Information Systems 36 (2011) 248–266

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

An effective 3-in-1 keyword search method over heterogeneous data sources

Guoliang Li [a,*], Jianhua Feng [a], Beng Chin Ooi [b], Jianyong Wang [a], Lizhu Zhou [a]

[a] *Department of Computer Science, Tsinghua University, Beijing 100084, China*
[b] *School of Computing, National University of Singapore, 117543 Singapore, Singapore*

ARTICLE INFO

Keywords:
Keyword search
Inverted index
Extended inverted index
Graph index
Ranking
Unstructured data
Semi-structured data
Structured data

ABSTRACT

Conventional keyword search engines are restricted to a given data model and cannot easily adapt to unstructured, semi-structured or structured data. In this paper, we propose an efficient and adaptive keyword search method, called EASE, for indexing and querying large collections of heterogeneous data. To achieve high efficiency in processing keyword queries, we first model unstructured, semi-structured and structured data as graphs, and then summarize the graphs and construct graph indices instead of using traditional inverted indices. We propose an extended inverted index to facilitate keyword-based search, and present a novel ranking mechanism for enhancing search effectiveness. We have conducted an extensive experimental study using real datasets, and the results show that EASE achieves both high search efficiency and high accuracy, and outperforms the existing approaches significantly.

1. Introduction

Keyword search is a proven and widely popular mechanism for querying document systems and the World Wide Web. Recently, it has even been extensively applied to extract useful and relevant information from the Internet. Furthermore, the database (DB) research community has also recognized the benefits of keyword search and has been introducing keyword search capability into relational DBs [1–8], XML DBs [9–16] and graph DBs [17–19]. However, the existing web search engines cannot integrate information from multiple interrelated pages to answer keyword queries meaningfully. Next-generation web search engines require *link-awareness*, or more generally, the capability of integrating correlative information items that are linked through hyperlinks. Meanwhile, the efficiency of keyword search on structured and semi-structured data remains a challenging problem. This is so because the traditional approaches have always employed the inverted index to process keyword queries, which is effective for unstructured data but inefficient for semi-structured and structured data. This is because the inverted index is inadequate for identifying the "`best`" answers with complex structural information, which is rather rich in XML documents or relational DBs.

To the best of our knowledge, very few existing studies could be universally applied to unstructured data (e.g., text documents), semi-structured data (e.g., XML documents), structured data (e.g., relational DBs) and graph data. Therefore, providing both effective and efficient search ability over such heterogeneous collections within a single search engine remains a big challenge. As it is, the structure of the data, such as the potentially hierarchical embedding in XML documents, is not fully exploited for answering keyword queries. It is also not taken into account for result ranking in most search engines. Consequently, current implementations focus on either IR-style search to meaningfully rank the results but ignore the rich structural information, or DB-style search to discover answers by identifying structural relationships but employ a very straightforward ranking mechanism.

* Corresponding author. Tel.: +86 10 62789150; fax: +86 10 62771138.
E-mail address: liguoliang@tsinghua.edu.cn (G. Li).

0306-4379/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.is.2008.08.001

This less-than-ideal situation calls for a framework for indexing and querying over large collections of unstructured, semi-structured or structured data, and adaptive ranking of the results retrieved over those heterogeneous data. In this paper, we propose EASE, an Efficient and Adaptive keyword SEarch method, as an attempt in that direction. Our work is in line with the current trend of seamlessly integrating DBs and information retrieval (IR) techniques [20,21]. EASE seamlessly integrates efficient query evaluation and adaptive scoring for ranking results. From the DB point of view, EASE provides an efficient algorithmic basis for scalable *top-k*-style processing of large amounts of heterogeneous data for the discovery of rich structural relationships. It works by employing an adaptive, efficient and novel index beyond the inverted index. From the IR viewpoint, EASE integrates an effective ranking mechanism to improve search effectiveness.

In our approach, we model unstructured, semi-structured and structured data as graphs, with nodes being documents, elements and tuples, respectively, and edges being hyperlinks, parent–child relationships (or IDREFS) and primary–foreign-key relationships, respectively. We enable efficient keyword queries on these heterogeneous data by summarizing, clustering the graphs and constructing graph indices. To facilitate efficient keyword-based query processing, we examine the issues of indexing and ranking to improve search quality. To the best of our knowledge, this is the first attempt to efficiently and adaptively process keyword queries on such heterogeneous data, and also the first work to propose the novel graph index, which is efficient in identifying rich structural relationships.

Our contributions in this paper are as follows:

- We model unstructured, semi-structured and structured data as graphs and propose an efficient keyword search method, EASE, to adaptively process keyword queries over the heterogeneous data. We devise an effective graph index as opposed to the inverted index, to improve search efficiency and effectiveness.
- We propose a partition-based method to maintain the graph index so as to reduce the graph-index size.
- We present a novel ranking mechanism for effective keyword search by taking into account both the structural compactness of answers from the DB viewpoint and the textual relevancy from the IR point of view.
- We examine the issues of indexing and ranking, and devise a simple and yet efficient indexing mechanism to index the structural relationships between the transformed data. The index is amenable to the deployment of existing *top-k* ranking methods.
- We have conducted an extensive performance study using real datasets and various queries with different characteristics. The results show that EASE achieves both high search efficiency and accuracy, and outperforms existing state-of-the-art methods.

The rest of this paper is organized as follows. We present the *r-radius Steiner graph problem* in Section 2. Section 3 introduces a novel graph index. We present a novel scoring function in Section 4. We examine the issues of indexing and ranking, and propose an indexing mechanism in Section 5. Extensive experimental evaluations are provided in Section 6. We review the related work in Section 7 and conclude the paper with Section 8.

2. Adaptive keyword search model

2.1. Motivation

2.1.1. Unstructured data

Although many prior studies of keyword search over text documents (e.g., HTML documents) have been proposed, they all produce a list of individual pages as results. In the event that there are no pages that contain all the keywords, they will return pages with some of the input keywords ranked by relevancy. Even if two or more interrelated pages contain all the keywords, the existing methods cannot integrate the pages into one relevant and meaningful answer. For example, to search for conferences covering the topic of "`Data Integration`" held in "Canada" in 2008, one may issue a keyword query of "`Conference 2008 Canada Data Integration`" to a search engine such as Google. As we all know, the venue of "`SIGMOD 2008`" is "`Canada`" and "`Data Integration`" is one of its major research topics. Yet surprisingly, the homepage of "`SIGMOD 2008`" is neither in the top 10 results nor even in the first 100 answers. Why? The reason is that "`SIGMOD 2008`" homepage splits its information into several pages methodically as shown in Fig. 1 and IMPORTANT-DATE page contains the keywords "`2008, Conference`" while "`Data Integration`" is contained in the CALL-FOR-PAPER page. Such data lineage problem also persists in most recently proposed community information management platforms [22].

Consequently, the existing search engines often include a number of false negatives due to the limitation of their models, which take only a list of individual pages as search results but neglect the fact that interrelated pages linked by hyperlinks may be more meaningful. Yet, this is not an *ad hoc* problem but a ubiquitous one over the Internet. As another example, most researchers organize their homepages according to content, as shown in Fig. 2. Suppose a user searches for professors who teach a specific course and have a specified project, and types in some keywords. Although there may be no page that

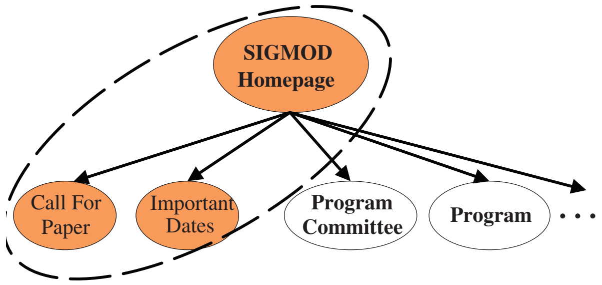


Fig. 1. Conference homepages.

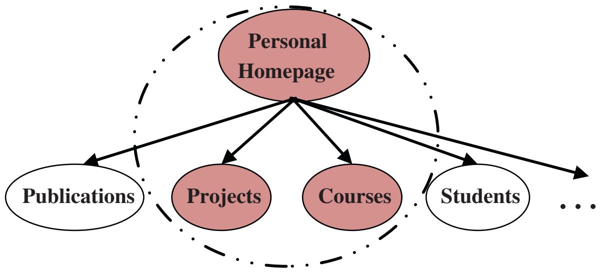


Fig. 2. Personal homepages.

contains all input keywords, the page units composed of HOMEPAGE, PROJECT and COURSE of some professors may answer this query meaningfully.

The accuracy and lineage of data have recently received considerable attention, but mainly from a theoretical perspective. While existing studies such as Trio [23] extend conventional data management by incorporating accuracy and lineage as integral components of both data and queries, few works are *link aware* to search text documents over the Internet and take into consideration the fact that multiple interrelated pages linked by hyperlinks may be more meaningful. Indeed, the accuracy problem of traditional search engines is ubiquitous over the Internet. This is so because most web sites organize their data systematically and relevant data may be separated into different pages but linked through hyperlinks. Although Li et al. [24] have proposed a method of retrieving and organizing web pages by "Information Unit", they model the problem of retrieving web pages as the minimum-weighted group Steiner tree problem, which is an NP-hard problem. It is rather difficult to identify Steiner trees over large graphs. Also, their method employs a heuristic method to identify the *top-k* answers, and it may fall into the local optimal point and fail to reach the global optimum. In this paper, we propose an effective search method, EASE, to address the problem.

2.1.2. *Semi-structured, structured and graph data*

Traditional studies of keyword search over semi-structured data always compute LCAs (lowest common ancestors) or its variants [10,16] of *content nodes* which directly contain input keywords and take the subtrees rooted at LCAs as answers. However, the problem as to which subtrees are more meaningful for answering the keyword queries remains open. This is because it is not a straightforward task to decide which subtrees with meaningful and complementary elements besides the content nodes should be used as results in order to meaningfully expand their answerability.

Prior works [2] of keyword search over structured data always identify connected trees with minimal cost in the labeled graph as answers. Called *Steiner trees*, they have nodes which are tuples in the DB and links which are primary–foreign-key relationships. The trees are identified with the use of an approximation to the Steiner tree problem. However, it is fairly difficult to extract all the Steiner trees in a large graph, which is NP-hard [2]. Moreover, the Steiner tree problem is difficult to adapt for complicated graph DBs [17], e.g., complex biological DBs, as it only discovers simple tree structures but cannot identify the more meaningful graph structures with rich structural relationships, such as circles. Although Guo et al. [17] have proposed data topology search to improve search effectiveness, their method is constrained by the input of only two keywords.

Traditionally, the inverted index is employed to answer keyword queries. It has been shown to be effective for text and document-based retrieval. However, it is inadequate for supporting keyword queries over structured, semi-structured and graph data because it is fairly difficult to identify the "`best`" answers that capture rich structural relationships through the inverted index. To address the above-mentioned issues, we propose an effective graph index to improve search performance in this paper.

2.2. *r-Radius Steiner graph problem*

EASE models unstructured data (e.g., text documents), semi-structured data (e.g., XML documents) and structured data (e.g., relational DBs) as graphs, where the nodes are, respectively, documents, elements and tuples, and the edges are, respectively, hyperlinks,[1] parent–child relationships (or IDREFS) and primary–foreign-key relationships. The advantage is obvious as EASE addresses the problem of keyword search over graph data so as to adaptively answer keyword queries over the heterogeneous data. We now formally define the problem of modeling heterogeneous data as graphs.

Inspired by the Steiner tree problem [2], we introduce the *Steiner graph problem*. However, graphs with a larger diameter (which is defined as the longest distance between any two nodes in a graph) are not so meaningful and relevant to queries as users are generally frustrated by large and complex graphs. Consequently, we introduce the *r-radius Steiner graph problem*, which is a more interesting and challenging problem of identifying meaningful Steiner graphs with acceptable sizes. To formally describe this problem, we first present several concepts as follows.

Definition 1 (*Centric distance*). Given graph $\mathcal{G}$ and any node v in $\mathcal{G}$, the centric distance of v, denoted as $\mathcal{CD}(v)$, is the maximal value among the distances between v and any node u in $\mathcal{G}$, i.e., $\mathcal{CD}(v) = \max_{u \in \mathcal{G}}\{\mathcal{D}(v,u)\}$, where $\mathcal{D}(v,u)$ denotes the distance between v and u, i.e., the length of the shortest path between v and u.

Definition 2 (*Radius*). The radius of a graph $\mathcal{G}$, denoted as $\mathcal{R}(\mathcal{G})$, is the minimal value among the centric distances of every node in $\mathcal{G}$, i.e., $\mathcal{R}(\mathcal{G}) = \min_{v \in \mathcal{G}}\{\mathcal{CD}(v)\}$. $\mathcal{G}$ is called an r-radius graph if the radius of $\mathcal{G}$ is exactly r.

Definition 3 (r-*Radius Steiner graph*). Given an r-radius graph $\mathcal{G}$ and a keyword query $\mathcal{K}$. Node ω in $\mathcal{G}$ is called a content node if ω directly contains some input keywords in $\mathcal{K}$. Node s in $\mathcal{G}$ is called a Steiner node if there exist two

[1] There may be a large number of hyperlinks among pages on the Web; however, to extract more meaningful and relevant answers, we can only consider the hyperlinks between pages in the same domain.

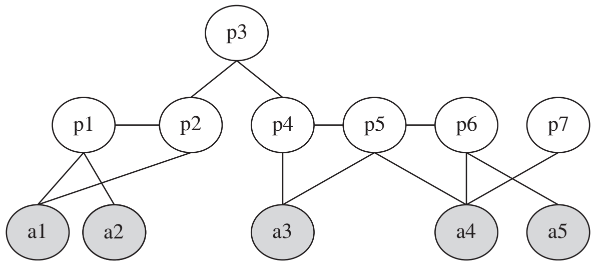


Fig. 3. The graph model for the publication database in Table 1.

content nodes, u and v, and s is on the path $u \leftrightsquigarrow v$ (s may be u or v), where $u \leftrightsquigarrow v$ denotes a path between u and v. The subgraph of $\mathscr{G}$ composed of the Steiner nodes and associated edges is called an r-radius Steiner graph. The radius of an r-radius Steiner graph may be smaller than r but cannot be larger than r.

Obviously, we can take the r-radius graphs that contain all or a portion of the input keywords as answers, as the r-radius graphs are very compact and meaningful, and also contain some relevant and complementary nodes for expanding their answerability. Moreover, r-radius Steiner graphs are more concise since non-Steiner nodes are excluded. Although we can take either r-radius graphs or r-radius Steiner graphs as answers of keyword search over graph data, we adopt the latter option in this paper.

However, it is very difficult to identify all r-radius Steiner graphs from a large graph, and hence, we will introduce an effective index later to improve the efficiency of extracting r-radius Steiner graphs. Here, to better explain our proposal, we give a running example as described in Example 1.

Example 1. Consider the DB in Table 1. We model it to a graph $\mathscr{G}$ as illustrated in Fig. 3. $\mathscr{D}(a1,a3)=4$, $\mathscr{CD}(p2)=5$ and $\mathscr{R}(\mathscr{G})=4$. Given a query "`IR`, `Hristidis`" on the DB in Table 1, we compute the Steiner graph composed of $p4$, $p5$ and $a3$ with associated edges between them as an answer. This differs from the Steiner tree (i.e., $p5-a3$) of prior studies, which can cause the loss of meaningful information, especially in DBs with complicated structures.

We now formally state *the r-radius Steiner graph problem* for identifying the most relevant subgraphs with acceptable sizes to answer keyword queries over graph data.

The r-radius Steiner graph problem: *Given a graph $\mathscr{G}$ and an input keyword query $\mathscr{K}$, the* r-radius Steiner graph problem *is to find all the r-radius Steiner graphs in $\mathscr{G}$, which contain all or a portion of the input keywords in $\mathscr{K}$, ranked by relevancy with $\mathscr{K}$.*

As users are usually interested in the *top-k* answers, we mainly discuss how to identify *top-k* r-radius Steiner graphs with the highest scores.

3. EASE: an effective and adaptive search method

The efficiency and advantages of using inverted indices for facilitating the computation of the "`best`" answers for online keyword queries are well recognized. However, the inverted indices are not effective for discovering the much richer structural relationships existing in DBs with complicated structures [17]. It is therefore important to be able to efficiently and effectively discover these structural relationships, and index them for fast and accurate response. Intuitively, a straightforward way is to enumerate all the combinations of keywords, compute the corresponding r-radius Steiner graphs for each combination, and index these graphs. However, it is prohibitively expensive to discover all these structures since the number of combinations of all keywords in real DBs is very large.

Consequently, we propose an effective strategy to discover a portion of the r-radius graphs such that the number of which is proportional to the number of nodes in the graph, and we only need to index and materialize these graphs. More importantly, all of the r-radius graphs can be effectively identified through the indexed ones. In a later section, we will address the issue of extracting r-radius Steiner graphs on the fly by removing non-Steiner nodes from the corresponding indexed r-radius graphs.

3.1. Adjacency matrix

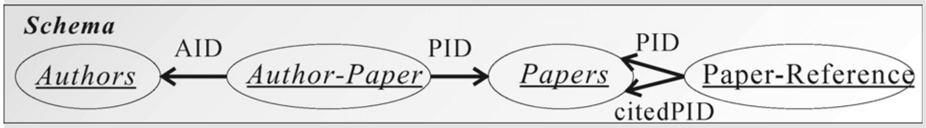
In order to efficiently extract r-radius graphs from a given graph $\mathscr{G}(\mathscr{R}(\mathscr{G}) \geqslant r)$,[2] we introduce the concept of *adjacency matrix*, $\mathscr{M}=(m_{ij})_{n\times n}$, with respect to $\mathscr{G}$, which is an $n\times n$ boolean matrix. In $\mathscr{M}$, the element m_{ij} is 1, iff, there is an edge between v_i and v_j in $\mathscr{G}$, i.e., $v_i \leftrightsquigarrow^1 v_j$, where $v_i(v_j)$ denotes the node at the i-th(j-th) row or column in $\mathscr{M}$ while $v_i \leftrightsquigarrow^d v_j$ denotes that there is a path between v_i and v_j with distance no larger than d; otherwise, m_{ij} is 0 (m_{ii} is always 1). Iteratively, $m^r_{ij}=1$, iff, $v_i \leftrightsquigarrow^r v_j$, where $\mathscr{M}^r = \overbrace{\mathscr{M}\times\cdots\times\mathscr{M}}^{r} = (m^r_{ij})_{n\times n}$.

To ease the discussion that follows, we summarize the notation we use in Table 2. $\mathscr{M}^r$ is said to be the r-th power of $\mathscr{M}$. $\mathscr{N}^r_i=\{v_j|\mathscr{M}^r_{ij}=1\}$ is the set of nodes which have a path to v_i with distance no larger than r. $\mathscr{G}^r_i$ denotes the subgraph of $\mathscr{G}$ with respect to the i-th row of $\mathscr{M}^r$, which is composed of the nodes in $\mathscr{N}^r_i$ and the associated edges. $\mathscr{G}^r_{v_i}(\mathscr{N}^r_{v_i})$ can be interchangeably employed instead of $\mathscr{G}^r_i(\mathscr{N}^r_i)$ if there is no ambiguity. We use $\mathscr{G}_i \unlhd \mathscr{G}_j$ to denote that $\mathscr{G}_i$ is a subgraph of $\mathscr{G}_j$ (equivalently, $\mathscr{G}_j$ is also called a super-graph of $\mathscr{G}_i$). $\mathscr{G}_i \lhd \mathscr{G}_j$ denotes that $\mathscr{G}_i$ is a proper subgraph of $\mathscr{G}_j$, i.e., $\mathscr{G}_i \lhd \mathscr{G}_j$ and $\mathscr{G}_i \neq \mathscr{G}_j$. $|\mathscr{G}|$ denotes the number of nodes in $\mathscr{G}$.

Example 2. We can construct the adjacency matrix of the graph in Fig. 3 as illustrated in Table 3. We note that $v_1=a_1$, $v_2=p_1$, $v_3=a_2$ and $v_4=p_2$. $\mathscr{N}^2_2=\mathscr{N}^2_{p_1}=\{a_1,p_1,a_2,p_2,p_3\}$ and $\mathscr{N}^2_4=\mathscr{N}^2_{p_2}=\{a_1,p_1,a_2,p_2,p_3,p_4\}$.

[2] Without loss of generality, we suppose $\mathscr{G}$ is a connected graph, the radius of which is no smaller than r, i.e., $\mathscr{R}(\mathscr{G}) \geqslant r$. Even if $\mathscr{G}$ is an unconnected graph, we can decompose it into a set of connected graphs and thus EASE can adapt to the unconnected graphs.

Table 1
A publication database



Authors

AID	Name
a1	J. Shanmugasundaram
a2	L. Guo
a3	V. Hristidis
a4	Y. Papakonstantinou
a5	A. Balmin

Paper-reference

PID	citedPID
p1	p2
p2	p3
p3	p4
p4	p5
p5	p6

Author-paper

AID	PID
a1	p1
a1	p2
a2	p1
a3	p4
a3	p5
a4	p5
a4	p6
a4	p7
a5	p6

Papers

PID	Title
p1	Topology search over biological databases
p2	XRANK: ranked keyword search over XML documents
p3	Bidirectional expansion for keyword search on graphs
p4	Finding *top-k* answers in keyword proximity search
p5	Efficient IR-style keyword search over relational databases
p6	Keyword proximity search on XML graphs
p7	DISCOVER: keyword search in relational databases

Table 2
Summary of notations

Notation	Descriptions
$\mathcal{G}$	A graph
$\vert\mathcal{G}\vert$	The number of nodes in $\mathcal{G}$
$\mathcal{SG}$	A Steiner graph
$\mathcal{K}$	An input keyword query
$\mathcal{M}$	The adjacency matrix w.r.t. $\mathcal{G}$
$\mathcal{M}^r$	The r-th power of $\mathcal{M}$
$\mathcal{N}_i^r$ (or $\mathcal{N}_{v_i}^r$)	$\{v_j \vert \mathcal{M}_{ij}^r = 1\}$
$\mathcal{G}_i^r$ (or $\mathcal{G}_{v_i}^r$)	The subgraph of $\mathcal{G}$ composed of the nodes in $\mathcal{N}_i^r$
$v_i \leftrightsquigarrow v_j$	There is a path between v_i and v_j
$\mathcal{D}(v_i, v_j)$	The distance of v_i and v_j
$\mathcal{CD}(v)$	$\max_{u\in\mathcal{G}}\{\mathcal{D}(v,u)\}$
$\mathcal{R}(\mathcal{G})$	The radius of $\mathcal{G}$ and $\mathcal{R}(\mathcal{G}) = \min_{v\in\mathcal{G}}\{\mathcal{CD}(v)\}$
$v_i \leftrightsquigarrow^d v_j$	$v_i \leftrightsquigarrow v_j$ and $\mathcal{D}(v_i, v_j) \leqslant d$
$\mathcal{G}_i \trianglelefteq \mathcal{G}_j$	$\mathcal{G}_i$ is a subgraph of $\mathcal{G}_j$ (or $\mathcal{G}_j$ is a super-graph of $\mathcal{G}_i$)
$\mathcal{G}_i \lhd \mathcal{G}_j$	$\mathcal{G}_i \trianglelefteq \mathcal{G}_j$ and $\mathcal{G}_i \neq \mathcal{G}_j$
$\mathcal{S}(\mathcal{G}_i, \mathcal{G}_j)$	The similarity between $\mathcal{G}_i$ and $\mathcal{G}_j$

$\mathcal{N}_{p_1}^2$ and $\mathcal{N}_{p_2}^2$ are composed of the nodes that can connect to p_1 and p_2 within distance 2, respectively. We can construct two subgraphs centered at nodes p_1 and p_2 by connecting the nodes in the two node sets $\mathcal{N}_{p_1}^2$ and $\mathcal{N}_{p_2}^2$. The two subgraphs $\mathcal{G}_2^2(\mathcal{G}_{p_1}^2)$ and $\mathcal{G}_4^2(\mathcal{G}_{p_2}^2)$ are illustrated in Fig. 4. We can see that the two graphs are compact within distance 2. We have $\mathcal{G}_{p_1}^2 \lhd \mathcal{G}_{p_2}^2$. In the remainder of this paper, we set r as 2 for all the running examples.

To effectively extract r-radius graphs according to the adjacency matrix, we provide Lemma 1 to determine the subgraphs in $\mathcal{G}$ that are r-radius graphs.

Lemma 1. *Given a graph $\mathcal{G}(\mathcal{R}(\mathcal{G}) \geqslant r > 1)$, $\forall i,\ 1 \leqslant i \leqslant |\mathcal{G}|$, $\mathcal{G}_{v_i}^r$ is an r-radius graph, if, $\forall v_k \in \mathcal{N}_{v_i}^r$, $\mathcal{N}_{v_i}^r \nsubseteq \mathcal{N}_{v_k}^{r-1}$.*

Proof. As all the nodes in $\mathcal{N}_{v_i}^r$ connect to v_i and all the nodes with distances no larger than r to v_i are in $\mathcal{N}_{v_i}^r$, the nodes in $\mathcal{N}_{v_i}^r$ and the edges associated with them can construct a subgraph of $\mathcal{G}$, $\mathcal{G}_{v_i}^r$. We need to prove $\mathcal{R}(\mathcal{G}_{v_i}^r) = r$.

We first prove that $\mathcal{R}(\mathcal{G}_{v_i}^r) \leqslant r$.

$\forall u \in \mathcal{N}_{v_i}^r$, we have $\mathcal{D}(v_i, u) \leqslant r$ according to the definition of $\mathcal{M}^r$, and thus $\mathcal{CD}(v_i) = \max_{u\in\mathcal{G}_{v_i}^r}\{\mathcal{D}(v_i,u)\} \leqslant r$. Hence, based on Definition 2, we have $\mathcal{R}(\mathcal{G}_{v_i}^r) = \min_{v\in\mathcal{G}_{v_i}^r}\{\mathcal{CD}(v)\} \leqslant \mathcal{CD}(v_i) \leqslant r$.

We then prove that $\mathcal{R}(\mathcal{G}_{v_i}^r) \geqslant r$.

$\forall v_k \in \mathcal{N}_{v_i}^r$, as $\mathcal{N}_{v_i}^r \nsubseteq \mathcal{N}_{v_k}^{r-1}$, there must exist a node u_k, $u_k \in \mathcal{N}_{v_i}^r$ and $u_k \notin \mathcal{N}_{v_k}^{r-1}$, thus $\mathcal{D}(v_k, u_k) \geqslant r$ (otherwise, if $\mathcal{D}(v_k, u_k) < r$, $u_k \in \mathcal{N}_{v_k}^{r-1}$, which contradicts $u_k \notin \mathcal{N}_{v_k}^{r-1}$). Thus, $\forall v_k \in \mathcal{N}_{v_i}^r$, we have $\mathcal{CD}(v_k) = \max_{u\in\mathcal{G}_{v_i}^r}\{\mathcal{D}(v_k,u)\} \geqslant \mathcal{D}(v_k,u_k) \geqslant r$ in $\mathcal{G}_{v_i}^r$. Accordingly, $\mathcal{R}(\mathcal{G}_{v_i}^r) = \min_{v_k\in\mathcal{G}_{v_i}^r}\{\mathcal{CD}(v_k)\} \geqslant r$.

Therefore, $\mathcal{R}(\mathcal{G}_{v_i}^r) = r$ and $\mathcal{G}_{v_i}^r$ is an r-radius graph. □

As formalized in Lemma 1, we can determine whether the subgraph $\mathcal{G}_{v_i}^r$ with respect to the i-th row of $\mathcal{M}^r$ is

Table 3
Adjacency matrix of the graph for the publication database

	a1	p1	a2	p2	p3	p4	a3	p5	a4	p6	p7	a5
(a) $\mathscr{M}$												
a1	1	1	0	1	0	0	0	0	0	0	0	0
p1	1	1	1	1	0	0	0	0	0	0	0	0
a2	0	1	1	0	0	0	0	0	0	0	0	0
p2	1	1	0	1	1	0	0	0	0	0	0	0
p3	0	0	0	1	1	1	0	0	0	0	0	0
p4	0	0	0	0	1	1	1	1	0	0	0	0
a3	0	0	0	0	0	1	1	1	0	0	0	0
p5	0	0	0	0	0	1	1	1	1	1	0	0
a4	0	0	0	0	0	0	0	1	1	1	1	0
p6	0	0	0	0	0	0	0	1	1	1	0	1
p7	0	0	0	0	0	0	0	0	1	0	1	0
a5	0	0	0	0	0	0	0	0	0	1	0	1
(b) $\mathscr{M}^2$												
a1	1	1	1	1	1	0	0	0	0	0	0	0
p1	1	1	1	1	1	0	0	0	0	0	0	0
a2	1	1	1	1	0	0	0	0	0	0	0	0
p2	1	1	1	1	1	1	0	0	0	0	0	0
p3	1	1	0	1	1	1	1	1	0	0	0	0
p4	0	0	0	1	1	1	1	1	1	1	0	0
a3	0	0	0	0	1	1	1	1	1	1	0	0
p5	0	0	0	0	1	1	1	1	1	1	1	1
a4	0	0	0	0	0	1	1	1	1	1	1	1
p6	0	0	0	0	0	1	1	1	1	1	1	1
p7	0	0	0	0	0	0	0	1	1	1	1	0
a5	0	0	0	0	0	0	0	1	1	1	0	1

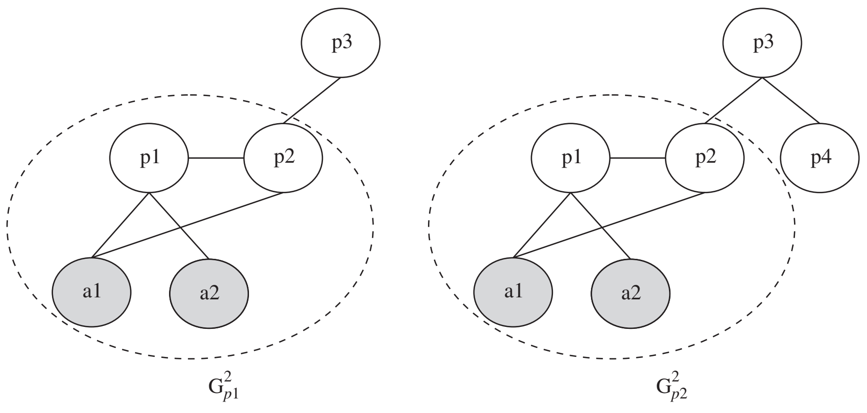


Fig. 4. Two 2-radius graphs.

an r-radius graph. For example, $\mathscr{G}^2_{p_2}$ is a 2-radius graph while $\mathscr{G}^2_{a_2}$ is not as $\mathscr{N}^2_{a_2} \subseteq \mathscr{N}^1_{p_1}$ as shown in Table 3.

In order to extract all of the r-radius Steiner graphs from $\mathscr{G}$, we must prove that any r-radius Steiner graph in $\mathscr{G}$ corresponds to a subgraph $\mathscr{G}^r_{v_i}$, and we provide Theorem 1 for extracting r-radius Steiner graphs.

Theorem 1. *Suppose* $\{\mathscr{SG}^r_1, \mathscr{SG}^r_2, \ldots, \mathscr{SG}^r_p\}$ *is the set of the r-radius Steiner graphs with respect to a graph* $\mathscr{G}(\mathscr{R}(\mathscr{G}) \geqslant r)$ *and a keyword query*, $\forall 1 \leqslant j \leqslant p$, $\exists 1 \leqslant i \leqslant |\mathscr{G}|$, $\mathscr{SG}^r_j \trianglelefteq \mathscr{G}^r_{v_i}$ *holds.*

Proof. $\forall \mathscr{SG}^r_j$, there must exist a corresponding r-radius graph $\mathscr{G}_{r_j}$ according to Definition 3, such that $\mathscr{SG}^r_j \trianglelefteq \mathscr{G}_{r_j}$. We then prove that $\exists \mathscr{G}^r_{v_i}$, $\mathscr{G}_{r_j} \trianglelefteq \mathscr{G}^r_{v_i}$.

As $\mathscr{G}_{r_j}$ is an r-radius graph, there must exist a node $v_i \in \mathscr{G}_{r_j}$ and $\mathscr{CD}(v_i) = r$. Since for any node $u \in \mathscr{G}_{r_j}$, $\mathscr{D}(v_i, u) \leqslant r$, thus u must be in the node set of $\mathscr{G}^r_{v_i}$ (i.e., $\mathscr{N}^r_{v_i}$) according to the definitions of $\mathscr{G}^r_{v_i}$ and $\mathscr{M}^r$. Hence, all the nodes in $\mathscr{G}_{r_j}$ must be in $\mathscr{G}^r_{v_i}$. Thus, $\mathscr{G}_{r_j} \trianglelefteq \mathscr{G}^r_{v_i}$. Hence, $\mathscr{SG}^r_j \trianglelefteq \mathscr{G}_{r_j} \trianglelefteq \mathscr{G}^r_{v_i}$. □

Based on Theorem 1, we can extract r-radius Steiner graphs through the adjacency matrix. To facilitate efficient retrieval of r-radius graphs, we construct a novel *graph index*. The entries of the graph index are terms contained in the graph and each entry preserves the r-radius graphs that contain the term. To construct the graph index, we first extract r-radius graphs as stated in Lemma 1, then for each term k_i, we keep the set of all r-radius graphs that contain k_i, denoted as $\mathscr{I}_{k_i}$, i.e., $\mathscr{I}_{k_i} = \{\mathscr{G}^r_{v_j} | \mathscr{G}^r_{v_j}$ contains $k_i\}$.

To process a keyword query $\mathscr{K} = \{k_1, k_2, \ldots, k_m\}$, we first retrieve the set $\mathscr{I}_{k_i}$ of those r-radius graphs which contain k_i based on the graph index, and then union every $\mathscr{I}_{k_i}$ to compute $\bigcup_{i=1}^m \mathscr{I}_{k_i}$, which is the set of r-radius graphs that contain all or a portion of the keywords in $\mathscr{K}$.[3] Finally, we extract the r-radius Steiner graphs by removing the non-Steiner nodes from the corresponding r-radius graphs, and rank the results to return the *top-k* answers.

Given an r-radius graph $\mathscr{G}^r$ and its *content nodes*, $c_1, c_2, \ldots, c_q$, which directly contain some of the input keywords, we compute the Steiner nodes and construct the corresponding r-radius Steiner graph as follows:

(i) Compute $\mathscr{P}(c_i)$, the set of those nodes which have a path to c_i in $\mathscr{G}_i$, i.e., $\mathscr{P}(c_i) = \{u | u \leftrightsquigarrow c_i \text{ in } \mathscr{G}_i\}$, where $\mathscr{G}_i$ is the subgraph of $\mathscr{G}^r$, by removing the nodes in $\{c_1, c_2, \ldots, c_{i-1}, c_{i+1}, \ldots, c_q\}$ and the associated edges.
(ii) Extract the set of Steiner nodes in $\mathscr{G}^r$, $\mathscr{P}$, where $\mathscr{P} = \bigcup_{i=1}^q \bigcup_{j=i+1}^q (\mathscr{P}(c_i) \cap \mathscr{P}(c_j)) \cup \{c_1, c_2, \ldots, c_q\}$.
(iii) Construct the r-radius Steiner graph, i.e., the subgraph of $\mathscr{G}^r$, which is composed of the nodes in $\mathscr{P}$ and the associated edges.

Example 3. Consider the DB described in Table 1, we obtain its adjacency matrix $\mathscr{M}$ and compute $\mathscr{M}^r$ as shown in Table 3 (r is set to 2 in the remaining examples throughout this paper). Subsequently, we derive the r-radius graphs, e.g., $\mathscr{G}^r_{p1}$ and $\mathscr{G}^r_{p2}$ based on the second and fourth rows of $\mathscr{M}^r$, respectively, which are illustrated in Fig. 4. We note that $\mathscr{G}^r_{p1} \lhd \mathscr{G}^r_{p2}$. To answer the keyword query "`Shanmugasundaram, Guo, XRANK`", we first retrieve the r-radius graphs, $\mathscr{G}^r_{p1}$ and $\mathscr{G}^r_{p2}$, which contain the three keywords, based on the graph index, and then extract the corresponding r-radius Steiner graphs, i.e., the circled subgraphs as illustrated in Fig. 4, by removing the non-Steiner nodes.

From Example 3, we observe that some r-radius graphs are contained in others. As a further illustration, we get the following expressions from the graph in Fig. 3:

(i) $\mathscr{G}^r_{a2} \lhd \mathscr{G}^r_{a1} = \mathscr{G}^r_{p1} \lhd \mathscr{G}^r_{p2}$;
(ii) $\mathscr{G}^r_{a3} \lhd \mathscr{G}^r_{p4}$;
(iii) $\mathscr{G}^r_{a5} \lhd \mathscr{G}^r_{p6} = \mathscr{G}^r_{a4} \lhd \mathscr{G}^r_{p5}$;
(iv) $\mathscr{G}^r_{p7} \lhd \mathscr{G}^r_{p6} = \mathscr{G}^r_{a4} \lhd \mathscr{G}^r_{p5}$.

Consequently, it is sufficient for us just to keep the graphs $\mathscr{G}^r_{p2}$, $\mathscr{G}^r_{p3}$, $\mathscr{G}^r_{p4}$ and $\mathscr{G}^r_{p5}$ in the graph index. We shall address this problem next.

[3] If we consider "AND" predicate, we merge $\mathscr{I}_{k_i}$ to compute $\bigcap_{i=1}^m \mathscr{I}_{k_i}$, which is the set of r-radius graphs that contain all keywords.

3.2. *Maximal r-radius graph*

To avoid maintaining redundant overlapping r-radius graphs in the graph index, we introduce the concept of *maximal r-radius graph* in this section.

Definition 4 (*Maximal r-radius graph*). Given a graph $\mathscr{G}$ and an r-radius subgraph $\mathscr{G}^r_{v_i}$ in $\mathscr{G}$, $\mathscr{G}^r_{v_i}$ is called a maximal r-radius subgraph if there is no other r-radius subgraph that contains $\mathscr{G}^r_{v_i}$.

Based on Definition 4, we only need to keep the maximal r-radius graphs in the graph index to minimize storage without affecting the search results. This is because all other r-radius graphs can be reconstructed from their corresponding super-graphs. For example, $\mathscr{G}^r_{p2}$, $\mathscr{G}^r_{p3}$, $\mathscr{G}^r_{p4}$ and $\mathscr{G}^r_{p5}$ are maximal r-radius graphs and will be kept in the graph index while other graphs, e.g., $\mathscr{G}^r_{p1}$ and $\mathscr{G}^r_{a4}$, can be reconstructed when necessary.

In fact, the maximal r-radius graphs can be directly extracted from $\mathscr{M}^r$, as captured by Lemma 2.

Lemma 2. *Given a graph $\mathscr{G}(\mathscr{R}(\mathscr{G}) \geqslant r)$, $\forall i$, $1 \leqslant i \leqslant |\mathscr{G}|$, $\mathscr{G}^r_{v_i}$ is a maximal r-radius graph if $\forall k \in \{t | \mathscr{M}^r_{it} = 1,\ \mathscr{N}^r_t \neq \mathscr{N}^r_i\}$, $\mathscr{N}^r_i \not\subset \mathscr{N}^r_k$.*

Proof. We first prove that $\mathscr{G}^r_{v_i}$ is an r-radius graph.

(i) $\forall k \in \{t | \mathscr{M}^r_{it} = 1,\ \mathscr{N}^r_t \neq \mathscr{N}^r_i\}$, as $\mathscr{N}^r_i \not\subset \mathscr{N}^r_k$, there must exist u_k, $u_k \in \mathscr{N}^r_i$ and $u_k \notin \mathscr{N}^r_k$, and thus we have $\mathscr{D}(v_k, u_k) > r$ (otherwise, if $\mathscr{D}(v_k, u_k) \leqslant r$, u_k must be in $\mathscr{N}^r_k$, which contradicts $u_k \notin \mathscr{N}^r_k$). Hence, $\mathscr{CD}(v_k) = \max_{u \in \mathscr{G}^r_{v_i}} \{\mathscr{D}(v_k, u)\} \geqslant \mathscr{D}(v_k, u_k) > r$.
(ii) $\forall j \in \{t | \mathscr{M}^r_{it} = 1,\ \mathscr{N}^r_t = \mathscr{N}^r_i\}$, as $\mathscr{R}(\mathscr{G}) \geqslant r$, $\mathscr{CD}(v_j) = r$ in $\mathscr{G}^r_{v_i}$.

Thus, $\forall v_k \in \mathscr{G}^r_{v_i}$, $\mathscr{CD}(v_k) \geqslant r$ and $\mathscr{CD}(v_i) = r$. Hence, we have $\mathscr{R}(\mathscr{G}^r_{v_i}) = \min_{v_k \in \mathscr{G}^r_{v_i}} \{\mathscr{CD}(v_k)\} = r$, and $\mathscr{G}^r_{v_i}$ is an r-radius graph.

We then prove that $\mathscr{G}^r_{v_i}$ must be a maximal r-radius graph.

As $\forall k \in \{t | \mathscr{M}^r_{it} = 1, \mathscr{N}^r_t \neq \mathscr{N}^r_i\}$, $\mathscr{N}^r_i \not\subset \mathscr{N}^r_k$, there cannot exist another r-radius graph that is a super-graph of $\mathscr{G}^r_{v_i}$. Thus, $\mathscr{G}^r_{v_i}$ must be a maximal r-radius graph. □

3.3. *Subgraph maintenance*

In this section, we discuss how to maintain the maximal r-radius subgraphs. There are two possible ways to index the subgraphs. One straightforward method is to keep all the maximal r-radius subgraphs. Once we find the subgraphs that are relevant to a given input keyword query, we use them to answer the query. However, there are still some overlaps between different maximal r-radius graphs. For example, $\mathscr{G}^r_{p2}$ and $\mathscr{G}^r_{p3}$ both contain the nodes, a_1, p_1, p_2, p_3 and p_4. Thus, if there are a large number of subgraphs, the graph index is rather large, and it is expensive to maintain such a huge index. Another approach is that we can maintain the whole graph, and keep the centered node (v_i is called the centered node of

$\mathcal{G}^r_{v_i}$) for each maximal r-radius subgraph. Based on the centered node, we can reconstruct the maximal r-radius subgraph by traversing the whole graph from the centered node using the depth-first traversal. However, if the graph is very large, the memory cannot hold the graph. Conventional Steiner-tree based methods [2] typically assume that the graph can be held in memory. They traverse the graph initializing from the content nodes to identify the Steiner trees with the minimal cost. They have to maintain the whole graph in memory, which is not practical for large graphs. Moreover, it is expensive to identify the Steiner trees by traversing the whole graph.

To alleviate these problems, we propose a graph partitioning-based method to achieve the needed search efficiency and reduce the graph index size. If the graph is small, we can maintain the whole graph in the memory, and use centered node-based method, which is very efficient to identify the Steiner graphs. If the graph is very large, we partition the whole graph into some subgraphs. For each subgraph, we employ centered node-based method. In this way, we maintain the frequent used subgraphs in the memory and keep the other subgraphs on the disk. Based on the centered node, we can reconstruct the maximal Steiner subgraphs from the partitions. In this way, we can extend our method to support very large graphs using a disk-based method.

3.4. Graph partitioning

We cluster r-radius graphs so that we can partition the graph to facilitate identifying r-radius graphs. We first cluster the r-radius graphs and then partition the whole graph based on the clusters. Each cluster corresponds to a portion of the graph. Clustering maximal r-radius graphs has the following salient advantages: (i) We only need to maintain a physical graph for each cluster while all the maximal r-radius graphs only preserve their nodes instead of the corresponding overlapping graphs. This avoids the incurrence of huge storage, and is similar to the views on top of underlying physical tables in RDBMS, in which clusters correspond to physical tables while maximal r-radius graphs correspond to views. (ii) We only need to retrieve the corresponding relevant graph partitions instead of maintaining the whole graph in order to identify the r-radius graphs.

To meaningfully cluster r-radius graphs, we first define the similarity between any two graphs and then cluster the maximal r-radius graphs based on their similarities.

Definition 5 (*Graph similarity*). Given two maximal r-radius subgraphs $\mathcal{G}^r_i$ and $\mathcal{G}^r_j$ in a given graph, their graph similarity, $\mathcal{S}(\mathcal{G}^r_i, \mathcal{G}^r_j)$, is $|\mathcal{N}^r_i \cap \mathcal{N}^r_j| / |\mathcal{N}^r_i \cup \mathcal{N}^r_j|$, where $\mathcal{N}^r_i$ and $\mathcal{N}^r_j$ denote the node sets of $\mathcal{G}^r_i$ and $\mathcal{G}^r_j$, respectively.

A bigger overlap between the nodes of the two graphs implies a larger graph similarity between them, and consequently, a higher probability that they will be clustered together. It is obvious that the graph similarity scales well with the number of overlapping nodes. Moreover, given two maximal r-radius subgraphs, employing the number of their overlapping nodes is sound as the edges associated with the overlapping nodes are also the same. In addition, we note that the graph similarity preserves the following properties:

- *Symmetry*: $\mathcal{S}(\mathcal{G}_i, \mathcal{G}_j) = \mathcal{S}(\mathcal{G}_j, \mathcal{G}_i)$;
- *Positivity*: $0 \leqslant \mathcal{S}(\mathcal{G}_i, \mathcal{G}_j) \leqslant 1$, for any $\mathcal{G}_i$ and $\mathcal{G}_j$;
- *Reflexivity*: $\mathcal{S}(\mathcal{G}_i, \mathcal{G}_j) = 1$ iff $\mathcal{G}_i = \mathcal{G}_j$;

which indicate that graph similarity is a good metric to evaluate the similarity between any two graphs. Based on the graph similarity, we can now cluster the maximal r-radius subgraphs by employing an existing method such as the K-mean, K-medoids and EM algorithms.

To effectively compute the graph similarity between two maximal r-radius graphs, we introduce Lemma 3.

Lemma 3. *Given two maximal r-radius subgraphs w.r.t.* $\mathcal{M}^r$, $\mathcal{G}^r_i$ *and* $\mathcal{G}^r_j$, $\mathcal{S}(\mathcal{G}^r_i, \mathcal{G}^r_j) = |\{k|\mathcal{M}^r_{ik} = 1$ *and* $\mathcal{M}^r_{jk} = 1\}| / |\{k| \mathcal{M}^r_{ik} = 1$ *or* $\mathcal{M}^r_{jk} = 1\}|$.

Proof. We first prove that $\forall k$, if $k \in \{\mathcal{M}^r_{ik} = 1$ and $\mathcal{M}^r_{jk} = 1\}$, $v_k \in \mathcal{N}^r_i \cap \mathcal{N}^r_j$. As $\mathcal{M}^r_{ik} = 1$, $v_k \in \mathcal{N}^r_i$. As $\mathcal{M}^r_{jk} = 1$, $v_k \in \mathcal{N}^r_j$. Thus, $v_k \in \mathcal{N}^r_i \cap \mathcal{N}^r_j$. Similarly, if $v_k \in \mathcal{N}^r_i \cap \mathcal{N}^r_j$, then $k \in \{\mathcal{M}^r_{ik} = 1$ and $\mathcal{M}^r_{jk} = 1\}$.

We then prove that $\forall k$, if $k \in \{\mathcal{M}^r_{ik} = 1$ or $\mathcal{M}^r_{jk} = 1\}$, $v_k \in \mathcal{N}^r_i \cup \mathcal{N}^r_j$. As $\mathcal{M}^r_{ik} = 1$ or $\mathcal{M}^r_{jk} = 1$, $v_k \in \mathcal{N}^r_i$ or $v_k \in \mathcal{N}^r_j$. Thus, $v_k \in \mathcal{N}^r_i \cup \mathcal{N}^r_j$. Similarly, if $v_k \in \mathcal{N}^r_i \cup \mathcal{N}^r_j$, then $k \in \{\mathcal{M}^r_{ik} = 1$ or $\mathcal{M}^r_{jk} = 1\}$. □

Following Lemma 3, we can compute the graph similarity based on the adjacency matrix, which is much easier to manipulate than the original graphs.

Example 4. Consider the DB in Table 1, we note that the four graphs $\mathcal{G}^r_{p2}$, $\mathcal{G}^r_{p3}$, $\mathcal{G}^r_{p4}$ and $\mathcal{G}^r_{p5}$ are the maximal r-radius graphs. We can compute their graph similarities as shown in Table 4 based on the adjacency matrix in Table 3. If we cluster the four maximal r-radius graphs into two clusters, $\mathcal{G}^r_{p2}$ and $\mathcal{G}^r_{p3}$ will fall into the same cluster while $\mathcal{G}^r_{p4}$ and $\mathcal{G}^r_{p5}$ will be in another cluster as illustrated in Fig. 5. On the other hand, if we cluster them into three clusters, $\mathcal{G}^r_{p4}$ and $\mathcal{G}^r_{p5}$ will fall into the same cluster while $\mathcal{G}^r_{p2}$ and $\mathcal{G}^r_{p3}$ are in the other two clusters, respectively.

To summarize, given a graph, we first obtain its adjacency matrix $\mathcal{M}$ and compute $\mathcal{M}^r$. We then extract the maximal r-radius graphs according to Lemma 2 and compute the graph similarities between any two maximal r-radius graphs based on Lemma 3. Subsequently, we cluster the graphs by employing the existing K-means

Table 4
Graph similarities

	$\mathcal{G}^r_{p2}$	$\mathcal{G}^r_{p3}$	$\mathcal{G}^r_{p4}$	$\mathcal{G}^r_{p5}$
$\mathcal{G}^r_{p2}$	1	5/8	3/10	1/6
$\mathcal{G}^r_{p3}$	–	1	5/9	4/11
$\mathcal{G}^r_{p4}$	–	–	1	2/3
$\mathcal{G}^r_{p5}$	–	–	–	1

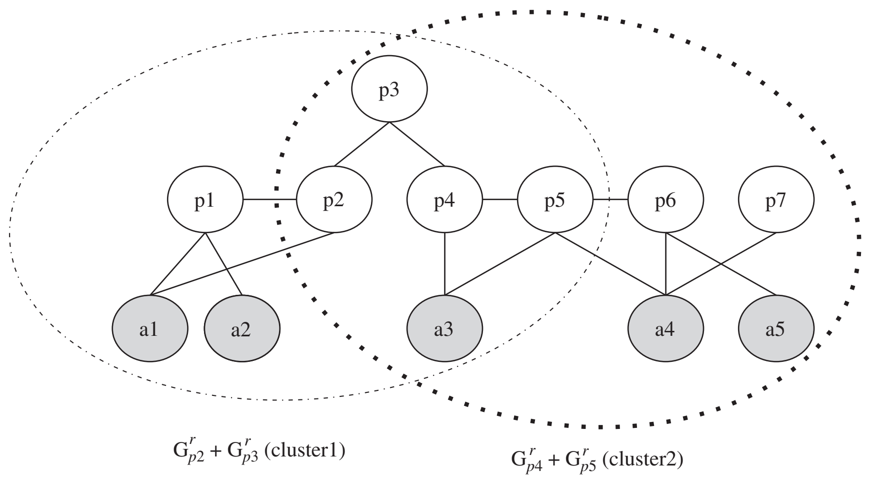

p3
p1
p2
p4
p5
p6
p7
a1
a2
a3
a4
a5
$G^r_{p2} + G^r_{p3}$ (cluster1)
$G^r_{p4} + G^r_{p5}$ (cluster2)

Fig. 5. Two clusters.

Table 5
Graph index

Terms	r-Radius graphs: $\mathscr{I}_{k_i}$
Database	$\mathscr{G}^r_{p_2}$, $\mathscr{G}^r_{p_3}$, $\mathscr{G}^r_{p_4}$, $\mathscr{G}^r_{p_5}$
DISCOVER	$\mathscr{G}^r_{p_5}$
Papakonstantinou	$\mathscr{G}^r_{p_4}$, $\mathscr{G}^r_{p_5}$
Relational	$\mathscr{G}^r_{p_3}$, $\mathscr{G}^r_{p_4}$, $\mathscr{G}^r_{p_5}$

algorithm and partition the graph. Finally, we construct the graph index to materialize the maximal r-radius graphs based on the graph partitions: we maintain each partition, and the centered nodes in each partition. To illustrate, we consider the example below, which computes the set of r-radius Steiner graphs that contain all or a portion of the input keywords.

Example 5. Given the DB in Table 1 and a keyword query "`DISCOVER`, `Relational`, `Database`, `Papakonstantinou`", we first retrieve the keyword lists based on the graph index (Table 5). We then derive the set of $\{\mathscr{G}^r_{p_2}, \mathscr{G}^r_{p_3}, \mathscr{G}^r_{p_4}, \mathscr{G}^r_{p_5}\}$, where each graph contains some input keywords. $\mathscr{G}^r_{p_5}$ contains all of the input keywords. Finally, we refine the maximal r-radius graphs on top of the corresponding graph partitions (instead of traversing the whole graph) to obtain the r-radius Steiner graphs. For instance, we can get $\mathscr{SG}^r_{p_5}$ by removing the non-Steiner nodes, e.g., $p3$, $p4$, $a3$ and $a5$, from $\mathscr{G}^r_{p_5}$, as shown in Fig. 6.

3.5. Update issue

In terms of update issues, we note that there is no need to re-index the whole graph from scratch. We can incrementally update the index as follow: (1) node relabeling: only the partitions that contain the relabeled node and the corresponding graph index are updated. Note that we only need to modify the entries that contain the label. (2) Node insertion: we only update the index corresponding to the partitions that have nodes connecting to the inserted node. We need not consider any other partitions. (3) Node deletion: which is the same as (2). (4) Edge insertion/deletion: which is the same as (2).

4. Ranking functions

In this section, we first discuss how to meaningfully rank r-radius Steiner graphs and identify the *top-k* answers based on the existing proposals. Next, we propose a new measure based on the structural compactness between content nodes and the structural relevancy between input keywords with respect to an r-radius Steiner graph.

4.1. IR ranking

4.1.1. TF · IDF-based ranking

The basic idea of the ranking method used in the existing literature, such as [5–7], is to first assign each r-radius graph a score using a standard IR-ranking formula (or its variants), and then combine the individual scores using a score aggregation function, such as SUM, to obtain the final score. For example, the TF · IDF-based IR-style ranking function weights an r-radius Steiner graph by considering textual relevancy in IR literature, which takes into account term frequency (*tf*), inverse document frequency (*idf*) and normalized document length (*ndl*). *tf* and *idf* are well employed to rank documents in the IR literature while *ndl* is used to normalize document length as a longer document has a higher likelihood to contain many more keywords. We can compute the three parameters as follows:

$$ntf_{(k_i,\mathscr{G})} = 1 + \ln(1 + \ln(1 + tf_{(k_i,\mathscr{G})})) \quad (1)$$

$$idf_{k_i} = \ln\frac{N+1}{N_{k_i}+1} \quad (2)$$

$$ndl_{\mathscr{G}} = (1-s) + s * \frac{tl_{\mathscr{G}}}{avg_{tl}} \quad (3)$$

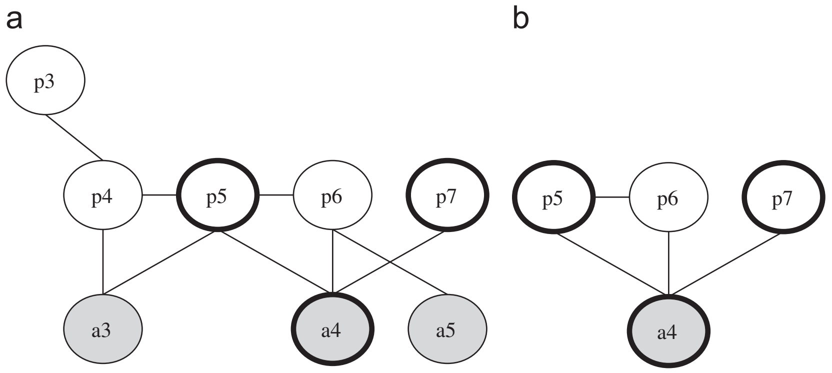


Fig. 6. (a) $\mathcal{G}^2_{p_5}$ and (b) $\mathcal{SG}^2_{p_5}$.

where $tf_{(k_i,\mathcal{G})}$ in Eq. (1) denotes the term frequency of k_i in $\mathcal{G}$; N and N_{k_i} in Eq. (2), respectively, denote the number of maximal r-radius graphs and the number of those maximal r-radius graphs that contain k_i. $tl_{\mathcal{G}}$ in Eq. (3) denotes the total number of terms in $\mathcal{G}$ and avg_{tl} is the average number of terms among all such r-radius graphs while s is a parameter taken from IR literature, which has been extensively discussed and typically set to 0.2 [6]. We combine the three parameters to evaluate the document relevancy between an input keyword k_i and a given Steiner graph $\mathcal{SG}$, denoted as $\text{Score}_{\text{IR}}(k_i,\mathcal{SG})$ as formalized in Eq. (4), where $\mathcal{G}$ is the corresponding r-radius graph w.r.t. $\mathcal{SG}$[4]:

$$\text{Score}_{\text{IR}}(k_i,\mathcal{SG}) = \frac{ntf_{(k_i,\mathcal{G})} * idf_{k_i}}{ndl_{\mathcal{G}}} \tag{4}$$

Based on the textual relevancy of k_i in $\mathcal{SG}$, we compute the overall score between an input keyword query $\mathcal{K}$ and $\mathcal{SG}$ by summing up $\text{Score}_{\text{IR}}(k_i,\mathcal{SG})$, as shown in the equation

$$\text{Score}_{\text{IR}}(\mathcal{K},\mathcal{SG}) = \sum_{k_i\in\mathcal{K}} \text{Score}_{\text{IR}}(k_i,\mathcal{SG}) \tag{5}$$

Although the TF · IDF-based IR ranking methods are efficient for textual documents, they do not consider node prestige. To address this issue, we propose a more effective IR ranking function in the following sections.

4.1.2. *Node prestige-based ranking*

This section considers node prestige. There exists a rich body of literature dealing with node ranking in graphs, such as PageRank [25], HITS [26], SALSA [27], ObjectRank [28] and EntityRank [29]. While these work mainly from the IR domain can be adopted to assign node weights in a DB graph, they only evaluate, however, the prestige (or authority, importance) of nodes in the graph. For example, the weight of node v, $\omega(v)$, is set to $\text{PR}(v)$, which can be defined as

$$\text{PR}(v) = (1-d) + d * \left(\frac{\text{PR}(u_1)}{\mathcal{D}_{out}(u_1)} + \cdots + \frac{\text{PR}(u_m)}{\mathcal{D}_{out}(u_m)}\right) \tag{6}$$

where $\text{PR}(v)$ is the prestige of node v; $\text{PR}(u_i)$ is the prestige of node u_i which directly links to node v; $\mathcal{D}_{out}(u_i)$ is the outdegree of node u_i; and d is a damping factor that can be set between 0 and 1.

Although node prestige-based ranking consider the relationships between nodes, they do not consider the term information. To address this problem, we introduce link-based ranking in the next section.

4.1.3. *Link-based ranking*

As tf · idf-based IR ranking does not consider the nodes which indirectly contain a keyword (i.e., those nodes which do not contain the keyword but have paths to the nodes that contain the keyword) and node prestige-based ranking does not consider term frequency and inverse document frequency, to address this issue, we combine these two features and propose a more effective ranking function so as to improve result quality. We assign the score of a Steiner graph by integrating PageRank and term importance scores defined by term frequency and inverse document frequency (tf · idf) as illustrated below:

$$\text{Score}_{\text{IR}}(k_i,\mathcal{SG}) = \alpha * \text{PR}(\mathcal{SG}) + (1-\alpha) * \mathcal{R}(k_i,\mathcal{SG}) \tag{7}$$

where

$$\mathcal{R}(k_i,\mathcal{SG}) = \begin{cases} \text{Score}_{\text{IR}}(k_i,\mathcal{SG}) & \mathcal{SG} \text{ directly contains } k \\ \eta^{\delta(\mathcal{SG},\mathcal{SG}')} & \\ \quad *\text{Score}_{\text{IR}}(k_i,\mathcal{SG}') & \mathcal{SG} \text{ indirectly contains } k \end{cases} \tag{8}$$

$\text{Score}_{\text{IR}}(k_i,\mathcal{SG})$ is the combined score of $\mathcal{SG}$ with respect to term k by integrating node prestige $\text{PR}(\mathcal{SG})$ and tf · idf-based relevance $\mathcal{R}(k_i,\mathcal{SG})$, which evaluates the relevance between $\mathcal{SG}$ and k_i. α is a tuning parameter to differentiate the importance of node prestige and tf · idf-based score, which is between 0 and 1 and usually set to 0.8. $\mathcal{SG}'$ is the subgraph which contains k_i and has the minimal distance with $\mathcal{SG}$.[5] Thus, $\text{Score}_{\text{IR}}(k_i,\mathcal{SG}')$ can be computed based on Eq. (4). η is a damping factor between 0 and 1 and usually set to 0.6. $\delta(\mathcal{SG},\mathcal{SG}')$ is the distance between $\mathcal{SG}$ and $\mathcal{SG}'$.

[4] $ntf_{(k_i,\mathcal{SG})} = ntf_{(k_i,\mathcal{G})}$; $tf_{(k_i,\mathcal{SG})} = tf_{(k_i,\mathcal{G})}$ for each input keyword k_i.

[5] The distance between two $\mathcal{SG}$'s is the minimal distance between the two nodes from the two graphs among all the node pairs.

Note that traditional tf · idf-based methods only consider the node which *contains* a given keyword. However, the node which *indirectly contains* a keyword is also relevant to the keyword. For example, consider p_1 in Fig. 3, although p_1 does not contain "keyword search", it should be relevant to them, as it cites (is cited by) papers which contain the keywords. Alternatively, we also score the node which does not directly contain a keyword and compute the extended tf · idf score as described in Eq. (8).

Accordingly, we incorporate both the PageRank score and the extended tf · idf score into the scoring function as described in Eq. (7), which can effectively measure the importance of a node and its relevance for a given keyword.

We note that node prestige values are very important in DB structures (capturing the primary–foreign-key relationship between tuples). Take the DBLP dataset as an example. Although some papers discussing "`Data Cube`" or "`Modeling Multidimensional Databases`" do not contain the keyword "`OLAP`" in their titles (not even in abstracts), they may still constitute relevant and potentially important papers in `OLAP` since they may be referenced by other papers in `OLAP` or written by the authors who authored other important `OLAP` papers. Based on the links between papers (citations), we can evaluate the prestige values, which can then be used for effective node ranking. tf · idf score is also important for node weight. As the more keywords contained in the nodes, the more likely the nodes are relevant to the keyword queries. We will experimentally prove that node weights can improve the result quality in Section 6.

Although the TF · IDF-based and node prestige-based IR ranking methods are efficient for textual documents, they are inefficient for semi-structured and structured data. From the IR perspective, traditional textual relevancy is important. However, due to our use of graph in modeling, the ranking of graph data becomes equally if not more important, and the structural compactness of *r*-radius Steiner graphs is the essence of the comparison. This is so because identifying rich structural relationships should be at least as important as discovering more keywords, and in some cases, even more crucial. Therefore, we propose a novel ranking function by incorporating structural compactness from the DB point of view.

4.2. Structural compactness-based DB ranking

Intuitively, when an *r*-radius Steiner graph $\mathcal{SG}$ is more compact, $\mathcal{SG}$ is more likely to be meaningful and relevant. Accordingly, the structural compactness score should be larger. As such, the compactness of $\mathcal{SG}$ should include the following parameters: (i) the structural compactness between content nodes in $\mathcal{SG}$ and (ii) the structural relevancy between input keywords w.r.t. $\mathcal{SG}$. We note that when the length of a path between two content nodes is larger, the relevancy between them is smaller. Further, there may be multiple paths between two content nodes, and we should consider all of them. Based on the above rationale, we propose Eq. (9) to score the overall structural compactness between any two content nodes:

$$\textsc{Sim}(n_i, n_j) = \sum_{n_i \leftrightsquigarrow n_j} \frac{1}{(|n_i \leftrightsquigarrow n_j| + 1)^2} \tag{9}$$

where n_i and n_j are two content nodes, $n_i \leftrightsquigarrow n_j$ is any path between n_i and n_j, and $|n_i \leftrightsquigarrow n_j|$ is the length of $n_i \leftrightsquigarrow n_j$. An important feature of $\textsc{Sim}(n_i, n_j)$ is that it can be precomputed and materialized off-line, based on the fact that $\textsc{Sim}_{\mathcal{G}}(n_i, n_j) = \textsc{Sim}_{\mathcal{SG}}(n_i, n_j)$ holds as formalized in Lemma 4, where $\mathcal{G}$ denotes the corresponding *r*-radius graph of $\mathcal{SG}$ while $\textsc{Sim}_{\mathcal{G}}(n_i, n_j)$ and $\textsc{Sim}_{\mathcal{SG}}(n_i, n_j)$ denote the structural compactness between n_i and n_j in $\mathcal{G}$ and $\mathcal{SG}$, respectively. It is clear that we can compute the overall structural compactness by summing up several materialized scores. For example, in Fig. 6, $\textsc{Sim}_{\mathcal{SG}^2_{p_5}}(a4, p5) = \textsc{Sim}_{\mathcal{G}^2_{p_5}}(a4, p5) = 1/(|a4 \leftrightsquigarrow p5| + 1)^2 + 1/(|a4 \leftrightsquigarrow p6 \leftrightsquigarrow p5| + 1)^2 = \frac{1}{4} + \frac{1}{9} = \frac{13}{36}$.

Lemma 4. *Given an r-radius graph $\mathcal{G}$ and its corresponding Steiner graph $\mathcal{SG}$ with respect to a given keyword query, the following equation holds*:

$$\textsc{Sim}_{\mathcal{G}}(n_i, n_j) = \textsc{Sim}_{\mathcal{SG}}(n_i, n_j)$$

where n_i and n_j are any two content nodes in $\mathcal{SG}$.

Proof. Note that if there is a path from n_i to n_j in $\mathcal{G}$, there must be a similar path in $\mathcal{SG}$ and vice versa based on Definition 3. Thus, we have $\textsc{Sim}_{\mathcal{G}}(n_i, n_j) = \textsc{Sim}_{\mathcal{SG}}(n_i, n_j)$. □

Although the structural compactness between two content nodes can measure the structural relevancy of *r*-radius graphs, it cannot evaluate the structural relevancy among input keywords, which captures the phrase-based relevancy between input keywords. It follows that a smaller distance between input keywords indicates a higher structural relevancy between them. This is particularly so for keywords in the same node that will represent a phrase. We therefore propose Eq. (10) to capture this parameter:

$$\textsc{Sim}(\langle k_i, k_j \rangle | \mathcal{SG}) = \frac{1}{|\mathcal{C}_{k_i} \cup \mathcal{C}_{k_j}|} \sum_{n_i \in \mathcal{C}_{k_i}; n_j \in \mathcal{C}_{k_j}} \textsc{Sim}(n_i, n_j) \tag{10}$$

where $\mathcal{C}_{k_i}$ denotes the set of all the content nodes that contain k_i in $\mathcal{SG}$, and $|\mathcal{C}_{k_i}|$ denotes the number of nodes in $\mathcal{C}_{k_i}$, which is used to normalize the structural relevancy between two input keywords. Consequently, a larger overall structural compactness score of $\mathcal{SG}$ indicates that $\mathcal{SG}$ is more likely to be relevant and meaningful to $\mathcal{K}$.

We note that the structural relevancy between input keywords has a salient feature that if $\textsc{Sim}(\langle k_i, k_t \rangle | \mathcal{SG})$ and $\textsc{Sim}(\langle k_t, k_j \rangle | \mathcal{SG})$ are large, $\textsc{Sim}(\langle k_i, k_j \rangle | \mathcal{SG})$ must be also large and thus k_i, k_j and k_t must be very relevant to each other w.r.t. $\mathcal{SG}$. Thus, a key feature of structural relevancy is that we can use the structural relevancy between any two input keywords to capture the relevancy between all of the input keywords as illustrated in Eq. (11). This feature can help capture the rich structural information of $\mathcal{SG}$ while the textual relevancy in Eq. (5) cannot. Formally, given a keyword query $\mathcal{K} = \{k_1, k_2, \ldots, k_m\}$ and an r-radius Steiner graph $\mathcal{SG}$, the overall structural compactness of $\mathcal{SG}$ w.r.t. $\mathcal{K}$, denoted as $\textsc{Score}_{\text{DB}}(\mathcal{K}, \mathcal{SG})$, can be computed as follows:

$$\textsc{Score}_{\text{DB}}(\mathcal{K}, \mathcal{SG}) = \sum_{1 \leqslant i < j \leqslant m} \textsc{Sim}(\langle k_i, k_j \rangle | \mathcal{SG}) \tag{11}$$

By taking into account both document relevancy from the IR perspective and structural compactness/relevancy from the DB perspective to capture structural relationships, we present a more accurate function for scoring r-radius Steiner graphs as given below[6]

$$\textsc{Score}(\mathcal{K}, \mathcal{SG}) = \sum_{1 \leqslant i < j \leqslant m} \textsc{Score}(\langle k_i, k_j \rangle | \mathcal{SG}) \tag{12}$$

where

$$\textsc{Score}(\langle k_i, k_j \rangle | \mathcal{SG}) = \textsc{Sim}(\langle k_i, k_j \rangle | \mathcal{SG}) * (\textsc{Score}_{\text{IR}}(k_i, \mathcal{SG}) + \textsc{Score}_{\text{IR}}(k_j, \mathcal{SG})) \tag{13}$$

$\textsc{Score}(\langle k_i, k_j \rangle | \mathcal{SG})$ measures the overall relevancy score of $\langle k_i, k_j \rangle$ in $\mathcal{SG}$ based on the structural compactness/relevancy and IR scores. Note that, $\textsc{Sim}(\langle k_i, k_j \rangle | \mathcal{SG})$ is taken as the weight of the sum of two IR scores, i.e., $\textsc{Score}_{\text{IR}}(k_i, \mathcal{SG})$ and $\textsc{Score}_{\text{IR}}(k_j, \mathcal{SG})$. A larger $\textsc{Sim}(\langle k_i, k_j \rangle | \mathcal{SG})$ means that k_i and k_j are more relevant w.r.t. $\mathcal{SG}$, and thus, the overall score of $\langle k_i, k_j \rangle$ in $\mathcal{SG}$ is expected to be larger.

5. Indexing

To efficiently identify the *top-k* answers with the highest scores, we examine the issues of indexing in this section.

Given any two keywords k_i and k_j in the graph, and an r-radius graph $\mathcal{SG}$, the scores of $\textsc{Score}_{\text{IR}}(k_i, \mathcal{SG})$ and $\textsc{Score}_{\text{IR}}(k_j, \mathcal{SG})$ in Eq. (4) and $\textsc{Sim}(\langle k_i, k_j \rangle | \mathcal{SG})$ in Eq. (10) share the key feature that they can be pre-computed and materialized off-line. Based on this observation, we can materialize $\textsc{Score}(\langle k_i, k_j \rangle | \mathcal{SG})$. We devise an extended inverted index (EI-Index) to maintain such scores. Different from the traditional inverted index, the entries of EI-Index are keyword pairs (combinations of two keywords), and the values of each entry is the maximal r-radius graphs that contain the keyword pair and the corresponding scores. For example, we can construct the EI-Index of the graph in Fig. 3 as illustrated in Table 6.

To answer a keyword query $\mathcal{K} = \{k_1, k_2, \ldots, k_m\}$, we first retrieve the maximal r-radius graphs for each keyword pair $\langle k_i, k_j \rangle$ according to EI-Index, and then compute the scores of every relevant maximal r-radius graph according to Eq. (12). Finally, we rank the results and return the *top-k* r-radius Steiner graphs with the highest scores by refining the corresponding r-radius graphs.

For example, in Example 5, recall the query "`DISCOVER, Relational, Database, Papakonstantinou`". We first retrieve the relevant maximal r-radius graphs

Table 6
EI-Index: an extended inverted index

Keyword pair	$\langle r$-radius graph, score$\rangle$
$\langle$Database, DISCOVER$\rangle$	$\mathcal{G}^r_{p_5}$,1.53
$\langle$Database, Papakonstantinou$\rangle$	$\mathcal{G}^r_{p_5}$,0.38; $\mathcal{G}^r_{p_4}$,0.19
$\langle$Database, Relational$\rangle$	$\mathcal{G}^r_{p_5}$, 0.85; $\mathcal{G}^r_{p_3}$, 0.35; $\mathcal{G}^r_{p_4}$, 0.34
$\langle$DISCOVER, Papakonstantinou$\rangle$	$\mathcal{G}^r_{p_5}$, 0.54
$\langle$DISCOVER, Relational$\rangle$	$\mathcal{G}^r_{p_5}$,1.95
$\langle$Papakonstantinou, Relational$\rangle$	$\mathcal{G}^r_{p_5}$, 0.57; $\mathcal{G}^r_{p_4}$, 0.28
$\langle\ldots,\ldots\rangle$	...

based on EI-Index and compute the corresponding overall scores. We then rank them based on such scores, i.e., $\mathcal{G}_5$, $\mathcal{G}_4$, $\mathcal{G}_3$. Finally, we identify the r-radius Steiner graphs on top of the corresponding graph partitions.

However, this method needs to first compute the scores for all relevant maximal r-radius graphs and then rank them. This leads to low efficiency in the presence of large numbers of r-radius Steiner graphs. To alleviate the problem, we introduce an effective technique of progressively identifying the *top-k* answers. Note that there are many studies of effectively retrieving *top-k* answers from multiple inverted lists, such as Fagin Algorithm (FA) [30] and Threshold Algorithm (TA) [31]. We can employ such techniques to identify the *top-k* answers on top of our EI-Index as follows: Given a keyword query $\mathcal{K} = \{k_1, k_2, \ldots, k_m\}$, we can retrieve the inverted lists composed of relevant r-radius graphs and corresponding scores for every keyword pair according to our EI-Index. Then, we adopt existing algorithms to progressively identify the *top-k* answers. The advantage of our approach is obvious—we need not discover the structural relationships by traversing the whole graph. Instead, we first materialize such rich relationships into our EI-Index off-line, and then identify the *top-k* answers according to EI-Index online.

6. Experimental study

We have designed and performed a comprehensive set of experiments to evaluate the search performance of EASE. We employed the datasets of DBLife,[7] DBLP,[8] and IMDB[9] to evaluate EASE on unstructured, semi-structured and structured data, respectively. There were about 10,000 pages in the DBLife dataset, and the raw file of DBLP was about 400 MB. IMDB contained about one million anonymous ratings of approximately 3900 movies made by 6040 users. All the experimental results in this paper were verified by the SIGMOD repeatability committee. Code and data used in the paper are available at http://www.sigmod.org/codearchive/sigmod2008/.

[6] Note that for the keyword query with a single keyword, we employ $\textsc{Score}_{\text{IR}}(k_i, \mathcal{SG})$ to rank the query.

[7] http://dblife.cs.wisc.edu/download/surfaceWeb-2007-07-25.tgz

[8] http://dblp.uni-trier.de/xml

[9] http://www.grouplens.org/node/73 (1,000,000 Data Set (5.73 MB)) http://www.grouplens.org/system/files/million-ml-data.tar__0.gz

Table 7
Queries employed in the experiments

Query ID	Queries
(a) Queries on unstructured data (DBLife)	
Q_1	2007 conference data integration
Q_2	XML relational keyword search
Q_3	Turkey conference 2007 uncertainty fuzziness
Q_4	Berkeley phone dataspaces information management
Q_5	Beijing conference 2007 data integration
(b) Queries on semi-structured data (DBLP)	
Q_6	Information retrieval database
Q_7	IR database
Q_8	DB IR XML
Q_9	XML relational keyword search
Q_{10}	Data mining algorithm 2006
(c) Queries on structured data (IMDB)	
Q_{11}	Lethal Weapon 4 academic
Q_{12}	Police Academy 3 customer
Q_{13}	Halloween 5 college
Q_{14}	Love 45 tradesman
Q_{15}	Robocop 3 college
(d) Queries on heterogeneous data	
Q_{16}	XML keyword search 2007
Q_{17}	Dennis Shasha database tuning
Q_{18}	Dataspaces 2006 information management
Q_{19}	Database indexing ranking search
Q_{20}	Romance action educator

The experiments were conducted on an Intel(R) Core(TM) 2.0 GHz computer with 2 GB of RAM running Windows Vista, and the algorithms were implemented in Java. We used MYSQL 5.0.45[10] to maintain the graph index. We employed the TA [31] to progressively identify the *top-k* answers based on the EI-Index. We compared our approach with existing state-of-the-art approaches. For the unstructured data, we compared EASE with DBLife [22] by submitting keyword queries to its interface and InfoUnit [24]. For the semi-structured data, we compared EASE with SLCA [16] while for structured data, we compared it with DPBF [3]. We selected 20 queries as illustrated in Table 7 for testing different aspects of search problems such as the ability of the methods in capturing data lineage and relationships, and their search accuracy.

6.1. Graph index evaluation

The graph w.r.t. DBLife contains about 10,000 nodes, which are far fewer than the approximate 1,000,000 nodes of IMDB and 12,000,000 nodes of DBLP. Note that the graph w.r.t. IMDB is much denser than that of DBLP, which in turn is denser than that of DBLife. The elapsed time of indexing DBLife, IMDB and DBLP is, respectively, 13, 25 and 87 min. The sizes of the graph indices of DBLife, IMDB and DBLP using the maximal *r*-radius-based method are, respectively, 246, 218 and 2896 MB, compared with the data sizes of 131, 30 and 405 MB as shown in Table 8.

[10] http://dev.mysql.com/downloads/mysql/5.0.html

Table 8
Graph index size

Datasets	Index size (MB)	
	r-Radius subgraph-based method	Partition-based method
DBLife	246	121
IMDB	218	88
DBLP	2896	862

While the graph index by employing the graph-partition-based method brings the index sizes down to 121, 88 and 862 MB, respectively. This is attributed to that the graph-partition-based method can reduce the redundancy of the graph index. Moreover, the graph index of DBLife is smaller than its data size due to: (i) tokenization for removing tags and (ii) its sparser structure.

6.2. Search efficiency on different datasets

We evaluate the search efficiency of EASE in this section. We tested every algorithm on the selected queries, identified the *top*-100 results and compared their corresponding elapsed time. Fig. 7 summarizes the experimental results. We observe that EASE always achieves the best performance for various keyword queries and outperforms the other methods significantly on different datasets.

On unstructured data, InfoUnit identifies Steiner trees by traversing the whole graph to discover structural relationships online, which is not as efficient as our method that materializes rich structural information into the graph indices for facilitating online keyword-based query processing. On semi-structured data, SLCA has to first identify all the relevant results, and then ranks them and returns the *top-k* answers with the highest scores. In contrast, EASE employs the threshold-based algorithm [31] to progressively identify *top-k* answers and thus produces a dramatic improvement over the existing methods. On structured data, although DPBF can progressively compute the *top-k* answers, it has to discover the structural information between tuples in different relational tables online. At the same time, it also needs to identify the Steiner trees on top of the whole graph. These needs lead to low efficiency. In contrast, EASE first identifies the *r*-radius graphs based on EI-Index and then constructs *r*-radius Steiner graphs on top of the corresponding graph partitions. These partitions are much smaller than the whole graph, and the method therefore achieves better search performance and outperforms DPBF significantly.

To better evaluate the selected algorithms, we identified the *top-k* answers with different values of *k* and compared the corresponding average elapsed time. Fig. 8 illustrates the experimental results obtained. We observe that EASE outperforms the other methods significantly. We note that, with the increase of *k* values, the elapsed time of InfoUnit and DPBF also increases, but the elapsed time of EASE varies only slightly (as SLCA cannot

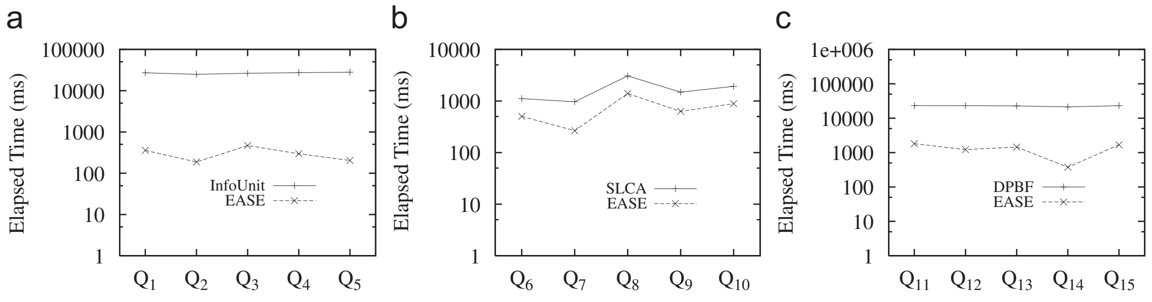


Fig. 7. Search efficiency on various queries. (a) Queries (unstructured), (b) Queries (semi-structured) and (c) Queries (structured).

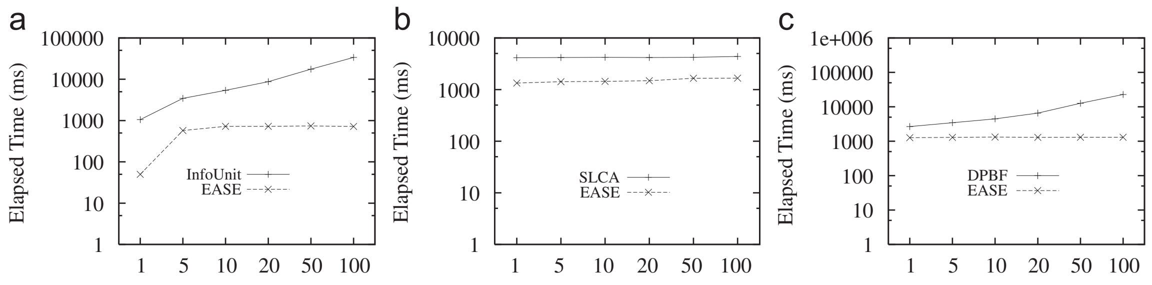


Fig. 8. Search efficiency with different values of *k*. (a) Top-k (unstructured), (b) top-k (semi-structured) and (c) top-k (structured).

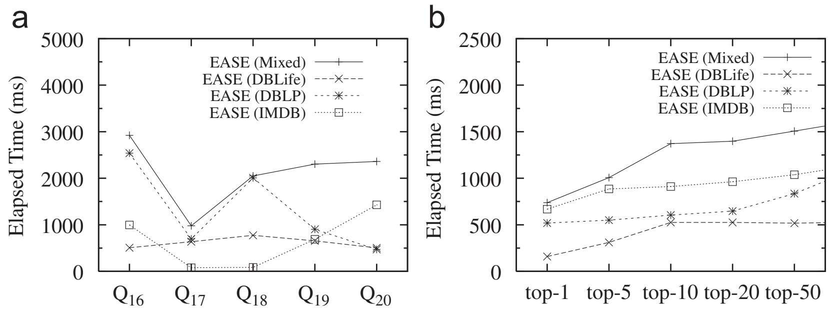


Fig. 9. Search efficiency on heterogeneous data. (a) Queries (Top-100) and (b) Top-k.

progressively identify answers, its elapsed time also varies little). The elapsed time of InfoUnit and DPBF to return *top*-100 results is always a little more than that of *top*-5 while the elapsed time of EASE remains about the same. This is so because we adopt the threshold-based technique to progressively identify the *top-k* answers.

6.3. *Search efficiency on heterogeneous datasets*

To evaluate the overall performance of EASE on heterogeneous data, we mixed the three datasets and tested EASE on the combined data. We constructed the graph index on top of the heterogeneous data and employed $Q_{16}-Q_{20}$ to evaluate the search efficiency of EASE over heterogeneous data. To better understand the performance of EASE, we first evaluated the elapsed time of identifying the *top*-100 answers for every query. We then varied the values of *k* to evaluate the average elapsed time. We also provide the elapsed time of EASE on different datasets as a baseline to show the search efficiency of EASE over heterogeneous data. The results are summarized in Fig. 9. We can see that EASE is still capable of achieving high search efficiency over heterogeneous data and is not much worse off than on homogeneous datasets. That is, even for a large volume of heterogeneous data, EASE can still effectively identify the *top-k* answers.

6.4. *Search accuracy on different datasets*

This section evaluates search accuracy, indicating the fraction of relevant results in the approximate answer that

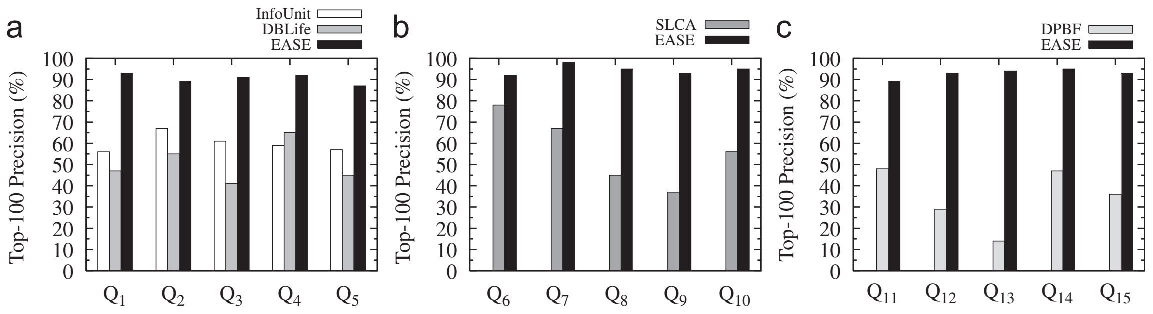


Fig. 10. *Top*-100 precision on various queries. (a) Queries (unstructured), (b) Queries (semi-structured) and (c) Queries (structured).

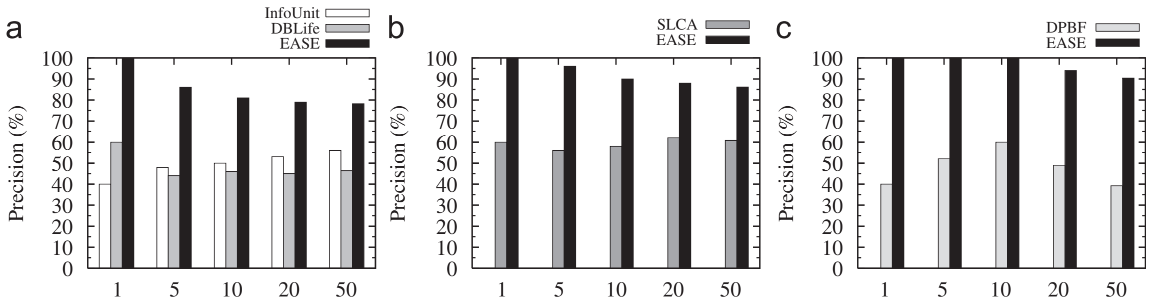


Fig. 11. *Top-k* precision with different values of *k*. (a) Top-k (unstructured), (b) top-k (semi-structured) and (c) top-k (structured).

are correct. We employed the metric, *top-k* precision, which measures the ratio of the number of relevant answers among the first *k* answers with the highest scores of an algorithm to *k*. Answer relevance is judged from discussions of researchers in our DB group. Interested readers can test our algorithm through our deposited demo as per submission requirement. Moreover, to evaluate the search accuracy of the selected algorithms, we identified the *top*-100 results for every query and compared the corresponding *top*-100 precision. Fig. 10 illustrates the experimental results obtained. We observe that EASE achieves much higher precision than existing methods such as, DBLife, InfoUnit, SLCA and DPBF on the corresponding datasets.

On unstructured data, as InfoUnit and EASE integrate relevant pages to answer keyword queries, they achieve much higher precision than DBLife. EASE is better than InfoUnit because EASE employs a ranking mechanism that takes into account both the structural compactness of answers from the DB viewpoint and textual relevancy from the IR viewpoint. On semi-structured data, *r*-radius Steiner graphs, alongside content nodes, also contain some relevant elements, such as the Steiner nodes, which may expand the answerability. They are more meaningful than the subtrees/path-trees rooted at LCAs (or its variants) of existing methods, which may miss some relevant information. Thus, EASE yields higher precision. On structured data, EASE also outperforms DPBF in that: (i) *r*-radius Steiner graphs are more meaningful than Steiner trees and (ii) again, this is due to the consideration of structural compactness between input keywords in our ranking function.

For example, consider query Q_7. Its answer should be the papers about “`Database`” and “IR”. However, “IR” may appear in the name of an author, and traditional methods such as SLCA cannot distinguish the papers entitled with “`Database`” and “IR” and the papers with “`IR`” being a term of the author and “`Database`” being a term of the title. They may mistakenly take the latter as answers, leading to low search accuracy. As another example, two input keywords may appear in two different authors of a paper, and the conventional methods will mistakenly take the two keywords as an author. EASE ranks the results with structures that are less compact lower. This is done by means of our structural relevancy-based ranking method. In addition, some answers which even if do not contain “`Database`” and “IR” but include “`DB`” and “IR” are also relevant to query Q_7. Our method still can find such answers. Hence, EASE achieves much higher precision.

As users are usually interested in *top-k* answers, we varied different values of *k* to evaluate the selected algorithms. The average results of the *top-k* precision for Q_1-Q_{15} are illustrated in Fig. 11. As expected, EASE consistently achieved high precision in many queries, which is approximately 10–30% higher than those of InfoUnit, DBLife, DPBF and SLCA.

In addition, we evaluate our method with different ranking functions, such as tf · idf-based ranking, link-based ranking, and DB ranking. The experimental results are

Table 9
Top-*k* precision with different ranking functions

Query	tf · idf	tf · idf + Link	tf · idf + DB	tf · idf + Link + DB
Q_7	67	75	89	98
Q_8	45	64	83	95
Q_{15}	36	62	88	93
Q_{19}	30	44	82	93
Q_{20}	35	53	85	95

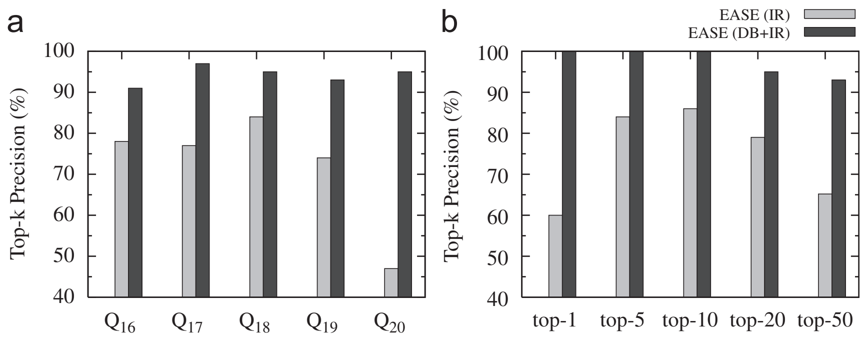


Fig. 12. Search accuracy on heterogeneous data. (a) Queries (Top-100), and (b) Top-k.

illustrated in Table 9. We note that all of the three ranking functions are important to rank the results. tf · idf-based ranking ranks the answers with more keywords higher. Link-based ranking considers the results which even do not include input keywords but have relationships with the input keywords through links between nodes. DB ranking ranks the result with more compact structures higher.

6.5. *Search accuracy on heterogeneous datasets*

To evaluate the robustness of our algorithm, we evaluate its search accuracy using the heterogeneous dataset. We first evaluated search accuracy by identifying the *top*-100 results on various queries, and then varied different values of *k* to evaluate the average precision. For a better understanding of our ranking method, we compared EASE employing IR ranking parameters (ref. Eq. (5)) with EASE considering both DB and IR (ref. Eq. (12)). Fig. 12 summarizes the experimental results. We observe that EASE(DB+IR) consistently achieves much higher accuracy than EASE(IR). This again confirms that TF · IDF-based IR ranking cannot capture the structural relationships effectively. For example, consider query Q_{20}. Some results on DBLife dataset will obtain higher IR scores than those on IMDB dataset as the term frequencies of some documents on DBLife dataset are larger than those on IMDB dataset. Thus, the IR-based ranking method will rank the results on DBLife dataset higher. However, in the IMDB dataset, which is a collection of ratings of movies made by users, the three input keywords can describe the instances of users with occupation of Educator scoring the movies with genres of Romance and Action. Hence, the three keywords are more compact and highly relevant to IMDB dataset. Our DB+IR ranking method considers the structural relevancy and ranks the results on the IMDB dataset higher, and thus achieves much higher precision. This comparison reflects the effectiveness and applicability of our proposed ranking mechanism.

Table 10
Index update

Datasets	Elapsed time (min)	
	Indexing from scratch	Incrementally indexing
DBLife	13	1.8
IMDB	25	3.1
DBLP	87	10.2

6.6. *Update of index*

In this section, we evaluate the cost of DB update. We first created the indices on top of the three datasets by using 90% data, and then inserted the other 10% data. We evaluate the running time of indexing the new 10% data as shown in Table 10. We note that it is rather expensive to re-index the data from scratch. Instead, we can incrementally update the index based on the partitions, which is much more efficient than indexing the whole data from scratch.

6.7. *Sensitivity of indexing*

In this section, we evaluate the sensitivity of our indexing method. We first created the indices on top of

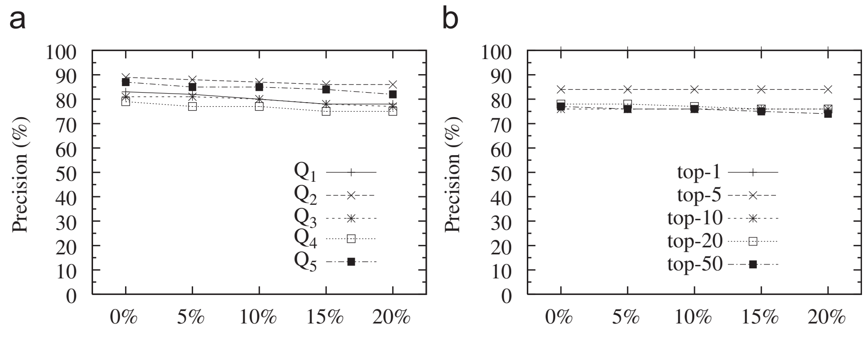


Fig. 13. Sensitivity of indexing. (a) Inserted data/original data (Top-100) and (b) Inserted data/original data (Top-k).

the DBLife dataset, and then inserted 5% additional new irrelevant data in each step. After each set of insertions, we evaluated EASE on the updated indices by using queries Q_1-Q_5. Fig. 13 summarizes the effects of insertions on our indexing method. We observe that the search accuracy of EASE does not degrade drastically with the increase of new data. This demonstrates the robustness and scalability of our method with respect to data insertions.

7. Related work

The first area of research related to our work is keyword search over relational DBs by identifying Steiner trees. As opposed to the traditional Steiner tree-based methods, which identify the structural relationships online, EASE identifies and materializes the rather rich structural relationships so as to improve the online processing of keyword queries.

DBXplorer [1], DISCOVER-I [5], DISCOVER-II [4], BANKS-I [2] and BANKS-II [19] are systems built on top of relational DBs. DISCOVER and DBXplorer generate trees of tuples connected through primary–foreign-key relationships that contain all of the input keywords. BANKS identifies connected trees in a labeled graph by using an approximation of the Steiner tree problem. DISCOVER-II considers the problem of keyword proximity search in terms of disjunctive semantics, as opposed to DISCOVER-I which only considers conjunctive semantics. Kacholia et al. [19] presented the bidirectional strategy (BANKS-II) to improve the efficiency of keyword search over graph data. However, their method still works by identifying Steiner trees from the whole graph, which is inefficient as it is rather difficult to identify structural relationships through inverted indices. Liu et al. [6] proposed a novel ranking strategy to solve the effectiveness problem for relational DBs. It employs phrase-based and concept-based models to improve search effectiveness by introducing IR techniques.

More recently, Ding et al. [3] employed dynamic programming (DPBF) to improve the efficiency of identifying Steiner trees. Guo et al. [17] proposed data topology search to retrieve meaningful structures from richer structural data, such as complex biological DBs. He et al. [18] proposed a partition-based method to improve search efficiency with a novel BLINKS index. Markowetz et al. [8] studied the problem of keyword search over relational data streams in a first attempt to answer keyword search over relational data streams. Luo et al. [7] proposed a new ranking method that adapts state-of-the-art IR ranking function and principles into the ranking trees of joined DB tuples. In addition, Yu et al. [32] studied the problem of relational data source selection in P2P environments by summarizing the relationships between keywords in underlying DBs.

In terms of keyword search over XML documents, the subtrees rooted at the LCAs of content nodes have been proposed as answers. As an extension of LCA, SLCA, Multiway-SLCA, XSeek and GDMCT have recently been proposed to answer keyword queries over XML documents in [11,14–16], respectively. SLCA [16] can avoid the false positives of LCA but it does so at the expense of false negatives. Multiway-SLCA [15] offers a search paradigm in support of keyword search beyond the traditional AND semantics, including both AND and OR boolean operators. Xu and Papakonstantinou [41] proposed the semantics of exclusive lowest common answer (ELCA) to improve result quality. GDMCT [11] returns grouped connected trees as answers but it still needs to traverse the whole graph to identify answers by employing the inverted index. XSeek [14] generates return nodes which can be explicitly inferred from keywords or dynamically constructed according to entities in the data that are relevant to the search. Li et al. [37] proposed a more effective ranking mechanism to improve search effectiveness by combining structural compactness and textual relevance to rank the answers. The ranking functions are independent of the search algorithms, and thus could be applied to any search algorithm.

XRANK [10] and XSEarch [9] are systems facilitating keyword search for XML documents, and they return connected subtrees as answers for keyword queries. XRANK presents a ranking method, where for a given tree $\mathscr{T}$ containing all the keywords, a score is assigned to $\mathscr{T}$ with an adaptation of PageRank for XML documents. XSEarch focuses on semantics and the ranking of results; during execution, it uses an all-pairs interconnection index to check the connectivity between nodes. XKeyword [12] is a system that offers keyword proximity search over

XML documents that conform to an XML schema. However, it needs to compute candidate networks and thus is constrained by schemas. Schema-Free XQuery [35] uses the idea of meaningful LCA (MLCA), and proposes a stack-based sort-merge algorithm by considering a part of structures and incorporating a new function `mlcas` into XQuery. TopX [33] is a prototype search engine for the ranked retrieval of XML, but it processes XML queries with support for XPath axes but not the more simple keyword queries, and it cannot adapt to relational DBs. Graupmann et al. [34] presented the SphereSearch engine to provide unified ranked retrieval on heterogeneous XML and Web data. However, it is orthogonal to our method in that: (i) it does not support relational DBs and it transforms HTML documents into XML and (ii) it depends on its own query language to discover structural relationships and thus is not a pure keyword-based search method. Chaudhuri et al. [20] point out a number of interesting research opportunities for integrating DB and IR technologies. In [21], Weikum provides a summary of existing DB and IR techniques and discusses the difficulties, opportunities and challenges of combining DB and IR. Li et al. [38] proposed a unified keyword search framework, namely SAILER, for answering keyword queries over unstructured and semi-structured data. Li et al. [39] proposed EASE to adaptively and effectively answer keyword search over heterogeneous data sources composed of unstructured, semi-structured and structured data by summarizing the graphs transformed from the heterogeneous data sources. They proposed Steiner subgraphs to effectively answer keyword queries over graphs. Li et al. [40] presented a demonstration of keyword search over heterogeneous data.

8. Conclusion

In this paper, we have proposed an efficient and adaptive keyword search method, EASE, to answer keyword queries over unstructured, semi-structured and structured data. EASE seamlessly integrates the efficient query evaluation of DB and the adaptive scoring models of IR for the ranking of results. EASE models heterogeneous data as graphs and processes keyword queries on the graphs. To the best of our knowledge, this is the first attempt to efficiently and adaptively process keyword queries on heterogeneous data. We have proposed summarizing and clustering the graphs, and devised effective graph indices to materialize structural relationships for fast and accurate response. To facilitate efficient keyword-based query processing, we have examined the issues of indexing and ranking by taking into account both the structural compactness of *r*-radius graphs from the DB point of view and textual relevancy from the IR viewpoint. Finally, we have conducted an extensive performance study to evaluate the efficiency and effectiveness of our approach using real datasets. The experimental results show that EASE achieves both high search efficiency and quality for keyword search over heterogeneous data, and significantly outperforms the existing methods.

Acknowledgments

The research of B.C. Ooi was in part funded by NUS Grant R-252-000-338-112. This work was also in part supported by the National Natural Science Foundation of China under Grant No. 60573094, the National High Technology Development 863 Program of China under Grant No. 2007AA01Z152, the National Grand Fundamental Research 973 Program of China under Grant No. 2006CB303103, and 2008 HP Labs Innovation Research Program.

References

[1] S. Agrawal, S. Chaudhuri, G. Das, Dbxplorer: a system for keyword-based search over relational databases, in: ICDE, 2002, pp. 5–16.
[2] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, S. Sudarshan, Keyword searching and browsing in databases using banks, in: ICDE, 2002, pp. 431–440.
[3] B. Ding, J.X. Yu, S. Wang, L. Qin, X. Zhang, X. Lin, Finding top-*k* min-cost connected trees in databases, in: ICDE, 2007.
[4] V. Hristidis, L. Gravano, Y. Papakonstantinou, Efficient IR-style keyword search over relational databases, in: VLDB, 2003, pp. 850–861.
[5] V. Hristidis, Y. Papakonstantinou, Discover: keyword search in relational databases, in: VLDB, 2002.
[6] F. Liu, C. Yu, W. Meng, A. Chowdhury, Effective keyword search in relational databases, in: SIGMOD, 2006.
[7] Y. Luo, X. Lin, W. Wang, X. Zhou, Spark: top-k keyword query in relational databases, in: SIGMOD, 2007.
[8] A. Markowetz, Y. Yang, D. Papadias, Keyword search on relational data streams, in: SIGMOD, 2007.
[9] S. Cohen, J. Mamou, Y. Kanza, Y. Sagiv, Xsearch: a semantic search engine for XML, in: VLDB, 2003.
[10] L. Guo, F. Shao, C. Botev, J. Shanmugasundaram, Xrank: ranked keyword search over XML documents, in: SIGMOD, 2003, pp. 16–27.
[11] V. Hristidis, N. Koudas, Y. Papakonstantinou, D. Srivastava, Keyword proximity search in XML trees, in: IEEE TKDE, vol. 18(4), 2006, pp. 525–539.
[12] V. Hristidis, Y. Papakonstantinou, A. Balmin, Keyword proximity search on XML graphs, in: ICDE, 2003, pp. 367–378.
[13] G. Li, J. Feng, J. Wang, L. Zhou, Efficient keyword search for valuable LCAs over XML documents, in: CIKM, 2007.
[14] Z. Liu, Y. Chen, Identifying return information for XML keyword search, in: SIGMOD, 2007.
[15] C. Sun, C.Y. Chan, A.K. Goenka, Multiway SLCA-based keyword search in XML data, in: WWW, 2007.
[16] Y. Xu, Y. Papakonstantinou, Efficient keyword search for smallest LCAs in XML databases, in: SIGMOD, 2005, pp. 527–538.
[17] L. Guo, J. Shanmugasundaram, G. Yona, Topology search over biological databases, in: ICDE, 2007.
[18] H. He, H. Wang, J. Yang, P. Yu, Blinks: ranked keyword searches on graphs, in: SIGMOD, 2007.
[19] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, H. Karambelkar, Bidirectional expansion for keyword search on graph databases, in: VLDB, 2005, pp. 505–516.
[20] S. Chaudhuri, R. Ramakrishnan, G. Weikum, Integrating DB and IR technologies: What is the sound of one hand clapping? in: CIDR, 2005, pp. 1–12.
[21] G. Weikum, DB & IR: both sides now (keynote), in: SIGMOD, 2007, pp. 25–30.
[22] P. DeRose, W. Shen, F. Chen, Y. Lee, D. Burdick, A. Doan, R. Ramakrishnan, Dblife: a community information management platform for the database research community, in: CIDR, 2007.
[23] M. Mutsuzaki, M. Theobald, A. Keijzer, J. Widom, P. Agrawal, et al., Trio-one: layering uncertainty and lineage on a conventional DBMS, in: CIDR, 2007.
[24] W.-S. Li, K.S. Candan, Q. Vu, D. Agrawal, Retrieving and organizing web pages by information unit, in: WWW, 2001.
[25] S. Brin, L. Page, The anatomy of a large-scale hypertextual web search engine, in: WWW, 1998.
[26] J.M. Kleinberg, Authoritative sources in a hyperlinked environment, J. ACM 46 (5) (1999) 604–632.
[27] R. Lempel, S. Moran, The stochastic approach for link-structure analysis (SALSA) and the tkc effect, in: WWW, 2000.

[28] A. Balmin, V. Hristidis, Y. Papakonstantinou, Objectrank: authority-based keyword search in databases, in: VLDB, 2004, pp. 564–575.
[29] S. Chakrabarti, Dynamic personalized pagerank in entity relation graphs, in: WWW, 2007.
[30] R. Fagin, Combining fuzzy information from multiple systems, in: PODS, 1996, pp. 216–226.
[31] R. Fagin, Fuzzy queries in multimedia database systems, in: PODS, 1998, pp. 1–10.
[32] B. Yu, G. Li, K. Sollins, A.K.H. Tung, Effective keyword-based selection of relational databases, in: SIGMOD, 2007.
[33] M. Theobald, R. Schenkel, G. Weikum, An efficient and versatile query engine for topx search, in: VLDB, 2005.
[34] J. Graupmann, R. Schenkel, G. Weikum, The spheresearch engine for unified ranked retrieval of heterogeneous XML and web documents, in: VLDB, 2005, pp. 529–540.
[35] Y. Li, C. Yu, H.V. Jagadish, Schema-free xquery, in: VLDB, 2004, pp. 72–84.
[37] G. Li, J. Feng, J. Wang, L. Zhou, RACE: finding and ranking compact connected trees for keyword proximity search over XML documents, in: WWW, 2008.
[38] G. Li, J. Feng, J. Wang, L. Zhou, SAILER: an effective search engine for unified retrieval of heterogeneous XML and web documents, in: WWW, 2008.
[39] G. Li, B.C. Ooi, J. Feng, J. Wang, L. Zhou, Ease: efficient and adaptive keyword search on unstructured, semi-structured and structured data, in: SIGMOD, 2008.
[40] G. Li, J. Feng, J. Wang, L. Zhou, An effective and versatile keyword search engine on heterogeneous data sources, in: VLDB, 2008.
[41] Y. Xu, Y. Papakonstantinou, Efficient LCA based keyword search in XML data, in: EDBT, 2008, pp. 535–546.