Finding Top-k Answers in Keyword Search over Relational Databases Using Tuple Units

Jianhua Feng, *Member*, *IEEE*, Guoliang Li, and Jianyong Wang, *Senior Member*, *IEEE*

Abstract—Existing studies on keyword search over relational databases usually find Steiner trees composed of connected database tuples as answers. They on-the-fly identify Steiner trees by discovering rich structural relationships between database tuples, and neglect the fact that such structural relationships can be precomputed and indexed. Recently, tuple units are proposed to improve search efficiency by indexing structural relationships, and existing methods identify a single tuple unit to answer keyword queries. However, in many cases, multiple tuple units should be integrated to answer a keyword query. Thus, these methods will involve false negatives. To address this problem, in this paper, we study how to integrate multiple related tuple units to effectively answer keyword queries. To achieve a high performance, we devise two novel indexes, single-keyword-based structure-aware index and keyword-pair-based structure-aware index, and incorporate structural relationships between different tuple units into the indexes. We use the indexes to efficiently identify the answers of integrated tuple units. We develop new ranking techniques and algorithms to progressively find the top-k answers. We have implemented our method in real database systems, and the experimental results show that our approach achieves high search efficiency and result quality, and outperforms state-of-the-art methods significantly.

Index Terms—Keyword search, relational databases, single-keyword-based index, keyword-pair-based index, tuple units.

1 Introduction

Keyword search is a proven and widely accepted mechanism for querying in textual document systems and the World Wide Web. The database research community has recently recognized the benefits of keyword search and has been introducing keyword-search capabilities into relational databases [5], [20], [41], [29], [7], XML databases [8], [16], graph databases [17], [23], [9], [14], and heterogenous data sources [30]. Keyword search provides an alternative means of querying relational databases, which is simple to people who are familiar with using web search engines. One important advantage of keyword search is that it enables users to search for information without having to know complex structured query languages (e.g., SQL) or prior knowledge about the structures of the underlying data.

Keyword search over relational databases finds the answers of tuples in the databases which are connected through primary/foreign keys and contain query keywords. Existing studies can be broadly classified into three types of methods: candidate-network-based methods [18], [20], [39], Steiner-tree-based algorithms [5], [10], [17], [23], and tuple-unit-based approaches [28], [45]. The former two methods on-the-fly discover the connections (primary/foreign keys) between tuples to find the connected tuples. As there are usually large numbers of tuples in the databases, these methods are rather expensive to find answers by on-the-fly enumerating the connections. In other words, these two types of methods neglect the fact that the connections between relevant database tuples can be precomputed and indexed.

To address this problem, Li et al. [28] proposed *tuple units* to efficiently answer keyword queries. A tuple unit is a set of highly relevant tuples which contain query keywords. Moreover tuple units can be precomputed and indexed, and we can use the indexed tuple units to efficiently answer a keyword query. Existing methods only identify a single tuple unit to answer keyword queries. However, they neglect the fact that in many cases, we need to integrate multiple related tuple units to answer a keyword query. To address this problem, in this paper, we propose a structure-aware-index-based method to integrate multiple related tuple units to effectively answer keyword queries.

We devise two structure-aware indexes, *single-keyword-based structure-aware index* (SKSA-Index) and *keyword-pair-based structure-aware index* (KPSA-Index), to capture structural relationships between tuple units. We use the indexes to on-the-fly integrate multiple tuple units to answer a keyword query. We develop new ranking techniques and efficient algorithms to efficiently and progressively find the top-k answers. To summarize, we make the following contributions:

- We propose a tuple-unit-based method for effective keyword search over relational databases, which integrates multiple relevant database tuple units to effectively answer keyword queries.
- We propose two structure-aware indexes and store the structural relationships between different tuple units into the indexes. We use the indexes to efficiently find the relevant answers.
- We develop effective ranking techniques to rank the tuple units by taking into account both the structural compactness of answers from the database point of view and the textual relevancy from the information

• *The authors are with the Department of Computer Science and Technology, Tsinghua University, Room 10-204, East Main Building, Beijing 100084, China. E-mail: {fengjh, liguoliang, jianyong}@tsinghua.edu.cn.*

Manuscript received 30 Apr. 2010; revised 29 Oct. 2010; accepted 23 Jan. 2011; published online 3 Mar. 2011.
Recommended for acceptance by S. Chaudhuri, Y. Chen, and J.X. Yu.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDESI-2010-04-0251.
Digital Object Identifier no. 10.1109/TKDE.2011.61.

TABLE 1
An Example Database

Authors

AID	Name
a1	X. Zhou
a2	X. Lin
a3	J. Yu
a4	L. Guo
a5	J. Shanmugasundaram
a6	F. Shao
a7	V. Hristidis
a8	Y. Papakonstantinou
a9	A. Balmin

Author-Paper

AID	PID
a1	p1
a2	p1
a2	p2
a3	p2
a4	p3
a4	p4
a5	p3
a6	p4
a7	p5
a7	p6
a8	p5
a9	p6

Papers

PID	Title	Conf	Year
p1	Spark: Top-k keyword query in relational databases	SIGMOD	2007
p2	Finding top-k min-cost connected trees in databases	ICDE	2007
p3	Topology search over biological databases	ICDE	2007
p4	XRank: Ranked keyword search over XML documents	SIGMOD	2003
p5	Discover: Keyword search in relational databases	VLDB	2002
p6	Keyword proximity search on XML graphs	ICDE	2003

retrieval viewpoint. We devise efficient algorithms to progressively find top-k answers.

- We have implemented our method in MYSQL. The experimental results show that our method achieves high efficiency and result quality, and outperforms state-of-the-art methods significantly.

The remainder of this paper is organized as follows: Section 2 presents the concept of $tupleunits$ and gives an overview of our method. We discuss how to model the tuple units in Section 3. Section 4 gives several effective ranking functions to rank tuple units. We propose effective indexing techniques in Section 5, and discuss how to use the index structures to answer queries in Section 6. Section 7 reports experimental results. We review related works in Section 8 and make a conclusion in Section 9.

2 Overview of Our Approach

We first give some notations (Section 2.1) and then introduce the concept of tuple units (Section 2.2). Finally, we present the overview of our method (Section 2.3).

2.1 Notations

For ease of presentation, we first introduce some notations. Given a database $\mathcal{D}$ with m tables, $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_m$. Let $\mathcal{R}_i \xrightarrow{\kappa} \mathcal{R}_j$ denote that table $\mathcal{R}_i$ has a foreign key κ referring to the primary key of table $\mathcal{R}_j$. Two tables $\mathcal{R}_i$ and $\mathcal{R}_j$ are connected, denoted by, $\mathcal{R}_i \rightsquigarrow \mathcal{R}_j$, if, 1) $\mathcal{R}_i \xrightarrow{\kappa} \mathcal{R}_j$; 2) $\mathcal{R}_j \xrightarrow{\kappa} \mathcal{R}_i$; or 3) $\exists \mathcal{R}_k$, $\mathcal{R}_i \rightsquigarrow \mathcal{R}_k$ and $\mathcal{R}_k \rightsquigarrow \mathcal{R}_j$. For example, given a database with three tables in Table 1, we have Author-Paper $\xrightarrow{\text{AID}}$ Authors, Author-Paper $\xrightarrow{\text{PID}}$ Papers, Authors $\rightsquigarrow$ Author-Paper, Author-Paper $\rightsquigarrow$ Papers, and Authors $\rightsquigarrow$ Papers.

Without loss of generality, we suppose any two relational tables in the given database are connected. If some relational tables are not connected, we first decompose the tables into some groups of connected tables and then apply our method on the decomposed groups.

2.2 Tuple Units

Given a database with m connected tables, the tuples that can be joined together through the primary/foreign keys must be very relevant with each other. Based on this observation, Li et al. [28] proposed the concept of $tupleunits$ to answer keyword queries.

Definition 1 (Tuple Units). *Given a database $\mathcal{D}$ with m connected tables, $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_m$, for each tuple t_i in table $\mathcal{R}_i$, let $\mathcal{R}_{t_i}$ denote the table with the same primary/foreign keys as $\mathcal{R}_i$, having a single tuple t_i. The joined result of table $\mathcal{R}_{t_i}$ and other tables $\mathcal{R}_j (j \neq i)$ based on foreign keys,[1] denoted by $\Re = \bowtie_{j \neq i} \mathcal{R}_j \bowtie \mathcal{R}_{t_i}$, is called a tupleset. Given two tuple sets t_1 and t_2, if any tuple in t_2 is contained in t_1, we call that t_1 covers t_2 (t_2 is covered by t_1). A tuple set is called a tuple unit if it is not covered by any tuple set.*

To better understand the notation of tuple unit, we give a running example.

Example 1. Consider a publication database in Table 1. For each tuple in a table, we join the three tables and get the tuple sets as shown in Table 2. Tuple set T_{a_1} is not a tuple unit as it is covered by T_{p_1}. T_{a_2} is a tuple unit as any tuple set does not cover it. In this way, we can find all tuple units as shown in Table 2. Each tuple unit can represent a meaningful and integral information unit, and can be taken as an answer of a keyword query. Considering a keyword query {`relational, database, keyword, search, Hristidis`}, the underlined tuple unit T_{p_5} (Table 2) contains all the input keywords. We can take this tuple unit as an answer. Note that we do not need to on-the-fly identify structural relationships between tuples in different tables, and the tuple-unit-based method can improve search performance.

The tuple-unit-based method has the following features:

1. Tuple units are effective to answer keyword queries as they capture structures and can represent a meaningful and integral information unit.
2. The relationships between tuples connected through primary/foreign keys can be identified and indexed, and thus we can efficiently answer keyword queries by using such indexed structural information.
3. The number of tuple units will not be large, which is not larger than that of the total tuples in the underlying database. If each value of the primary key is referred by the foreign key, the number of tuple units is the same as the number of tuples in the tables with foreign keys. In practice, the number of tuple units is much smaller than the total number of tuples in the underlying database as experimentally proved in [28].
4. We can employ database capabilities to generate and materialize the tuple units by creating a view on top of the underlying relational tables. We need no additional indexes to maintain tuple units. Interested readers are referred to [28] for more details about how to generate and materialize tuple units for answering a keyword query.

1. Note that we will employ left join/right join in some cases to avoid missing tuples in the table with primary keys. For example, given two relational tables, $\mathcal{R}_i$, $\mathcal{R}_j$, and $\mathcal{R}_i \xrightarrow{\kappa} \mathcal{R}_j$. If there exists a value v in attribute κ of $\mathcal{R}_j$ and v is not referred by any value of attribute κ in $\mathcal{R}_i$, we will employ the right join on $\mathcal{R}_i$ and $\mathcal{R}_j$.

TABLE 2
Tuple Sets and Tuple Units (Tuple Unit $\surd$; Nontuple Unit $\times$)

(a) For tuples in *Authors*

TID	AID	PID	Name	Title	···	
Ta_1	a1	p1	···	···	···	×
Ta_2	a2	p1	···	···	···	$\surd$
	a2	p2	···	···	···	
Ta_3	a3	p2	···	···	···	×
Ta_4	a4	p3	···	···	···	$\surd$
	a4	p4	···	···	···	
Ta_5	a5	p3	···	···	···	×
Ta_6	a6	p4	···	···	···	×
Ta_7	a7	p5	···	···	···	$\surd$
	a7	p6	···	···	···	
Ta_8	a8	p5	···	···	···	×
Ta_9	a9	p6	···	···	···	×

(b) *For tuples in Papers*

TID	AID	PID	Name	Title	···	
Tp_1	a1	p1	···	···	···	$\surd$
	a2	p1	···	···	···	
Tp_2	a2	p2	···	···	···	$\surd$
	a3	p2	···	···	···	
Tp_3	a4	p3	···	···	···	$\surd$
	a5	p3	···	···	···	
Tp_4	a4	p4	···	···	···	$\surd$
	a6	p4	···	···	···	
Tp_5	a7	p5	···	···	···	$\surd$
	a8	p5	···	···	···	
Tp_6	a7	p6	···	···	···	$\surd$
	a9	p6	···	···	···	

(c) *For tuples in Author-Paper*

TID	AID	PID	Name	Title	···	
T_{ap_1}	a1	p1	···	···	···	×
T_{ap_2}	a2	p1	···	···	···	×
T_{ap_3}	a2	p2	···	···	···	×
T_{ap_4}	a3	p2	···	···	···	×
T_{ap_5}	a4	p3	···	···	···	×
T_{ap_6}	a4	p4	···	···	···	×
T_{ap_7}	a5	p3	···	···	···	×
T_{ap_8}	a6	p4	···	···	···	×
T_{ap_9}	a7	p5	···	···	···	×
$T_{ap_{10}}$	a7	p6	···	···	···	×
$T_{ap_{11}}$	a8	p5	···	···	···	×
$T_{ap_{12}}$	a9	p6	···	···	···	×

Li et al. [28] have studied how to generate and materialize tuple units using database capabilities, and how to identify and rank a single tuple unit to answer keyword queries. However, noticed that multiple tuple units may be interrelated and can be integrated together to answer a query. For example, consider the database in Table 1. Suppose a user issues a query "Zhou Yu." There is no tuple unit containing the two keywords. We can integrate the tuples T_{p_1} and T_{p_2} in Table 2 to answer the query. To address this problem, in this paper, we study effective ranking functions, index structures, and efficient algorithms for integrating multiple relevant tuple units to answer keyword queries.

2.3 Architecture

This section gives an overview of our proposed method, namely, SAINT (Structure-Aware INdexing for finding and integrating Tuple units for effective keyword search over relational databases). Fig. 1 depicts the overall architecture of SAINT.

SAINT takes the underlying relational databases and keyword queries as input, materializes the tuple units, and integrates the highly relevant tuple units connected by the primary/foreign key relationships to answer keyword queries. Finally, we rank the answers and deliver the top-k relevant answers with the highest scores to corresponding users. The basic components of SAINT are as follows:

- **Summarization.** Summarization generates and materializes the tuple units offline.
- **Indexing.** Indexing constructs structure-aware indexes on top of the summarized tuple units.
- **Progressive rank-aware answer extraction.** Extraction identifies the most relevant top-k answers with the highest scores to answer keyword queries using the structure-aware indexes.

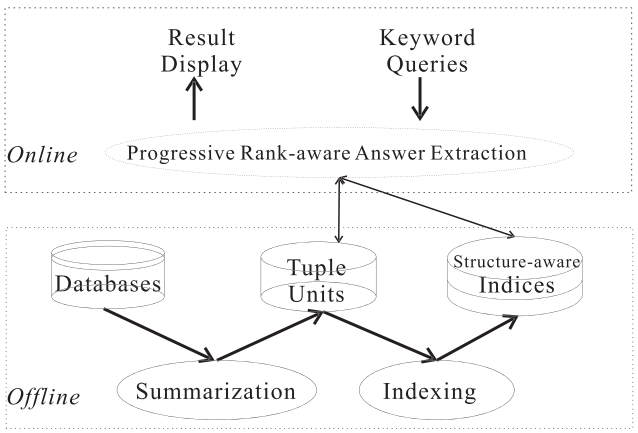

Fig. 1. The overall architecture of SAINT.

3 Tuple Unit Modeling

We model the tuple units w.r.t. a relational database as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where nodes $(\mathcal{V})$ are tuple units and edges $(\mathcal{E})$ are the relationships between two tuple units. Given two tuple units, if they share the same value on any primary key attribute, they will be related, and thus we connect the two tuple units. Different from Steiner-tree-based graphs in which nodes are tuples in relational tables and edges are primary/foreign key relationships [5], note that the graph we construct has much smaller size, as we group many tuples into a compact tuple unit. In other words, our constructed graph has smaller numbers of nodes and edges than the Steiner-tree-based graph, and thus is much easier to be manipulated. For example, consider the data in Table 1. The Steiner graph contains 24 nodes, and our graph has only nine nodes.

For ease of presentation, we use a boolean adjacency matrix ($\mathcal{BAM}$) to represent the modeled graph. The rows or columns of $\mathcal{BAM}$ are tuple units, and the value at the ith row and the jth column is 1 if and only if there is an edge between tuple unit u_i and tuple unit u_j; otherwise, the value is 0. For example, considering the database in Table 1, we first identify the tuple units as shown in Table 2 and then transform the tuple units into a graph as illustrated in Fig. 2. For instance, tuple unit T_1 and tuple unit T_2 are connected as they share the same vale on the primary key AID of table Author.

Notice that in the implementation, we use the adjacency list to represent all edges in the graph, which only keeps the

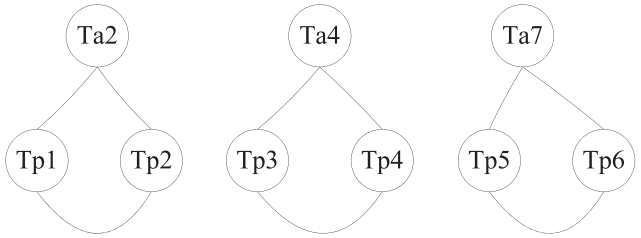

Fig. 2. Graph built from tuple units in Table 2.

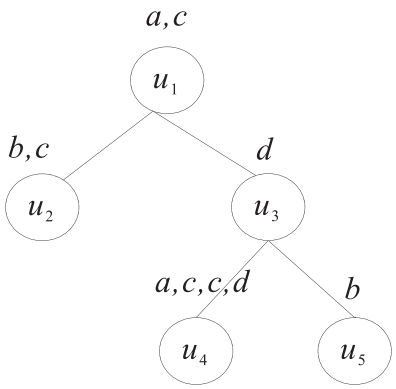

Fig. 3. A running example graph.

pairs of two nodes with an edge. As a database graph is typically a sparse graph, the indexing cost is acceptable (experimentally proved in Section 7).

As an another example, we model the graph in Fig. 3 as a $\mathcal{BAM}$ as shown in Table 3a, where the italic characters denote the keywords contained in the corresponding tuple unit. For instance, u_4 contains four keywords of a, c, c, d. We will take this graph as a running example throughout this paper. In the matrix (Table 3a), the cell for tuple unit u_3 and tuple unit u_4 is 1 as there is an edge between the two tuple units in the graph (Fig. 3).

To evaluate the relationship between two tuple units, we use their minimal distance to capture their relevancy. To keep such information, we use another matrix, minimal distance matrix, abbreviated as $\mathcal{MDM}$. Different from $\mathcal{BAM}$, the value of the ith row and the jth column of $\mathcal{MDM}$ is the minimal distance of the two tuple units u_i and u_j. In order to preserve the paths between two tuple units with minimal distance, we use a minimal path matrix, abbreviated as $\mathcal{MPM}$. The entry of $<u_i, u_j>$ in $\mathcal{MPM}$ preserves the path with minimal distance between u_i and u_j. For example, we can construct $\mathcal{MDM}$ and $\mathcal{MPM}$ of the graph in Fig. 3 as illustrated in Tables 3b and 3c, respectively. In $\mathcal{MDM}$, the value of entry $<u_4, u_2>$ is 3. This means that the minimal distance from u_4 to u_2 is 3. In $\mathcal{MPM}$, the value of entry $<u_4, u_2>$ is u_3, u_1. This means that the minimal path from u_4 to u_2 is u_4-u_3-u_1-u_2.

Although many existing literatures also model the tuples in the underlying databases as a graph and identify the Steiner trees as the answers [5], these methods are very inefficient. The reason is that it is rather hard to identify the Steiner trees by discovering the structural relationships between tuples, and such relationships are very rich in relational databases. As opposed to traditional Steiner-tree-based methods, we proposed to store the relationships between tuple units into structure-aware indexes and identify answers based on the indexes. Next, we study how to evaluate the importance of a tuple unit.

4 Ranking

In this section, we study several effective ranking functions to rank tuples units.

4.1 Structure-Aware Ranking

In this section, we discuss how to seamlessly incorporate the structural relationships between tuples into the ranking functions.

4.1.1 Directly Scoring

Traditional methods employ $TF \cdot IDF$-based methods in IR literature to score tuple units (or documents). In this section, we also use this technique to score a tuple unit w.r.t. an input keyword that is directly contained by the tuple unit. We model every tuple unit as a document and take the terms in the tuple units as keywords. Accordingly, the $TF \cdot IDF$-based methods can be borrowed to score tuple units as follows:

Suppose that we have the set (denoted as $\mathcal{U}$) of the tuple units in a given underlying relational database, and there are p distinct tuple units and q keywords in $\mathcal{U}$. Given a tuple unit $u \in \mathcal{U}$ and a keyword $k_i (1 \leq i \leq q)$ in u, we denote $tf(k_i, u)$ as the term frequency of k_i in u, which is the number of occurrences of k_i in u; we denote $idf(k_i)$ as the inverse document frequency of k_i, where $idf(k_i) = \frac{p+1}{O_{k_i}+1}$ and O_{k_i} is the number of such tuple units which directly contain k_i; we denote $ntl(u)$ as the normalized term length of u, where $ntl(u) = \frac{|u|}{\frac{1}{p} * \sum_{u' \in \mathcal{U}} |u'|}$ and $|u|$ denotes the number of terms in u.

In IR literature, the ranking methods usually integrate the three parameters to score a tuple unit w.r.t. a query $\mathcal{K} = \{k_1, k_2, \ldots, k_n\}$ as illustrated in (1) and (2), where s is a constant and usually set to 0.2 [35].

$$\text{SCORE}_{\text{IR}}(u, k_i) = \frac{ln(1 + tf(k_i, u)) * ln(idf(k_i))}{(1 - s) + s * ntl(u)}, \quad (1)$$

$$\text{SCORE}_{\text{IR}}(\mathcal{K}, u) = \sum_{k=1}^{n} \text{SCORE}_{\text{IR}}(u, k_i). \quad (2)$$

For example, considering the tuple units in Fig. 3, we can compute the scores according to (1) as illustrated in Table 4d. Especially, we give the tf, idf, and ntl in Tables 4a, 4b, and 4c to help users understand. For instance, idf of keyword a is $\frac{5+1}{2+1}$, since there are totally five nodes and two nodes contain the keyword.

TABLE 3
Graph Matrix

(a $\mathcal{BAM}$ (Boolean Adjacency Matrix))

	u_1	u_2	u_3	u_4	u_5
u_1	0	1	1	0	0
u_2	1	0	0	0	0
u_3	1	0	0	1	1
u_4	0	0	1	0	0
u_5	0	0	1	0	0

(b $\mathcal{MDM}$ (Minimal Distance Matrix))

	u_1	u_2	u_3	u_4	u_5
u_1	0	1	1	2	2
u_2	1	0	2	3	3
u_3	1	2	0	1	1
u_4	2	3	1	0	2
u_5	2	3	1	2	0

(c $\mathcal{MPM}$ (Minimal Path Matrix))

	u_1	u_2	u_3	u_4	u_5
u_1	-	-	-	u_3	u_3
u_2	-	-	u_1	u_1,u_3	u_1,u_3
u_3	-	u_1	-	-	-
u_4	u_3	u_3,u_1	-	-	u_3
u_5	u_3	u_3,u_1	-	u_3	-

TABLE 4
Directly Scoring

(a) tf

$tf(k_i,u)$	a	b	c	d
u_1	1	0	1	0
u_2	0	1	1	0
u_3	0	0	0	1
u_4	1	0	2	1
u_5	0	1	0	0

(b) idf

$Keyword$	$idf(k_i)$
a	$\frac{6}{3}$=2
b	$\frac{6}{3}$=2
c	$\frac{6}{4}$=1.5
d	$\frac{6}{3}$=2

(c) ntl

$TupleUnit$	$ntl(u)$
u_1	$\frac{2}{\frac{2+2+1+4+1}{5}}$=1
u_2	1
u_3	0.5
u_4	2
u_5	0.5

(d) $\text{SCORE}_{\text{IR}}(u, k_i)$

	a	b	c	d
u_1	0.48	0	0.28	0
u_2	0	0.48	0.28	0
u_3	0	0	0	0.53
u_4	0.40	0	0.37	0.40
u_5	0	0.53	0	0

Although the $TF \cdot IDF$-based scoring method in IR literature can rank the tuple units which directly contain input keywords, it cannot rank the tuple units that indirectly contain input keywords. Note that even if a tuple unit indirectly contains an input keyword, this tuple unit may also be relevant to the input keyword through the primary/foreign key relationships. For example, in Fig. 3, although node u_3 does not contain keywords a and b, it can connect the nodes u_4 and u_5 which contain the keywords. Thus, u_3 should also be relevant to the keywords. Considering another example, although a paper entitled with "`Data Warehousing and Multidimensional Data Models`" does not contain keyword "`OLAP`," this paper may also be relevant to "`OLAP`" as it cites a paper with keyword "`OLAP`."

However, traditional ranking methods [28] cannot capture the structural compactness between relevant tuple units. Most importantly, the structural information between relevant tuple units is at least as important as the textual information, and is even much more crucial in many cases. To address this issue, we propose a novel scoring method in Section 4.1.2.

4.1.2 Indirectly Scoring

This section proposes to rank tuple units w.r.t. input keywords that are indirectly contained by the tuple units and extends the score functions in (1).

For ease of presentation, we first introduce a notion of *pivotal tuple unit*, which can be used to measure the relationship between a keyword and a tuple unit that indirectly contains this keyword.

Definition 2 (Pivotal Tuple Unit). *Given a keyword k_j and a tuple unit u_i that indirectly contains k_j, the tuple unit $ku_{(k_j,u_i)}$, which directly contains k_j and has the minimal distance with u_i in the corresponding graph, is called a pivotal tuple unit.*[2] *That is, $ku_{(k_j,u_i)} \in argmin_{u_r}\{\delta(u_r, u_i)$, where u_r directly contains k_j and $\delta(u_r, u_i)$ denotes the distance between u_r and $u_i\}$.*

It is obvious that the smaller the distance between two tuple units, the more the relevancy between them. Based on this observation, given a keyword k_j and a tuple unit u_i that indirectly contains k_j, we first identify the pivotal tuple unit $ku_{(k_j,u_i)}$ w.r.t. u_i and k_j, which directly contains k_j and has the minimal distance with u_i among all the tuple units. Then, we score u_i w.r.t. k_j by combining $\text{SCORE}_{\text{IR}}(ku_{(k_j,u_i)}, k_j)$ and the distance between u_i and $ku_{(k_j,u_i)}$. There are many ranking functions to combine them as long as the score increases with $\text{SCORE}_{\text{IR}}(ku_{(k_j,u_i)}, k_j)$ and decreases with $\delta(ku_{(k_j,u_i)}, u_i)$. As an example, we give the following score function, which is effective as proved in our experiments.

$$\text{SCORE}_{\text{IR}}(u_i, k_j) = \frac{\text{SCORE}_{\text{IR}}(ku_{(k_j,u_i)}, k_j)}{(\delta(ku_{(k_j,u_i)}, u_i) + 1)^2}. \quad (3)$$

We note that $\text{SCORE}_{\text{IR}}(ku_{(k_j,u_i)}, k_j)$ can be computed using (1) as $ku_{(k_j,u_i)}$ directly contains k_j. $\delta(ku_{(k_j,u_i)}, u_i)$ can be derived from the minimal distance matrix. Accordingly, based on the minimal distance matrix and the scores of tuple units which directly contain input keywords, we can efficiently score the tuple units that indirectly contain the keywords.

To maintain the scores w.r.t. every tuple unit and every input keyword, we construct a score matrix, where each row is a tuple unit and each column is a keyword. The value at the ith row and the jth column of score matrix is denoted by $\text{SCORE}_{\text{IR}}(u_i, k_j)$, which is the score of u_i directly or indirectly containing keyword k_j. If u_i directly contains k_j, we compute the score according to (1); otherwise, we compute the score of u_i indirectly containing k_j by using (3).

In order to identify and construct the relevant answers composed of interrelated tuple units, we introduce another matrix, pivotal tuple unit matrix, where each row is the tuple unit and each column is a keyword, and the value at the ith row and the jth column is the path with minimal distance from tuple unit u_i to the pivotal tuple unit $ku_{(k_j,u_i)}$. Accordingly, the path from u_i to the corresponding pivotal tuple unit is preserved in pivotal tuple unit matrix, which can be used to reconstruct the answers of keyword queries. Note that, pivotal tuple unit matrix and score matrix can be obtained according to $\mathcal{MDM}$, $\mathcal{MPM}$, and the traditional IR scores as described in (1) and (2).

For example, considering the tuple unit u_1 and keyword b, as u_1 does not directly contain b, we identify the pivotal tuple unit u_2, which directly contains b and has the minimal distance with u_1. According to (3), we have

$$\text{SCORE}_{\text{IR}}(u_1, b) = \frac{\text{SCORE}_{\text{IR}}(u_2, b)}{(\delta(u_1, u_2) + 1)^2} = \frac{0.48}{2^2} = 0.12.$$

Although this structure-aware model can capture the structural compactness between tuple units, it cannot capture the structural relevancy between input keywords. Moreover, the relevancy of input keywords is at least as important as that of tuple units, even more crucial in some cases. For example, considering that a user searches for a paper written by "`Guoliang Feng`," although the paper coauthored by "`Guoliang Li`" and "`Jianhua Feng`" contains all the input keywords, it is less relevant to the query than the paper written by "`Guoliang Feng`." This is not an *adhoc* problem as the traditional methods may mistakenly take input keywords in different irrelevant nodes as an answer. Based

2. If there are more than one tuple unit that contains k_j and has the minimal distance with u_i, we randomly select one of them. Usually, we select the tuple unit with the highest IR score w.r.t. k_j.

on this observation, we introduce another structure-aware index to address this issue.

4.2 Keyword-Pair-Based Ranking

Given two keywords k_i and k_j, and a tuple unit u which directly or indirectly contains the two keywords, it is obvious that if k_i and k_j are in the same pivotal tuple unit, they will be relevant to each other. Moreover, the smaller the distance between the pivotal tuple units w.r.t. k_i and k_j, the higher the relevancy between them. There are many similarity functions for computing the score as long as the score decreases with the distance between k_i and k_j. As an example, we use the following equation to measure the keyword-pair-based structural relevancy between any pair of two keywords $<k_i, k_j>$ w.r.t. a tuple unit u, $\mathrm{REL}(<k_i, k_j>|u)$,

$$\mathrm{REL}(<k_i, k_j>|u) = \frac{1}{(\delta(ku_{(k_i,u)}, ku_{(k_j,u)}) + 1)^2}, \quad (4)$$

where $ku_{(k_i,u)}$ denotes the pivotal tuple unit w.r.t. k_i and u, and $\delta(ku_{(k_i,u)}, ku_{(k_j,u)})$ denotes the minimal distance between $ku_{(k_i,u)}$ and $ku_{(k_j,u)}$.

We note that if $\mathrm{REL}(<k_i, k_t>|u)$ and $\mathrm{REL}(<k_t, k_j>|u)$ are large, $\mathrm{REL}(<k_i, k_j>|u)$ must be also large, and thus k_i, k_j, and k_t must be very relevant with each other w.r.t. u. Accordingly, we can use the keyword-pair relevancy between any two input keywords to capture the overall structural relevancy of an input keyword query. Lemma 1 describes the key feature of the keyword-pair relevancy: the keyword-pair relevancy can capture the overall structural relevancy of all input keywords.

Lemma 1.

$$\mathrm{REL}(<k_i, k_j>|u) > \frac{1}{4} * min(\mathrm{REL}(<k_i, k_t>|u), \mathrm{REL}(<k_t, k_j>|u)).$$

Proof. Let $\delta_{(i,j)} = \delta(ku_{(k_i,u)}, ku_{(k_j,u)})$. If $\delta_{(i,t)} \leq \delta_{(t,j)}$, we have

$$\delta_{(i,t)} + \delta_{(t,j)} + 2 \leq 2 * (\delta_{(t,j)} + 1);$$

if $\delta_{(t,j)} \leq \delta_{(i,t)}$, we have

$$\delta_{(i,t)} + \delta_{(t,j)} + 2 \leq 2 * (\delta_{(i,t)} + 1).$$

Based on the two equations, we have

$$\delta_{(i,t)} + \delta_{(t,j)} + 2 \leq max\{2 * (\delta_{(i,t)} + 1), 2 * (\delta_{(t,j)} + 1)\}.$$

Based on the triangle inequality of the distances between nodes in a graph, we have $\delta_{(i,j)} \leq \delta_{(i,t)} + \delta_{(t,j)}$, and $\delta_{(i,j)} + 1 < \delta_{(i,t)} + \delta_{(t,j)} + 2$. Thus,

$$\delta_{(i,j)} + 1 < max\{2 * (\delta_{(i,t)} + 1), 2 * (\delta_{(t,j)} + 1)\}.$$

As $\mathrm{REL}(<k_i, k_j>|u) = \frac{1}{(\delta_{(i,j)}+1)^2}$, we have

$$\mathrm{REL}(<k_i, k_j>|u) > \frac{1}{4} * min(\mathrm{REL}(<k_i, k_t>|u), \mathrm{REL}(<k_t, k_j>|u)).$$

□

The keyword-pair relevancy can help to capture the rather rich structural information between two input keywords. Based on this good metric, we propose (5) to compute the structure-aware score of u w.r.t. keyword-pair $<k_i, k_j>$ from the DB viewpoint, i.e., $\mathrm{SCORE}_{\mathrm{DB}}(<k_i, k_j>, u)$, by combining the two IR scores, $\mathrm{SCORE}_{\mathrm{IR}}(u, k_i)$ and $\mathrm{SCORE}_{\mathrm{IR}}(u, k_j)$.

$$\mathrm{SCORE}_{\mathrm{DB}}(<k_i, k_j>, u) = \mathrm{REL}(<k_i, k_j>|u) * (\mathrm{SCORE}_{\mathrm{IR}}(u, k_i) + \mathrm{SCORE}_{\mathrm{IR}}(u, k_j)). \quad (5)$$

According to the structure-aware keyword-pair scores, we propose the overall DB score by considering the structural relevancy of keyword pairs as below:

$$\mathrm{SCORE}_{\mathrm{DB}}(\mathcal{K}, u) = \sum_{1 \leq i < j \leq n} \mathrm{SCORE}_{\mathrm{DB}}(<k_i, k_j>, u). \quad (6)$$

4.3 Score Expectation

To answer keyword queries effectively, we combine the IR score and the DB score, and propose the concept of SCORE EXPECTATION to score a keyword query $\mathcal{K} = \{k_1, k_2, \ldots, k_n\}$ w.r.t. a tuple unit u as described in (7), which is inspired from mathematical expectation.

$$\begin{aligned} \mathrm{EXPECTATION}(\mathcal{K}, u) &= \mathrm{SCORE}(\mathcal{K}, u) \\ &= \sum_{i=1}^{n} (w_{k_i} * \mathrm{SCORE}_{\mathrm{IR}}(u, k_i)), \end{aligned} \quad (7)$$

where

$$w_{k_i} = \sum_{j=1}^{n} \mathrm{REL}(<k_j, k_i>|u). \quad (8)$$

In the formula, w_{k_i} is the weight of k_i among the input keywords in $\mathcal{K}$, which is the same as the probability in mathematical expectation. Moreover, the score expectation can be computed by summing up the IR score and the DB score as formalized in Lemma 2.

Lemma 2.

$$\mathrm{SCORE}(\mathcal{K}, u) = \mathrm{SCORE}_{\mathrm{IR}}(\mathcal{K}, u) + \mathrm{SCORE}_{\mathrm{DB}}(\mathcal{K}, u).$$

Proof.

$$\begin{aligned} \mathrm{SCORE}(\mathcal{K}, u) &= \sum_{i=1}^{n} \mathrm{REL}(<k_i, k_i>|u) * \mathrm{SCORE}_{\mathrm{IR}}(u, k_i) \\ &\quad + \sum_{i=1}^{n} \left(\sum_{j \neq i} \mathrm{REL}(<k_j, k_i>|u) * \mathrm{SCORE}_{\mathrm{IR}}(u, k_i) \right) \\ &= \sum_{i=1}^{n} 1 * \mathrm{SCORE}_{\mathrm{IR}}(u, k_i) \\ &\quad + \sum_{i=1}^{n} \left(\sum_{j \neq i} \mathrm{REL}(<k_j, k_i>|u) * \mathrm{SCORE}_{\mathrm{IR}}(u, k_i) \right) \\ &= \sum_{i=1}^{n} 1 * \mathrm{SCORE}_{\mathrm{IR}}(u, k_i) \\ &\quad + \sum_{1 \leq i < j \leq n} \mathrm{SCORE}_{\mathrm{DB}}(<k_i, k_j>, u) \\ &= \mathrm{SCORE}_{\mathrm{IR}}(\mathcal{K}, u) + \mathrm{SCORE}_{\mathrm{DB}}(\mathcal{K}, u). \end{aligned}$$

□

TABLE 5
Indirectly Scoring

(a) Pivotal Tuple Unit Matrix

	a	b	c	d
u_1	-	u_2	-	u_3
u_2	u_1	-	-	u_1,u_3
u_3	u_1	u_5	u_4	-
u_4	-	u_3,u_5	-	-
u_5	u_3,u_1	-	u_3,u_4	u_3

(b) Score Matrix

	a	b	c	d
u_1	0.48	0.12	0.28	0.13
u_2	0.12	0.48	0.28	0.06
u_3	0.12	0.13	0.09	0.53
u_4	0.40	0.06	0.37	0.40
u_5	0.05	0.53	0.04	0.13

4.4 Top-k Answers of Keyword Queries

Given a keyword query $\mathcal{K} = \{k_1, k_2, \ldots, k_n\}$ and a relational database, we first identify the tuple units with the *top-k* highest scores (using the scoring function $\mathrm{SCORE}(\mathcal{K}, u)$). Then, we construct *the subtrees* which are rooted at each identified tuple unit and contain the paths from the tuple unit to the corresponding pivotal tuple units for each keyword. Finally, we take the subtrees as the answers. Note that the subtree is composed of multiple highly relevant tuple units. We introduce how to efficiently answer queries in the following sections.

5 Indexing

This section proposes two structure-aware indexes for efficiently answering a keyword query.

5.1 SKSA-Index

To efficiently retrieve IR scores, we propose a single-keyword-based structure-aware index, called SKSA-Index, which is similar to the traditional inverted index. The entries of SKSA-Index are also the keywords that are contained in the underlying database. Different from inverted indexes which only maintain the tuple units that directly contain the keyword, each entry of SKSA-Index preserves the tuple units that *directly* or *indirectly* contain the keyword in the form of a triple <TupleUnit, Score, TupleUnitLists>, where the Score is the assigned score of the keyword in the tuple unit TupleUnit, and TupleUnitLists preserves the tuple unit lists from Unit to the corresponding $pivotaltupleunits$, which can be obtained from the pivotal tuple unit matrix. The tuple units are sorted by the corresponding scores in descending order. Most importantly, SKSA-Index captures the rich structural relationships, as each entry preserves the paths from a given tuple unit to the corresponding pivotal tuple unit and also keeps the structure-aware scores of tuple units indirectly or directly containing keywords.

Based on the SKSA-Index, given a relevant tuple unit u, it is easy to get $\mathrm{SCORE}_{\mathrm{IR}}(u, k_i)$ for each $k_i \in \mathcal{K}$. Thus, we can efficiently compute the overall IR score, $\mathrm{SCORE}_{\mathrm{IR}}(\mathcal{K}, u)$, by summing up $\mathrm{SCORE}_{\mathrm{IR}}(u, k_i)$ for every $k_i \in \mathcal{K}$. To better understand how to construct and use the SKSA-Index, we walk through our method with a running example as follows:

Example 2. Considering the graph in Fig. 3, suppose that we have computed the IR scores, and stored the scores in Table 4d, the minimal distance matrix in Table 3b, the pivotal tuple unit matrix in Table 5a, and the score matrix in Table 5b, respectively. We construct the SKSA-Index as illustrated in Table 6a.

Then, suppose that a user issues a keyword query $\mathcal{K} = \{a, c, d\}$. We compute the IR scores of tuple units w.r.t. $\mathcal{K}$ using the SKSA-Index as follows:

$$\mathrm{SCORE}_{\mathrm{IR}}(\mathcal{K}, u_1) = 0.48 + 0.28 + 0.13 = 0.89,$$
$$\mathrm{SCORE}_{\mathrm{IR}}(\mathcal{K}, u_2) = 0.12 + 0.28 + 0.06 = 0.46,$$
$$\mathrm{SCORE}_{\mathrm{IR}}(\mathcal{K}, u_3) = 0.12 + 0.09 + 0.53 = 0.74,$$
$$\mathrm{SCORE}_{\mathrm{IR}}(\mathcal{K}, u_4) = 0.40 + 0.37 + 0.40 = 1.17,$$
$$\mathrm{SCORE}_{\mathrm{IR}}(\mathcal{K}, u_5) = 0.05 + 0.04 + 0.13 = 0.22.$$

□

For each keyword, the SKSA-index keeps the tuple units that directly/indirectly contain the keyword. Suppose there are N tuple units and the number of distinct keywords is M.

TABLE 6
Structure-Aware Indexes

(a) SKSA-Index

$Keyword$	$<TupleUnit, Score, TupleUnitLists>$
a	<u_1,0.48,u_1>; <u_4,0.40,u_4>; <u_2,0.12,u_2-u_1>; <u_3,0.12,u_3-u_1>; <u_5,0.05,u_5-u_3-u_1>
b	<u_5,0.53,u_5>; <u_2,0.48,u_2>; <u_3,0.13,u_3-u_5>; <u_1,0.12,u_1-u_2>; <u_4,0.06,u_4-u_3-u_5>
c	<u_4,0.37,u_4>; <u_1,0.28,u_1>; <u_2,0.28,u_2>; <u_3,0.09,u_3-u_4>; <u_5,0.04,u_5-u_3-u_4>
d	<u_3,0.53,u_3>; <u_4,0.40,u_4>; <u_1,0.13,u_1-u_3>; <u_5,0.13,u_5-u_3>; <u_2,0.06,u_2-u_1-u_3>

(b) KPSA-Index

$<Keyword_1, Keyword_2>$	$<TupleUnit, Score>$
<a,b>	<u_1,0.15>; <u_2,0.15>; <u_5,0.06>; <u_4,0.05>; <u_3,0.03>
<a,c>	<u_4,0.77>; <u_1,0.76>; <u_2,0.10>; <u_3,0.02>; <u_5,0.01>
<a,d>	<u_4,0.80>; <u_3,0.16>; <u_1,0.15>; <u_5,0.05>; <u_2,0.04>
<b,c>	<u_2,0.76>; <u_1,0.10>; <u_5,0.06>; <u_4,0.05>; <u_3,0.02>
<b,d>	<u_3,0.17>; <u_5,0.16>; <u_2,0.06>; <u_4,0.05>; <u_1,0.03>
<c,d>	<u_4,0.77>; <u_3,0.16>; <u_1,0.10>; <u_2,0.04>; <u_5,0.04>

The maximal size of SKSA-index is $O(M * N)$. Actually, given any term contained in the database, we can only keep the tuple units, which have scores larger than a given threshold τ. For example, for each term t, we only preserve the tuple units which have a score (w.r.t. t) larger than $\tau = 0.2$. This strategy can prune many irrelevant tuple units and can improve the performance. Thus, the index size of SKSA-index is much smaller than $O(M * N)$. Next, we give a deep analysis about the index size of SKSA. Suppose the average number of distinct keywords in a tuple unit is m and the average number of neighbors of a tuple unit is F. For any tuple unit and a keyword in the tuple unit, we only need to keep the tuple units with score to the keyword larger than τ. As we normalize the IR score in (1) into (0, 1], based on the ranking function (3), we only need to maintain the tuple units with distances to the keyword smaller than

$$\sqrt{\tfrac{1}{\tau}} - 1.$$

That is, for each tuple unit and each keyword in the tuple unit, we keep at most

$$O\left(1 + F + F^2 + \cdots + F^{\sqrt{\frac{1}{\tau}}-1}\right) = O\left(\frac{F^{\sqrt{\frac{1}{\tau}}}}{F-1}\right)$$

tuple units for the keyword. Thus, the total index size of SKSA is

$$O\left(N * m * \frac{F^{\sqrt{\frac{1}{\tau}}}}{F-1}\right).$$

5.2 KPSA-Index

This section presents another structure-aware index, keyword-pair-based structure-aware index, called KPSA-Index, to efficiently get the IR scores.

The entries of KPSA-Index are keyword pairs (two keywords), and each entry keeps the tuple units which contain the two keywords and the corresponding structure-aware scores, i.e., $\mathrm{SCORE}_{\mathrm{DB}}(<k_i, k_j>, u)$. For example, we can construct the KPSA-Index of the graph in Fig. 3 as illustrated in Table 6b. We can compute the overall DB scores according to our KPSA-Index as computing IR scores based on SKSA-Index.

Example 3. Consider the graph in Fig. 3. We compute the DB scores, and store the scores in Table 6b.

Then, suppose that a user issues a keyword query $\mathcal{K} = \{a, c, d\}$. We compute the DB scores of tuple units w.r.t. $\mathcal{K}$ using the KPSA-Index as follows:

$$\begin{aligned}
&\mathrm{SCORE}_{\mathrm{DB}}(\mathcal{K}, u_1) = 0.76 + 0.15 + 0.10 = 1.01,\\
&\mathrm{SCORE}_{\mathrm{DB}}(\mathcal{K}, u_2) = 0.10 + 0.04 + 0.04 = 0.18,\\
&\mathrm{SCORE}_{\mathrm{DB}}(\mathcal{K}, u_3) = 0.02 + 0.16 + 0.16 = 0.34,\\
&\mathrm{SCORE}_{\mathrm{DB}}(\mathcal{K}, u_4) = 0.77 + 0.8 + 0.77 = 2.34,\\
&\mathrm{SCORE}_{\mathrm{DB}}(\mathcal{K}, u_5) = 0.01 + 0.05 + 0.04 = 0.10.
\end{aligned}$$

We note that, the scores of u_4 w.r.t. a and d are not the maximum when compared with those of other tuple units, such as u_1 and u_3, as shown in Table 6a. However, it is obvious that u_4 is very relevant to this query. Interestingly, for each keyword pair, such as $<a, c>$, $<a, d>$, and $<c, d>$, u_4 has the maximum scores as illustrated in Table 6b, which reflects the salient feature of our keyword-pair-based scoring method.

For a pair of two keywords, the KPSA-index keeps the tuple units that directly/indirectly contain the two keywords in the pair. The maximal size of KPSA-index is $O(M^2 * N)$. Actually as we only keep those keyword pairs whose relevancy is larger than a threshold τ, the index size of KPSA-index is much smaller than $O(M^2 * N)$. Based on (5), the sum of the two scores must be no larger than 2. Then, based on (4), if a keyword pair is kept in the KPSA-index, their distance is smaller than

$$\sqrt{\frac{2}{\tau}} - 1.$$

Given a tuple unit u and a keyword pair $\langle k_1, k_2\rangle$, suppose the minimal distance between u and keyword k_1 is d_1 and the minimal distance between u and keyword k_2 is d_2. Without loss of generality, support $d_1 \leq d_2$. If u is kept for the keyword pair, based on (5), the score between k_1 and u is not larger than $\frac{\tau}{2}$. Based on (2), d_1 must be smaller than

$$\sqrt{\frac{2}{\tau}} - 1.$$

Based on triangle inequality, d_2 is smaller than

$$2 * \left(\sqrt{\frac{2}{\tau}} - 1\right).$$

Thus, for each tuple unit, we keep at most

$$\left(m * \frac{F^{\sqrt{\frac{2}{\tau}}}}{F-1}\right) * \left(m * \frac{F^{2*\sqrt{\frac{2}{\tau}}-1}}{F-1}\right)$$

keyword pairs. Thus, the total index size of KPSA is

$$O\left(N * m^2 * \frac{F^{\sqrt{\frac{2}{\tau}}}}{F-1} * \frac{F^{2*\sqrt{\frac{2}{\tau}}-1}}{F-1}\right).$$

6 Query Processing

Index-based method. Based on the SKSA-Index and KPSA-Index, we answer a keyword query as follows: consider a keyword query $\mathcal{K} = \{k_1, k_2, \ldots, k_n\}$. We first retrieve the relevant tuple units that contain an input keyword, and then compute the score of each relevant tuple-unit-based on Lemma 2. Finally, we rank the tuple units and return the *top-k* answers with the highest scores. Note that, given a relevant tuple unit u, it is easy to get $\mathrm{SCORE}_{\mathrm{IR}}(u, k_i)$ for each $k_i \in \mathcal{K}$ based on the SKSA-Index. Thus, we can efficiently compute the overall IR score, $\mathrm{SCORE}_{\mathrm{IR}}(\mathcal{K}, u)$, by summing up $\mathrm{SCORE}_{\mathrm{IR}}(u, k_i)$ for every $k_i \in \mathcal{K}$. Similarly, we can efficiently compute $\mathrm{SCORE}_{\mathrm{DB}}(\mathcal{K}, u)$ by using the KPSA-Index.

TABLE 7
Score Expectation Table

$TupleUnit$	a	b	c	d	$<a,b>$	$<a,c>$	$<a,d>$	$<b,c>$	$<b,d>$	$<c,d>$
u_1	0.48	0.12	0.28	0.13	0.15	0.76	0.15	0.10	0.03	0.10
u_2	0.12	0.48	0.28	0.06	0.15	0.10	0.04	0.76	0.06	0.04
u_3	0.12	0.13	0.09	0.53	0.03	0.02	0.16	0.02	0.17	0.16
u_4	0.40	0.06	0.37	0.40	0.05	0.77	0.80	0.05	0.05	0.77
u_5	0.05	0.53	0.04	0.13	0.06	0.01	0.05	0.06	0.16	0.04

Example 4. Suppose that a user issues a keyword query $\mathcal{K} = \{a, c, d\}$ and wants to get the top-2 answers. We compute the scores of tuple units w.r.t. $\mathcal{K}$, and get

$$\begin{aligned}
&\text{SCORE}(\mathcal{K}, u_1) = 0.89 + 1.01 = 1.90,\\
&\text{SCORE}(\mathcal{K}, u_2) = 0.46 + 0.18 = 0.64,\\
&\text{SCORE}(\mathcal{K}, u_3) = 0.74 + 0.34 = 1.08,\\
&\text{SCORE}(\mathcal{K}, u_4) = 1.17 + 2.34 = 3.51,\\
&\text{SCORE}(\mathcal{K}, u_5) = 0.22 + 0.10 = 0.32.
\end{aligned}$$

We return the top-2 tuple units with the highest scores, i.e., u_4 and u_1. Finally, we construct the subtrees composed of relevant tuple units according to the pivotal tuple unit matrix, i.e., $\{u_4\}$ and $\{u_1, u_3\}$.

However, the above method first computes the scores for all the tuple units, and then ranks the tuple units. This leads to low efficiency if there are large numbers of tuple units. Alternatively, we can employ threshold-based techniques to efficiently identify the top-k answers.

Threshold-based method. There have been many studies of identifying top-k answers in relational databases, such as Fagin Algorithm (FA) [11], Threshold Algorithm (TA) [12], and other methods [2], [22], [43]. To find the top-k tuple units with the highest scores (summing up $\text{SCORE}(u, k_i)$ for every keyword k_i), the Fagin algorithm and Threshold algorithm do not need to scan the full inverted lists of query keywords. Instead they can do an early termination. The basic idea is as follows: they scan the tuple units in the inverted lists in a round-robin way and maintain an upper bound for the scores of all unscanned tuple units. If there are k scanned tuple units whose scores are larger than the upper bound, the top-k answers have been found. We can borrow their ideas to compute the top-k answers on top of our index structures so as to improve the search efficiency.

SQL-based method. In addition, we can also use the database capabilities to compute the top-k answers. We construct and materialize a score expectation table (SET) to store $\text{SCORE}_{\text{IR}}(u, k_i)$ and $\text{SCORE}_{\text{DB}}(<k_i, k_j>, u)$, where attributes are keywords and keyword pairs, and records are corresponding scores as shown in Table 7. Using the score expectation table, we issue the following SQL statement to compute top-k tuple units for answering a keyword query:

SELECT top k TupleUnit
FROM SET
ORDER BY $k_1 + \cdots + k_n + <k_1, k_2> \cdots <k_{n-1}, k_n>$ DESC

Then, we find the pivotal tuple units based on SKSA-Index and construct the top-k answers with the highest score expectations. As there are many zero values in table SET, SET is a sparse table. We can employ the sparse table-based techniques in [49] to facilitate identifying top-k answers. To understand our method better, we give a running example as follows:

Example 5. Consider the modeled graph in Fig. 3. We construct the score expectation table as illustrated in Table 7. Suppose that a user issues a keyword query $\{a, c, d\}$ on the graph in Fig. 3, we first create the table $\mathcal{V}_{\mathcal{K}} = \mathcal{V}_{a,c,d,<a,c>,<a,d>,<c,d>}$ by projecting the six columns from the score expectation table, and then find top-k answers by issuing the following SQL statement:

SELECT top k TupleUnit
FROM $\mathcal{V}_{\mathcal{K}}$
ORDER BY a+c+d+<a,c>+<a,d>+<c,d> DESC

7 Experimental Study

We have implemented our proposed methods in real relational database system MYSQL 5.0.22. We report some experimental results in this section.

We employed the DBLP[3] and IMDB[4] data sets to evaluate our algorithms. We converted the DBLP data set into four relational tables as shown in Table 8, where the underlined attributes are the primary/foreign keys. The schema of the IMDB data set is illustrated in Table 9. The raw file of the DBLP data set was about 470 MB. IMDB contains approximately one million anonymous ratings of 3,900 movies made by 6,040 users. We selected 100 keyword queries with different numbers of input keywords to compare the algorithms. The keyword number of DBLP data set is about 0.5 million and that of the IMDB data set is about 0.4 million.

We compared our algorithm with the state-of-the-art algorithms, BLINKS [17], Retune [28] and SPARK [39].[5] All the algorithms were implemented in C++. All the experiments were conducted on a computer with an Intel(R) Core(TM) 2@2.0 GHz CPU, a 2 GB of RAM and a 120 G Disk running Windows XP.

7.1 Index Time and Sizes

In this section, we report some experimental results of indexing. We give the elapsed time of indexing, the number of entries, the sizes of our structure-aware indexes, and scalability. We set $\tau = 0.2$. The experimental results are summarized in Table 10. We observe that the elapsed time, the number of entries, and the size of KPSA-Index are a bit larger than those of SKSA-Index, as KPSA-Index needs to

3. http://dblp.uni-trier.de/xml/.
4. http://www.grouplens.org/node/73.
5. We did not compare with [45] and compared with SPARK, as SPARK extends the method [45] by incorporating new ranking functions and principles, and achieves better performance.

TABLE 8
The Schema of DBLP Data Set

Table	Attributes			
Papers	PaperID	Title	Year	...
Authors	AuthorID	Name		
Paper-Author	AuthorID	PaperID		
Paper-Reference	PaperID	CitedPaperID		

TABLE 9
The Schema of IMDB Data Set

Table	Attributes			
Movies	MID	MovieName	Genres	...
Users	UID	UserName	OID	...
Ratings	MID	UID	Rating	
Occupation	OID	Occupation		

TABLE 10
Evaluation of Indexing

(a) Index Time

Elapsed Time of Indexing ($minute$)	DBLP	IMDB
SKSA-Index	23	6
KPSA-Index	53	15

(b) # of Entries (# of Attributes) in Score Expectation Table

# of Entries	DBLP	IMDB
SKSA-Index	46350	16735
KPSA-Index	326887	47346

(c) Index Sizes

Size (MB)	DBLP	IMDB
Dataset	405	27
SKSA-Index	615	31
KPSA-Index	1031	89

(d) Scalability of Index Sizes

DBLP Data Size (MB)	80	160	240	320	400
SKSA-Index	119	245	368	492	603
KPSA-Index	203	405	617	808	1012

compute and materialize the keyword-pair-based scores. More importantly, the two indexes scaled very well as the data set increases.

In addition, the two structure-aware indexes can significantly improve the search efficiency and result quality. KPSA-Index achieves much higher search quality than SKSA-Index, which in turn outperforms the other methods as discussed in following sections. The experimental results of indexing reflect that our structure-aware indexes are practicable in real database systems.

7.2 Search Efficiency

This section evaluates the search efficiency of the three algorithms. We selected 100 keyword queries from query logs of our deployed systems for each data set and evaluated different algorithms on them. To better understand the performance of our indexing method, we tested our algorithm with SKSA-Index (SAINT+SKSA) and KPSA-Index (SAINT+KPSA).

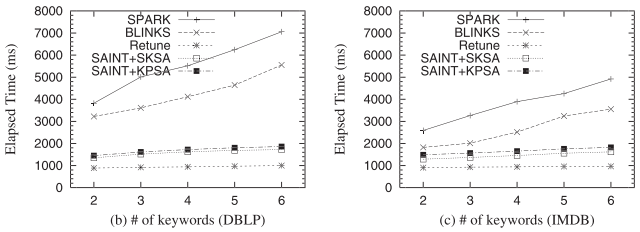

Fig. 4. Search efficiency.

We first varied different numbers of input keywords to identify all the answers and compared the corresponding average elapsed time. Fig. 4 summarizes the experimental results on the two data sets. We observe that our algorithms achieve much higher search efficiency than SPARK, as we use structure-aware indexes to efficiently identify the answers. Our method also outperforms BLINKS as the graph our method constructs has smaller number of nodes than that of BLINKS. For example, on IMDB data set, SAINT costs less than 1,500 ms to answer the keyword queries with six keywords while SPARK even costs more than 5,000 ms. This comparison shows the significance of our proposed structure-aware indexes, which can help to identify the answers through our indexed tuples units, and thus significantly improve the search efficiency. Although SAINT+SKSA costs a little longer time than SAINT+KPSA, SAINT+KPSA archives much higher result quality than SAINT+SKSA, which will be further discussed in Section 7.3. SAINT is a bit slower than Retune, as Retune only identifies a single tuple unit and need not discover the relationships between different tuple units. We will prove that Retune leads to low search effectiveness in the following sections.

To further evaluate the performance of our algorithms, we identified the top-k relevant answers and compared the corresponding elapsed time. Figs. 5 and 6 give the experimental results on the two data sets, respectively. We see that our method beats SPARK significantly. For example, on the DBLP data set, considering finding top-10 results of the queries with six keywords, our methods cost about 1,000 ms while SPARK and BLINKS, respectively, consumed 3,000 and 2,000 ms. This is so because, we adopt the threshold-based techniques to identify the top-k answers, which can help to progressively and efficiently

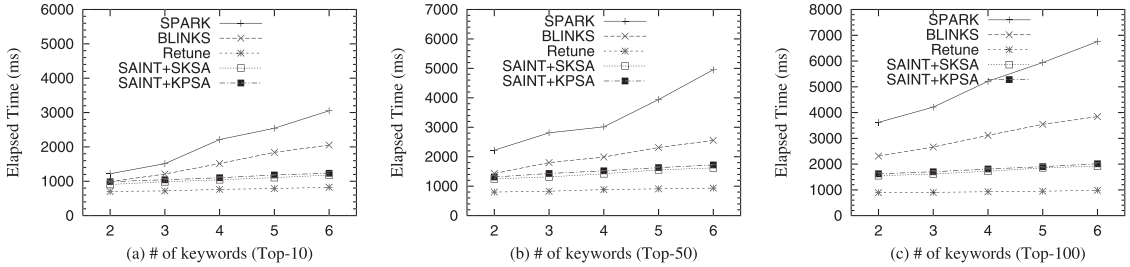

Fig. 5. Search efficiency of top-k answers on DBLP data set.

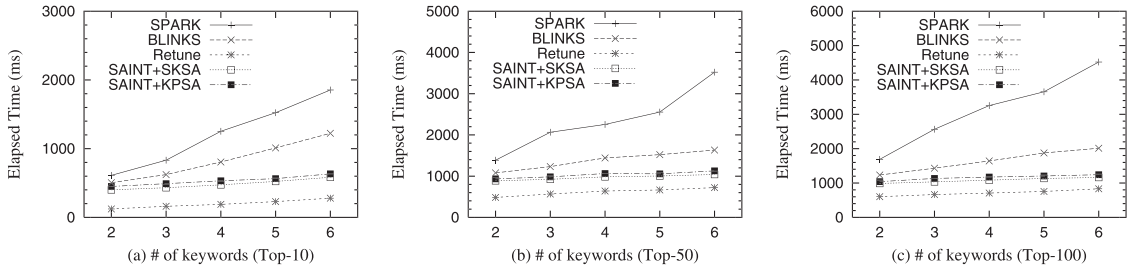

Fig. 6. Search efficiency of top-k answers on IMDB data set.

identify the *top-k* results with the highest scores and thus lead to high search efficiency.

7.3 Result Quality

This section evaluates the result quality of a search technique in terms of search accuracy and completeness using standard precision and recall metrics, where the correct results are the answers returned by the corresponding schema-aware languages such as SQL. Given a keyword query, we enumerate all possible SQL queries for a set of keywords (as discussed in the paper [15]) to generate our baseline query results, which are assumed as accurate and complete, and used to compare with the results for these keywords using different algorithms. For each query, we generated all possible SQL queries as follows: first, we join the tables to formulate SQL queries with the WHERE clauses containing keywords. Second, the tables that contain the keywords can be connected within a specific distance. (We set the distance as 4.) Third, the tables that contain no keyword must be connected by some tables containing keywords; otherwise, we remove them. For the top-k queries, we evaluated the precision and recall by user study, and the answer relevance of a query is judged from discussions of the 15 people attending the user study.

7.3.1 Search Accuracy

We evaluate the search accuracy of the selected algorithms in this section. We first varied the numbers of input keywords and evaluated the corresponding precision. Fig. 7 illustrates the experimental results. We observe that SAINT achieves much higher precision than existing methods such as Retune, SPARK, and BLINKS significantly. This is because our method integrates multiple tuple units to answer a keyword query. For example, on IMDB data set, SAINT achieves more than 90 percent precision, which leads to about 20-30 percent over those of Retune and SPARK. Moreover, SAINT+KPSA outperforms SAINT+SKSA as the former considers the structural relevancy of input keywords as well as the structural compactness of the answers.

In addition, as users are usually interested in the *top-k* answers, we employed another good metric, *top-k* answer relevancy to evaluate the search accuracy of SAINT, which measures the ratio of the number of relevant answers among the first k answers with the highest scores of an algorithm to k. The experimental results of the average *top-k* answer relevancy for the 100 queries on the two data sets are summarized in Figs. 8 and 9.

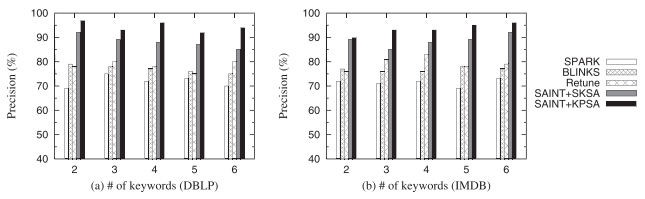

Fig. 7. Search accuracy.

As expected, SAINT+SKSA always achieves more than 85 percent precision, which is about 10-30 percent higher than the existing methods for various queries and different data sets. This reflects our relevancy-based ranking is very effective and practicable. Moreover, we note that SAINT+KPSA always gets more than 90 percent precision and achieves the best performance. This is so because we employ the keyword-pair-based ranking, which can capture the structural relevancy between input keywords. This comparison also reflects the effectiveness of our overall ranking method by taking into account both structural compactness between tuple units from DB point of view and textual relevancy from IR perspective.

7.3.2 Search Completeness

This section evaluates the search completeness. To evaluate the ranking mechanism, we compared the overall precision and recall. Fig. 10 shows the precision/recall curves of different algorithms. We observe that SAINT outperforms the alternative methods and always achieves higher precision than state-of-the-art proposals on whatever values of recall. Moreover, the precision of alternative methods falls sharply with the increase of recall, while that of SAINT varies little. This comparison further demonstrates the high search effectiveness of our proposed ranking mechanism. For example, on IMDB data set, when recall is 0.6, the precision of SAINT+KPSA is 0.9, which leads to 20-30 percent over those of Retune, SPARK, and BLINKS.

7.4 Usability Study

To further evaluate result quality of different methods, we evaluate the query results by human judgement. We have deployed a system on the DBLP data set which has been widely used. We selected the 20 most frequently issued queries to test different methods. Answer relevance of the queries was judged from discussions of researchers in our group. As users are usually interested in the top-k answers, we employed the top-k precision, i.e., the ratio of the number of answers deemed to be relevant in the first k results to k. Table 11 shows the average top-k precision of the selected 20 queries. We observe that our methods still achieve higher search quality. For example, SAINT+KPSA always achieves more than 90 percent precision while

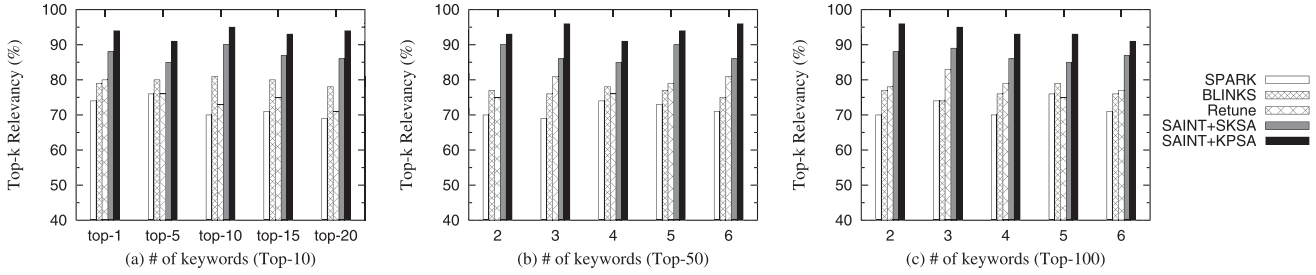

Fig. 8. Search accuracy of top-k answers on DBLP data set.

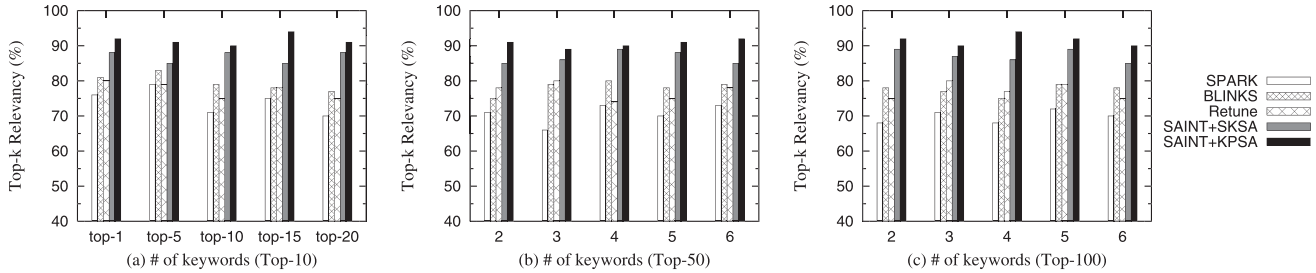

Fig. 9. Search accuracy of top-k answers on IMDB data set.

Retune, SPARK, and BLINKS only have less than 80 percent precision. This is attributed to our effective ranking methods by considering both DB and IR and integrating multiple tuple units to answer a query.

8 Related Work

Existing studies of keyword search in relational databases can be broadly classified into three types of methods: candidate-network-based methods [18], [20], [39], Steiner-tree-based algorithms [5], [10], [17], [23], and tuple-unit-based approaches [28], [45]. Steiner-tree-based methods first model tuples in the relational database as a graph, where nodes are database tuples and edges are primary/foreign keys, and then identify the Steiner trees which contain all or some input keywords to answer keyword queries. The candidate-network-based methods identify the answers composed of relevant tuples by generating and extending the candidate networks (i.e., the primary/foreign keys).

DISCOVER-I [20] and DBXplorer [1] identified trees of tuples connected through primary/foreign key relationships that contain all the input keywords of a given keyword query. BANKS-I [5] modeled relational data as graphs and identified the connected trees, called Steiner trees, to answer a query. DISCOVER-II [18] studied the problem of keyword proximity search in terms of disjunctive semantics, compared with DISCOVER-I only considering the conjunctive semantics. BANKS-II [23] improved the efficiency beyond BANKS-I by introducing a novel technique of bidirectional expansion. Liu et al. [35] proposed a novel ranking strategy to improve the result quality in relational databases, using the phrase-based and concept-based models. ObjectRank [3] extended the idea of hub-authority-based ranking to rank the answers of keyword queries in relational databases to improve the search effectiveness. Although ObjectRank is effective to rank objects, pages, and entities similar to HITS [25], it cannot rank tree-structured results, such as Steiner trees. Kimelfeld and Sagiv [24] studied keyword proximity search in relational databases. The method shows that the answer of keyword proximity search can be enumerated in ranked order with a polynomial delay, under data complexity. Sayyadian et al. [42] introduced schema mapping into keyword search and proposed a method to answer keyword search across heterogenous databases by seamlessly integrating schema mapping and keyword search. Ding et al. [10] proposed a dynamic-programming-based method to improve the search efficiency for identifying Steiner trees. He et al. [17] proposed a partition-based method to improve the search efficiency using a novel BLINKS index. Guo et al. [15] proposed data topology search to retrieve meaningful structures from biological databases. Markowetz et al. [40] studied the problem of keyword search over relational data streams, which is the first attempt to process keyword search over relational data streams. Luo et al. [39] proposed a new ranking method that adapts the state-of-the-art IR ranking

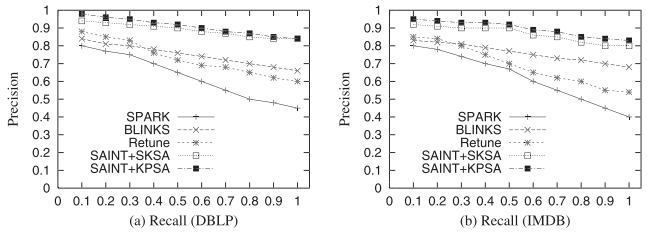

Fig. 10. Precision-recall curves.

TABLE 11
Top-k Precision by Human Judgement

Precision (%)	Top-1	Top-10	Top-50	Top-100
SPARK	65	67.5	63.2	64.1
BLINKS	74.5	78.5	76	66.5
Retune	70	73	76.5	75.4
SAINT+SKSA	85	88.5	90.5	90
SAINT+KPSA	90	92	91.6	92.5

functions and principles into ranking tree-structured results composed of joined database tuples. Li et al. [31], [32] proposed compact Steiner trees to improve search efficiency, by introducing an approximate algorithm to identify Steiner trees.

However, the above methods compute the Steiner trees composed of relevant tuples on-the-fly, and neglect the fact that the relevant tuples can be identified and materialized offline for facilitating the online processing of keyword queries. Su and Widom [45] proposed a technique of indexing tuples to improve the search efficiency. They proposed *text objects* and *virtual documents*, which are tuples connected through primary/foreign keys to answer keyword queries. Li et al. [28] proposed tuple units to improve search effectiveness by grouping the relevant text objects and virtual documents. They computed single text object or tuple unit to answer keyword queries. However, different text objects or tuple units may also be relevant, and multiple relevant text objects or tuple units can be integrated together to answer a keyword query. Thus, those methods may lead to miss some relevant results and introduce a serious problem—false negative. Different from our previous study [28], we study how to identify multiple related tuple units to answer keyword queries, as opposed to taking a single tuple unit as an answer. In addition, Su and Widom [45] only considered the document relevancy in IR literature to rank the answers but neglected the rich structural information, which is at least as important as the textual information, and is even more crucial in many cases. Instead, we propose a novel ranking method by taking into account both the textual relevancy from the IR point of view and the structural compactness between the relevant tuple units from the DB viewpoint to improve the result quality. Moreover, we devise two structure-aware indexes to improve the search efficiency and accuracy, which incorporate the structural information into the indexes. Different from our poster paper [26], we added a new structure-aware index, a new ranking technique, and an extensive new performance study.

In addition, there are many studies on keyword search in XML documents [8], [16], [21], [13], [37], [19], [38], [4], [6], [13], [33], [36]. They modeled XML documents as tree structures and computed connected subtrees to answer a keyword query. To find such relevant trees, they used a concept of "lowest common ancestors (LCAs)" or their variants to answer keyword queries. As an extension of LCA, XRank [16], XSEarch [8], Meaningful LCA (MLCA) [34], Smallest LCA (SLCA) [47], Multiway-SLCA (MSLCA) [46], Valuable LCA (VLCA) [27], RACE [13], Exclusive LCA (ELCA) [48], [44], and XSeek [36] have recently been proposed to answer keyword queries over XML documents.

9 Conclusion

We have studied the problem of effective keyword search over relational databases. We proposed to integrate multiple relevant tuple units to effectively answer keyword queries. We devised two novel structure-aware indexes, SKSA-Index and KPSA-Index, which incorporate the structural relationships between tuple units and the textual relevancy between input keywords into the indexes. We proposed a novel ranking mechanism by taking into consideration both the textual relevancy in IR literature and the structural compactness of tuple units from the DB viewpoint. We have implemented our method, and the experimental results show that our method achieves high performance and outperforms state-of-the-art approaches significantly.

In future work, we will study how to support online updates and how to reduce the index sizes.

Acknowledgments

This work is partly supported by the National Natural Science Foundation of China under Grant No. 61003004 and No. 60873065, the National High Technology Development 863 Program of China under Grant No. 2009AA011906, the National Grand Fundamental Research 973 Program of China under Grant No. 2011CB302206, and National S&T Project of China under Grant No. 2011ZX01042-001-002.

References

[1] S. Agrawal, S. Chaudhuri, and G. Das, "DBXplorer: A System for Keyword-Based Search over Relational Databases," *Proc. Int'l Conf. Data Eng. (ICDE)*, pp. 5-16, 2002.

[2] B. Arai, G. Das, D. Gunopulos, and N. Koudas, "Anytime Measures for Top-k Algorithms," *Proc. Int'l Conf. Very Large Data Bases (VLDB)*, 2007.

[3] A. Balmin, V. Hristidis, and Y. Papakonstantinou, "Objectrank: Authority-Based Keyword Search in Databases," *Proc. Int'l Conf. Very Large Data Bases (VLDB)*, pp. 564-575, 2004.

[4] Z. Bao, T.W. Ling, B. Chen, and J. Lu, "Effective XML Keyword Search with Relevance Oriented Ranking," *Proc. IEEE Int'l Conf. Data Eng. (ICDE)*, pp. 517-528, 2009.

[5] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan, "Keyword Searching and Browsing in Databases Using Banks," *Proc. Int'l Conf. Data Eng. (ICDE)*, pp. 431-440, 2002.

[6] L.J. Chen and Y. Papakonstantinou, "Supporting Top-k Keyword Search in Xml Databases," *Proc. IEEE Int'l Conf. Data Eng. (ICDE)*, pp. 689-700, 2010.

[7] E. Chu, A. Baid, X. Chai, A. Doan, and J.F. Naughton, "Combining Keyword Search and Forms for Ad Hoc Querying of Databases," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 349-360, 2009.

[8] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv, "Xsearch: A Semantic Search Engine for XML," *Proc. Int'l Conf. Very Large Data Bases (VLDB)*, pp. 45-56, 2003.

[9] B.B. Dalvi, M. Kshirsagar, and S. Sudarshan, "Keyword Search on External Memory Data Graphs," *Proc. VLDB Endowment*, vol. 1, no. 1, pp. 1189-1204, 2008.

[10] B. Ding et al., "Finding Top-k Min-Cost Connected Trees in Databases," *Proc. IEEE Int'l Conf. Data Eng. (ICDE)*, 2007.

[11] R. Fagin, "Combining Fuzzy Information from Multiple Systems," *Proc. ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems (PODS)*, pp. 216-226, 1996.

[12] R. Fagin, "Fuzzy Queries in Multimedia Database Systems," *Proc. ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems (PODS)*, pp. 1-10, 1998.

[13] J. Feng, G. Li, J. Wang, and L. Zhou, "Finding and Ranking Compact Connected Trees for Effective Keyword Proximity Search in XML Documents," *Information Systems*, vol. 35, no. 2, pp. 186-203, 2010.

[14] K. Golenberg, B. Kimelfeld, and Y. Sagiv, "Keyword Proximity Search in Complex Data Graphs," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 927-940, 2008.

[15] L. Guo, J. Shanmugasundaram, and G. Yona, "Topology Search over Biological Databases," *Proc. IEEE Int'l Conf. Data Eng. (ICDE),*, 2007.

[16] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram, "Xrank: Ranked Keyword Search over XML Documents," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 16-27, 2003.

[17] H. He, H. Wang, J. Yang, and P. Yu, "Blinks: Ranked Keyword Searches on Graphs," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 2007.
[18] V. Hristidis, L. Gravano, and Y. Papakonstantinou, "Efficient Ir-Style Keyword Search over Relational Databases," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* pp. 850-861, 2003.
[19] V. Hristidis, N. Koudas, Y. Papakonstantinou, and D. Srivastava, "Keyword Proximity Search in Xml Trees," *IEEE Trans. Knowledge and Data Eng.,* vol. 18, no. 4, pp. 525-539, Apr. 2006.
[20] V. Hristidis and Y. Papakonstantinou, "Discover: Keyword Search in Relational Databases," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* pp. 670-681, 2002.
[21] V. Hristidis, Y. Papakonstantinou, and A. Balmin, "Keyword Proximity Search on Xml Graphs," *Proc. Int'l Conf. Data Eng. (ICDE),* pp. 367-378, 2003.
[22] M. Hua, J. Pei, A.W.C. Fu, X. Lin, and H.-F. Leung, "Efficiently Answering Top-k Typicality Queries on Large Databases," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* 2007.
[23] V. Kacholia et al., "Bidirectional Expansion for Keyword Search on Graph Databases," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* pp. 505-516, 2005.
[24] B. Kimelfeld and Y. Sagiv, "Finding Approximating Top-k Answers in Keyword Proximity Search," *Proc. ACM SIGMOD-SIGACT-SIGART Symp. Principles of Database Systems (PODS),* 2006.
[25] J.M. Kleinberg, "Authoritative Sources in a Hyperlinked Environment," *J. ACM,* vol. 46, no. 5, pp. 604-632, 1999.
[26] G. Li, J. Feng, and J. Wang, "Structure-Aware Indexing for Keyword Search in Databases," *Proc. ACM Conf. Information and Knowledge Management (CIKM),* pp. 1453-1456, 2009.
[27] G. Li, J. Feng, J. Wang, and L. Zhou, "Efficient Keyword Search for Valuable Lcas over XML Documents," *Proc. ACM Conf. Information and Knowledge Management (CIKM),* 2007.
[28] G. Li, J. Feng, and L. Zhou, "Retune: Retrieving and Materializing Tuple Units for Effective Keyword Search over Relational Databases," *Proc. Int'l Conf. Conceptual Modeling (ER),* pp. 469-483, 2008.
[29] G. Li, S. Ji, C. Li, and J. Feng, "Efficient Type-Ahead Search on Relational Data: A Tastier Approach," *Proc. SIGMOD Int'l Conf. Management of Data,* pp. 695-706, 2009.
[30] G. Li, B.C. Ooi, J. Feng, J. Wang, and L. Zhou, "Ease: An Effective 3-in-1 Keyword Search Method for Unstructured, Semi-Structured and Structured Data," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 903-914, 2008.
[31] G. Li, X. Zhou, J. Feng, and J. Wang, "Progressive Keyword Search in Relational Databases," *Proc. IEEE Int'l Conf. Data Eng. (ICDE),* pp. 1183-1186, 2009.
[32] G. Li, J. Feng, X. Zhou, and J. Wang, "Providing Built-in Keyword Search Capabilities in RDBMS," *The VLDB J.,* vol. 20, no. 1, pp. 1-19, 2011.
[33] J. Li, C. Liu, R. Zhou, and W. Wang, "Suggestion of Promising Result Types for XML Keyword Search," *Proc. Int'l Conf. Extending Database Technology (EDBT),* pp. 561-572, 2010.
[34] Y. Li, C. Yu, and H.V. Jagadish, "Schema-Free Xquery," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* pp. 72-84, 2004.
[35] F. Liu, C. Yu, W. Meng, and A. Chowdhury, "Effective Keyword Search in Relational Databases," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 563-574, 2006.
[36] Z. Liu and Y. Chen, "Identifying Return Information for Xml Keyword Search," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 2007.
[37] Z. Liu and Y. Chen, "Reasoning and Identifying Relevant Matches for Xml Keyword Search," *Proc. VLDB Endowment,* vol. 1, no. 1, pp. 921-932, 2008.
[38] Z. Liu, P. Sun, and Y. Chen, "Structured Search Result Differentiation," *Proc. VLDB Endowment,* vol. 2, no. 1, pp. 313-324, 2009.
[39] Y. Luo, X. Lin, W. Wang, and X. Zhou, "Spark: Top-k Keyword Query in Relational Databases," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 2007.
[40] A. Markowetz, Y. Yang, and D. Papadias, "Keyword Search on Relational Data Streams," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 2007.
[41] L. Qin, J.X. Yu, and L. Chang, "Keyword Search in Databases: The Power of RDBMS," *Proc. SIGMOD Int'l Conf. Management of Data,* pp. 681-694, 2009.
[42] M. Sayyadian, H. LeKhac, A. Doan, and L. Gravano, "Efficient Keyword Search across Heterogeneous Relational Databases," *Proc. IEEE Int'l Conf. Data Eng. (ICDE),* 2007.
[43] K. Schnaitter, J. Spiegel, and N. Polyzotis, "Depth Estimation for Ranking Query Optimization," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* 2007.
[44] F. Shao, L. Guo, C. Botev, A. Bhaskar, M. Chettiar, F. Yang, and J. Shanmugasundaram, "Efficient Keyword Search over Virtual XML Views," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* 2007.
[45] Q. Su and J. Widom, "Indexing Relational Database Content Offline for Efficient Keyword-Based Search," *Proc. Int'l Database Eng. and Application Symp. (IDEAS),* 2005.
[46] C. Sun, C.Y. Chan, and A.K. Goenka, "Multiway SLCA-Based Keyword Search in XML Data," *Proc. Int'l Conf. World Wide Web (WWW),* pp. 1043-1052, 2007.
[47] Y. Xu and Y. Papakonstantinou, "Efficient Keyword Search for Smallest LCAs in XML Databases," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 527-538, 2005.
[48] Y. Xu and Y. Papakonstantinou, "Efficient LCA Based Keyword Search in XML Data," *Proc. Int'l Conf. Extending Database Technology (EDBT),* pp. 535-546, 2008.
[49] B. Yu, G. Li, K. Sollins, and A.K.H. Tung, "Effective Keyword-Based Selection of Relational Databases," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 139-150, 2007.

Jianhua Feng received the BS, MS, and PhD degrees in computer science and technology from Tsinghua University. He is currently working as a professor in the Department of Computer Science and Technology in Tsinghua University. His main research interests include native XML database, data mining, and keyword search over structure and semistructure data. He has published papers in top conferences and top journals. He is a member of the IEEE and a senior member of China Computer Federation (CCF).

Guoliang Li received the PhD degree in computer science from Tsinghua University, Beijing, China, in 2009. Since then, he has worked as an assistant professor in the Department of Computer Science and Technology, Tsinghua University. His research interests mainly include integrating databases and information retrieval, data cleaning, and data integration. He has published papers in top conferences and journals, such as ACM SIGMOD, VLDB, IEEE ICDE, WWW, *VLDB Journal*, and *IEEE TKDE*.

Jianyong Wang received the PhD degree in computer science in 1999 from the Institute of Computing Technology, Chinese Academy of Sciences. He is currently an associate professor in the Department of Computer Science and Technology, Tsinghua University, Beijing, China. He was ever an assistant professor at Peking University, and visited Simon Fraser University, University of Illinois at Urbana-Champaign, and the University of Minnesota at Twin Cities. His research interests mainly include data mining and knowledge discovery, and web information management. He has coauthored more than 40 research papers in leading international conferences and top international journals. He is a senior member of the IEEE.