Progressive Keyword Search in Relational Databases

Guoliang Li[†] Xiaofang Zhou[‡] Jianhua Feng[†] Jianyong Wang[†]

[†] *Tsinghua University, China* {liguoliang,fengjh,jianyong}@tsinghua.edu.cn

[‡] *The University of Queensland, Australia* zxf@itee.uq.edu.au

***Abstract*— A common approach to performing keyword search over relational databases is to find the minimum Steiner trees in database graphs. These methods, however, are rather expensive as the minimum Steiner tree problem is known to be NP-hard. Further, these methods cannot benefit from DBMS capabilities. We propose a new concept called *Compact Steiner Tree* (CSTree), which can be used to approximate the Steiner tree problem for answering top-k keyword queries efficiently. We propose a structure-aware index, together with an effective ranking mechanism for fast, progressive and accurate retrieval of top-k highest ranked CSTrees. The proposed techniques can be implemented using a standard RDBMS to benefit from its indexing and query processing capability. The experimental results show that our method achieves high search efficiency and result quality comparing to existing state-of-the-art approaches.**

I. Introduction

The database research community has recently recognized the benefits of keyword search, starting to introduce keyword search capabilities into relational databases [1]. The existing approaches of keyword search over relational databases can be broadly classified into two categories: those based on the candidate network [2] and others based on the Steiner tree [1]. The candidate network based methods identify answers composed of relevant tuples by generating and extending a candidate network following the primary-foreign-key relationship. The Steiner tree based methods first model the tuples in a relational database as a graph, where nodes are tuples and edges are the primary-foreign-key relationships. Steiner trees which contain all or some of input keywords are then identified to answer keyword queries. These methods all search for the optimal solutions on-the-fly by traversing the database graph to discover structural relationship. As the minimum Steiner tree problem is known to be NP-hard [1], a Steiner tree based approach can be inefficient, demanding new research to find polynomial time solutions that can effectively approximate the minimum Steiner tree problem.

We make the following two observations here. First, the existing DBMS technologies have not been designed with supporting keyword search in mind; and the keyword search methods recently proposed by the database community are also largely independent of the underlying DBMS. While the advantage for keyword-based search to be supported by the underlying DBMS is quite clear, this integration task remains to be an open challenge. It is further complicated by other constraints such as user friendliness (using keyword search, not SQL queries) and no modification of any source code of an existing DBMS. Second, most of the existing methods to support keyword search over relational data start with generating *all* possible results composed of relevant tuples such that these results can be sorted according to their individual ranks. This approach is ineffective for top-k keyword queries, and can be infeasible for some applications where the number of possible answers are prohibitively large. A progressive approach that returns highly ranked results first is much more attractive. In addition to eliminating the need of enumerating all possible results, such an approach can reduce the query response time, and provide the possibility of early termination of the search when the user finds sufficient number of answers.

To address the above-mentioned problems, we propose in this paper a polynomial time approximate solution for the minimum Steiner tree problem with a theoretical bound. A novel concept called *Compact Steiner Tree* (CSTree) is used to answer keyword queries more efficiently. We devise an effective structure-aware index and materialize the index in the form of relational tables. Our proposed techniques can be seamlessly integrated into an existing DBMS such that the capabilities of the DBMS can be explored to effectively and progressively identify the top-k relevant CSTrees using SQL statements. This has been achieved without the need of changing any DBMS source code. To the best of our knowledge, this is the first attempt to integrate structure-aware indices into DBMS and use the capabilities of the DBMS to support effective and progressive keyword-based search.

II. Minimum Path Weight Steiner Tree

A. Database Graph

A relational database can be modeled as a database graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ such that there is a one-to-one mapping between a tuple in the database and a node in $\mathcal{V}$. $\mathcal{G}$ can be considered as a directed graph with two types of edges [1]: a forward edge $(u, v) \in \mathcal{E}$ iff there is a foreign key from u to v, and a back edge (v, u) iff (u, v) is a forward edge in $\mathcal{E}$. An edge (u, v) indicates a close relationship between tuples u and v (i.e., they can be directly joined together), and the introduction of two edge types allows differentiating the importance of u to v and vise versa. When such differentiation is not necessary for some applications, $\mathcal{G}$ becomes an undirected graph (i.e., the forward edge and its corresponding back edge are combined into one non-directional edge). To support keyword search over relational data, $\mathcal{G}$ is typically modeled as a weighted graph, with a node weight $\omega(v)$ to represent the "prestige" level of each node $v \in \mathcal{V}$ and an edge weight $\omega(u, v)$ for each edge in $\mathcal{E}$ to represent the strength of the proximity relationship between the two tuples.

1084-4627/09 $25.00 © 2009 IEEE
DOI 10.1109/ICDE.2009.196

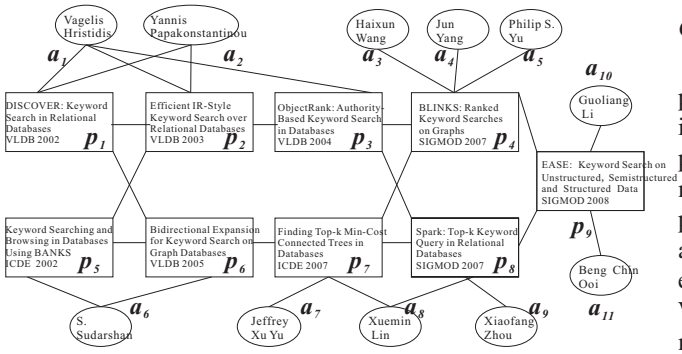

Fig. 1. A Running Example of Keyword Search over A Publication Database

For example, Figure 1 gives a partial database graph for a publication database, containing 9 publications on the topic of keyword search in relational databases. For presentation simplicity, we take the undirected graph as a running example in this paper and edge weights are not shown here. Our method applies to directed graphs with any edge weight function.

B. The Steiner Tree-based Approach

Definition 1: MINIMUM STEINER TREE: Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and $\mathcal{V}' \subseteq \mathcal{V}$, $\mathcal{T}$ is a Steiner tree of $\mathcal{V}'$ in $\mathcal{G}$ if $\mathcal{T}$ is a connected subtree in $\mathcal{G}$ covering all nodes in $\mathcal{V}'$. Let $\varpi_\xi(\mathcal{T}) = \sum_{(u,v)\in\mathcal{T}} \omega(u,v)$. $\mathcal{T}$ is a minimum Steiner tree if $\varpi_\xi(\mathcal{T})$ is the minimum among all Steiner trees of $\mathcal{V}'$ in $\mathcal{G}$.

This definition applies to both directed and undirected graphs (so trees are directed and undirected respectively). The minimum Steiner tree problem should not be confused with the minimum spanning tree problem: a Steiner tree of $\mathcal{V}'$ in $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ may contain nodes in $\mathcal{V}$-$\mathcal{V}'$. Next we introduce an extension of the minimum Steiner tree problem, allowing a node in $\mathcal{V}'$ be any representative of a group of nodes in $\mathcal{G}$.

Definition 2: MINIMUM GROUP STEINER TREE: Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and groups $\mathcal{V}_1, \mathcal{V}_2, \cdots \mathcal{V}_n \subseteq \mathcal{V}$, $\mathcal{T}$ is a minimum group Steiner tree of these given groups in $\mathcal{G}$ if $\mathcal{T}$ is a minimum Stiener tree that contains at least one node from each group $\mathcal{V}_i$, $1 \leq i \leq n$.

The problem of finding the best answer of a keyword query in a database can be translated into the problem of finding the minimum group Steiner tree in the database graph [1]. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a database graph derived from an underlying database. Let $\mathcal{K}$ be a query containing keywords k_1,k_2,$\cdots$,k_n against the database. Denote $\mathcal{V}_{k_i} \in \mathcal{V}$ for all nodes whose corresponding tuples contain keyword k_i, $1 \leq i \leq n$ (note that such groups can be easily obtained by using an inverted index). The best answer to $\mathcal{K}$ is then represented by the minimum Steiner tree $\mathcal{T}$ of $\mathcal{G}$, such that $\mathcal{V}(\mathcal{T}) \cap \mathcal{V}_{k_i} \neq \phi$ for $1 \leq i \leq n$.

The above approach returns only one solution which is represented by the minimum group Steiner tree. In the context of keyword search, however, a user is typically not satisfied with just one answer deemed by the system as the best; rather, they are interested in more answers ranked by their relevance to the query, i.e., k Steiner trees with smallest $\varpi_\xi(\mathcal{T})$ values.

C. Minimum Path Weight Steiner Tree

Finding *minimum (group) Steiner tree* is an important problem in many applications. These problems have been investigated extensively over the last three decades. These problems are known to be NP-hard [3]. A number of approximation algorithms with polynomial time complexity have been proposed [3]. The existing polynomial algorithms, however, are designed to find minimum Steiner trees. They are not efficient to identify top-k answers for keyword search problem. We propose *minimum path weight Steiner tree* to approximate minimum Steiner tree. Let $u \rightsquigarrow v$ be the shortest path from node u to node v, and $\omega(u \rightsquigarrow v)$ be its path weight, which is the sum of the weight of each edge along the path. The root of a Steiner tree $\mathcal{T}$ is denoted as $\mathsf{root}(\mathcal{T})$.

Definition 3: MINIMUM PATH WEIGHT STEINER TREE: Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and $\mathcal{V}' \subseteq \mathcal{V}$. $\mathcal{T}$ is a Steiner tree of $\mathcal{V}'$ in $\mathcal{G}$. Let $\varpi_\rho(\mathcal{T}) = \sum_{u\in\mathcal{V}'} \omega(\mathsf{root}(\mathcal{T}) \rightsquigarrow u)$. $\mathcal{T}$ is a minimum path weight Steiner tree if $\varpi_\rho(\mathcal{T})$ is the minimum among all Steiner trees of $\mathcal{V}'$ in $\mathcal{G}$.

It is easy to extend the above definition to define the concept of *minimum path weight group Steiner tree* (MPWGST), in a way similar to how the minimum group Steiner tree is defined.

D. Polynomial Algorithms

Now we consider how the MPWST problem can be solved using a polynomial time complexity algorithm. Given a graph $\mathcal{G}$=$(\mathcal{V}, \mathcal{E})$ and a Steiner node set $\mathcal{V}'$⊆$\mathcal{V}$, a minimum path weight Steiner tree can be found in the following three steps: (1) the shortest paths from each node $s \in \mathcal{V}'$ to all other nodes in $\mathcal{G}$ are computed using, for example, the *Dijkstra* algorithm in O($|\mathcal{V}'|$*$|\mathcal{V}|^2$) time. (2) for each node v∈$\mathcal{V}$, a Steiner tree rooted at v and contains the shortest paths from v to all the nodes in $\mathcal{V}'$ is constructed. (3) the minimum path weight Steiner tree can be found among the trees constructed in the previous step, O($|\mathcal{V}'|$*$|\mathcal{V}|$). Hence, the total complexity of the algorithm is O($|\mathcal{V}'|$*$|\mathcal{V}|^2$), which can be further improved, by using a more efficient shortest path algorithm, to O($|\mathcal{V}'|$*($|\mathcal{V}|log(|\mathcal{V}|) + |\mathcal{E}|$)). In a similar way, the MPWGST problem can also be solved in polynomial time.

E. Performance Ratio Analysis

We use a standard measurement of *performance ratio* [3] to quantify MPWST and MPWGST. For an approximation algorithm $\mathcal{A}$, the performance ratio $\mathcal{R}_\mathcal{A}$ is defined as the maximum ratio between the solutions returned by $\mathcal{A}$ and the optimum solution($\mathcal{OPT}$) for all instances. For a minimization problem like minimum Steiner tree problem, the performance ratio is defined in [3] as below:

$$\mathcal{R}_\mathcal{A} = supremum\{\frac{\mathcal{A}(I)}{\mathcal{OPT}(I)} | all\ instances\ I\} \quad (1)$$

Theorem 1: Given an undirected graph $\mathcal{G}$=$(\mathcal{V}, \mathcal{E})$ and a node set $\mathcal{V}'$⊆$\mathcal{V}$ ($|\mathcal{V}'| \geq 2$), the performance ratio of MPWST is $\lfloor \frac{|\mathcal{V}'|}{2} \rfloor$. Given a set $\{\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_n\}$ of sets of nodes in $\mathcal{G}$, the performance ratio of MPWGST is $\lfloor \frac{n}{2} \rfloor$.

We can always find a solution to the minimum (group) Steiner tree problem that is larger than the optimum solution by at most $\lfloor\frac{|\mathcal{V}'|}{2}\rfloor(\lfloor\frac{n}{2}\rfloor)$. In a keyword-based search problem, the number of input keywords (i.e., n) is usually not large, generally no larger than 5. Thus, the performance ratio of MPWST (MPWGST) is usually at most 2. Furthermore, the average query length (the average number of input keywords) is 1.7 terms for popular queries and 2.2 terms over all queries [4]. Thus the performance ratio of our proposed methods is exactly 1 for most of keyword queries. We give more tighter performance ratios for database graphs.

Theorem 2: Given an undirected graph $\mathcal{G}$=$(\mathcal{V},\mathcal{E})$ where edge weights are assigned as 1, the performance ratio of MPWST is $\min\{\lfloor\frac{|\mathcal{V}'|}{2}\rfloor,\frac{\gamma+1}{2}\}$, where γ is the maximal path weight in the Steiner tree. Given a set $\{\mathcal{V}_1,\mathcal{V}_2,\cdots,\mathcal{V}_n\}$ of sets of nodes, the performance ratio of MPWGST is $min\{\lfloor\frac{n}{2}\rfloor,\frac{\gamma+1}{2}\}$.

Theorem 3: Given a directed graph $\mathcal{G}=(\mathcal{V},\mathcal{E})$ and a node set $\mathcal{V}'\subseteq\mathcal{V}$ ($|\mathcal{V}'|\geq 2$), the performance ratio of MPWST is $|\mathcal{V}'|-1$. Given a set $\{\mathcal{V}_1,\mathcal{V}_2,\cdots,\mathcal{V}_n\}$ of sets of nodes in $\mathcal{G}$, the performance ratio of MPWGST is $n-1$.

Theorem 4: Given a directed graph $\mathcal{G}=(\mathcal{V},\mathcal{E})$, the performance ratio of MPWST is $\min\{|\mathcal{V}'|-1,\ 2{*}\gamma{-}1\}$, where γ is the maximal path weight in the Steiner tree. Given a set $\{\mathcal{V}_1,\mathcal{V}_2,\cdots,\mathcal{V}_n\}$ of sets of nodes in $\mathcal{G}$, the performance ratio of MPWGST is $min\{n-1,\ 2{*}\gamma{-}1\}$.

III. Compact Steiner Tree

In this section, we introduce a novel concept called *Compact Steiner Tree* to improve the search efficiency of top-k queries.

A. Compact Steiner Tree

Clearly, not all nodes in a database graph are of equal importance to a term. For any term t, a node v in the graph can *contain* the term if t appears in the tuple of v; or *imply* the term if v does not contain t but there exists a path in the graph from v to any node that contains t; or *irrelevant* if it does not contain nor imply t. A node is *relevant* to t if it either contains or implies t. Intuitively, the relevance of a node u to a term t can be partially measured by the path weight of the shortest path from u to any node that contains t. Using the notion of the shortest path, we define *Vironoi partition*.

Definition 4: Voronoi Partition: Let $\mathcal{G}=(\mathcal{V},\mathcal{E})$ be a database graph and t be a term. Let $\{v_1,v_2,\cdots,v_n\}\subseteq\mathcal{V}$ be the nodes containing t, and $U=\{u_1,u_2,\cdots,u_m\}\subseteq\mathcal{V}$ be the nodes implying t. A Voronoi partition of U for term t is defined as $U_1,U_2\cdots U_n\subseteq U$ such that $\cup_{i=1}^{n}U_i=U$ and for any node $u\in U_i$, $\omega(u\rightsquigarrow v_i)\leq\omega(u\rightsquigarrow v_j)$, $1\leq j\leq n$.

When $u\in U_i$, u is said to be *dominated* by v_i with respect to t. We denote this fact as $\mathsf{vp}(u,t)$=v_i. We call $\mathsf{vp}(u,t)$ the *Voronoi-node* of u with respect to t; and the shortest path $u\rightsquigarrow\mathsf{vp}(u,t)$ the *Voronoi-path*. Note that such a Voronoi partition for a term can be effectively pre-computed [5] and materialized. Based on the notion of Voronoi partitions, we introduce the concept of *Compact Steiner Tree* as follows.

Definition 5: Compact Steiner Tree (CSTree): Given a graph $\mathcal{G}=(\mathcal{V},\mathcal{E})$ and groups $\mathcal{V}_1,\mathcal{V}_2,\cdots\mathcal{V}_n\subseteq\mathcal{V}$, $\mathcal{T}$ is a compact Steiner tree of these given groups in $\mathcal{G}$ if $\mathcal{T}$ is a group Steiner tree of these groups, and the root of $\mathcal{T}$ is dominated by at least one node in $\mathcal{T}$ from each group $\mathcal{V}_i$, $1\leq i\leq n$.

A CSTree is said to be compact because all the Steiner nodes in such a tree must have the shortest path to the root. If each $\mathcal{V}_i$ means all the nodes containing a unique term, then from the above definition the root of a CSTree must be highly relevant to all n terms. Note that a CSTree is a special type of group Steiner tree. The intuition of finding top-k minimum path weight CSTrees, instead of finding top-k minimum path weight group Steiner trees, is that the CSTrees are highly compact and can offer much more information than top-k minimum path weight group Steiner trees.

B. CSTrees vs. MPWGSTs

Theorem 5: Let ST be a minimum MPWGST and CST be a minimum CSTree in the same database graph for the same set of search terms. We have ST≡CST.

There is no difference between MPWGSTs and CSTrees for top-k keyword queries when $k=1$. When $k>1$, there is a significant difference. First, let us explain why it can be prohibitively expensive to compute top-k MPWGSTs using the algorithm described in Section II-D. Given groups $\mathcal{V}_1,\mathcal{V}_2,\cdots,\mathcal{V}_n$, there exist $|\mathcal{V}|{*}\prod_{i=1}^{n}|\mathcal{V}_i|$ possible Steiner trees. It is inefficient to identify the top-k minimum MPWGSTs from this rather large number of possible Steiner trees. On the other side, for each CSTree, its root must be dominated by its Steiner nodes and there are no more than $|\mathcal{V}|$ CSTrees. We can then construct the top-k CSTrees by adapting the algorithm of generating the minimum MPWGST. One important property of CSTrees is that the top-k MPWGSTs can easily be constructed from the top-k CSTrees.

C. Finding CSTrees

As not all nodes that contain or imply a term are equally important to the term, we propose a more effective scoring function to assign node weights.

$$\omega(v|k)=\alpha*\omega(v)+(1-\alpha)*\mathcal{R}(v,k) \tag{2}$$

where

$$\mathcal{R}(v,k)=\begin{cases} ln(1+\mathsf{tf}(v,k))*ln(\mathsf{idf}(k)) & v \text{ contains } k \\ \mathcal{P}(v\rightsquigarrow\mathsf{vp}(v,k))*\mathcal{R}(\mathsf{vp}(v,k),k) & v \text{ implies } k \end{cases} \tag{3}$$

$$\mathcal{P}(v\rightsquigarrow\mathsf{vp}(v,k))=\begin{cases} \prod_{(x,y)\in(v\rightsquigarrow\mathsf{vp}(v,k))}\frac{1}{\mathcal{D}_{in}(y)} & \text{directed graph} \\ \prod_{(x,y)\in(v\rightsquigarrow\mathsf{vp}(v,k))}\frac{1}{\mathcal{D}(y)} & \text{undirected graph} \end{cases} \tag{4}$$

$\omega(v|k)$ is the node weight of v w.r.t. term k by integrating node prestige $\omega(v)$ and tf·idf based relevance $\mathcal{R}(v,k)$, which evaluates the relevance between v and k. α is a tuning parameter to

TABLE I
STRUCTURE-AWARE TABLE

Term	Node	EdgeWeight	NodeWeight	Voronoi-path
Graph	p_4	0	$\omega(p_4)$+$ln2$*$ln10$	p_4
Top-k	p_4	1	$\omega(p_4)$+$\frac{1}{5}$*$ln2$*$ln10$	p_4-p_7
...	...	...	...	...

differentiate the importance of node prestige and tf·idf based score, and is usually set to 0.8. $\mathsf{vp}(v,k)$ is the node which dominates v with respect to k. Note that $\mathsf{vp}(v,k)$ must contain k, thus $\mathcal{R}(\mathsf{vp}(v,k),k)$ can be computed based on tf and idf. $v \rightsquigarrow \mathsf{vp}(v,k)$ denotes the directed (undirected) path from v to $\mathsf{vp}(v,k)$. $\mathcal{P}(v \rightsquigarrow \mathsf{vp}(v,k))$ denotes the probability of randomly walking from v to $\mathsf{vp}(v,k)$. $\mathsf{tf}(v,k)$ denotes the term frequency of k in v. $\mathsf{idf}(k)$ denotes the inverse document frequency of k in the database. By integrating the node weight and edge weight, we propose a more effective ranking function:

$$\begin{aligned}\psi(\mathsf{CSTree},\mathcal{K}) &= \mathsf{Node\text{-}Score} + \beta * \mathsf{Edge\text{-}Score} \\ &= \sum\nolimits_{k_i \in \mathcal{K}} \omega(r|k_i) + \beta * \sum\nolimits_{k_i \in \mathcal{K}} \omega(r \rightsquigarrow \mathsf{vp}(r,k_i)) \\ &= \sum\nolimits_{k_i \in \mathcal{K}} (\omega(r|k_i) + \beta * \omega(r \rightsquigarrow \mathsf{vp}(r,k_i)))\end{aligned} \tag{5}$$

where r=root(CSTree) and β is a tuning parameter.

To maintain $\omega(r \rightsquigarrow \mathsf{vp}(r,k_i))$ and $\omega(r|k_i)$, we construct a structure-aware table, which maintains scores and Voronoi-paths as shown in Table I. To effectively answer a keyword query, we issue an SQL statement:

```
SELECT top-k Node,SUM(NodeWeight+β*EdgeWeight) AS Cost
FROM Structure-Aware-Table
WHERE Term in (k_1, k_2, ..., k_n)
GROUP BY Node
HAVING COUNT(Term)=n
ORDER BY Cost DESC
```

IV. EXPERIMENTAL STUDY

We compare with existing state-of-the-art algorithms including BLINKS [6] and EASE [7]. The datasets used include a publication database DBLP (http://dblp.uni-trier.de/xml/) and a movie database IMDB (http://www.imdb.com/). All the algorithms are coded in JAVA. All the experiments were conducted on a computer with an Intel(R) Core(TM) 2@2.33GHz CPU, 2GB RAM running Windows XP.

A. Search Efficiency

We evaluate efficiency of various algorithms. Figure 2 illustrates the experimental results. We observe that our algorithms achieve much better search performance than the existing methods EASE and BLINKS. CSTree clearly yields high search efficiency as it does not need to identify answers by discovering the relationships between tuples in different relational tables on the fly, relying instead on SQL capabilities of the DBMS to identify the answer.

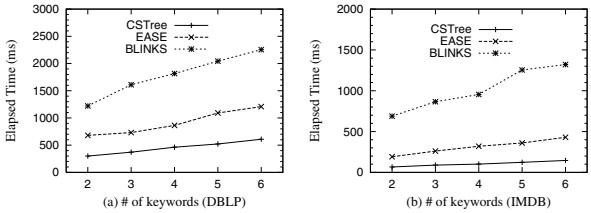

Fig. 2. Search Efficiency

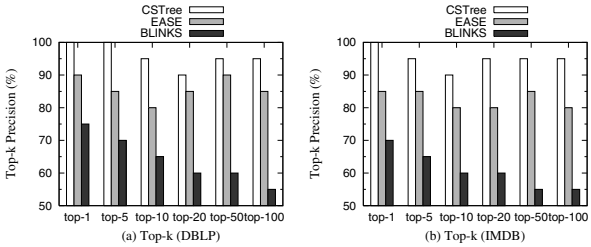

Fig. 3. Top-k Precision by Human Judgement

B. Evaluating Result Quality by Human Judgement

To further evaluate result quality of different methods, we evaluate the query results by human judgement. We select ten keyword queries on the two datasets. Answer relevance of the selected queries is judged from discussions of researchers in our database group. As users are usually interested in the top-k answers, we employ top-k precision, i.e., the ratio of the number of answers deemed to be relevant in the first k results to k, to compare those algorithms. We vary different values of k and evaluate the average top-k precision of the selected queries. The results of the average top-k precision are illustrated in Figure 3. CSTree consistently achieves high precision in all the queries, which is approximately 5-15% higher than EASE and 10-30% higher than BLINKS.

ACKNOWLEDGEMENT

This work is partly 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, Basic Research Foundation of Tsinghua National Laboratory for Information Science and Technology (TNList), and 2008 HP Labs Innovation Research Program.

REFERENCES

[1] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan, "Keyword searching and browsing in databases using banks," in *ICDE*, 2002, pp. 431–440.

[2] V. Hristidis, L. Gravano, and Y. Papakonstantinou, "Efficient ir-style keyword search over relational databases," in *VLDB*, 2003, pp. 850–861.

[3] G. Robins and A. Zelikovsky, "Improved steiner tree approximation in graphs," in *SODA*, 2000.

[4] S. M. Beitzel, E. C. Jensen, A. Chowdhury, D. Grossman, and O. Frieder, "Analysis of a very large web search engine query," in *SIGIR*, 2004.

[5] F. Aurenhammer, "Voronoi diagrams: a survey of a fundamental data structure," in *ACM Computing Surveys, 23:345-405*, 1991.

[6] H. He, H. Wang, J. Yang, and P. Yu, "Blinks: Ranked keyword searches on graphs," in *SIGMOD*, 2007.

[7] G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou, "Ease: Efficient and adaptive keyword search on unstructured, semi-structured and structured data," in *SIGMOD*, 2008.