Effective Top-k Keyword Search in Relational Databases Considering Query Semantics

Yanwei Xu[1,2], Yoshiharu Ishikawa[1], and Jihong Guan[2]

[1] Graduate School of Information Science, Nagoya University, Japan
[2] School of Electronics and Information Engineering, Tongji University, China

Abstract. Keyword search in relational databases has recently emerged as a new research topic. As a search result is often assembled from multiple relational tables, existing IR-style ranking strategies can not be applied directly. In this paper, we propose a novel IR ranking strategy considering query semantics for effective keyword search. The experimental results on a large-scale real database demonstrate that our method results in significant improvement in terms of retrieval effectiveness as compared to previous ranking strategies.

Key words: top-k,keyword search,effective,relational database

1 Introduction

With the amount of available text data in relational databases growing rapidly, the need for ordinary users to effectively search such information is increasing dramatically. Keyword search is the most popular information retrieval method because the user needs to know neither a query language nor the underlying structure of the data. Keyword search in relational databases has recently emerged as an active research topic. In this paper, we focus on how to support effective top-k keyword search in relational databases.

Although most of the popular DBMSs support full-text search, they only provide support for retrieving tuples relevant to a query within the same relation. A unique feature of keyword search in relational databases is that search results are often joined tuples from multiple relations.

Example 1: Suppose a user wants to search papers written by *"Ralf Steinmetz"* with *"p2p"* in their titles from the DBLP[3] database (its schema is shown in Figure 1). He might give a query containing two keywords: *"p2p Steinmetz"*. Our system will return the results shown in Table 1, where relevant tuples from multiple relations (presented in bold font) are joined together to form a meaningful answer to the query. Table 1 shows that three papers with *"p2p"* in their titles were written by *"Ralf Steinmetz"*.

Recently, there have been many studies dedicated to keyword search in relational databases [1–5]. Among these, [3] was the first to consider top-k keyword

[3] http://dblp.mpi-inf.mpg.de/dblp-mirror/index.php

Table 1. Top-3 results for query *"p2p Steinmetz"*

1	**Article:**Token-Based Accounting for P2P-Systems.→**Author:** Ralf Steinmetz
2	**Article:**An Adaptable, Role-Based Simulator for P2P Networks.→ **Author:**Ralf Steinmetz
3	**Article:**Self-protection in P2P Networks: Choosing the Right Neighbourhood.→ **Author:**Ralf Steinmetz

search in relational databases; it incorporates a state-of-the-art IR ranking formula to address the retrieval effectiveness issue and presents several efficient query execution algorithms optimized for returning top-k relevant answers. [4] improves the ranking formula in [3] by adapting four normalizations. [5, 6] further modify the ranking formula of [3] by introducing the concept of a *virtual document* and present two efficient query evaluation algorithms for their ranking formula. [7] takes another approach to address keyword search based on the Steiner tree.

Due to the fuzzy nature of keyword queries, result ranking is vital for retrieval effectiveness. Despite the results from previous studies, there are still several issues with existing ranking methods, some of which may discourage users to use keyword search systems. In this paper, we present a method for improving the ranking formula by considering query semantics.

The main contributions of this paper are as follows:

- We introduce the concept of query semantics for keyword search in relational databases. Although this concept has been mentioned in previous works [5], it was not considered as a factor for ranking search results.
- We propose a method for incorporating query semantics into the ranking formulas proposed in [3, 5]. To our knowledge, our paper is the first to rank CNs.[4]
- We conduct comprehensive experiments on large-scale real databases. The experimental results show that our approach is better than existing ones in terms of effectiveness.

The rest of the paper is organized as follows: Section 2 presents the basic concepts and the method for generating the relevant answers of a query. Section 3 introduces the ranking strategies used in previous works. Section 4 presents the concept of query semantics and our method of ranking answers by considering query semantics. Section 5 shows the experimental results. Section 6 discusses the related works. Section 7 concludes this paper.

2 Preliminaries

In this section, we describe the framework for generating answers for a given keyword query. Section 2.1 describes some basic concepts such as Candidate Network (CN) and Joint-Tuples-Tree (JTT). We follow the definitions of previous work [5, 8]. Section 2.2 describes the framework of generating query answers.

[4] CN is short for *candidate network*, which will be introduced in Section 2

2.1 Basic Concepts

We first define some terms used throughout the paper. A relational database composed of a set of relations $R_1, R_2, \cdots, R_n$. A *Schema Graph* (SG) is a directed graph with the relations as its nodes and the foreign key to primary key relationships of the relations as its edges. Figure 1 shows the schema graph of DBLP used in this paper. A *Joint-Tuple-Tree* (JTT) T is a joining tree of different tuples. Each node t_i is a tuple in the database, and each pair of adjacent tuples in T is connected via a foreign key to primary key relationship. The three results in Table 1 are examples of JTTs. A JTT is an answer to a keyword query if it contains more than one keyword of the query and each of its leaf tuples must contain at least one keyword. A *Query Tuple Set* R^Q is a set of all tuples which belong to relation R; these tuples contain at least one keyword of the query Q. We call R^F the *free tuple set*, which is the set of all tuples in relation R and we use R^* to denote a *tuple set*, which can be either a query tuple set or a free tuple set. A *Candidate Network* (CN) is a tree of tuple sets R^Q or R^F with the restriction that every leaf node must be a query tuple set. Every edge (R_i^*, R_j^*) in a CN corresponds to an edge (R_i, R_j) in the schema graph SG. A CN can be easily transformed into its equivalent SQL statement and executed through the DBMS. The *size* of a CN is the number of its tuple sets.

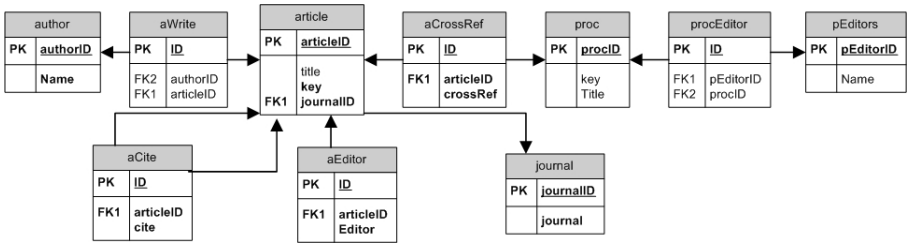

Fig. 1. DBLP schema graph

2.2 Answer Generation

Given a keyword query, the system first generates all the non-empty query tuple sets R^Q for all the relations R. These non-empty query tuple sets and the schema graph are inputted to the CN generator to generate all the valid CNs. For this purpose, [8] has proposed a breadth-first algorithm that is both sound and complete. It can enumerate all the CNs of size no more than a specified number without violating any pruning rules. There are three pruning rules used in [5], which are listed below. We show the traces of the CN generation algorithm for query *"p2p Steinmetz"* in Example 1 in Table 2 (for simplicity, suppose there are only two non-empty query tuple sets $article^Q$ and $author^Q$, and omit relation $aCite$).

Rule 1 Prune duplicate CNs
Rule 2 Prune non-minimal CNs i.e., CNs with free tuple sets as leaf nodes

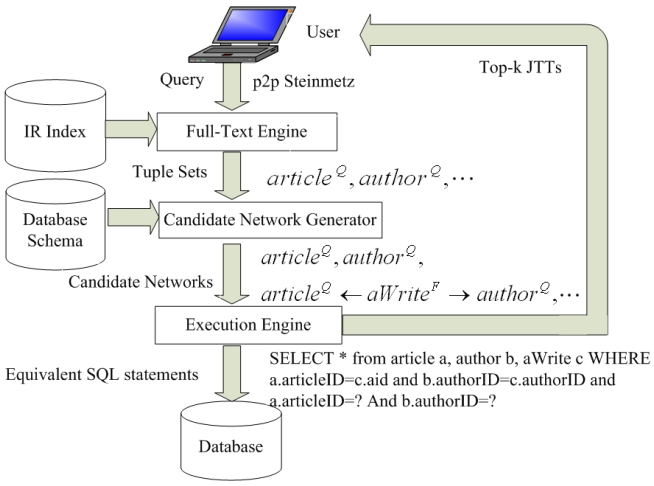

Fig. 2. Query processing framework

Rule 3 Prune CNs of type: $R^Q \leftarrow S^* \rightarrow R^Q$. The rationale is that every resulting JTT would contain the same tuple from R^Q for two times.

Finally, the generated CNs are evaluated to identify the top-k query results based on some relevance formulas. Figure 2 shows query processing framework, which is a modified version of [3].

Table 2. Enumerating CNs for query *"p2p Steinmetz"*

Size	CN ID	CN	Valid?
1	CN_1	$article^Q$	Y
1	CN_2	$author^Q$	Y
2		$article^Q \leftarrow aWrite^F$	n
2		$article^Q \rightarrow journal^F$	n
2		$article^Q \leftarrow aCrossRef^F$	n
3	CN_3	$article^Q \leftarrow aWrite^F \rightarrow author^Q$	Y
3		$article^Q \leftarrow aWrite^F \rightarrow author^F$	n
3	CN_4	$article^Q \rightarrow journal^F \leftarrow article^Q$	Y
3		$article^Q \rightarrow journal^F \leftarrow article^F$	n
3		$article^Q \leftarrow aCrossRef^F \rightarrow proc^F$	n
3		$author^Q \leftarrow aWrite^F \rightarrow article^Q$	n
4	⋮	⋮	⋮

3 Ranking Strategy

In this section, we first present the ranking strategies of previous works, then motivate our work by presenting an observation that reveals a problem in existing schemes.

3.1 Existing Ranking Strategies

Due to the fuzzy nature of keyword queries, result ranking is vital for retrieval effectiveness. The initial attempt was to simply rank results according to the size

of JTTs [2, 8]. Later, IR-Style [3] proposed a ranking formula based on a state-of-the-art IR scoring function e.g., formulas based on the TF-IDF weighting. The basic idea of the ranking method used in [3] is:

1. Assign to each tuple in the JTT a score by using a standard IR-ranking formula[5]; and
2. Combine the individual scores together by using a monotonic aggregation function to obtain the final score.

[4] suggested four sophisticated normalizations to the scoring function in [3]: tuple tree size normalization, document length normalization, document frequency normalization and inter-document weight normalization. The scoring function of [4] is not monotonic due to the four normalizations, and therefore the optimized query evaluation algorithms in [3] cannot be applied.

SPARK [5] models the entire JTT as a virtual document while the entire results produced by a CN is modeled as a document collection. SPARK computes the relevance score for a JTT T as follows:

$$score(T, Q) = score_a(T, Q) \cdot score_b(T, Q) \cdot score_c(T, Q), \tag{1}$$

$$score_a(T, Q) = \sum_{w \in T \cap Q} \frac{1 + ln(1 + ln(tf_w(t)))}{1 - s + s \cdot \frac{dl_T}{avdl(CN^*(T))}} \cdot ln(idf_w), \tag{2}$$

$$\text{where } tf_w(T) = \textstyle\sum_{t \in T} tf_w(t),\ idf_w = \frac{N(CN^*(T))+1}{df_w(CN^*(T))}\ ,$$

$$score_b(T, Q) = 1 - (\frac{\sum_{1 \le i \le m} (1 - T.i)^p}{m})^{\frac{1}{p}}, \tag{3}$$

$$\text{where } T.i = \frac{tf_{w_i}(T)}{max_{1 \le j \le m} tf_{w_j}(T)} \cdot \frac{idf_{w_i}}{max_{1 \le j \le m} idf_{w_j}}\ ,$$

$$score_c = (1 + s_1 - s_1 \cdot size(CN)) \cdot (1 + s_2 - s_2 \cdot size(CN^{nf})), \tag{4}$$

where $tf_w(t)$ denotes the number of instances of w in t, dl_T denotes the length of all the text attributes of T, $CN(T)$ denotes the CN T belongs to, $CN^*(T)$ is identical to $CN(T)$ with the exception that each tuple set is free, $avdl(CN^*(T))$ is the average length of JTTs for $CN^*(T)$, $N(CN^*(T))$ denotes the number of JTTs for $CN^*(T)$, and $size(CN^{nf})$ is the number of non-free tuple sets for the CN. $score_a$ is an IR-style ranking score based on the TF-IDF weighting. $score_b$ acts as the completeness factor and gives biases toward the JTTs which contain all of the keywords in query Q to those which only contain a few keywords. The tuning parameter p in Eq.(3) can smoothly switch the completeness factor biased towards the OR semantics to the AND semantics. $score_c$ is a JTT size factor and its degree of penalties for large CN is between [3] and [4]. The ranking function of SPARK addresses an important deficiency in existing methods and results in substantial improvement of the quality of search results [5].

[5] This score is often automatically computed by the DBMS by using the full-text indexing engine.

3.2 Problems with Existing Ranking Functions

The vector space model based on TF-IDF weighting is used to compute the relevance between a keyword query and documents from a document collection. In the setting of the keyword search in relational databases, there will be multiple document collections (each collection is either a relation or a CN). The above scoring functions only compute a document's relative relevance to the query in the document collection it belongs to. However, these document collections have different levels of importance to the query, and therefore the final score of a document must reflect the importance of the document collection it belongs to. For example, the top ten results of the query in Example 1 returned by SPARK with $p = 2$ (a value of 2.0 is already good enough to enforce the AND-semantics [5]) are listed in Table 3. We can see that most of the results are not useful to the user: no papers written by *"Ralf Steinmetz"* with *"p2p"* in the title are returned.

Table 3. Top ten results for query *"Steinmetz p2p"* by SPARK

JTT	Score	CN
E. Steinmetz	3.40	$author^Q$
Uli Steinmetz	3.37	$author^Q$
2 P2P or Not 2 P2P?	3.35	$article^Q$
2 P2P or Not 2 P2P?	3.35	$article^Q$
Ralf Steinmetz	3.34	$author^Q$
Arnd Steinmetz	3.34	$author^Q$
Rita Steinmetz	3.34	$author^Q$
Aase Steinmetz	3.34	$author^Q$
Oliver Steinmetz	3.28	$author^Q$
Ulrich Steinmetz	3.28	$author^Q$

The total number of tuples in every relation that contain the two keywords in DBLP are shown in Table 4, and enumerated CNs whose size is less than 4 are listed in Table 5.

Table 4. Statistics of keyword *Steinmetz* and *p2p*

Relation	Column	Keyword	Count
proc	title	P2P	11
article	title	P2P	1855
procEditor	Name	Steinmetz	1
author	author	Steinmetz	20

Table 5. Enumerated CNs for query *"Steinmetz p2p"*

CN ID	CN
CN_1	$article^Q$
CN_2	$author^Q$
CN_3	$procEditor^Q$
CN_4	$proc^Q$
CN_5	$article^Q \leftarrow aCrossRef^F \rightarrow proc^Q$
CN_6	$article^Q \leftarrow aWrite^F \rightarrow author^Q$
CN_7	$article^Q \rightarrow journal^F \leftarrow article^Q$
CN_8	$pEditors^Q \leftarrow procEditor^F \rightarrow proc^Q$

Table 4 shows that the two keywords *Steinmetz* and *p2p* mostly occur at relation *author* and *article*, respectively. From a human perspective, results for

CN_6 should be ranked higher than results from other CNs on the basis of the data in Table 4, even if we do not know the user's intentions. Unfortunately, *p2p* is such a popular keyword in *article* as compared to *Steinmetz* in *author* that idf_{p2p} in Eqs. (2) and (3) is very small as compared to $idf_{Steinmetz}$. As a result, answers from CN_1 and CN_2 containing *Steinmetz* are ranked as the top ten answers as Table 3 shows. Of course, $score_a$ and $score_b$ of the answers for CN_6 will be larger than the answers for CN_1 and CN_2. However, the degree of increase is very small because $idf_{p2p} \ll idf_{Steinmetz}$ and is counteracted by the decrease of $score_c$, as the JTT size is 3.

We will show our solution to this problem in next section.

4 Ranking with Query Semantics

In this section, we will first introduce the concept of query semantics, and then discuss how to use it to improve the ranking strategy. Our proposed method shows a notable improvement in the effectiveness of keyword search.

4.1 Query Semantics

We believe that the problem in Table 3 is caused by the fact that the keyword search system does not understand the user's true intention. This is why some keyword search systems allow a query to contain database schema data such as "author:Steinmetz". However, requiring an ordinary user to write queries containing database schema data is not realistic and violates the original motivation for keyword search systems.

In actual commercial databases, there is always a large number of relations. Due to the E-R model and the normalization requirement, each relation stores information of a certain kind of entities, and hence it has its own special keyword set. For example, keywords in relation *author* in DBLP are unlikely to occur in relation *article* or *proc*.

When a user inputs a short query, we can assume that there is a strong possibility that he has a preference for the relation selection for each keyword. For example, he prefers the relation *author* to relation *article* when he inputs the name of person. The implicit relationship between keywords and relations specifies the hidden user's intention for the query.

We refer to such relation preference of keywords as the *semantics* of the query. If a CN contains all the relations in the semantic set of a query, results from this CN will be more relevant to the query

Example 2: a query contains an author name *Steinmetz* and a research area keyword *p2p*, which shows that the user wants to search papers about *p2p* that were written by an author named *Steinmetz*. Hence, the semantic set of query *"Steinmetz p2p"* is $\{article, author\}$. JTTs for CN_6 in Table 5 are more relevant to the query *"Steinmetz p2p"* than JTTs for CN_8.

If the semantics for a keyword query can be obtained exactly and used to rank query results, query effectiveness can be drastically improved. For example,

the papers with *p2p* in the title written by *Ralf Steinmetz* will be ranked at the top of the answers. We present our method for obtaining the query semantics and using it to rank answers in the next section.

4.2 Incorporating Query Semantics into Ranking

We propose modeling a relation as a document, in which case tuples in relations will be modeled as words or sentences. Consequently, the database is a document collection composed of relations as documents. By adopting such a model, we can naturally compute the IR-style relevance score of each relation for a keyword.

For a keyword w, we can easily find all the relations $R_1, R_2, \cdots, R_t$ that have tuples containing w and the corresponding numbers of tuples by using a full-text index. Then, for each $R_i (1 \leq i \leq t)$, we use the following formula, which is based on TF-IDF weighting, to compute its relevance to keyword w:

$$p_w(R_i) = \frac{p_0 + ln(1 + ln(1 + df_w(R_i)))}{(1-s) + s \cdot ln(1 + \frac{tc_{R_i}}{avtc})}, \tag{5}$$

where p_0 indicates the initial preference score for a keyword to an arbitrary relation, $df_w(R_i)$ is the number of tuples of R_i that contains keyword w, tc_{R_i} is the number of total tuples in R_i, $avtc$ is the average number of tuples of the relations in the database.

p_0 and s in Eq.(5) acts as two tuning parameters: small p_0 give higher preference to relations that have a large $df_w(R_i)$, while larger s gives higher preference to relations that have a small number of tuples. In our experiments, p_0 is tuned between 0.2 and 1 and s is tuned between 0.1 and 0.5. We observed that $p_0 = 0.6$ and $s = 0.2$ is appropriate for all the tested queries.

Then the preference of CN for a query Q is computed as:

$$score_s(CN, Q) = \sum_{w \in Q} max_{R_i \in CN} p_w(R_i). \tag{6}$$

We refer to $score_s(CN, Q)$ as the *semantic score* of a CN.

Finally, the relevance score of a JTT T to a keyword query Q is computed as:

$$SCORE(T, Q) = score(T, Q) \cdot score_s(CN(T), Q), \tag{7}$$

where $score(T, Q)$ is computed by Eq.(1).

The calculated $score_s$ for the eight CNs in Table 5 are listed in Table 6 with $p_0 = 1.0, s = 0.25$. We also suggest the application of a modification to Eq.(3) as

$$T.i = \frac{ln(1 + tf_{w_i}(T))}{ln(1 + max_{1 \leq j \leq m} tf_{w_j}(T))} \cdot \frac{ln(idf_{w_i})}{ln(max_{1 \leq j \leq m} idf_{w_j})}, \tag{8}$$

to increase the impact of a keyword w that has a small *idf* (e.g., keyword *p2p*). Table 7 shows the top ten results of SPARK for query *"p2p Steinmetz"* with $p = 1.8$ and our modification when computing $score_b$. We find that the six results which belong to CN_6 are ranked at the top of the results. More importantly, the result belonging to CN_8 is ranked appropriately.

Table 6. Calculated $score_s$ of the eight CNs

CN_1	CN_2	CN_3	CN_4	CN_5	CN_6	CN_7	CN_8
3.24	2.59	3.04	2.14	3.44	4.66	3.24	3.7

Table 7. Top ten results for query *"p2p Steinmetz"*

JTT	CN ID	score
Token-Based Accounting for P2P-Systems.→ Ralf Steinmetz	CN_6	33.25
An Adaptable, Role-Based Simulator for P2P Networks.→ Ralf Steinmetz	CN_6	32.28
Self-protection in P2P Networks: Choosing the Right Neighbourhood.→ Ralf Steinmetz	CN_6	31.29
Globase.KOM - A P2P Overlay for Fully Retrievable Location-based Search.→ Ralf Steinmetz	CN_6	30.88
Ralf Steinmetz→ Proceedings P2P'08, Eighth International Conference on Peer-to-Peer Computing, 8-11 September 2008, Aachen, Germany	CN_8	30.78
Overlay Design Mechanisms for Heterogeneous Large-Scale Dynamic P2P Systems.→Ralf Steinmetz	CN_6	30.62
Working Group Report on Managing and Integrating Data in P2P Databases.→ Rita Steinmetz	CN_6	30.55
E. Steinmetz	CN_2	16.03
Uli Steinmetz	CN_2	15.89
Arnd Steinmetz	CN_2	15.75

5 Experiments

In this section, we experimentally discuss the impact of our proposed method on the effectiveness of top-k keyword search. We incorporate the semantic score into the ranking strategies of [3, 5], and then compare its impact to the effectiveness.

5.1 Experimental Settings

Database: For our evaluation,we use the DBLP data set, which we decomposed into relations from a downloaded XML file according to the schema shown in Figure 1. We use many relations in order to represent the original data in the XML file as closely as possible. The size of the XML file is 478MB. Table 8 shows the basic statistics after the decomposition.

Table 8. Statistics of DBLP database

Relation Schema	# Tuples
article(articleID,key,title,journalID,$\cdots$)	1,092,239
aCite(id,articleID,cite)	109,625
author(authorID,author)	658,461
aWrite(id,articleID,authorID)	2,752,673
journal(journalID,journal)	730
proc(procID,key,title,$\cdots$)	11,108
pEditors(pEditorID,Name)	12,001
procEditor(id,procEditorID,procID)	23,540

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 set, such as the selectivity of keywords, the size of the most relevant answers, the number of potential relevant answers, etc. We focus on 20 queries with query length ranging from 2 to 4.

5.2 Measures

To measure the effectiveness, we adopt two metrics used in previous studies [4, 5]: *a*) number of top-1 answers that are relevant (#**Rel**), and *b*) *reciprocal rank* (**R-Rank**), for a given query. The reciprocal rank is 1 divided by the rank at which the first correct answer is returned or 0 if no correct answers are returned.

In order to identify the relevant answers for each query, we used all the ranking strategies ([3],[5] and ours) for each query and merged their top-50 results. Then, we manually evaluated the results and selected the relevant answer(s) for each query.

5.3 Results and Discussion on Effectiveness

We show the **#Rel** of [3, 5] and our proposed method on the DBLP dataset in Table 9. Figure 3 and 4 show the *reciprocal ranks* of 13 and 10 queries, respectively. We use [3](S), [5](S) to denote that the relevance score is computed by considering the semantic score of CNs. M in [5](M) and [5](SM) denotes the modification shown in Eq.(8). $p = 1.5(2)$ denotes the parameter p in Eq.(3) is set to 1.5(2). From Table 9, we can see the notable improvement brought by our method to the ranking strategies of [3, 5] in terms of effectiveness. [5](SM) can always return the relevant answer(s) for a query. We also find that the semantic score works well with the method of [3]. Although the **#Rel** of [3](S) is not large, we can find relevant answers in the Top-20 answers returned by [3](S) in most cases, as shown in Figure 4.

The experimental results in Table 9 and Figure 3 show that Eq.(8) is characterized by a notable improvement as compared to the method of [5] in terms of **#Rel** and *reciprocal ranks*. There are two reasons for this: Eq.(8) produces a stronger bias to answers that contain all of the keywords in a query; our manually evaluated relevant answers are based on the **AND** semantics for queries. Although they have similar **#Rel** and **R-Rank**, there is a great difference between [5](S) and [5](SM) in the structure of the top-k answers as compared to [5](M): there is always a larger CN number. For example, [5](M) ranks all the answers for CN_6 at the top for the query *"p2p Steinmetz"*. However, we can also find answers for CN_1, CN_2 and CN_8 in the top-ten answers returned by [5](S) and [5](SM). Therefore, although we believe users might target different results with their queries, a larger CN number can meet the demands of more users.

Table 9. Impacts on **#Rel**

[5]($p = 2$)	[5](M)($p = 1.5$)	[5](M)($p = 2$)	[5](S)($p = 1.5$)	[5](SM)($p = 1.5$)	[3]	[3](S)
4	17	18	12	19	0	5

6 Related Work

Keyword search in relational databases has recently emerged as a new research topic [9]. Existing approaches can be broadly classified into two categories: those based on candidate networks [2, 3, 8] and others based on Steiner trees [1, 10, 11].

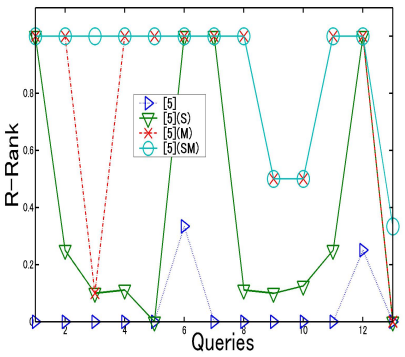

Fig. 3. Impacts on **R-Rank** (1)

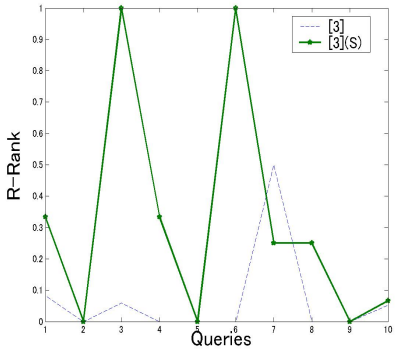

Fig. 4. Impacts on **R-Rank** (2)

Mragyati [12], Discover [8], DBXplorer [2], IR-Style [3] and ObjectRank [13] are several early keyword search systems for relational databases. Discover and DBXplorer only rank tuple trees according to their sizes. IR-Style proposed ranking of tuple trees according to their IR relevance scores to a query. Our work adopt the same framework with [2–4, 8], and can be viewed as a further improvement along the line of enhancing the retrieval effectiveness. ObjectRank and ObjectRank2 [14] apply authority-based ranking to keyword search in databases modeled as labeled graphs. Authority originates at the nodes (objects) containing the keywords and flows to objects according to their semantic connections.

Banks [1] also finds tuple trees from the data graph directly by using the Steiner tree algorithm. For a data graph, it uses PageRank style methods to assign weights to tuples and edges between them. Banks2 [10] is an improvement of Banks which introduces a novel technique of bidirectional expansion to improve search efficiency. Li et al. [7] proposed a new concept referred to as a *compact Steiner Tree*, which can be used to approximate the Steiner tree problem for answering top-k keyword queries efficiently. They also proposed a novel structure-aware index to support keyword search.

Most recently, keyword search has been studied in a few generalized contexts as well [11, 15]. [15] describes a solution to the keyword-search problem over heterogeneous relational databases. The scoring function of [15] is adapted from [3] by adding two more equally weighted terms that capture the match confidence of *FK joins* and the corresponding attribute value pairs in an answer, which may be a JTT composed of tuples come from multiple databases.

7 Conclusions

Keyword search allows non-expert users to find text information in relational databases with much higher flexibility. In this paper, we proposed a novel ranking strategy for effective keyword search considering query semantics. Our method can solve the problems with the ranking strategies proposed in previous works.

We also presented a modification of an existing ranking strategy. Our method can be incorporate into existing ranking strategies and does not require excessive additional computation. The results of experiments performed on a large-scale real dataset show that our method results in a significant improvement in terms of retrieval effectiveness.

Acknowledgement

This research was partly supported by the Grant-in-Aid for Scientific Research, Japan (#19300027).

References

1. Aditya, B., Bhalotia, G., Chakrabarti, S., Hulgeri, A., Nakhe, C., Parag, Sudarshan, S.: BANKS: Browsing and keyword searching in relational databases. In: VLDB. (2002) 1083–1086
2. Agrawal, S., Chaudhuri, S., Das, G.: DBXplorer: enabling keyword search over relational databases. In: ACM SIGMOD. (2002) 627
3. Hristidis, V., Gravano, L., Papakonstantinou, Y.: Efficient ir-style keyword search over relational databases. In: VLDB. (2003) 850–861
4. Liu, F., Yu, C., Meng, W., Chowdhury, A.: Effective keyword search in relational databases. In: ACM SIGMOD. (2006) 563–574
5. Luo, Y., Lin, X., Wang, W., Zhou, X.: SPARK: top-k keyword query in relational databases. In: ACM SIGMOD. (2007) 115–126
6. Luo, Y., Wang, W., Lin, X.: SPARK: A keyword search engine on relational databases. In: ICDE. (2008) 1552–1555
7. Li, G., Feng, J., Lin, F., Zhou, L.: Progressive ranking for efficient keyword search over relational databases. In: BNCOD. (2008) 193–197
8. Hristidis, V., Papakonstantinou, Y.: DISCOVER: Keyword search in relational databases. In: VLDB. (2002) 670–681
9. Wang, S., Zhang, K.: Searching databases with keywords. J. Comput. Sci. Technol. 20(1) (2005) 55–62
10. Kacholia, V., Pandit, S., Chakrabarti, S., Sudarshan, S., Desai, R., Karambelkar, H.: Bidirectional expansion for keyword search on graph databases. In: VLDB. (2005) 505–516
11. Li, G., Ooi, B.C., Feng, J., Wang, J., Zhou, L.: EASE: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In: ACM SIGMOD. (2008) 903–914
12. Sarda, N.L., Jain, A.: Mragyati: A system for keyword-based searching in databases. CoRR cs.DB/0110052 (2001)
13. Balmin, A., Hristidis, V., Papakonstantinou, Y.: ObjectRank: Authority-based keyword search in databases. In: VLDB. (2004) 564–575
14. Hristidis, V., Hwang, H., Papakonstantinou, Y.: Authority-based keyword search in databases. ACM Trans. Database Syst. 33(1) (2008) 1–40
15. Sayyadian, M., LeKhac, H., Doan, A., Gravano, L.: Efficient keyword search across heterogeneous relational databases. In: ICDE. (2007) 346–355