ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.9, September 2008, pp.2362−2375
DOI: 10.3724/SP.J.1001.2008.02362

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

关系数据库中支持语义的 Top-*K* 关键字搜索*

王 斌[1+], 杨晓春[1,2], 王国仁[1]

[1](东北大学 信息学院,辽宁 沈阳 110004)

[2](中国人民大学 数据工程与知识工程教育部重点实验室,北京 100872)

A Top-*K* Keyword Search for Supporting Semantics in Relational Databases

WANG Bin[1+], YANG Xiao-Chun[1,2], WANG Guo-Ren[1]

[1](College of Information Science and Engineering, Northeastern University, Shenyang 110004, China)

[2](Key Laboratory of Data Engineering and Knowledge Engineering for the Ministry of Education, Renmin University of China, Beijing 100872, China)

+ Corresponding author: E-mail: binwang@mail.neu.edu.cn

Wang B, Yang XC, Wang GR. A Top-*K* keyword search for supporting semantics in relational databases. *Journal of Software*, 2008,19(9):2362−2375. http://www.jos.org.cn/1000-9825/19/2362.htm

Abstract: In order to enhance the search results of keyword search in relational databases, semantic relationship among relations and tuples is employed and a semantic ranking function is proposed. In addition to considering current ranking principles, the proposed semantic ranking function provides new metrics to measure query relevance. Based on it, two Top-*k* search algorithms BA (blocking algorithm) and EBA (early-stopping blocking algorithm) are presented. EBA improves BA by providing a filtering threshold to terminate iterations as early as possible. Finally, experimental results show the semantic ranking function guarantees a search result with high precision and recall, and the proposed BA and EBA algorithms improve query performance of existing approaches.

Key words: Top-*K*; keyword search; relational databases; information retrieval; semantic similarity

摘 要: 为了增强关系数据库中的关键字搜索查询结果,考虑了多表之间以及元组之间的语义关系,提出了一种语义评分函数.该语义评分函数不仅涵盖了当前的评分思想,并且加入新指标来衡量查询结果与查询关键字之间的相关性.基于该评分函数,提出两种以数据块为处理单位的 Top-*K* 搜索算法,分别为 BA(blocking algorithm)算法和 EBA(early-stopping blocking algorithm)算法.EBA 在 BA 基础上引入了过滤域值,以便尽早终止算法的迭代次数.最后实验结果显示语义评分函数保证了搜索结果的高查准率和查全率,所提出的 BA 算法和 EBA 算法改善了现有方法的查询性能.

关键词: Top-*K*;关键字搜索;关系数据库;信息检索;语义相似度

* Supported by the Program for New Century Excellent Talents in University of China under Grant No.NCET-06-0290 (新世纪优秀人才计划); the National Natural Science Foundation of China under Grant No.60503036 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant Nos.2006AA09Z139, 2007AA01Z192 (国家高技术发展计划(863)); the Fok Ying Tong Education Foundation under Grant No.104027 (霍英东优选资助课题); the Key Laboratory of Data Engineering and Knowledge Engineering for the Ministry of Education, Renmin University of China (中国人民大学数据与知识工程教育部重点实验室开放课题)

Received 2007-08-30; Accepted 2007-11-02

中图法分类号: TP311 文献标识码: A

1 Introduction

Integration of IR and database technologies has been a hot research topic. One of the driving forces is the fact that more and more data is stored in relational databases[1,2]. Two advantages for integrating keyword search into relational databases are users need to neither understand the underlying database schemas and structures in advance nor complex query languages like SQL. Instead, users are only required to submit a list of keywords, and search engines will return ranked answers based on their relevance to query keywords.

However, due to the inherit nature of relational databases, information retrieval (IR) techniques in text databases cannot be straightforwardly applied to relational databases (DBs). Figure 1 shows an example of four relations: *author*, *writes*, *paper*, and *cites*. Relations are related with each other through reference constraints. For instance, *cites*→*paper* represents two foreign key constraints *cites*[Cited]⊆*paper*[Pid] and *cites*[Citing]⊆*paper*[Pid].

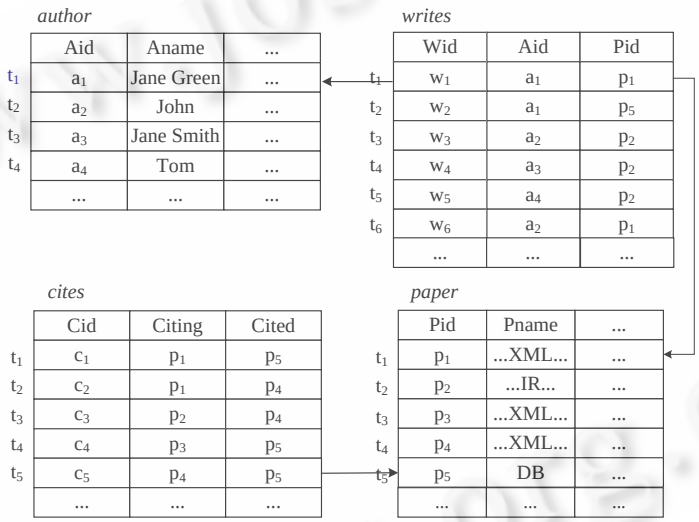


Fig.1 An example of tables in conference domain

Table 1 lists examples of three keyword queries and their results. Query Qry_1 contains two keywords "John" and "Tom." Assume the search engine returns three results R_{11}, R_{12}, and R_{13} to answer Qry_1. R_{11} is a tuple containing one of the keyword "John," and R_{12} is a tuple containing another keyword "Tom." Whereas, R_{13} is a **tuple tree**[3], in which tuples are related with each other through reference constraints and the non-free tuples cover all the query keywords ("John" and "Tom"). The size of the tuple tree R_{13} is the number of tuples in R_{13}, i.e. $|R_{13}|=5$. R_{11} and R_{12} can be regarded as tuple trees such that $|R_{11}|=|R_{12}|=1$.

1.1 Drawbacks of current ranking functions

Ranking functions and search algorithms are two core aspects in IR technologies. Current ranking functions in relational databases can be classified into two categories: tuple tree size based ranking function and IR-style relevance ranking function. Tuple tree size based ranking function[3−6] is a simple and straightforward ranking measurement, which roughly considers inverse proportion between the size of a tuple tree and the score that tuple tree gets. IR-style relevance ranking function[1,3,5,7] makes further exploration and incorporates into DB's ranking

field the relevance-ranking strategies developed by IR community over the years. Specifically, it treats each attribute text as a document, all attribute texts in a database as the document collection, and leverage state-of-the-art IR ranking functionality[8] to compute a tuple tree's relevance to query keywords.

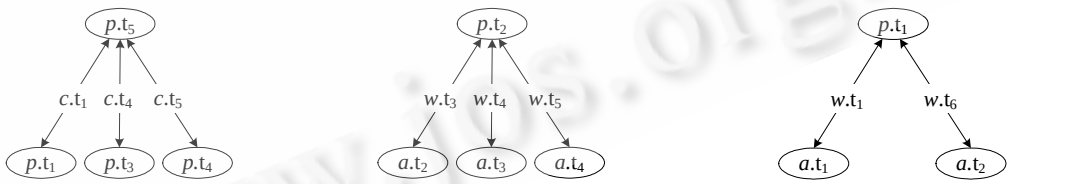
Table 1 Keyword search instances and their search results

Queries		Results	Size of results
	R_{11}:	*author*.t_2	1
Qry_1: {John, Tom}	R_{12}:	*author*.t_4	1
	R_{13}:	*author*.t_2 ← *writes*.t_3→ *paper*.t_2 ← *writes*.t_5→ *author*.t_4	5
	R_{21}:	*paper*.t_1	1
Qry_2: {XML}	R_{22}:	*paper*.t_3	1
	R_{23}:	*paper*.t_4	1
	R_{31}:	*author*.t_1	1
	R_{32}:	*author*.t_2	1
Qry_3: {Jane, John}	R_{33}:	*author*.t_3	1
	R_{34}:	*author*.t_1 ← *writes*.t_1→ *paper*.t_1 ← *writes*.t_6 → *author*.t_2	5
	R_{35}:	*author*.t_3 ← *writes*.t_4→ *paper*.t_2 ← *writes*.t_3 → *author*.t_2	5

Current search algorithms can also be classified to two categories: Candidate Networks (CN)-based search algorithm[1,3,4,6,7] and graph-based algorithm[5,11,12]. Both of the two types of algorithms start from tuples that contains partial or all keywords, discover shortest paths that could connect those tuples according to database schemas or pre-created data models, and finally return ranked joining networks of tuples (tuple trees) as answers.

However, the existing keywords search approaches[1,3,4,6,7,13−15] cannot capture semantic relevant to the query due to overlooking the following two cases.

Case (**1**) Indirect containment of query keywords. For instance, consider the query Qry_2 in Table 1. Tuple trees *paper*.t_1, *paper*.t_3, and *paper*.t_4 are returned as query answers because at least one of their respective attribute text contains query keyword "XML." The tuple *paper*.t_5 is not an answer since none of its attribute text contains "XML." However, relation *cites* in Fig.1 shows that paper *paper*.t_5 (with Pid=p_5) is cited by *paper*.t_1 (with Pid=p_1), *paper*.t_3 (with Pid=p_3), and *paper*.t_4 (with Pid=p_4), which means there is a high probability that the topic of *paper*.t_5 is related to Qry_2. Fig.2(a) shows the indirect containment relationship between *paper*.t_5 and {*paper*.t_1, *paper*.t_3, *paper*.t_4}, which describes the semantic correlation between *paper*.t_5 and the query Qry_2.



(a) Indirect containing Qry_2 (b) R_{35} partially matches an answer to Qry_3 (c) R_{34} completely matches an answer to Qry_3

Fig.2 Semantic relevance (*p* stands for relation *paper*, *c* for *cites*, *w* for *writes*, and *a* for *author* in Fig.1)

Case (**2**) Semantic correlation. The existing approaches do not distinguish the difference between partial matching and complete matching among non-free tuples. For example, consider the query Qry_3 in Table 1. The tuple trees R_{34} and R_{35} have the same size and contain both the query keyword {Jane, John}. Using the existing ranking function, R_{34} and R_{35} have the same scores. However, further investigation would reveal that the scores of R_{34} and R_{35} should be different. Figure 2(b) and (c) show the reason. The authors of *paper*.t_2 are *author*.t_2, *author*.t_3, and *author*.t_4 (shown in Fig.2(b)), in which, two of them construct R_{35} to answer query Qry_3. Figure 2(c) shows authors of *paper*.t_1, where all of them construct R_{34} to answer the query Qry_3. Apparently, the score of R_{34} should be higher than the score of R_{35}.

1.2 Our contributions

In this paper, a semantic ranking function is proposed to solve the above problems. It measures semantic relevance between tuple trees and query keywords. In addition, two Top-*k* search algorithms supporting the new ranking function namely BA (Blocking Algorithm) and EBA (Early-Stopping Blocking Algorithm) are proposed. BA and EBA process data in block, minimize database probes, and perform effectively as a result. Additionally, they can discover not only tuple trees that actually contain query keywords but those related with query keywords in a less obvious fashion. EBA improves BA by providing a filtering threshold to terminate iterations as early as possible. Our main contributions are as follows:

1) The concepts of semantic relevance as well as a novel ranking function to encompass this concept are proposed.
2) Two efficient algorithms namely BA and EBA in support of the new ranking function are presented.

2 Related Work

Keyword search in structured or semi-structured data has attracted a lot attention. DBExplorer[6], DISCOVER1[4], BANKS[5], DISCOVER2[3], and SPARK[1] support keyword search in relational databases[7] and return ranked tuple trees as answers[9]. The former three systems require answers to cover all query keywords while the latter ones[1,3,7] support searching out answers that partially contain query keywords. Current keyword search algorithms can be classified into two categories: Candidate Networks (CN)-based search algorithm and graph-based search algorithm. BANKS models a database into a tuple graph, where nodes denote tuples and edges represent key and foreign key constraints in the database. Based on the tuple graph, BANKS starts from tuples that actually contain query keywords, carries out heuristic search, and halt until it finds out a sub-graph that can connect all initial tuples containing query keywords. DBExplorer, DISCOVER1, DISCOVER2, and SPARK are divided into two steps: (1) determine appropriate CN-based on database schemas; and (2) evaluate CNs produced in the first step and sort the results in a descending order according to their respective ranking function. CN-based search algorithms use similar methods to fulfill the first step but diverge with respect to the second step. Specifically, DBExplorer directly uses SQL to evaluate CNs, obtains the results and return those results in a descending order. However, DBMSs are required to process a huge amount of data, therefore, jeopardize search performance. DISCOVER1 is an improvement of DBExplorer in that it stores some temporary data to avoid repeated evaluation of some joining networks of relations. DISCOVER2 and SPARK use different strategies such as Top-K or skyline to further improve search performance. When it comes to ranking functions, DBExplorer and DISCOVER1 explore the structure of an answer and favor tuple trees of small size over those with a large size. BANKS measures a tuple tree's relevance in terms of two aspects: the weight of each tuple member (similar to Google's PageRank), and the weight of each edge member. BANKS, DBExplorer, and DISCOVER1 do not leverage state-of-the-art IR ranking methods. Reference [7] and DISCOVER2 use IR-style relevance ranking methods to compute relevance. Specifically, they treat each attribute text as a document, and each column text as a document collection, and then apply the classic equation tf×idf or its variants to score research results. SPARK proposes similar ranking function. Different from the above methods, it treats each tuple tree produced by CN as a document. Therefore, all possible tuple trees are regarded as a document collection. Those ranking functions share the same ground that if and only if a tuple actually contains some query keyword, can it be called relevant to the query keyword. Kaushik *et al*. in Ref.[15] extend the term "relevance" to another dimension: if a tuple is referenced by another which actually contain a query keyword, the referenced tuple is relevant to the query keyword[10]. However, they do not consider tuples that actually contain query keywords as relevant[10].

3 Problem Definition

Given a set of keywords $Q=\{w_1,w_2,\ldots,w_n\}$, design a ranking function *score*, so that *score* considers not only tuples that actually contain keywords but also those semantically contain keywords in Q. Based on the ranking function *score*, determine k results $R(Q,k)$. For any tuple tree $T\in R(Q,k)$, there does not exist a tuple tree $T'\in R(Q,k)$ such that the score of T is less than the score of T'. This paper bases on the following two assumptions:

(1) For any keyword w_i, all tuples actually containing w_i can be achieved in a descending order w.r.t. a given ranking function *score*.

(2) Tuple trees that contain partial query keywords in Q are also useful.

Definition 1. (Directly containing query keywords) Given a tuple tree T. Let t be a tuple in T, $A(t)$ be the set of attributes of t, and w be a query keyword. T directly contains the query keyword w, if for any tuple t in T, $\exists\ a\in A(t)$ and the value of t for attribute a, denoted $t[a]$, contains w.

Definition 2. (Indirectly containing query keywords) Given two tuple trees T and T'. Let w_i be a query keyword. T indirectly contains the query keyword w_i, if for any $t'\in T'$, $\exists t\in T$, such that $t'\rightarrow t$.

Definition 3. (Semantic relevance) Given a tuple t and a query keyword w. The tuple t is semantically relevant to w, if t directly contains w or indirectly contains w.

For the sake of simplicity, Table 2 displays notations and their descriptions used in the rest part of the paper.

Table 2 Notations used in this paper

Notations	Descriptions
$Q=\{w_1, w_2, \ldots, w_n\}$	A query Q containing a set of keywords $\{w_1, w_2, \ldots, w_n\}$
$R(Q, k)$	Results to the top-k query Q
$A(t)=\{a_1, a_2, \ldots, a_m\}$	The set of attributes of a tuple t
$score_D(t, w_i)$	The direct contribution ratio for a tuple t to the query keyword w_i (will be discussed in Section 4.1.1)
$score_I(t, w_i)$	The indirect contribution ratio for a tuple t to query keyword w_i (will be discussed in Section 4.1.2)
$score(t, Q)$	The overall contribution ratio for a tuple t to a set of query keywords $\{w_1, w_2, \ldots, w_n\}$
$score_s(T, Q)$	The semantic similarity between a tuple tree T and query keywords Q

4 Semantic Ranking Function

Our semantic ranking function considers both cases of directly and indirectly containing query keywords, use *direct contribution ratio* and *indirect contribution ratio*, respectively, to quantify the relevance between a tuple t and a query keyword w. We show how to quantify the contribution ratios in Section 4.1 and how to compute the semantic correlation in Section 4.2. Based on the discussions in Section 4.1 and Section 4.2, we propose a semantic ranking function in Section 4.3.

4.1 Containment relationship between query and tuples

4.1.1 Direct contribution ratio

Direct contribution ratio measures the degree of which that a tuple tree directly contains a set of query keywords. In this paper, we adopt the ranking method in DISCOVER2[5] to compute direct contribution ratio.

Let t be a tuple, $Q=\{w_1,w_2,\ldots,w_n\}$ be a query, and $A(t)=\{a_1,a_2,\ldots,a_m\}$ be the attributes of t. Let t directly contains a keyword $w\in Q$. The direct contribution ratio for an attribute value $t[a_i]$ to w is given in Eq.(1).

$$score_D(t[a_i],w)=\frac{1+\ln(1+\ln tf)}{(1-\alpha)+\alpha\times\frac{len(t[a_i])}{avg\_len}}\cdot\ln\frac{N+1}{df} \tag{1}$$

where, tf is the frequency of w in $t[a_i]$, df is the number of tuples in a_i's relation with word w in this attribute, N is the total number of tuples in a_i's relation, $len(t[a_i])$ is the size of $t[a_i]$, avg_len is the average attribute-value size, and α is a parameter with a range of [0, 1].

Therefore, the direct contribution for t to w is given in Eq.(2).

$$score_D(t,w)=\sum_{a\in A(t)} score_D(t[a],w) \tag{2}$$

Accordingly, the direct contribution ratio for t to the query Q is shown in Eq.(3).

$$score_D(t,Q)=\sum_{w\in Q} score_D(t,w) \tag{3}$$

4.1.2 Indirect contribution ratio

Indirect contribution ratio measures the degree of which a tuple tree indirectly contains a set of query keywords in Q. Considering the scenario that a tuple t is referenced by other tuples directly containing the query Q. Tuple t is also relevant to Q.

In order to clearly describing how t is related to other tuples using foreign-key constraints, we employ a matrix M. We use $S(t)$ to represent a tuple set $\{t_1,t_2,\ldots,t_h\}$, such that for each $t_i\in S(t)$, $t[key]=t_i[fkey]$ and t_i directly contains query keywords in Q, where *key* is the key attribute of t, and *fkey* is the foreign key attribute of t_i.

$$M=\begin{bmatrix} m_{11} & \cdots & m_{1(n-1)} & m_{1n} \\ \cdots & \cdots & \cdots & \cdots \\ m_{(h-1)1} & \cdots & m_{(h-1)(n-1)} & m_{(h-1)n} \\ m_{h1} & \cdots & m_{h(n-1)} & m_{hn} \end{bmatrix}$$

$$m_{ij}=\begin{cases} score_D(t_i,w_j), & t_i \text{ directly contains } w_j, \\ 0, & t_i \text{ does not directly contain } w_j. \end{cases} \tag{4}$$

The row of M denotes a query $Q=\{w_1,w_2,\ldots,w_n\}$, the column of M denotes $S(t_i)$, and m_{ij} represents the direct contribution ratio for t_i to w_j. When t_i does not directly contain w_j, $m_{ij}=0$.

First, consider the indirect contribution ratio for t_i to a single keyword w. We obtain the indirect contribution ratio $score_I(t_i, w)$ by using Eq.(5).

$$score_I(t_i,w)=\frac{\sum_{t\in S(t_i)} score_D(t,w)^2}{\sum_{t\in S(t_i)} score_D(t,w)} \tag{5}$$

The indirect contribution ratio defined in Eq.(5) satisfies the following two properties:

(1) Given a tuple t_i and a query keyword w, the larger size of $S(t_i)$ is, the larger value of $score_I(t_i, w)$ should be;

(2) Given two tuples t_1 and t_2. Let t_1' and t_2' be tuples such that $t_1'\in S(t_1)$ and $t_2'\in S(t_2)$. If $score_D(t_1',w)\geq score_D(t_2',w)$, then $score_I(t_1,w)\geq score_I(t_2,w)$.

When it comes to multi-term keyword search, $Q=\{w_1,w_2,\ldots,w_n\}$ say, the indirect contribution ratio for t to Q is shown in Eq.(6).

$$score_I(t,Q)=\begin{cases} \sum_{w\in Q} score_I(t,w), & \text{if } t \text{ indirectly contains a keyword in } Q, \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

Lemma 1. Given a tuple t and its related tuple set $S(t)=\{t_1, t_2, \ldots, t_h\}$. Let w be a query keyword, and $score_D(t_1,w)\geq score_D(t_2,w)\geq\ldots\geq score_D(t_h,w)$, then $score_I(t,w)\leq score_D(t_1,w)$.

Proof: By Eq.(5), we need prove $\sum_{i=1}^{h} score_D(t_i,w)^2\leq score_D(t_1,w)\cdot\sum_{i=1}^{h} score_D(t_i,w)$, which can be rewritten

as $\sum_{i=2}^{h} t_i \cdot (t_i - t_1) \le 0$. Since $t_i \le t_1 (1 \le i \le h)$, we conclude $score_I(t,w) \le score_D(t_1,w)$. □

4.1.3 Overall contribution ratio for a tuple tree to query keywords

As mentioned before, Eq.(3) computes the direct contribution ratio for t to Q and Eq.(6) computes the indirect contribution ratio for t to Q. We use Eq.(7) to combine the direct and indirect contribution ratios to quantify the overall contribution ratio for t to Q, denoted $score(t, Q)$.

$$score(t,Q) = (1-\theta) \cdot score_D(t,Q) + \theta \cdot score_I(t,Q) \tag{7}$$

where, θ is a coefficient to balance the two contribution ratios, $\theta < 0.5$. When θ equals 0, we do not consider the indirect contribution ratio.

Example. Consider the query Qry_2 and its results in Table 1. The direction contribution ratio for $paper.t_5$ to "XML" is $score_D(paper.t_5$, "XML")=0, since it does not directly contains "XML." However, $paper.t_5$ is cited by $paper.t_1$, $paper.t_3$, and $paper.t_4$, which directly contain "XML," therefore, the indirection contribution ratio for $paper.t_5$ is a non-zero value. Suppose $score_D(paper.t_1$,"XML") = 0.9, $score_D(paper.t_3$, "XML") = 0.7, $score_D(paper.t_4$, "XML") = 0.2, and θ=0.4, then the indirect contribution $score_I(paper.t_5$, "XML") $= \frac{0.9^2 + 0.7^2 + 0.2^2}{0.9 + 0.7 + 0.2} =$ 0.744, and the overall contribution ratio $score(paper.t_5$, "XML")=(1–0.4)×0+0.4×0.74=0.296.

The overall contribution ratio for a tuple tree T to a query Q is defined in Equation(8).

$$score(T,Q) = \sum_{t \in T} score(t,Q) \tag{8}$$

4.2 Semantic correlation between query and tuples

In this subsection, we take care of Case (2) discussed in Section 1.1. Consider a two-keyword query $Q = \{w_1, w_2\}$. Suppose a tuple tree T: $t_1 \leftarrow t \rightarrow t_2$, where t_1, t_2, and t are tuples of R_1, R_2, and R, respectively, $t_1 \in \delta_{a_1=w_1}(R_1)$, $t_2 \in \delta_{a_2=w_2}(R_2)$ and $t \in R$. Let S_1 and S_2 be subsets of R where all tuples connect with t_1 and t_2 through foreign key constraints among R, R_1, and R_2, respectively.

$$S_1 = \{t \mid t \in \pi_{A(R)}(\delta_{a_1=w_1}(R_1) \infty R)\},\ S_2 = \{t \mid t \in \pi_{A(R)}(\delta_{a_2=w_2}(R_2) \infty R)\} \tag{9}$$

Here, $A(R)$ denotes the set of all attributes of relation R. The intersection of S_1 and S_2 denotes tuples connecting with both t_1 and t_2 through foreign key constraints. For instance, a_1 and a_2 are coauthors to a paper, then S_1 means all papers written by a_1, S_2 means all papers written by a_2, and $S_1 \bigcap S_2$ is a collection of the papers in which both a_1 and a_2 join. The *semantic correlation* between tuples in the tuple tree T w.r.t. Q can be expressed using the similarity between the two sets S_1 and S_2, i.e. $\frac{S_1 \bigcap S_2}{S_1 \bigcup S_2}$. Futhermore, for a multi-keywords search $Q = \{w_1, w_2, \ldots, w_n\}$, let S_i be $S_i = \{t \mid t \in \pi_{A(R)}(\delta_{a_i=w_i}(R_i) \infty R)\}$, then the *semantic correlation* between tuples in T w.r.t. Q is shown in Eq.(10).

$$correlation(T_1,Q) = \frac{\bigcap_{i=1}^{n} S_i}{\bigcup_{i=1}^{n} S_i} \tag{10}$$

For instance, the semantic correlation between tuples in R_{34} w.r.t. the Qry_3 in Table1 is *correlation* (R_{34}, "Jane, John")=1/(1+2)=1/3, *correlation* (R_{35}, "Jane, John")=1/2, therefore, we conclude that $author.t_2$ cooperates more closely with $author.t_3$ than with $author.t_1$.

4.3 Semantic ranking function

As discussed above, semantic ranking function should take into consideration three aspects: (1) the total contribution ratio for a single tuple to keywords, (2) semantic correlation between non-free tuples, and (3) the size

of the tuple tree. Therefore, the semantic ranking function is defined in Equation (11), where $|T|$ is the number of tuples in T.

$$score_s(T,Q) = score(T,Q) \cdot \frac{correlation(T,Q)}{|T|} \tag{11}$$

5 Semantic-Based Search Algorithms

Our semantic-based search algorithm (SSA) is based on that of the DISCOVER2 systems, that is, it first creates the *tuple set graph* from the pre-defined *database schema graph* and the tuple sets returned by the IR Engine module. Figure 3 shows an example of the tuple set graph derived from relations in Fig.1.

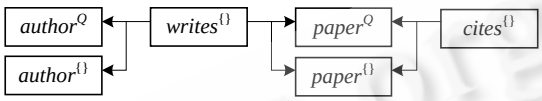


Fig.3 An example of tuple set graph

SSA first generates a set of candidate CNs. Each CN consists of the non-free tupes sets to the query, e.g. $author^Q$ and $paper^Q$ in Fig.3. Differently from the existing approaches, SSA progressively adds a CN by expanding non-free tuples by using a tuple set adjacent to the CN in the tuple set graph. For instance, when a keyword in Q is "XML," it firstly identifies "XML" in the relation *paper*, then it expands all terms that directly contains the keyword "XML." SSA uses *paper*'s adjacent relation *cites* to do the expansion. Since all papers in the relation *paper* cite *paper*.t5, SSA expands values of non-free tuples to "DB" (the title of *paper*.t5) to construct a new CN.

In this section, we first propose a pre-processing approach to extending non-free tuples in CNs that indirectly contain query keywords, we then discuss that the proposed semantic ranking function satisfies the tuple monotonicity property[5], so that the existing approach can be employed by using the ranking function. We finally present two improved algorithms in Section 5.3.

5.1 Expanding Non-Free tuples in CNs

Given a query keyword w, the search algorithm expands w to a term set indirectly containing w. Firstly, the algorithm identifies relation R that contains the query keyword w, then it finds relations S_1, S_2, …, and S_m that have foreign key constraints with R, i.e. $S_i.a_i \subseteq S_m.a_m (1 \le i < m)$, the attribute $R.key$ has the same domain with the attributes $S_1.a_1$ and $S_m.a_m$. Equation (12) shows the set of terms $T_s(R,w)$ that indirectly contain w in relation R.

$$T_s(R,w) = R \underset{R.key=S_m.a_m}{\infty} (\pi_{S_m.a_m}(\delta_w(R) \underset{R.key=S_1.a_1}{\infty} S_1 \infty ... \underset{S_{m-1}.a_{m-1}=S_m.a_m}{\infty} S_m)) \tag{12}$$

Different from the existing approaches, non-free tuples in CNs can be terms in either Q or $T_s(R,w)$. For example, in Fig.1, papers with ids p_1, p_3, and p_4 cite paper p_5 in relation *cites*, thus $T_s(R,$ "XML")= {*paper*.t_5}. Given a query Q={Jane, XML}, under indirect containment semantic, we can get two results: R_1= *author*.t_1← *writes*.t_1→ *paper*.t_1 and R_2= *author*.t_1← *writes*.t_2→ *paper*.t_5, while the existing approaches can only get one result R_1.

5.2 Tuple monotonicity

A naive algorithm to get semantic top-k results includes the following two steps: (i) calculate candidate CNs using non-free tuples $R \cup T_s(R,w_i)$, and (ii) for each candidate tuple tree T, calculate $score_s(T,Q)$ using Equation (11) and choose k tuple trees with largest ranking values. Obviously, the naive approach calculates many tuple trees that do not belong to the top-k results. Literature[3] uses sparse algorithm to calculate top-k results. In this section, we first prove that the sparse algorithm[3] can also be employed under our semantic-based scenarios. That is, semantic-based ranking function satisfies *tuple monotonicity* property[3].

Theorem 1. Given a CN, the semantic ranking function $score(T,Q)$ satisfies the tuple monotonicity property, if for every query Q and joining trees of tuples T and T' derived from the same CN such that (i) T consists of tuples $t_1,...,t_m$ while T' consists of tuples $t'_1,...,t'_m$, and (ii) $score_D(t_i,Q) \geq score_D(t'_i,Q)$ for all i, it follows that $score(T,Q) \geq score(T',Q)$.

Proof: We consider the cases that the non-free tuples in T and T' directly and indirectly contain the query keywords in Q respectively.

(1) We start from the non-free tuple set $R_1^Q,\ldots, R_m^Q$. Suppose that $t_1,...,t_m$ are non-free tuples in the tuple tree T that directly contain $w_1,...,w_m (w_i \in Q)$, respectively. If non-free tuples in T' also directly contain keywords in Q, then it follows that the same observation with[4,5], i.e. $score(T,Q) \geq score(T',Q)$ holds. If non-free tuples in T' indirectly contain keywords in Q, then we prove for each i, $(1-\theta)\cdot score_D(t_i,Q)+\theta\cdot score_I(t_i,Q) \geq (1-\theta)\cdot score_D(t'_i,Q)+\theta\cdot score_I(t'_i,Q)$.

According to Equation (5), $score(t'_i,w)=(1-\theta)\cdot score_D(t'_i,w)+\theta\cdot\dfrac{\sum_{t_i\in S(t'_i)} score_D(t,w)^2}{\sum_{t_i\in S(t'_i)} score_D(t,w)}$.

From Lemma 1, we get $score(t'_i,w) \leq (1-\theta)\cdot score_D(t'_i,w)+\theta\cdot score_D(t_i,w)$.

Now we prove $(1-\theta)\cdot score_D(t'_i,w)+\theta\cdot score_D(t_i,w) \leq (1-\theta)\cdot score_D(t_1,w)$, i.e.

$$\theta \leq \frac{score_D(t_i,w)-score_D(t'_i,w)}{2score_D(t_i,w)-score_D(t'_i,w)}.$$

Furthermore, $\theta \leq \dfrac{score_D(t_i,w)-score_D(t'_i,w)}{2(score_D(t_i,w)-score_D(t'_i,w))+score_D(t'_i,w)} < 0.5$. It is consistent with the definition of the score function in Equation (7), i.e. $score(T,Q) \geq score(T',Q)$ holds.

(2) Suppose that $t_1,...,t_m$ are non-free tuples in the tuple tree T that indirectly contain $w_1,...,w_m (w_i \in Q)$, respectively. Obviously, when we construct a CN using terms that indirectly contain the query keywords, the algorithm SSA already calculated the score value $score(T,Q)$ in the current CN. That is, it follows the same observation with[4,5], i.e. $score(T,Q) \geq score(T',Q)$ holds. □

Theorem 2. Given a set of CNs. Let C be a CN such that its $score_s(T,Q)$ is the k-th largest score. For any CN C', if its $score(T',Q)$ is less than $score_s(T,Q)$, then the remaining tuple trees in C' are not candidates.

Proof: From Equations (10) and (11), we know $score_S(T,Q) \leq score(T,Q)$ So, if a tuple tree T' whose $score(T',Q)$ is less than $score_s(T,Q)$, then its semantic score $score(T',Q)$ must be less than $score_s(T,Q)$. Since T is the k-th closest CN to answer the query Q, the tuple tree T' can be safely pruned. □

5.3 Semantic Top-k algorithm

For each CN, the existing approaches probe the database to evaluate tuples one by one and generate k candidate tuple trees. The final top-k results are chosen from the candidate tuples trees of all CNs. In order to improve query performance in a single CN, the tuples can be grouped into blocks to save the times of probing the database. Two algorithms namely BA (Block Algorithm) and EBA (Early-stopping Block Algorithm) are proposed in this paper.

5.3.1 Block algorithm (BA)

Given a CN C. For each query keyword w_i, we rank tuples in $R \cup T_S(R,w_i)$ in descending order. In order to avoid frequently probing database, we divide tuples in $R \cup T_S(R,w_i)$ into several equal-sized data blocks. In each iteration step, a data block, which contains tuples with highest scores are fetched from $R \cup T_S(R,w_i)$ to generate candidate k tuple trees with k largest semantic scores.

Figure 4 shows the BA algorithm. BA fetches a data block B from each R_i that directly contains the query keyword w_i. It obtains tuples $T_S(B,w_i)$ that indirectly contain w_i by joining with other relations in the tuple set graph. A temporary relation S_{Ti} is used for storing all the tuples that provide either direct contribution ratio or indirect contribution ratio to w_i. S_{Ti} is defined as a triple $\langle$id, $score_D$, $score_I\rangle$, where id is the identifier attribute of tuples in B, $score_D$ records the direct contribution ratio to w_i, and $score_I$ records the indirect contribution ratio to w_i. S_{Ti} ranks its tuples according to the descending order of $score(t,\{w_i\})$ shown in Eq.(7), where t is a tuple in $B\cup T_S(B,w_i)$. BA then invokes the function GenCandidate() to generate candidate tuple trees.

Algorithm 1. BA
Input: Query $Q=\{w_1, w_2, \ldots, w_n\}$; tuple set graph G; a CN C; k;
Output: A queue containing top-k results *Res* in C;

```
1.  FOR (each w_i in Q)  {                    //find all tuples that directly or indirectly contain w_i
2.      B = next block from R_i;
3.      Calculate T_s(B, w_i) using Equation (12);
4.      Order tuples in T_s(B_i,w_i) in descending order using their semantic ranking scores;
5.      Generate S_Ti such that tuples in it are ranked in descending order according to score(t, {w_i});
    }
6.  Res = GenCandidate(Q, G, C, k, S_T1,..., S_Tn);
7.  Return Res;
```

Fig.4 BA algorithm

Function: **GenCandidate**
Input: Relations $R, S_1, \ldots, S_m$; query $Q=\{w_1, w_2, \ldots, w_n\}$; a CN C; k;
Output: A queue containing top-k results *Res* in C;

```
1.  Res = ∅;
2.  FOR (each w_i in Q)
3.      checked_i = ∅;
4.  WHILE (|Res| < k) {
5.      checked_i = checked_i ∪ S_Ti;        // checked_i is a set of checked tuples that contain w_i
6.      Construct a tuple tree T such that its non-free tuple contains tuples in checked_i;
7.      Res.push_back(T);
    }
8.  Return Res;
```

Fig.5 GenCandidate() function in BA algorithm

For each CN, the function GenCandidate() joins adjacent relations with R_i in the tuple set graph, and calculates the semantic ranking function $score_s(T,Q)$ for each tuple tree T. The tuples trees with the first k largest semantic scores are candidate tuple trees.

5.3.2 Early-Stopping block algorithm (EBA)

It is unnecessary to generate all k results for each CN. EBA enhances the BA algorithm by providing a filtering threshold to prune non-candidates. At the beginning of the search algorithm, the threshold is set to be 0. During the processing of all CNS, EBA always keeps the k-th maximal possible score value as a filtering threshold. Only a candidate tuple tree whose score larger than the threshold, it can be returned as a candidate.

Figure 6 shows the GenCandidate() function in EBA algorithm. Differ from the function in Fig.5, EBA stores the filtering threshold. In Line 4, EBA generates a candidate only when there is no enough results generated and the semantic score is larger than the filtering threshold T^k. The value of T^k increases when more tuples in CNs are processed, which saves more iterative steps.

For example, given 5 CNs. BA algorithm chooses 5 tuple trees within each CN and chooses the top-5 results among these candidate CNs. EBA also chooses 5 tuple trees within the first processing CN. Then, it uses the semantic score of the 5th tuple tree as the filtering threshold T^k. When processing the second CN, only a tuple tree whose score is larger than T^k and is ranked within top-5 tuple trees can be regarded as candidate. Therefore, the filtering threshold increases when more CNs are processed.

```
Function: GenCandidate
Input:  Relations R, S1, …, Sm; query Q = {w1, w2, …, wn}; filtering threshold T^k; a CN C;
Output: A queue containing top-k results Res in C;
…
4.   max_score = 0;   flag = TRUE;
5.   WHILE (|Res| < k && flag) {
6.       checked_i = checked_i ∪ S_Ti;          // checked_i is a set of checked tuples that contain w_i
7.       Construct a tuple tree T such that its non-free tuple contains tuples in checked_i;
8.       max_score = score_S(T,Q);              // calculate semantic ranking function as the maximal possible score
9.       IF (max_score > T^k) {
10.          Res.push_back(T);
11.          T^k = the (k–1)-th semantic score in the candidate tuple trees;
         }
         ELSE flag = false;
     }
11.  Return Res;
```

Fig.6 GenCandidate() function in EBA algorithm

6 Experimental Results

In order to evaluate the effectiveness of the proposed semantic ranking function and the efficiency of BA and EBA algorithms, we have conducted extensive experiments on large-scale real datasets.

We used two real data sets. The first one was about course information of University of Washington, download from the data archive in University of Illinois. The second data set was from DBLP. All the algorithms were implemented using JDK 1.5 and JDBC to connect database Oracle 9i. The experiments were run on a PC with an Intel Pentium 3.0 GHz CPU and 512M memory with a 160GB disk, running a Windows XP operating system.

Experiments are conducted to test two aspects: (i) quality of search results, and (ii) performance of the search algorithms. In the remaining of the paper, search time includes time cost to generate CNs (Candidate Networks) without considering the time of obtaining tuples directly containing w_i. The primary reason of ignoring the time of fetching tuples from DBMS is that we can employ the DBMS search engine to do exactly search and get all tuples directly contain the keyword w_i. In this paper, we focus on the following orthogonal issues: expanding terms that indirectly contain w_i, choosing candidate CNs, and using filtering threshold to stop iterations in a single CN.

We manually constructed two sets of queries (Q_1,Q_2,…,Q_{20}) for DBLP dataset and the third set of queries (Q_{21}, Q_{22},…,Q_{30}) for courses dataset. The first set of queries include ten queries (Q_1,Q_2,…,Q_{10}), which involve single CNs that derived from the DBLP tuple set graph, while the second set of queries include another ten queries (Q_{11},Q_{12},…, Q_{20}), which involve multiple CNs, i.e. a wide variety of keywords and their combinations. The queries (Q_{21},Q_{22},…, Q_{30}) were involved multiple CNs derived from the courses tuple set graph. We implemented the global pipeline (GP) algorithm[3] to generate top-k results without considering semantic correlation, and compared BA and EBA algorithms with the GP algorithm. We manually determined 150 tuples as one block for DBLP and 200 tuples as one block for course according to their different data distributions.

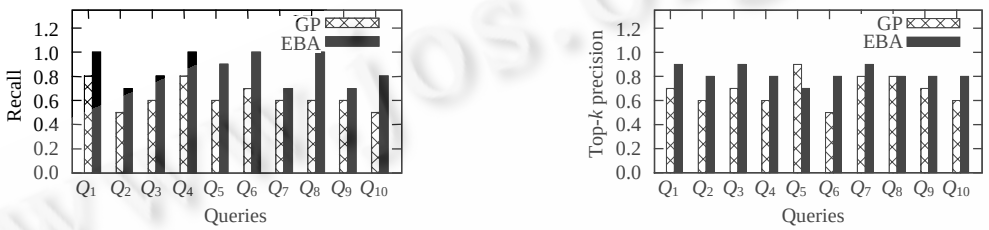
6.1 Effective of semantic ranking function

In order to test the effectiveness of the proposed semantic ranking function, we used *recall* and *top-k precision* to do the evaluation. *Recall* is a ratio of the number of relevant tuple trees searched over the overall number of relevant tuple trees in the database. Recall=1 means search algorithm can successfully retrieve all relevant tuple trees. *Top-k precision* is a ratio of the number of returned results that are among tuple trees of top-k highest scores over k results.

We compared the recall and top-10 precision between GP and EBA algorithms using queries Q_1, Q_2,…, and Q_{10} (The test results of queries Q_{11}, Q_{12},…, and Q_{20} are similar). We used 0.2 as the coefficient θ. Figure 7(a) shows the recall values of GP and EBA, respectively. Since BA and EBA always have the same recall values, the recall of

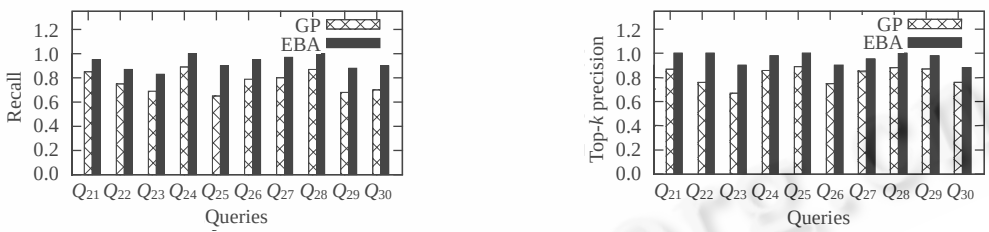
BA is not plotted. As the figure shows, the recall values of EBA values are all exceed 0.7, and among which the recall values of four queries reached 1. On the other hand, the highest recall value of GP was 0.8. Therefore, EBA can search more relevant results. The reason of achieving high recalls is that the proposed semantic ranking function can discover answers that indirectly contain query keywords and considers the semantic correlation, while GA only gets answers directly containing query keywords.

Figure 7(b) shows the top-10 precisions of GP and EBA. 90% of top-*k* precisions of EBA were higher than GA. Using EBA, 90% queries achieved more than 0.8 top-*k* precisions, while using GP, only 30% queries exceeded 0.8 top-*k* precisions. The results show that GP determines the top-*k* results merely based on their direct contribution ratio, therefore, it runs the risk of missing answers with lower direct contribution ratios but higher indirect contribution ratios. Instead, EBA strikes a balance between the direct contribution ratios, indirect contribution ratios, and semantic similarity. It determines the relevance of answers to queries more accurately and comprehensively. Figure 8(a) and (b) show the similar results on courses dataset.



(a) Comparison of recalls (b) Comparison of top-10 precisions

Fig.7 Recalls and top-10 precisions on DBLP dataset

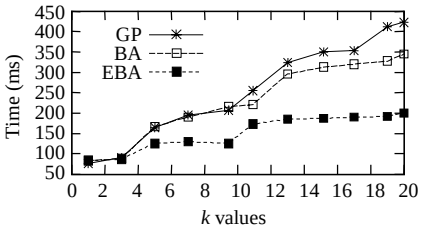


(a) Comparison of recalls (b) Comparison of top-10 precisions

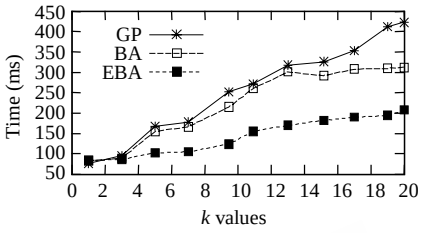
Fig.8 Recalls and top-10 precisions on courses dataset

6.2 Effect of *k* values on query performance

Figure 9 shows the execution time when varying *k* values from 1 to 20. We choose query Q_7 to test the execution time of different top-*k* queries. Figure 9(a) shows the running time of query Q_7. When *k* was small, GP performed quicker than BA and EBA. The reason is that GP only needs small number of iterations to get tuple trees with highest scores, whereas BA and EBA accessed one data block at each iteration, which produces much more number of tuple trees that cannot answer the query. However, when *k* increased, GA needed more iterations than BA and EBA, which result in more frequent database probes. Furthermore, EBA and BA required the same running time when *k* was small (e.g. *k*=1) for additional computations. As *k* increased, EBA performed better than BA and GP algorithms, since EBA used filtering threshold to saves more iterative steps. Figure 9(b) shows average running time of queries Q_1,Q_2,…, and Q_{10} when varying *k* from 1 to 20. The test results were similar with the result of Q_7.



(a) Effect on k on query time (for Q_7)

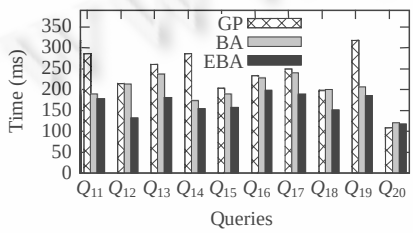


(b) Effect on k on average query time (for Q_1, Q_2,…, Q_{10})

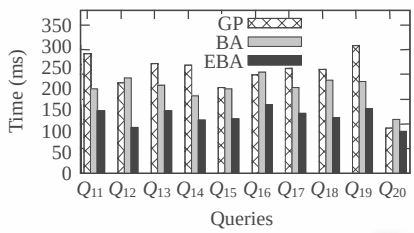
Fig.9 Effect of K on execution time

6.3 The effect of queries on query performance

Figure 10 shows how the effect of different queries (Q_{11},Q_{12},…,Q_{20}) on the running time. Figure 10(a) is the running time of top-5 queries and Fig.10(b) is the running time of top-15 queries. We can see that EBA performs better than BA and GP for all the queries that involved in multiple CNs. EBA retrieved all desirable top-5 results within 200ms and top-15 results within 300ms, compared with other algorithms, EBA sustained a high query performance. Different queries led to generate various CNs with different sizes as well as time complexity, which plays a major role in the overhead of a single database probe. That is, more complex and larger a CN is, more overhead it needs for a database probe.



(a) Top-5 query



(b) Top-15 query

Fig.10 Effect of queries (Q_{11},Q_{12},…,Q_{20}) on execution time

7 Conclusions

In this paper, we studied semantics-based Top-k keyword search over relational databases. We proposed a novel semantic ranking function, which not only adapts the state-of-the-art IR ranking function and more importantly, it encompasses semantic features. We also studied search methods tailored to support our ranking function. Two Top-k algorithms namely BA and EBA are proposed which process data in block, minimize database probes and can more comprehensively and effectively search out relevant results. We have conducted extensive experiments on large-scale databases. The experimental results show that the semantic ranking function is adequate and proposed algorithms are effective and efficient.

References:

[1] Luo Y, Lin XM, Wang W, Zhou XF. SPARK: Top-k keyword query in relational databases. In: Chan CY, Ooi BC, Zhou A, eds. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. Beijing: ACM, 2007. 115−126.

[2] Meng X, Zhou L, Wang S. The state-of-the-art and trends in database research. Journal of Software, 2004,15(12):1822−1836 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/15/1822.htm

[3] Hristidis V, Gravano L, Papakonstantinou Y. Efficient IR-style keyword search over relational databases. In: Freytag JC, Lockemann PC, Abiteboul S, Carey MJ, Selinger PG, Heuer A, eds. Proc. of the 29th Int'l Conf. on Very Large Data Bases. Berlin: Morgan Kaufmann Publishers, 2003. 850−861.

[4] Hristidis V, Papakonstantinou Y. DISCOVER: Keyword search in relational databases. In: Proc. of the 28th Int'l Conf. on Very Large Data Bases. Hong Kong: Morgan Kaufmann Publishers, 2002. 670−681.

[5] Bhalotia G, Hulgeri A, Nakhe C, Chakrabarti S, Sudarshan S. Keyword searching and browsing in databases using BANKS. In: Proc. of the 18th Int'l Conf. on Data Engineering. San Jose, CA. IEEE Computer Society, 2002. 431−440.

[6] Agrawal S, Chaudhuri S, Das G. DBXplorer: A system for keyword-based search over relational databases. In: Proc. of the 18th Int'l Conf. on Data Engineering. San Jose: IEEE Computer Society, 2002. 5−16.

[7] Liu F, Yu CT, Meng W, Chowdhury A. Effective keyword search in relational databases. In: Chaudhuri S, Hristidis V, Polyzotis N, eds. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. Chicago: ACM, 2006. 563−574.

[8] Singhal A. Modern information retrieval: A brief overview. IEEE Data Engineering Bulletin (Special Issue on Text and Databases), 2001,24(4):35−43.

[9] Li X, Meng W, Meng X: EasyQuerier: A keyword query interface for Web database integration system. In: Ramamohanarao K, Krishna PR, Mohania MK, Nantajeewarawat E, eds. Advances in Databases: Concepts, Systems and Applications, Proc. of the 12th Int'l Conf. on Database Systems for Advanced Applications, DASFAA 2007. LNCS 4443, Bangkok: Springer-Verlag, 2007. 936−942.

[10] Chakrabarti K, Ganti V, Han JW, Xin D. Ranking objects by exploiting relationships: Computing top-k over aggregation. In: Chaudhuri S, Hristidis V, Polyzotis N, eds. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. Chicago: ACM, 2006. 371−382.

[11] Ding B, Yu JX, Wang S, Qin L, Zhang X, Lin X. Finding top-k min-cost connected trees in databases. In: Proc. of the 23rd Int'l Conf. on Data Engineering. Istanbul: IEEE Computer Society, 2007. 836−845.

[12] Kacholica V, Pandit S, Chakrabarti S, Sudarshan S, Desai R, Karambelkar H. Bidirectional expansion for keyword search on graph databases. In: Böhm K, Jensen CS, Haas LM, Kersten ML, Larson P, Ooi BC, eds. Proc. of the 31st Int'l Conf. on Very Large Data Bases. Trondheim, Norway, 2005. 505−516.

[13] Balmin A, Hristidis V, Papakonstantinou Y. ObjectRank: Authority-Based keyword search in databases. In: Nascimento MA, Özsu MT, Kossmann D, Miller RJ, Blakeley JA, Schiefer KB, eds. Proc. of the 30th Int'l Conf. on Very Large Data Bases. Toronto: Morgan Kaufmann Publishers, 2004. 564−575.

[14] Chang KCC, Hwang SW. Minimal probing: Supporting expensive predicates for top-k queries. In: Franklin MJ, Moon B, Ailamaki A, eds. Proc. of the 2002 ACM SIGMOD Int'l Conf. on Management of Data. Madison: ACM, 2002. 346−357.

[15] Koutrika G, Simitsis A, Ioannidis Y. Precis: The essence of a query answer. In: Liu L, Reuter A, Whang KY, Zhang J, eds. Proc. of the 22nd Int'l Conf. on Data Engineering. Atlanta: IEEE Computer Society, 2006. 69−78.

附中文参考文献:

[2] 孟小峰,周龙骧,王珊.数据库技术发展趋势.软件学报,2004,15(12):1822−1836. http://www.jos.org.cn/1000-9825/15/1822.htm

WANG Bin was born in 1972. He is a Ph.D. candidate at the Institute of Computer System, Northeastern University. His current research areas are Web search and P2P management.

YANG Xiao-Chun was born in 1973. She is an associate professor at the Institute of Software and Theory, Northeastern University and a CCF senior member. Her research areas are database theory and techniques.

WANG Guo-Ren was born in 1966. He is a professor and doctoral supervisor at the Institute of Computer System, Northeastern University and a CCF senior member. His research areas are database theory and application.