

EFFICIENT SCHEMA BASED KEYWORD SEARCH IN RELATIONAL DATABASES

Myint Myint Thein¹ and Mie Mie Su Thwin²

¹University of Computer Studies, Mandalay, Myanmar
mmyintt@gmail.com

²University of Computer Studies, Mandalay, Myanmar
drmiemiesuthwin@mmcert.org.mm

ABSTRACT

Keyword search in relational databases allows user to search information without knowing database schema and using structural query language (SQL). In this paper, we address the problem of generating and evaluating candidate networks. In candidate network generation, the overhead is caused by raising the number of joining tuples for the size of minimal candidate network. To reduce overhead, we propose candidate network generation algorithms to generate a minimum number of joining tuples according to the maximum number of tuple set. We first generate a set of joining tuples, candidate networks (CNs). It is difficult to obtain an optimal query processing plan during generating a number of joins. We also develop a dynamic CN evaluation algorithm (D_CNEval) to generate connected tuple trees (CTTs) by reducing the size of intermediate joining results. The performance evaluation of the proposed algorithms is conducted on IMDB and DBLP datasets and also compared with existing algorithms.

KEYWORDS

Candidate Network, Connected Tuple Tree, Joining Tuples, Keyword Query, Keyword Search, Relational Database

1. INTRODUCTION

The most critical and valuable amount of data such as business data has been stored in relational databases. Relational database management system (RDBMS) is a DBMS in which data is saved in tables and the relationships among the data are saved in tables. The data can be reassembled and accessed in many different ways without change the table forms. Most commercial relational database management system uses SQL to access the database. With more and more data being stored in relational database, it has become crucial for users to be able to search and browse the information stored in them. Keyword search in relational databases enables ordinary users, who do not understand the database schema and SQL, to find the connected tuple sets among the tuples stored in relations, with a given set of keywords. The existing methods of keyword search in relational databases can be broadly classified into two categories that are schema based method and graph based method.

In schema based keyword search in relational database, it has a common method that is generating candidate network in schema graph transformed from relations. Data is stored in the form of columns, tables and primary key to foreign key relationships in relational databases. According to develop the schema graph, we illustrate two schema graphs as examples. Figure 1 shows the schema graph of publication database from DBLP dataset. It consists of six relation

schemas that are Person, InProceeding, RelationPersonInProceeding, Proceeding, Publisher and Series. Each relation has a primary key from except RelationPersonInProceeding relation. InProceeding relation has one foreign key that refers to the primary key defined on Proceeding relation. Proceeding relation has two foreign key that refers to the primary key defined on both Publisher and Series relations. The movies database schema graph of IMDB dataset shows in Figure 2. It consists of six relation schemas: Movies, Directors, Movies-Directors, Movies-Genres, Actors and Roles. Each relation has a primary key from except Movies-Directors relation. Roles relation has one foreign key that refers to the primary key defined on Actors relation.

The logical unit of answers needed by users is not limited to an individual column value or even an individual tuple for a given keyword query. It may be multiple tuples joined together. Given keyword search in relational databases, generating minimum joining tuples sets of relations that contained keyword is called candidate network, such as SQL. A candidate network must satisfy the two conditions, total and minimal. Because it is meaningless if two tuples in a candidate network are too far away from each other, the maximum numbers of tuples allowed in a candidate network are needed to specify [18].

Suppose user wants to get the papers written by “Jinlin Chen” from DBLP database. The system generates the relevant CNs, such as Person \bowtie Relation-Person-InProceeding \bowtie InProceeding, with multiple tuples from different relations joined by foreign keys. Generating all valid candidate networks that are called connected tuple trees by joining tuples from multiple relations. DISCOVER [4], S-KWS [10], Liu et al.[7], and SPARK2 [9] are systems that support keyword search on relational database. They generated tuple trees as answer for the CN generation. The first two systems need to reduce the cost of generating minimal CNs, while the last two systems cannot solve the growing number of CNs for small CN size. Existing candidate network generation, CN’s size is unbounded and the number of CNs grows very large for small CN’s size. This fact brings large overhead for CNs generation. Due to large number of generated CNs to be evaluated, multi-query optimization problem is caused on the CNs evaluation.

In this paper, we focus on generating the valid CNs on the data bond and producing the minimal connected tuple trees. We develop algorithms in order to generate all minimum connected trees of tuples in the database with no more than the maximum number of tuple set. We proposed candidate network generation algorithms to find relevant answers on-the-fly by joining tuples in the database. We also proposed the dynamic CN evaluation algorithm (D_CNEval) by evaluating the number of generated CNs. We conduct the experimental results on DBLP and IMDB databases and present the analysis of worst case time for these algorithms.

The rest of the paper is organized as follows: Section 2 discusses the related work. Section 3 presents the basic concept of keyword query and CN. The overview of proposed system is specified in Section 4. Section 5 presents the Candidate Network Generation. Section 6 illustrates the query execution and Section 7 shows the experimental results. Section 8 concludes this paper.

2. RELATED WORK

The main goal of a keyword search system is to find a set of closely inter-connected tuples that collectively match the keywords. One type of methods is based on modeling data as a graph, and the results as subtrees or sub-graphs. Another type of methods is based on relational databases where structured data are stored.

Several researchers have been done on early keyword search systems for relational databases [2, 7, 8, 15]. Yu et al. [18] surveyed the developments on finding structural information among tuples

in an RDB using an l-keyword query. They discussed the keyword search systems by comparing between schema-based keyword search and graph-based keyword search in RDB. The former evaluated the sets of answers by defining all minimal total joining networks of tuples between CNs and the latter showed how to answer keyword queries using graph algorithms focused on weighted directed graph. DBXplorer [1] used undirected graph to construct each SQL statements according to each tuple tree. This system accessed to symbol table to get tuples' information, and then calculated tuple tree according to schema graph.

DISCOVER [4] proposed the CN generation algorithm based on a breadth-first traversal in the search space. This proposed algorithm expanded the partial CNs generated to larger partial CNs until all CNs are generated. As the number of partial CNs can be exponentially large, arbitrarily expanding will make the algorithm extremely inefficient. The problem with this algorithm is that the cost of generating the set of CNs is high and kept in memory for further extension.

S-KWS [10] developed an algorithm that reduces the number of partial results generated by expanding from part of the nodes in a partial tree and avoid isomorphism testing by assigning a proper expansion order. Although it reduced the generated partial results, it existed overhead for generating minimal CNs to the query.

Liu et al. [7] described the answer graph generation algorithm to generate tuple trees. Although they produced duplication-free CNs by assigning the different alias, they had not considered the efficiency of answer generation. SPARK2 [9] developed the duplication-free algorithm by canonical form but it did not solve the number of CNs grows very large for small CN size.

3. PRELIMINARIES

3.1. Data Model

A relational database can be viewed as a graph which represents a relational model such as schema graph $G_s (V, E)$ [4, 8, 16, 18]. A relation database is a collection of relations. Each relation in the database corresponds to a vertex in G_s , denoted as the set of relation schemas $\{R_1, R_2, \dots\}$. Edges represent the foreign key to primary key relationships between pairs of relation schemas, R_i and R_j , denoted $R_i \rightarrow R_j$. A relation on relation schema R_j is an instance of the relation schema, such as a set of tuples, conforming to the relation schema. The graph can be as a directed or undirected graph. It can be captured every granularity level of the schema elements.

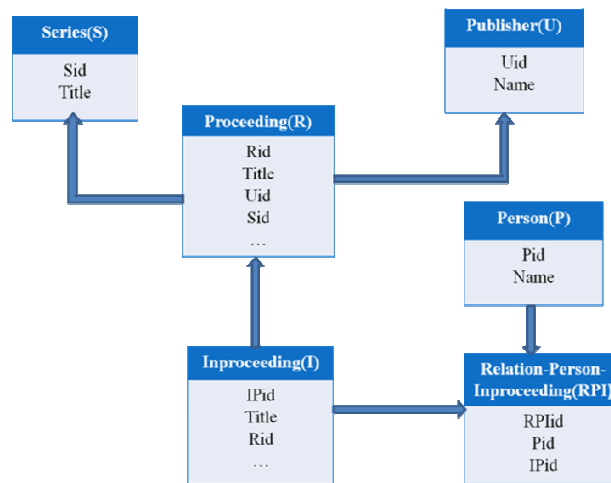


Figure 1. Publication Database Schema Graph

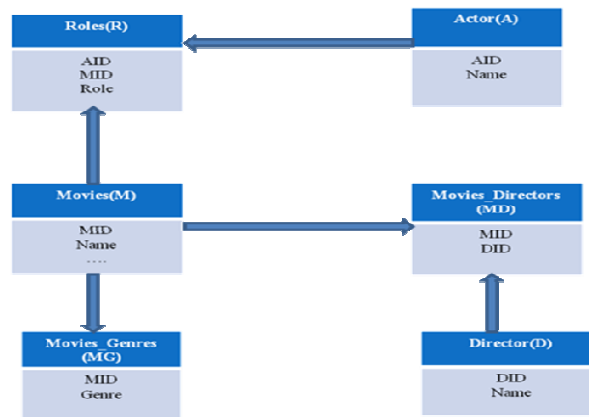


Figure 2. Movies Database Schema Graph

We use directed schema graphs that show in Figure 1 and Figure 2 as the schema graph of publication database and movies database schema graph. For simplicity, we assume all primary key and foreign key attributes are made of same attribute with attribute of related relation. There are no self loops and at most one primary-foreign key relationship between any two relations.

3.2. Connected Tuple Tree

A keyword query (Q) consists of a list of keywords $\{k_1, k_2, \dots, k_q\}$, and searches interconnected tuples that contain the given keywords. For a given query Q , a result is the set of all possible joining networks of tuples. A joining network of tuple is a connected tuple tree. Each node t_i is a tuple in the database, and each pair of adjacent tuples in CTT is connected via a foreign key to primary key relationship. Suppose (R_i, R_j) is an edge in the schema graph. Let $t_i \in R_i$, $t_j \in R_j$, and $(t_i \text{ join } t_j) \in (R_i \text{ join } R_j)$. Then (t_i, t_j) is an edge in the connected tuple tree. The size of a CTT is the number of tuples involved. Note that a single tuple is the connected tuple tree with size 1. The size of CTT can have arbitrarily large size, when there exists a many to many relationship in the schema graph. Therefore, the size of connected tuple tree is needed to only data bound.

3.3. Candidate Network

Each connected tuple tree is the sets consisting of relational names that produced by a relational algebra expression, if each tuple in one relation contains a term of the keywords. For a given keyword query Q , the query tuple set R^N is a set of all tuples which belong to relation R that contain at least one keyword of the query Q . We denote R^F the free tuple set which is the set of all tuples in relation R and we use R^Q to denote a tuple set, which can be either a non-free tuple set or a free tuple set. A candidate network is a tree of tuple sets R^N or R^F with the restriction that every node must be a query tuple set. Every edge (R_i^Q, R_j^Q) in a CN corresponds to an edge (R_i, R_j) in the schema graph G_s . The size of a CN is the number of its tuple sets.

In the framework of RDBMS, a keyword query is processed in the two main steps that are candidate network generation and candidate network evaluation. In candidate network generation step, it generates a set of CNs over schema graph G_s . The set of CNs shall be sound or complete and duplicate-free upon the maximal size. In candidate network evaluation step, it evaluates the generated CNs by reducing the size of intermediate joining results. We present how to generate minimal number of CNs and how to evaluate the generated CNs in Section 5 and Section 6.

4. PROPOSED SYSTEM OVERVIEW

In this section, we demonstrate the overview of keyword search on relational databases that is shown in Figure 3. The system supports free-style keyword search by generating connected tuple trees with user typed keywords. In this system, the final results are eliminated by processing the four phases that are following. The query cleaning phase filters out as potential index terms as removed stopwords query. This process reduces the size of the indexing structure considerably. The indexing unit in a relational document can be a field, attribute, tuple, table, or any combination of these. After the system has built the inverted index files as posting table for each relation, the indexer produces the matched tuple sets by using the filtered input query. The system generates a set of CNs by traversing on the schema graph in order to the tuple sets. Query executing phase executes queries for each CNs and generate the connected tuple trees as executed queries. Finally, the system returns the minimal connected tuple trees to the user for a given query.

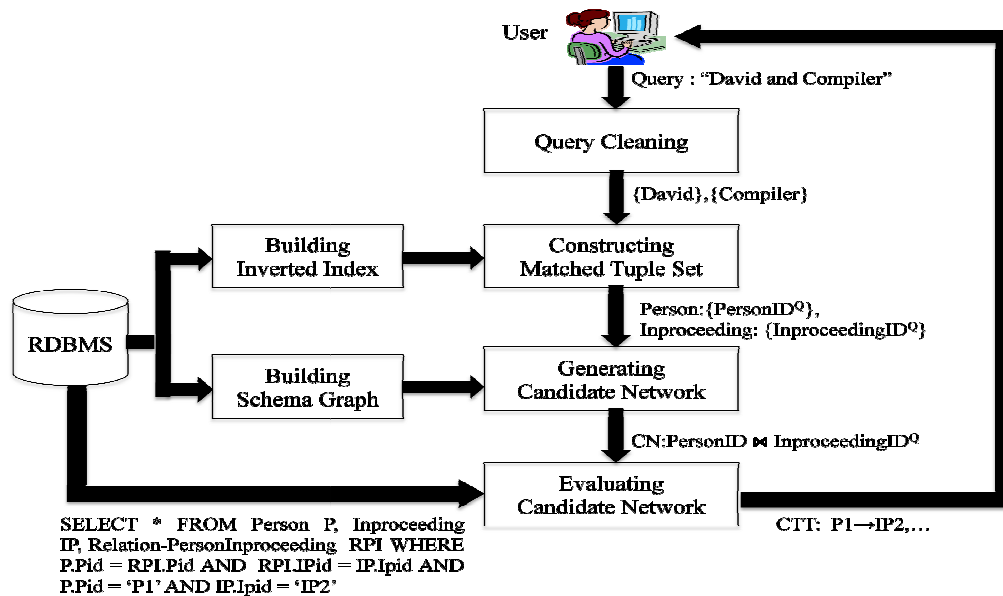


Figure 3. Proposed System Overview

5. CANDIDATE NETWORK GENERATION

In schema-based keyword search in relational database, the generating all candidate networks for keyword query Q satisfy the two properties, such as complete and duplication-free, which are listed below.

- Property 1. The set contains all CNs with no more than MAXN (completeness).
- Property 2. Every two CNs are not isomorphic to each other (duplication-free).

In this paper, we propose CN generation algorithms for schema-based keyword search in relational database. Existing keyword search systems, such as DISCOVER and S-KWS, generate CNs temporarily through a breadth-first traversal of schema graph for any user query. The result of CNs set is to avoid the generation of redundant joining networks of tuple sets. As the number of query keywords or maximum size of CN increases, or the database schema becomes

complicated, it will take much more time to generate CNs for a query Q. There can be two ways to reduce the time for the generation of CNs. One is to develop a more efficient CN generation algorithm and the other is to develop preprocessing techniques to generate CNs in advance. In this section, we develop the efficient CN generation algorithms to address the above problem.

5.1. Heuristic_CNGen Algorithm

In this section, we describe a new CN generation algorithm (Heuristic_CNGen) to generate valid CNs [13]. Given a keyword query Q, the system first receives all the query tuple set R^Q for all relations R as input. We use R^{NorQ} to define a tuple set, if CN is a result then each node belongs to the non-free query tuple set R^N and the free query tuple set R^F of each relation R for a given query. Note that the free query tuple set in CN cannot contain the query keyword, but they support to the non-free query tuple set as primary-foreign keys relationship.

```

Heuristic_CNGen(MAXN, f_limit, f_new)
Input: schema graph SG, query Q
Output: a set of candidate networks CN
1. E: queue generated all non-free tuple sets  $\{R_1^N, R_2^N, \dots, R_n^N\}$  and all
free tuple sets  $\{R_1^F, R_2^F, \dots, R_n^F\}$  for Q.
2. While E is not empty {
3.   Pop head T from E
4.   If  $T > MAXN$  Then T is pruned
5.   Else
6.     If T is a valid network graph Then add T to CN
7.     For each  $R_i^{NorQ}$  in T do
8.       For each  $R_j^N$  that is adjacent to  $R_i^N$  in SG do
9.          $f\_new = h(R_j^N)$ 
10.         $g(R_j^N) = c(R_j^N, R_i^N)$ 
11.         $f(R_j^N) = g(R_j^N) + f\_new$ 
12.        If  $f(R_j^N) \leq f\_limit$  Then
13.          Add  $R_j^N$  in front of E
14.        Else
15.          Add  $R_i^N$  in front of E
16.           $f\_new = f(R_j^N)$ 
17.           $f\_limit = MAX.VALUE$ 
18.        End if
19.      End for
20.    End for
21.  End if
  }
22. For each network graph in CN, if there is more than one network
graph in CN, then the same network graph is pruned.
23. Return CN.

```

Figure 4. Heuristic_CNGen Algorithm

We identify a network graph as a joined expression of the query tuple sets that produces candidate networks as result. We define the size of a network graph as the number of nodes the same as the generated CN's size. In Figure 4, we present the candidate network generation algorithm based on IDA* algorithm [3, 6] to generate all network graphs for a given query Q and schema graph SG.

We set up three parameters: MAXN, f_limit and f_new. First, the maximum number of tuple sets, denote MAXN, in a network graph to reduce generating meaningless results. Second, the node R_j^N add in front of queue E, if the estimated cost of the cheapest solution through node R_j^N is less than given f_limit value. If the estimated cost of node R_j^N is more than f_limit value, the node R_i^N that is adjacent node R_j^N in SG add in front of E. Third, f_new assign heuristic value of a new node that is adjacent by the existing node in schema graph. Finally, the number of CNs is only data bounded by the query and database. The Properties 1 and 2 prove the completeness and duplication-free on the results of the algorithm, if we do not violate any constraints.

5.2. AT_CNGen Algorithm

We present another CN generation algorithm (AT_CNGen) to improve the performance of Heuristic_CNGen. In CN generation, we used Heuristic_CNGen algorithm in order to reduce the computation cost and memory cost. This algorithm generated the valid CN in order to complete and duplication-free. It computes a heuristic value to estimate the cost of a new node that is adjacent by the existing node in SG at once. If the two nodes in SG have same heuristic value, the algorithm expands these nodes iteratively. We observe that this algorithm is not efficient because it does not reduce the large overhead for the above shortcomings. Therefore, we propose a new CN generation algorithm (AT_CNGen) based on adjacent tuple list to address the efficiency of CN generation.

We can identify an adjacent tuple following the primary and foreign keys into the schema graph, because the relational database is designed as a schema graph SG. An adjacent tuple is defined a set of adjacent tuple connected by primary-foreign key relationship in SG. In this way, we recognize an adjacent tuple as a joined expression of the query tuple sets that produces CNs as result. We define the size of an adjacent tuple as the number of tuples the same as the generated CN's size.

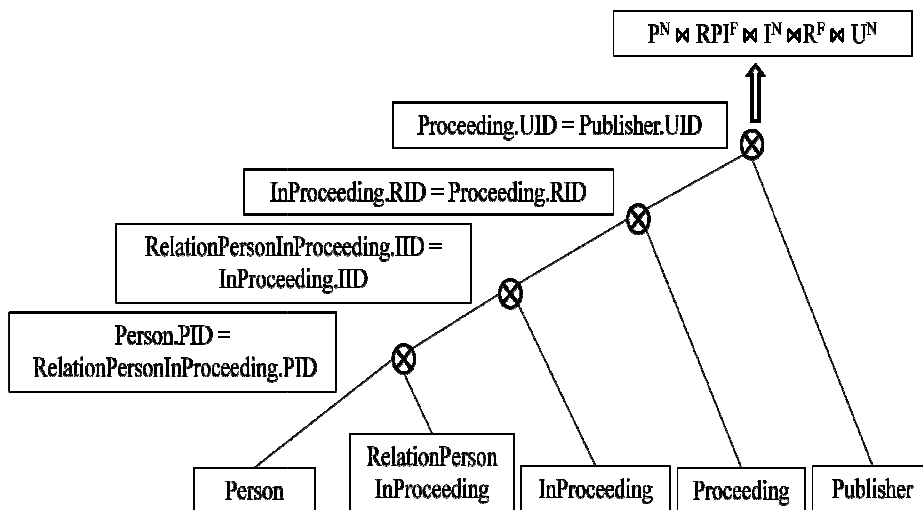


Figure 5. Adjacent Tuple List for DBLP

Consider the publication database schema graph in Figure 1 and the movies database schema graph in Figure 2, where we can iteratively get each tuple in SG followed by primary-foreign keys relationship and obtain the adjacent tuple. For example, we illustrate the adjacent tuple list for query Q = “Chen Web Springer” in DBLP and the adjacent tuple list for query Q = “Black Jack David” in IMDB that are shown in Figure 5 and Figure 6.

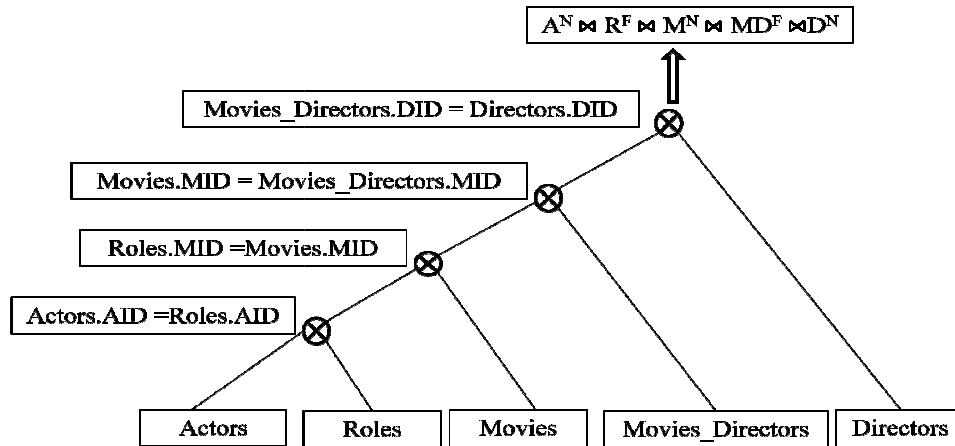


Figure 6. Adjacent Tuple List for IMDB

The adjacent tuple list based method has some features. First, adjacent tuple lists is effective to generate candidate networks as they capture structures. They can depict a meaningful and non-isomorphic. Second, the relationships between adjacent tuples through primary-foreign keys can be identified, so we can efficiently generate the valid candidate networks. Third, a set of adjacent tuple list is no larger than the total primary-foreign keys relationships in the underlying database.

```

AT_CNGen()
Input: A set of adjacent tuple list D
Output: A set of CN
1. CN ← Φ
2. For each T ∈ getTupleSet(ki) do
3.   For each D ∈ getAdjacentList(T) do
4.     For each d ∈ D do
5.       If( d is a valid CN and d ∉ CN ) Then
6.         Add d into CN.
7.       Else
8.         d is pruned.
9.     End if
10.   End for
11. End for
12. End for
13. Return CN.
    
```

Figure 7. AT_CNGen Algorithm

We demonstrate this algorithm to generate all CNs for a given query Q and schema graph SG that is shown in Figure 7. In order to generate valid CNs, AT_CNGen algorithm first accepts the adjacent tuple lists as input. During each CN generation, AT_CNGen calls $getTupleSet(K)$ to get a query tuple sets T for a given query Q . And then it calls $getAdjacentList(T,MAXN)$ to take the adjacent tuple list for getting query tuple sets. Each adjacent tuple list d adds CN, if d is a valid CN and is duplicated on each others. If d is invalid and identical, d is pruned.

```

getTupleSet(K)
Input: query  $Q=\{k_1,k_2,\dots,k_i\} \in K$ , all non-free tuple sets
 $\{R_1^N, R_2^N, \dots, R_i^N\}$  and all free tuple sets  $\{R_1^F, R_2^F, \dots, R_i^F\}$ 
for  $Q$ 
Output: A set of query tuple set  $T$ 
1.  $T \leftarrow \Phi$ 
2. While (K.length is not empty){
3.   If (K.length = 1) Then
4.      $R_i^N \leftarrow Index(k_i)$ 
5.     Add  $R_i^N$  into  $T$ .
6.   Else
7.      $R_i^N \leftarrow Index(k_i)$ 
8.     If ( $R_i^N \notin T$ ) Then
9.       Add  $R_i^N$  into  $T$ .
10.    Else
11.      Ignore  $R_i^N$ .
12.    End if
13.  End if
14. }
15. Return  $T$ .

```

Figure 8. $getTupleSet$ Algorithm

In Figure 8, $getTupleSet(K)$ returns a set of query tuple set T for a keyword query Q . We set up one parameter K that is the number of keyword. We receive all the non-free query tuple set R^N and the free query tuple set R^F of each relation R for a given query. First, the algorithm checks the length of input keyword query. If length of keyword query is one, it produces the non-free query tuple sets R_i^N by using the inverted index for that keyword. And it returns the query tuple sets. Where keyword's length is more than one, the non-free query tuple sets R_i^N is made by indexing for each keyword query. At that time, the algorithm returns T by adding R_i^N which are not identical.

$getAdjacentList(T,MAXN)$ is put two parameter: T and $MAXN$ that is shown in Figure 9. It first receives the non-free query tuple sets T and schema graph SG as input. For each tuple set R_i^N , it takes R_j^{NorF} that is adjacent tuple R_i^N in schema graph as adjacent tuple d . Next the algorithm checks each adjacent tuple d , which is not identical and no more than the maximum number of tuple sets. And it removes an invalid adjacent tuple. The algorithm returns the valid adjacent tuple list subsequently.

Eventually, AT_CNGen algorithm generates all candidate networks no more than the maximal number of tuple sets for the user input keywords. The generated CNs is only data bounded by following Properties 1 and 2.

```

getAdjacentList(T,MAXN)
Input: Tuple set T:  $\{R_1^N, R_2^N, \dots, R_i^N\} \in T$ , Schema graph SG
Output: Adjacent tuple list D
1.  $D \leftarrow \Phi$ 
2. For each  $R_i^N$  in T do
3.   For each  $R_j^{NorF}$  that is adjacent to  $R_i^N$  in SG do
4.     Add  $R_j^{NorF}$  with an edge into d.
5.     If (  $d \notin D$  and  $d < MAXN$ ) Then
6.       Add d into D.
7.     Else
8.       Ignore d.
9.     End if
10.  End for
11.  End for
12. Return D.

```

Figure 9. getAdjacentList Algorithm

6. QUERY EXECUTION

In this section, we present the generating CTTs by executing the generated CNs in order to get the results. For instance, consider the query $Q = \text{“Jinlin Content”}$. The corresponding result consists of the connected tuple trees: $P1 \rightarrow I1$, $P1 \rightarrow I2$. Each connected tuple trees corresponds to a tree at schema level. For example, both of the above trees correspond to the schema level tree $\text{Person}^{(Jinlin)} \rightarrow \text{Relation-Person-InProceeding}^{(Content)} \leftarrow \text{InProceeding}^{(Content)}$, where each R_i^K consists of the tuples of R_i that contain all keywords of K and no other keyword of Q .

Given a query Q , all possible tuple sets R_i^K are computed, where $R_i^K = \{t \mid t \in R_i \wedge \forall w_k \in K, t \text{ contains } w_k \wedge \forall w_j \in Q \setminus K, t \text{ does not contain } w_j\}$ [12]. After selecting a keyword query w_i , all tuple sets R_i^K for which $w_i \in K$ are located. These are the initial connected tuple tree with only one node. Then, these trees are expanded either by adding a tuple set that contains at least another keyword query or a tuple set that is free tuple set. These trees can be further expanded. The connected tuple trees that contain all keywords query are returned.

In RDBMS, the problem of evaluating all CNs in order to get all connected tuple trees is a multi-query optimization problem. There are two main issues: (1) How to share common sub-expressions among CNs generated in order to reduce computational cost when evaluating. (2) How to find a proper join order to fast evaluate all CNs. For a

keyword query, the number of CNs generated can be very large. Given a large number of joins, it is extremely difficult to obtain an optimal query processing plan. It is because one best plan for a CN may make others slow down, if its subtrees are shared by others CNs [11].

The idea of evaluating the sub-expressions is a well-known topic in query optimization. DISCOVER [4] proposed the algorithm to evaluate all CNs together using a greedy algorithm. In this algorithm, sub-expressions that are shared by most CNs should be evaluated first and may generate the smallest number of results should be evaluated first. S-KWS [10] constructed an operator mesh in order to share the computational cost of evaluating all CNs. When evaluating all CNs in a mesh, a projected relation with the smallest number of tuples is selected to start and to join. Qin et al. [11] observed that evaluating all CNs using only joins may always generate a large number of temporary tuples. They proposed to use semijoin/join sequences to evaluate a CN.

In this paper, we present the algorithm for executing CN in order to CN evaluation strategy. It is observed that there is substantial evaluating the common join expressions among CNs. As a consequence, the computational efforts can be saved if multiple CNs can be executed in a calculated way that minimizes the sizes of joining intermediate results.

6.1. Evaluating Candidate Network

We present the D_CNEval algorithm based on the idea of dynamic query optimization algorithm [14] for CN evaluation that is shown in Figure 12. The D_CNEval algorithm is devised to perform this task. In general, it will return the result of query execution.

We set up one parameter such as CN and use the non-free tuple sets as input, which belong to that CN. The algorithm evaluates the size of CN. If the size of CN is equal to one, the algorithm can directly return the executed result for the CN. But if the size of CN is more than one, non-free tuple sets of each relation in CN are projected, which contain keyword query, to reduce the size of intermediate results. These projected results are executed as SQL queries and also are merging all results. If the algorithm executes an empty tuple set, it can stop the current state and it will start to execute the next relation in CN. As a consequence, the algorithm returns all executed results for each CN. The examples of evaluated CN for Q = “Chen Web Springer” in DBLP and Q = “Black Jack David” in IMDB that are shown in Figure 10 and Figure 11.

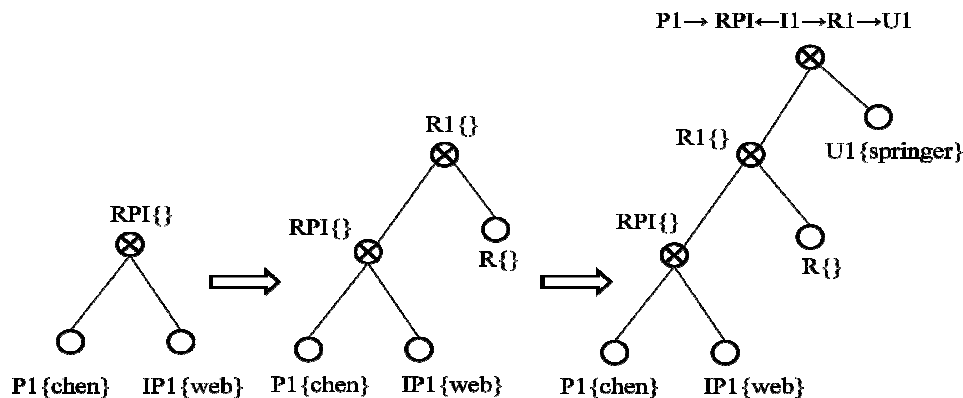


Figure 10. Processing of Evaluated CN on DBLP

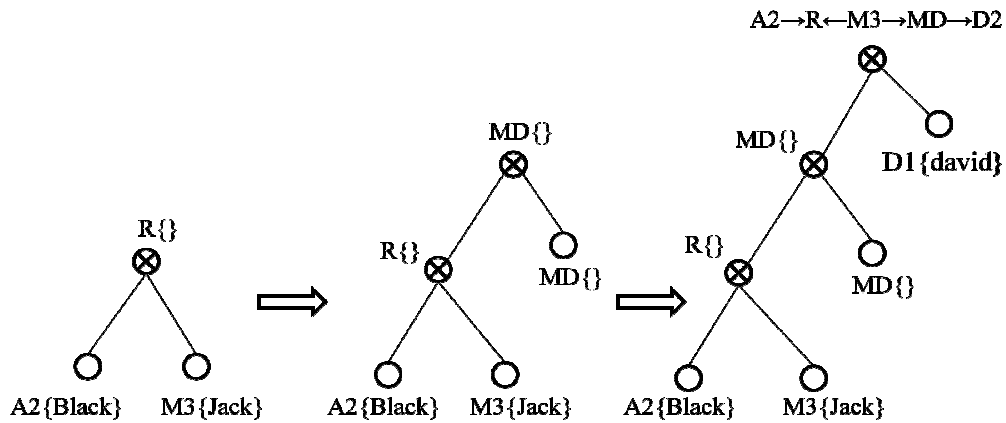


Figure 11. Processing of Evaluated CN on IMDB

```

D_CNEval(CN)
Input: non-free tuple sets  $\{R_1^N, R_2^N, \dots, R_i^N\}$ , in which belongs to CN
Output: result of query execution EQ
1. EQ =  $\Phi$ 
2. n = sizeof(CN)
3. If (EQ.isEMPTY()) then
4. Return  $\Phi$ 
5. Else {
6.   If (n = 1) then
7.     EQ = executeTo(  $R_i^N$  )
8.   Else{
9.     For each  $R_i^N \in CN$  {
10.       $R_i^{N'}$  = projectTo(  $R_i^N$  )
11.      EQ' = executeTo(  $R_i^{N'}$  )
12.      If (EQ'.isEMPTY()) then continue
13.      Else
14.        EQ = EQ  $\cup$  EQ' } }
15. Return EQ.
    
```

Figure 12. D_CNEval Algorithm

We perform a worst case time analysis of the D_CNEval algorithm. If the size of CN is equal 1, we assume that we execute the result in time $O(1)$. The for-loop is executed at most $|CN|$ times for every relation in CN, where $|CN|$ is the size of candidate network. Given a n-relation in CN, the attributes of relation are projected by the non-free tuple set R_i^N that are executed due to projected result. In each step, we assign the query results with the same array lists. Evaluating the query results in R_i^N takes time T, where T is the number of tuples, that are contained keyword, in

relation. Hence we check if a query result is empty in R_i^N and reduce its time in $O(1)$. The query result is merged at most $|T|$ times. Hence the total execution time takes in the worst case time $O(|CN|.|T|)$. The D_CNEval algorithm may output the results of query execution by reducing the size of intermediate joining results.

6.2. Generating Connected Tuple Trees

We display the processing of generated CTT as shown in Figure 13. For a given query Q, the connected tuple tree is generated according to an evaluated CN that is some tuples coming from different relations. For each pair of adjacent tuple sets R_i, R_j in connected tuple tree, there is an edge (R_i, R_j) in SG. Each CTT that defined satisfaction as follow:

- Property 3. If a node in connected tuple tree is one of tuples in relation, it contains at least one keyword in query Q (completeness).
- Property 4. There is no duplicate tuple with each other in the connected tuple tree (duplication-free).

In Figure 14, CTT1 and CTT2 for Query 1 and Query 2 in DBLP are presented as examples. In CTT1, a node P2 contains the keyword “Peter”, and I3 contains keyword “XML” and U1 contains the keyword “Springer” in Query 1. In CTT2, the nodes P3 and I4 contain the keywords “David” and “Modelling”, and U1 contains the keyword “Springer” in Query 2 respectively.

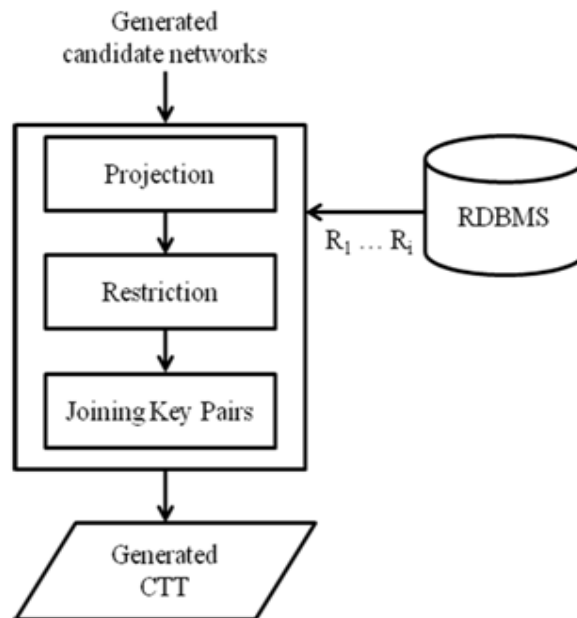


Figure 13. Processing of Generated CTT

For IMDB, CTT3 and CTT4 for Query 3 and Query 4 are illustrated as examples in Figure 15. In CTT3, a node M1 contains the two keywords “Love” and “Story”, and D1 contains keyword “Elley” in Query 3. In CTT4, the nodes A2 and M3 contain the keywords “Black” and “Jack”, and D2 contains the keyword “David” in Query 4 respectively. Except primary-foreign relation nodes, all remaining nodes contain the keywords in given query, and there are no duplicate nodes. In this paper, we consider a connected tuple tree as a result as long as it fulfills the properties.

Query 1:	“Peter XML Springer”
CTT1:	$P2 \rightarrow RPI \leftarrow I3 \rightarrow R2 \rightarrow U1$
CN1:	$P^N \bowtie RPI^F \bowtie I^N \bowtie R^F \bowtie U^N$
Query 2:	“David Modelling Springer”
CTT2:	$P3 \rightarrow RPI \leftarrow I4 \rightarrow R4 \rightarrow U1$
CN2:	$P^N \bowtie RPI^F \bowtie I^N \bowtie R^F \bowtie U^N$

Figure 14. Queries, CTTs and CNs for DBLP

Query 3:	“Elley love Story”
CTT3:	$M1 \rightarrow MD \leftarrow D1$
CN3:	$M^N \bowtie MD^F \bowtie D^N$
Query 4:	“Black Jack David”
CTT4:	$A2 \rightarrow R \leftarrow M3 \rightarrow MD \rightarrow D2$
CN4:	$A^N \bowtie R^F \bowtie M^N \bowtie MD^F \bowtie D^N$

Figure 15. Queries, CTTs and CNs for IMDB

7. PERFORMANCE EVALUATION

7.1. Evaluation Setup

We evaluate the search efficiency of proposed algorithms on DBLP and IMDB datasets. All queries generating algorithms were implemented in Java, and JDBC was used to connect to the database. We conducted all the experiments on Core(TM) 2 Duo CPU and 2GB memory laptop running XP. We take the average executing time on running 15 times.

Dataset: We use two real datasets the Original Digital Bibliography and Library Project (DBLP) dataset [5] and the Internet Movie Database (IMDB) [17] in our evaluation. DBLP contains publications records. IMDB contains movies records. Table 1 and Table 2. show the schema and statistic of two datasets.

Query Set: We manually picked a large number of queries for evaluation. We attempted to include a wide variety of keywords and their combinations in the query sets, such as the selectivity of keywords, the size of the most relevant answers, the number of potential relevant answers, etc. We focus on a subset of the queries in this experiment. There are 20 queries with query length ranging from 2 to 6.

Table 1. Statistics of DBLP Dataset

Relation Schema	#Tuples
Person(Pid,Name)	174,709
InProceeding(Iid,Title,Pages,Rid)	212,273
Proceeding(Rid,Title,Uid,Sid,...)	3,007
Publisher(Uid,Name)	86
Series(Sid,Title)	24
RelationPersonInProceeding(Pid,IPid)	491,777

Table 2. Statistics of IMDB Dataset

Relation Schema	#Tuples
Actors(Aid,Name)	817,718
Directors(Did,Name)	86,880
Movies(Mid,Name,Year,Rank)	388,269
Movies-Directors(Mid,Did)	406,967
Movies-Genres(Mid,Genre)	417,784
Roles(Aid,Mid,Role)	3,432,630

7.2. Evaluation

We implement the Heuristic_CNGen and AT_CNGen algorithms for CN generation. In practical, we observe that AT_CNGen algorithm is small overhead than Heuristic_CNGen algorithm. To compare the performance of these algorithms, we analyze the worst case time of Heuristic_CNGen algorithm and AT_CNGen algorithm.

In Heuristic_CNGen algorithm, while-loop is performed at most $|E|$ times for every tuple sets in queue E , where $|E|$ is the size of queue. We check the network graph T that is more than maximum number of tuple sets when T is put from queue to calculate as first. If T is greater than maximum number of tuple sets, we can reduce its time in $O(1)$. We add T into Hash table H with generated CN for each tuple set when T is not more than maximum number of tuple sets. This step is increasing the computation time in $O(1)$. And the firstly for-loop is achieved at most $|T|$ times for each tuple set in T , where $|T|$ is the size of network graph that is less than or equal to maximum number of tuple sets. The next for-loop is fulfilled at most $|T|$ times because each node in T traverses the adjacent node with it in schema graph. Generating all valid candidate networks in schema graph takes time $|T|$. In completeness, the CN generation time takes in $O(|E|-|T|)$. Then, we filter out the duplicated CN with for-loop that takes at most time $|T|$. Hence the total execution time takes in the worst case time $O((|E|-|T|)^2)$.

In AT_CNGen algorithm, we suppose that the number of keyword is K and the size of tuple set is T and the maximal adjacent tuple in schema graph is M . The for-loop in algorithm $getTupleSet(k)$ is executed at most $|K|$ times for each keyword in keyword query, time complexity to construct the tuple set is $O(|K|)$. After generating the tuple set, $getAdjacentList(T,MAXN)$ algorithm is transversed at most $|T|$ time for each tuple set that is adjacent tuple in the schema graph. The time complexity of transversing adjacent tuple is $O(|T|)$. Then, we check the duplicated CN with for-loop that takes at most time $|M|$. As a consequence, the total time complexity is $O(|K|.|T|.|M|)$.

In summary, AT_CNGen algorithm can be reduced the computation time and searching space, while Heuristic_CNGen algorithm expands the nodes in schema graph which have same heuristic values. Moreover, AT_CNGen algorithm generates the valid CNs due to completeness and duplication-free. Finally, we select AT_CNGen algorithm for the CN generation of relational keyword search system.

7.3. Experimental Results of the Candidate Network Generation

We compare the evaluation results of the proposed AT_CNGen algorithm and existing algorithms by using the same DBLP and IMDB datasets. We observe that proposed algorithm achieve better search performance over the two datasets than the existing algorithms, such as DISCOVER and SPARK, that is shown in Figure 16 and Figure 17.

The proposed AT_CNGen algorithm can generate the valid CNs by the maximal CN size. The proposed algorithm can produce the number of CNs by duplication-free. Then, the new CN generation algorithm is compared with all existing CN generations to eliminate redundancies. The proposed method reduces the number of CNs grows very large even for small CN size. The elapsed time of SPARK is faster than DISCOVER, but SPARK cannot bounded to produce the number of CN for the small CN's size. We can see that elapsed time of the proposed AT_CNGen algorithm is exponentially smaller than DISCOVER and SPARK.

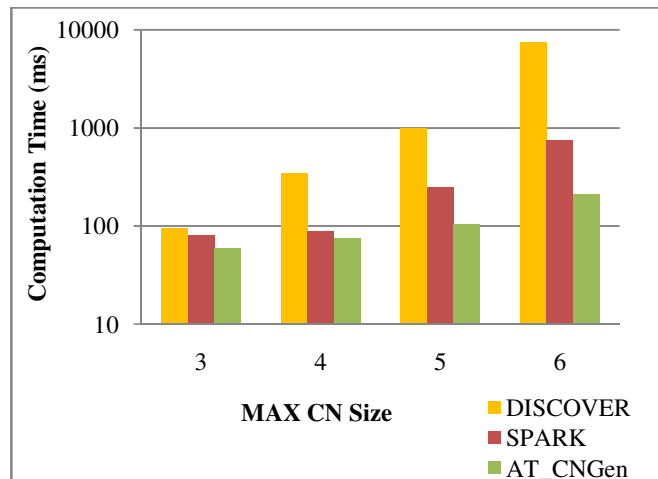


Figure 16. Comparison of AT_CNGen and Others previous algorithms on DBLP

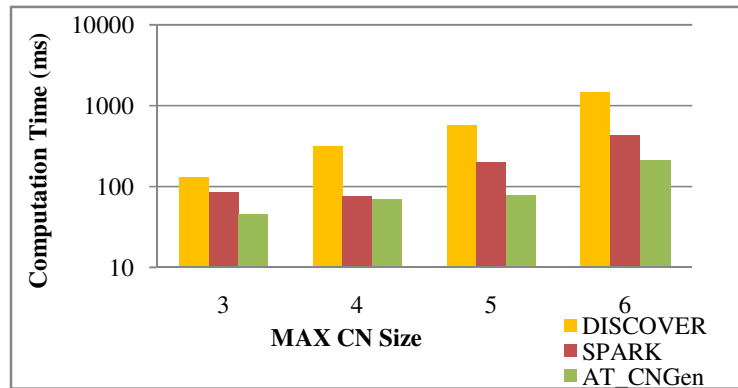


Figure 17. Comparison of AT_CNGen and Existing algorithms on IMDB

7.4. Experimental Results of Query Execution

In this section, we measure the computation time for sample queries of DBLP dataset in Figure 18 by implementing the D_CNEval algorithm. Moreover, we also present sample queries of IMDB dataset in Figure 20 to evaluate the efficiency of D_CNEval algorithm. Given a keyword query, the proposed algorithm generates the valid CNs. The generated CNs is evaluated by reducing the size of intermediate results to get the final results.

Query	Keywords
Q1	chen
Q2	chen web content
Q3	chen web springer
Q4	web content
Q5	content springer bychen
Q6	chen web springer 2000
Q7	david compiler generator
Q8	compiler springer by david
Q9	compiler generator
Q10	david compiler generator springer

Figure 18. Keywords Queries on DBLP

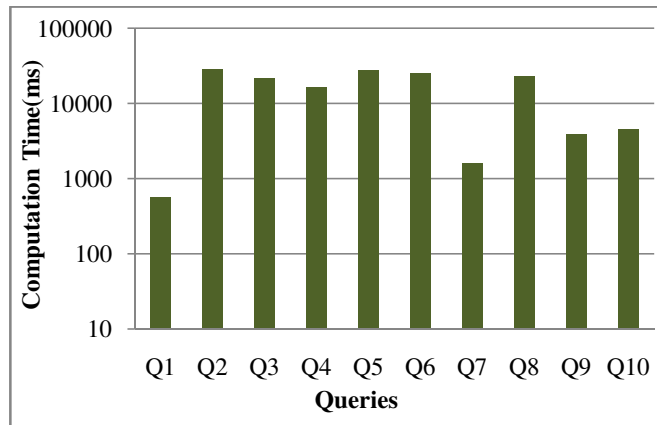


Figure 19. Execution Times for Queries of DBLP

Figure 19 shows the execution times by evaluating the number of queries. In this figure, Q1 can execute the result at minimum time because this query selects the executed query in a single relation. Also Q7, Q9 and Q10 can evaluate the result queries at minimum time although two or more relations joined due to the primary-foreign relationship.

Then, we present the evaluation of execution times for the queries in IMDB that is shown in Figure 21. We can see that Q14 and Q19 can execute the result at minimum time because this query selects the executed query in a single relation. Also Q13, Q15 and Q18 can evaluate the result queries at minimum time although two or more relations joined due to the primary-foreign relationship. So, we observe that the D_CNEval algorithm execute the final result to speed up.

Query	Keywords
Q11	alexander
Q12	hollywood
Q13	elley love story
Q14	monkey island
Q15	blake death
Q16	mile allen
Q17	godfather
Q18	come away 2005
Q19	2010:continues
Q20	black jack david

Figure 20. Keywords Queries on IMDB

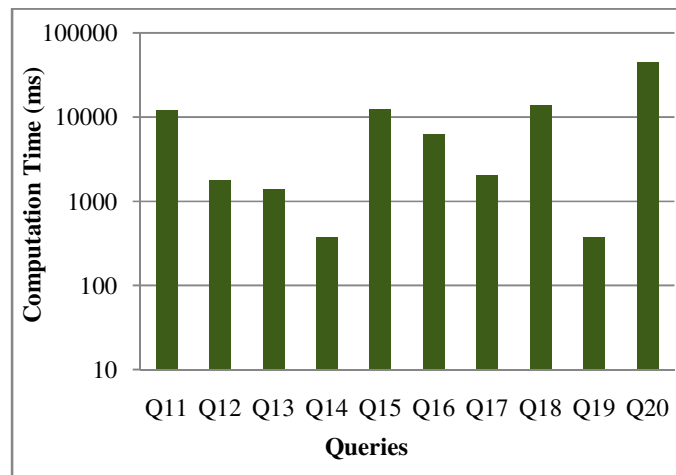


Figure 21. Execution Times for Queries of IMDB

8. CONCLUSIONS

Efficient keyword search in relational databases allows ordinary users to find text information in relational databases with much higher flexibility. A keyword query in the system is a list of keywords and does not need to specify any relation or attributes names. The result to such a keyword query consists of the minimal connected tuple trees, which potentially include tuples from multiple relations in database. We first proposed a CN generation algorithm (Heuristic_CNGen) to produce all CNs. Although this algorithm produces the valid CNs, it reduced the system performance when traversing the same heuristic values nodes. In order to improve the performance, we also propose a new CN generation algorithm (AT_CNGen) to generate all CNs for relational keyword search system. The proposed candidate network algorithm can solve the growing number of CNs for small CN size by comparing with existing algorithms. Moreover, we observe that AT_CNGen algorithm achieved the system performance as high as Heuristic_CNGen algorithm. Then, we propose the dynamic CN evaluation algorithm (D_CNEval) to produce the connected tuple trees by reducing the joining intermediate results. The proposed CN evaluation algorithm can generate the minimal number of CTTs by doing minimal accesses to the database, and does not have data bound with the maximum number of tuple set. We presented the experimental results on DBLP and IMDB show that the proposed algorithms generate the result approximately for the user desired query. And the experimental results are efficiently evaluated by using query execution strategy.

REFERENCES

- [1] Agrawal,S., Chaudhuri,S., Das,G. (2002). DBXplorer: A System for Keyword-Based Search over Relational Database. Proc. 18th Int. Conf. on Data Engineering, pp. 5-16.
- [2] Baid,A., Rae,I., Li,J., Doan,A., Naughton,J. (2010). Toward Scalable Keyword Search over Relational Data. Proc. VLDB Endowment, Vol. 3.
- [3] Felner,A., Korf,R.E., Hanan,S. (2004). Additive Pattern Database Heuristics. Journal of Artificial Intelligence Research, (p.279-318).
- [4] Hristidis,V., Papakonstantinou,Y. (2002). DISCOVER: Keyword Search in Relational Databases. Proc. 28th Int. Conf. on Very Large Data Bases, pp. 670-681.
- [5] <http://www.dblp.uni.trier.de>.
- [6] Korf,R.E. Depth-First Iterative-Deepening: An Optimal Admissible Tree Search.

- [7] Liu,F., Yu,C., Meng,W. (2006). Effective Keyword Search in Relational Databases. Proc. 2006 ACM SIGMOD Int. Conf. on Management of data, pp. 563-574.
- [8] Li,P., Zhu,Q., Wang,S. (2008). The Research on the Algorithms of Keyword Search in Relational Database. Springer, pp. 134-143.
- [9] Luo,Y., Wang,W., Lin,X., Zhou,X. (2011). SPARK2: Top-k Keyword Query in Relational Databases. TKDE Special Issue: Keyword Search on Structured Data.
- [10] Markowetz,A., Yang,Y., Papadias,D. (2007). Keyword Search on Relational Data Streams. Proc. 2007 ACM SIGMOD Int. Conf. on Management of data, pp. 605-616.
- [11] Qin,L., Yu,J.X., Chang,L. (2009). Keyword Search in Databases: The Power of RDBMs. Proc. 35th SIGMOD Int. Conf. on Management of data, pp. 681-694.
- [12] Stefanidis,K., Drosou,M., Pitoura,E. (2010). PerK: Personalized Keyword Search in Relational Databases through Preferences. Proc. 13th Int. Conf. on Extending Database Technology, EDBT, pp. 585-596.
- [13] Thein,M.M. (2012). Querying Connected Tuple Trees for Relational Keyword Search. Proc. Int. Conference on Information Retrieval and Knowledge Management, pp. 285-289.
- [14] Tamer Özsü.M, Valduriez.P. (2011). Principles of Distributed Database Systems. Springer.
- [15] Wang,S., Zhang,J., Peng,Z., Zhan,J., Wang,Q. (2007). Study on Efficiency and Effectiveness of KSORD. APWeb/WAIM Int. Workshops, pp. 6-17.
- [16] Xu,Y., Ishikawa,Y., Guan,J. (2009). Effective Top-k Keyword Search in Relational Databases Considering Query Semantics. APWeb/WAIM Int. Workshops, pp. 172-184.
- [17] <http://www.imdb.com/interfaces>.
- [18] YU,J.X., Qin,L., Chang,L. (2010). Keyword Search in Relational Databases: A Survey. IEEE Data Engineering Bulletin, Vol. 33, pp. 67-78.