          TempCCN.AddCN(CNa)
5:          SetTag(TempCCN)
6:          for all CNb∈CNs ∧
7:            SameFirstNode(TempCCN, CNb) do
8:              tag←HeadTag (TempCCN)
9:              for all stag∈SonTag(tag) ∧
10:               CompleteRepeat(stag, CNb) do
11:                 tag←stag
12:             end for
13:             if NoNodeRepeat(TempCCN, tag, CNb) then
14:                 InsertNode(TempCCN, CNb)
15:                 SetTag (TempCCN, tag)
16:             else
17:                 InsertNode(TempCCN, CNb)
18:                 SetTag(TempCCN, stag)
19:                 ResetTag (TempCCN, stag)
20:             end if
21:         end for
22:         CCNs.Add(TempCCN)
23:         Clear(TempCCN)
24:     end for
25: end function
```

new duplicate layer to the CCN (Line 15) and set *name* and *parent* for its nodes (Line 16). In addition, the non-repetitive structure of the current duplicate layer is separated as a new duplicate layer and the *parent* fields of all original children of the current duplicate layer are modified (Line 17). The final CCN generated is added to the CCN set (Line 20), and the TempCCN is reset (Line 21) for the next CCN.

C. CCN EVALUATION

The main part of CCN evaluation is how to generate the final results of MTJNTs. Traditional methods for MTJNT generation typically convert a CN into a SQL query and execute it in database, which is costly. Consequently, we propose a MTJNT generation algorithm which significantly reduces the complex database queries required by traditional methods and further improves the query efficiency. The CN set generated by our approach contains all possible CNs as discussed above. Therefore, the MTJNTs generated by the algorithm contain all possible query results. In the following, we will discuss how to evaluate the CCN to generate the MTJNTs.

For a given CCN, there exists a corresponding duplicate layer tree. As shown in Figure 3, the part with the solid line represents the CCN while the part with the dotted line represents the corresponding tree with six duplicate layers

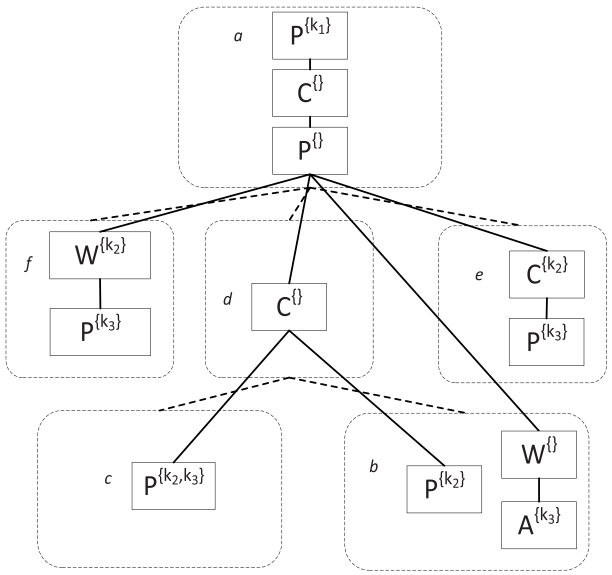


FIGURE 3. Combined candidate network.

$\{D_a, D_b, D_c, D_d, D_e, D_f\}$. The role of duplicate layer is to provide our approach with the capacity of identifying each CN in the CCN. For a duplicate layer tree, a path from the leaf to the root can form a CN. The reason why we consider the CCN in two different ways is that the view considering tuple sets as nodes is used to generate the joining network of tuples while the other view is used to restrict the generation of joining network of tuples within a CN, namely the generation of MTJNTs.

The way generating the MTJNTs in CCN evaluation is similar to DFS (Depth-First Search) in CCN, which is a recursion. The algorithm will begin with the root of a CCN and traverse all nodes in the CCN. When it arrives at a duplicate layer leaf, the path from the root to the leaf will form a MTJNT. The most fine-grained object operated in the algorithm is actually a tuple rather than a node and the links between tuples are also considered in addition to the links between tuple sets. The tuple ID and the join relationships between tuples are loaded during the preload process. Considering the path $(D_a\ D_d\ D_c)$ in the CCN in Figure 3, the algorithm starts at a tuple at_1 in $P^{\{k_1\}}$ of D_a, then traverses the adjacent tuple at_2 of at_1 in $C^{\{\}}$, at_3 of at_2 in $P^{\{\}}$, dt_1 of at_3 in $C^{\{\}}$ of D_d, ct_1 of dt_1 in $P^{\{k_2,k_3\}}$ of D_c, where we assume that each tuple has one adjacent tuple at least in the following node. Finally, the path which is composed of the tuples at_1, at_2, at_3, dt_1, ct_1 forms a real MTJNT for the CN $(D_a\ D_d\ D_c)$. It seems to be a simple task and DFS can be quite sufficient for this work. However, if DFS is used, the adjacent tuples in $W^{\{\}}$ and $A^{\{k_3\}}$ in D_b might be traversed after at_3 is traversed in D_a and this will lead to the generation of incomplete path which is just a part of a MTJNT for the CN $(D_a\ D_d\ D_b)$. Consequently, the duplicate layer tree is introduced to restrict the traversal of path within each of CN in a given CCN.

Given a tuple in a parent node, like at_1, its adjacent tuple of child node, like at_2, is also referred to as its eligible tuple.

All the eligible tuples in the duplicate layer leaf will be enumerated when the algorithm arrives at a duplicate layer leaf. Suppose that there exist three eligible tuples in D_c given dt_1. The algorithm will enumerate the three tuples to generate three MTJNTs. After the enumeration, it returns to the D_d. If there still exists a child of D_d which has some eligible tuples not enumerated, like D_b, the algorithm will go into D_b and enumerate all the eligible tuples to generate MTJNTs. There are still some problems which are needed to be solved in the enumeration of this situation; that is, there are multiple separate nodes or sub-trees in a duplicate layer, like D_b, rather than only one node. The algorithm will conduct the recursive process again over each separate part in the duplicate layer to enumerate all the combination. Suppose that there are three tuples dt_1, dt_2 and dt_3 in $C^{\{\}}$ in D_d. After the enumeration of D_b, the recursion for the tuple dt_1 is finished, and then the algorithm begins with dt_2 by the same procedure as dt_1, i.e., traversing the eligible tuples of dt_2 in the following D_b. The process keeps iterating until all tuples in $P^{\{k_1\}}$ of D_a complete their recursion; that is, the algorithm terminates at that point.

It can be seen that the proposed tree CCN efficiently shares the overlapping part between different CNs. The overlapping part $P^{\{k_1\}}/C^{\{\}}/P^{\{\}}/C^{\{\}}$ between the CN (D_a D_d D_c) and the CN (D_a D_d D_b) is handled for only one time during the generation of their MTJNTs. Moreover, our algorithm parses the complex join relationship between relations in the front end to generate MTJNTs while traditional algorithms typically evaluate the join relationship in CNs into a large number of complex SQL queries and execute them in back-end databases. The SQL queries with lots of complex join relationship between tables are really time-consuming. Therefore, our algorithm improves the efficiency from the two points above.

The MTJNT generation process is shown in Algorithm 2. The function of MTJNTGeneration is used to generate all MTJNTs (Lines 3-6). The function of MTJNTExtension is for the extension of TempMTJNT and its parameter *Node* is the tuple set where the extended tuple is located. When *Node* is the first node of the CCN (Line 9), a tuple of *Node* is added to TempMTJNT (Line 11) and the extension continues (Lines 10-14). If *Node* is not the first node, the algorithm looks for a tuple which belongs to the parent node of *Node* and has been added to TempMTJNT (Line 16), and then adds the adjacent tuple of this tuple in *Node* to TempMTJNT (Line 18) and continues the extension (Line 19). The function of AddNextTuple is for the extension of the next tuple. If there is a node in the CCN that is on the same duplicate layer as *Node* and has not been visited (Lines 27-29), the algorithm extends the tuple in that node (Line 28) or its child. When the duplicate layer of the *Node* has the child (Line 30), the algorithm extends the tuples of the nodes in the child (Lines 31-35). When the duplicate layer which the *Node* is in has been extended and there is no child, TempMTJNT is a final MTJNT (Line 37).

Algorithm 2 MTJNT Generation

```
Input:  A set of CCN CCNs
Output:  A set of MTJNT MTJNTs
1:  function MTJNTGeneration // stores intermediate MTJNT
2:      Initialize queue TempMTJNT
3:      for all CCN∈CCNs do
4:          Node←HeadNode (CCN)
5:          MTJNTExtension(TempMTJNT,Node,CCN)
6:      end for
7:  end function
8:  function MTJNTExtension(TempMTJNT, Node, CCN)
9:      if Node=HeadNodeOf(CCN) then
10:         for all tuple∈TupleSet(Node) do
11:             TempMTJNT.AddTuple(tuple)
12:             AddNextTuple(TempMTJNT, Node, CCN)
13:             TempMTJNT.Remove(tuple)
14:         end for
15:     else
16:         ft ←ParentTuple(TempMTJNT, Node)
17:         for all tuple∈AdjacentTupleOf(ft)∧tuple∈Node do
18:             TempMTJNT.AddTuple(tuple)
19:             AddNextTuple(TempMTJNT, Node, CCN)
20:             if AllNodeAdd(TempMTJNT, Node) then
21:                 TempMTJNT.Remove(tuple)
22:             end if
23:         end for
24:     end if
25: end function
26: function AddNextTuple(TempMTJNT, Node, CCN)
27:     for all n∈CCN∧SameTag(n, Node) do
28:         MTJNTExtension (TempMTJNT, n, CCN)
29:     end for
30:     if ExistSonTag(Node) then
31:         for all g∈SonTag (Node) do
32:             for all h∈HeadNodeOfBranch(g, CCN) do
33:                 MTJNTExtension (TempMTJNT, h, CCN)
34:             end for
35:         end for
36:     end if
37:     MTJNTs.Add(TempMTJNT)
38: end function
```

A set M of all possible MTJNTs are generated after the CCN evaluation. We show how to select the top-k MTJNTs in M. It is not the point of this paper how to measure the quality of the final results returned to users. Thus, we choose a classical evaluation strategy [12], [29] for the top-k selection. Their strategy prefers MTJNTs with less tuples, which means the MTJNT with the least tuples will be the top-1 answer. Our approach employs their strategy to score each MTJNT

TABLE 1. DBLP data set.

Relations	Size(MB)	Number of Tuples
AUTHOR	49	1189006
PAPER	392	1836075
WRITE	232	5516611
CITATION	5	79002

in M, and then returns top-k MTJNTs. Actually, any evaluation strategy can be easily exploited by our approach for the selection of the results returned according to different requirements or data circumstance.

V. EXPERIMENTAL ANALYSES

Our algorithms are implemented using Java language and experiments are carried on a compatible PC, with Intel I5 processor (3.2GHz) and 8 GB of RAM. The software platform is configured as Windows 7 and Oracle 11g. The experimental data used are from real datasets, DBLP (DataBase systems and Logic Programming) and IMDB (Internet Movie Database) respectively. DBLP is the on-line reference for bibliographic information on major computer science publications, while IMDB is the world's most popular and authoritative source for films, television programs and home videos, designed to help fans explore the world of movies. They provide two public datasets for research community, which are updated continuously and wildly used by researchers. The specific features of the two datasets are shown in Table 1 and Table 2.

TABLE 2. IMDB data set.

Relations	Size(MB)	Number of Tuples
NAME_BASIC	523	9097025
TITLE_AKAS	175	1701200
TITLE_BASICS	451	15164499
CREW	169	9484034
EPISODE	92	3834666
PRINCIPALS	1352	53599850
RATINGS	15	912960

In our experiments, we compare the query efficiency of our approach (CCNE for short) to the traditional approach DISCOVER-II along different dimensions and test the effect of changing various parameters of CCNE on its efficiency. CNRank is a representative CN ranking technique which reduces the number of CNs to be evaluated by ignoring the less important CNs for the improvement of query efficiency (there are several similar methods within this kind, such as SPARK, etc.). Thus, we make CNRank to be a complete end-to-end system by adding CN generation and evaluation process (as said in [21]), and compare our approach to CNRank on query efficiency. We also conduct experiments on two different datasets DBLP and IMDB and obtain similar results, which means our approach is stable and valid.

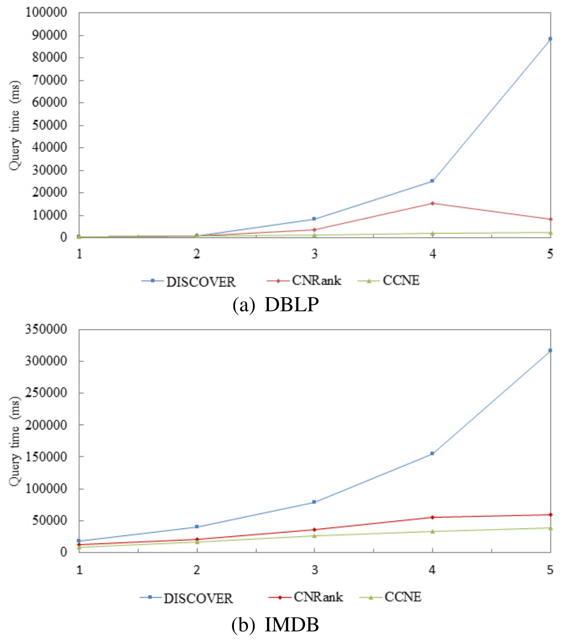


FIGURE 4. Comparison with traditional methods.

A. EFFECT OF NUMBER OF KEYWORDS

The experiment compares the efficiency of CCNE with the two methods DISCOVER-II and CNRank when varying the number of keywords. The maximum number of nodes in CN (namely the CN's threshold) and the k value of top-k are set to 5 and 10 respectively. The experimental results are shown in Figure 4. It can be seen that the efficiency of CCNE is higher than the other two methods. The query time of CCNE slowly increases with varying the number of keywords from 1 to 5. In CCNE, the process of creating tuple sets accounts for the main part of the query time. As the number of keywords increases, more tuple sets need to be created, which results in increased query time. For DISCOVER-II and CNRank, their evaluation process of CNs needs more time in addition to the time cost for creating the tuple sets, especially for DISCOVER-II. Therefore, CCNE has a better performance in query time than DISCOVER-II and CNRank.

B. EFFECT OF THE NUMBER OF RESULTS RETURNED

The influence of the number of results returned (namely, top-k) is tested for the three methods in this experiment. The number of keywords and CN's thresholds are fixed at 3 and 5, respectively. As shown in Figure 5, the query time of CCNE has almost no change with the increase of k comparing with DISCOVER-II and CNRank. The reason is that CCNE finds out the top-k results from the universe of MTJNTs; that is, CCNE does not need to generate more MTJNTs to return the top-k results as the value k increases. DISCOVER-II and CNRank implement the CN evaluation according to the level of relevance [18], [20], and stop the execution after generating top-k MTJNTs; that is, DISCOVER-II and CNRank need

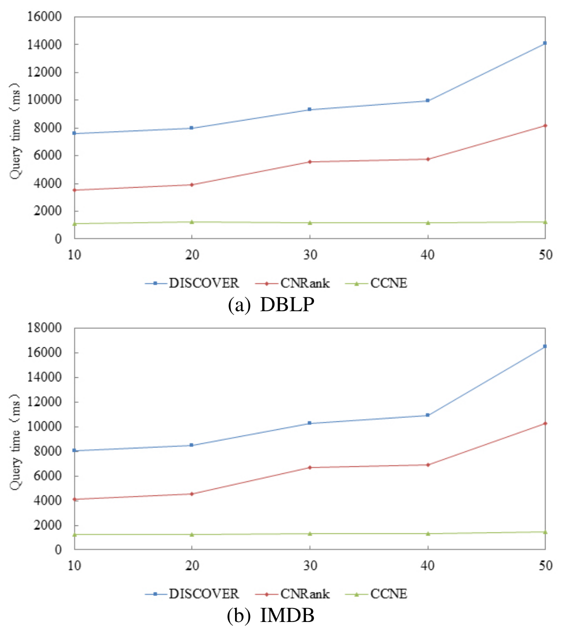


FIGURE 5. Query time varying top-*k*.

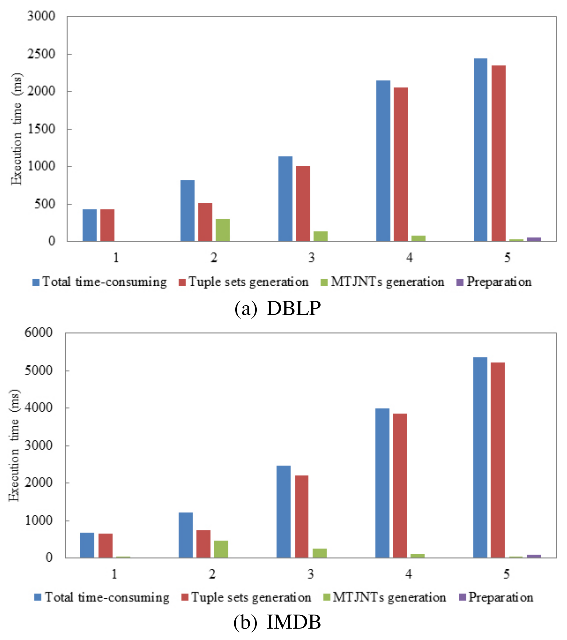


FIGURE 6. Time cost of different phases in CCNE.

to generate more MTJNTs with the increase of the value k, which decreases their efficiency.

C. TIME COST OF DIFFERENT PHASES IN CCNE

The time cost of different phases of CCNE is analyzed in this experiment. There are three main phases: the generation of tuple sets, the preparation for MTJNTs, and the generation of MTJNTs, as shown in Figure 6. It can be seen that the tuple set generation accounts for the most time of CCNE relative to the other two phases. A large number of queries are posed to the database and the execution of these queries takes up much time of CCNE. The other two phases have less access to the database, and the size of data manipulated is much smaller than the process of creating tuple sets. Therefore, the columns of the generation of tuple sets in Figure 6 are much higher than the other two phases. Time cost of generating MTJNTs tends to increase at the beginning and then decrease as the number of keywords increases. This is because for a given CN's threshold the number of CNs increases first and then decreases as the number of keywords increases. For example, if there is only one keyword, then CN contains only one key tuple set. Thus, the number of CNs is only the number of key tuple sets. However, the number of CNs containing all the keywords will significantly decrease when there are a large number of keywords.

D. COST ANALYSIS OF INITIALIZATION

The consumption of space and time in the initialization of CCNE is shown in Figure 7 where the abscissa represents DBLP and IMDB in different years, the histogram with the legend DBLP indicates the size of the DBLP dataset used in our experiments, the legend Initialization indicates the size of the memory occupied by the initialization of CCNE, and the

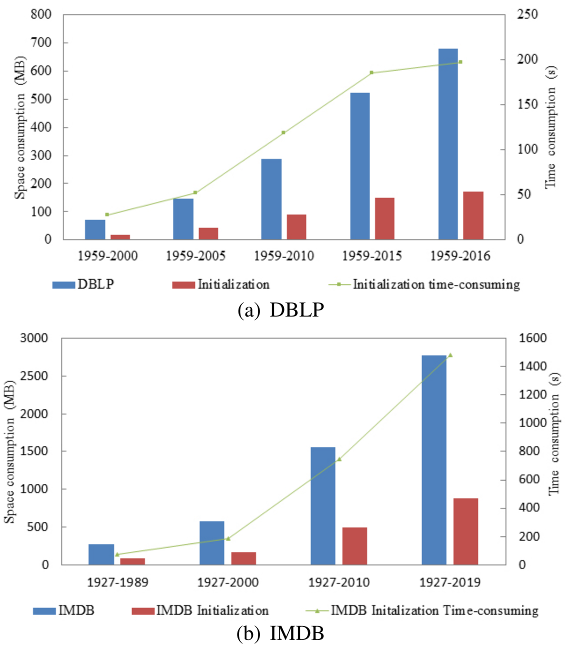


FIGURE 7. Cost analysis of initialization.

performance curve indicates the time cost of the initialization. As shown in Figure 7, the amount of data loaded during the initialization is much smaller than the actual size of the dataset, and has a small increase rate as the dataset increases. Meanwhile, CCNE consumes memory much less than traditional R-KWS based on Data Graphs, which considers a tuple as a node in their graphs. The time cost of the initialization in CCNE is about 200s when the whole DBLP dataset is used for the test, which is not an acceptable time cost for query.

However, the initialization is established for only one time at the beginning of execution during the whole life cycle of our system. The same experimental results were also presented on the IMDB dataset. Our approach is able to achieve high efficiency based on the initialization.

VI. CONCLUSION

We propose the approach based on CCN evaluation to solve the problem of low query efficiency of R-KWS based on Schema Graphs. The key of our approach is the new proposed tree CCN which is the base of efficiency improvement. CCN can efficiently share the overlap between CNs by compressing them into a single tree. Meanwhile, CCN possesses an important feature that each combined CN within the CCN is still identifiable. We propose an algorithm based on this feature to generate the final query results, which significantly improve the efficiency of R-KWS based on Schema Graphs. We compare the time cost of the proposed approach with two traditional approaches DISCOVER-II and CNRank in the DBLP dataset and IMDB dataset. Experimental results show that our approach has a better performance than the other two approaches, and the change of relative parameters has less negative effect on the efficiency of our approach than the other two approaches, such as the CN's threshold, the number of query result returned (top-k), the number of keywords, etc.

REFERENCES

[1] C. Lu, L. Chengfei, Y. Xiaochun, and W. Bin, "Efficient batch processing for multiple keyword queries on graph data," in *Proc. CIKM*, Indianapolis, IN, USA, 2016, pp. 1261–1270.

[2] L. Qin, J. X. Yu, and L. Chang, "Ten thousand SQLs: Parallel keyword queries computing," *Proc. VLDB Endowment*, vol. 3, nos. 1–2, pp. 58–69, Sep. 2010.

[3] L. Ronghua, Q. Lu, Y. Jeffrey, and M. Rui, "Efficient and progressive group Steiner tree search," in *Proc. SIGMOD*, San Francisco, CA, USA, 2016, pp. 91–106.

[4] H. He, H. Wang, and J. Yang, "BLINKS: Ranked keyword searches on graphs," in *Proc. SIGMOD*, Beijing, China, 2007, pp. 305–316.

[5] K. Golenberg, B. Kimelfeld, and Y. Sagiv, "Keyword proximity search in complex data graphs," in *Proc. SIGMOD*, Vancouver, BC, Canada, 2008, pp. 927–940.

[6] Z. Lin, Y. Li, and Y. Lai, "Improve the effectiveness of keyword search over relational database by node-temperature-based ant colony optimization," in *Proc. FSKD*, Zhangjiajie, China, 2015, pp. 1209–1214.

[7] B. B. Dalvi, M. Kshirsagar, and S. Sudarshan, "Keyword search on external memory data graphs," *Proc. VLDB Endowment*, vol. 1, no. 1, pp. 1189–1204, Aug. 2008.

[8] J. Coffman and A. C. Weaver, "Structured data retrieval using cover density ranking," in *Proc. KSSD Workshop*, New York, NY, USA, 2010, pp. 1–6.

[9] B. Ding, J. X. Yu, and S. Wang, "Finding top-K min-cost connected trees in databases," in *Proc. ICDE*, Ankara, Turkey, 2007, pp. 836–845.

[10] A. Baid, I. Rae, J. Li, A. Doan, and J. Naughton, "Toward scalable keyword search over relational data," *Proc. VLDB Endowment*, vol. 3, nos. 1–2, pp. 140–149, Sep. 2010.

[11] A. Markowetz, Y. Yang, and D. Papadias, "Keyword search on relational data streams," in *Proc. SIGMOD*, Beijing, China, 2007, pp. 605–616.

[12] V. Hristidis and Y. Papakonstantinou, "DISCOVER: Keyword search in relational databases," in *Proc. VLDB*, Hong Kong, 2002, pp. 670–681.

[13] J. X. Yu, L. Qin, and L. Chang, "Keyword search in relational databases: A survey schema-based keyword search on relational databases," *IEEE Data Eng. Bull.*, vol. 33, no. 1, pp. 67–78, Feb. 2010.

[14] L. Qin, J. X. Yu, and L. Chang, "Keyword search in databases: The power of RDBMS," in *Proc. SIGMOD*, Providence, RI, USA, 2009, pp. 681–694.

[15] Y. Tao and J. X. Yu, "Finding frequent co-occurring terms in relational keyword search," in *Proc. EDBT*, Saint Petersburg, Russia, 2009, pp. 839–850.

[16] B. Zhou and J. Pei, "Aggregate keyword search on large relational databases," *Knowl. Inf. Syst.*, vol. 30, no. 2, pp. 283–318, Feb. 2012.

[17] J. Park and S.-G. Lee, "A graph-theoretic approach to optimize keyword queries in relational databases," *Knowl. Inf. Syst.*, vol. 41, no. 3, pp. 843–870, Dec. 2014.

[18] N. Kuchmann-Beauger, F. Brauer, and M. A. Aufaure, "QUASL: A framework for question answering and its application to business intelligence," in *Proc. RCIS*, Paris, France, 2013, pp. 1–12.

[19] V. Hristidis, Y. Papakonstantinou, and L. Gravano, "Efficient IR-style keyword search over relational databases," in *Proc. VLDB*, Berlin, Germany, 2003, pp. 850–861.

[20] Y. Luo, X. Lin, and W. Wang, "Spark:top-k keyword query in relational databases," in *Proc. SIGMOD*, Beijing, China, 2007, pp. 115–126.

[21] P. D. Oliveira, A. D. Silva, and E. D. Moura, "Ranking candidate networks of relations to improve keyword search over relational databases," in *Proc. ICDE*, Seoul, South Korea, 2015, pp. 399–410.

[22] P. Zhao-hui, Z. Jun, and W. Shan, "S-CBR: Presenting results of keyword search over databases based on database schema," *Software*, vol. 2, no. 10, pp. 323–337, Feb. 2008.

[23] J. Coffman and A. C. Weaver, "A framework for evaluating database keyword search strategies," in *Proc. CIKM*, Toronto, ON, Canada, 2010, pp. 729–738.

[24] X. Meng, L. Cao, and X. Zhang, "Top-*k* coupled keyword recommendation for relational keyword queries," *Knowl. Inf. Syst.*, vol. 50, no. 3, pp. 883–916, Mar. 2017.

[25] L. Qin, J. X. Yu, and L. Chang, "Querying communities in relational databases," in *Proc. ICDE*, Shanghai, China, 2009, pp. 724–735.

[26] F. Liu, C. Yu, and A. Chowdhury, "Effective keyword search in relational databases," in *Proc. SIGMOD*, Chicago, IL USA, 2006, pp. 563–574.

[27] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar, "Bidirectional expansion for keyword search on graph databases," in *Proc. VLDB*, Trondheim, Norway, 2005, pp. 505–516.

[28] G. J. Fakas, "A novel keyword search paradigm in relational databases: Object summaries," *Data Knowl. Eng.*, vol. 70, no. 2, pp. 208–229, Feb. 2011.

[29] S. Agrawal, S. Chaudhuri, and G. Das, "DBXplorer: A system for keyword-based search over relational databases," in *Proc. ICDE*, San Jose, CA, USA, 2002, pp. 5–15.

[30] G. Bhalotia, C. Nakhe, A. Hulgeri, S. Chakrabarti, and S. Sudarshan, "Keyword searching and browsing in databases using banks," in *Proc. ICDE*, San Jose, CA, USA, 2002, pp. 431–440.

[31] X. Yang, C. M. Procopiuc, and D. Srivastava, "Summarizing relational databases," *Proc. VLDB Endowment*, vol. 2, no. 1, pp. 634–645, Aug. 2009.

[32] S. Bergamaschi, E. Domnori, F. Guerra, R. T. Lado, and Y. Velegrakis, "Keyword search over relational databases: A metadata approach," in *Proc. SIGMOD*, Athens, Greece, 2011, pp. 565–576.

[33] X. Yang, C. M. Procopiuc, and D. Srivastava, "Summary graphs for relational database schemas," *Proc. VLDB Endowment*, vol. 4, no. 11, pp. 899–910, 2011.

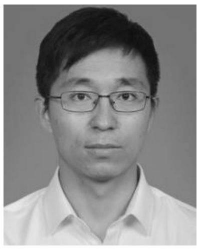
[34] M. Kargar and A. An, "Keyword search in graphs: Finding r-cliques," *Proc. VLDB Endowment*, vol. 4, no. 10, pp. 681–692, Jul. 2011.

[35] J. Feng, G. Li, and J. Wang, "Finding top-k answers in keyword search over relational databases using tuple units," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 12, pp. 1781–1794, Dec. 2011.

[36] M. Kargar, A. An, N. Cercone, P. Godfrey, J. Szlichta, and X. Yu, "Meaningful keyword search in relational databases with large and complex schema," in *Proc. ICDE*, Seoul, South Korea, 2015, pp. 411–422.

[37] Y. Xu, J. Guan, F. Li, and S. Zhou, "Scalable continual top-k keyword search in relational databases," *Data Knowl. Eng.*, vol. 86, pp. 206–223, Jul. 2013.

[38] Z. Zeng, M.-L. Lee, and T. W. Ling, "PowerQ: An interactive keyword search engine for aggregate queries on relational databases," in *Proc. EDBT*, Bordeaux, France, 2016, pp. 596–599.



GUOHUI DING received the B.S. and M.S. degrees in CS from Shenyang Aerospace University, in 2005 and 2008, respectively, and the Ph.D. degrees in CS from Northeastern University, in 2012. He is currently an Associate Professor with the School of Computer Science, Shenyang Aerospace University (SAU). His research interests include data integration, data exchange, and time series data.

HAOHAN SUN was born in 1995. He received the B.E. degree in computer science and technology from Liaoning Technical University, in 2018. He is currently pursuing the master's degree with Shenyang Aerospace University. His research interests include data mining and big data.

JIAJIA LI was born in 1987. She received the B.S. degree in software from Northeast Normal University, in 2008, and the M.S. and Ph.D. degrees in computer science from Northeastern University, in 2010 and 2014, respectively. She is currently an Associate Professor with the School of Computer Science, Shenyang Aerospace University. Her research interests include mobile data management, spatial-temporal database, machine learning, and uncertain data management.

CHENYANG LI was born in 1997. She received the B.E. degree in computer science and technology from the Lanzhou University of Finance and Economics, in 2019. She is currently pursuing the master's degree in computer science and technology with Shenyang Aerospace University. Her research interests include data mining and time series data analysis.

RU WEI was born in 1997. She received the B.S. degree in Internet of Things from Cangzhou Normal University, in 2019. She is currently pursuing the master's degree with the School of Computer Science, Shenyang Aerospace University. Her research interests include concept linking, entity recognition, and machine learning.

YUNFENG FEI received the B.E. degree in computer application technology from the Shenyang Institute of Engineering, in 2015, and the M.A. degree in computer science and technology from Shenyang Aerospace University, in 2018. He is currently with Beijing Huayu Information Technology Company, Ltd. His research interests include keyword retrieval, big data, and data management.