World Wide Web (2012) 15:485–515
DOI 10.1007/s11280-011-0145-1

Top-*k* answers for XML keyword queries

Khanh Nguyen · Jinli Cao

Received: 16 April 2011 / Revised: 5 September 2011 /
Accepted: 16 September 2011 / Published online: 12 October 2011

Abstract Searching XML data using keyword queries has attracted much attention because it enables Web users to easily access XML data without having to learn a structured query language or study possibly complex data schemas. Most of the current approaches identify the meaningful results of a given keyword query based on the semantics of lowest common ancestor (LCA) and its variants. However, given the fact that LCA candidates are usually numerous and of low relevance to the users' information need, how to effectively and efficiently identify the most relevant results from a large number of LCA candidates is still a challenging and unresolved issue. In this article, we introduce a novel semantics of relevant results based on mutual information between the query keywords. Then, we introduce a novel approach for identifying the relevant answers of a given query by adopting skyline semantics. We also recommend three different ranking criteria for selecting the top-*k* relevant results of the query. Efficient algorithms are proposed which rely on some provable properties of the dominance relationship between result candidates to rapidly identify the top-*k* dominant results. Extensive experiments were conducted to evaluate our approach and the results show that the proposed approach has a good performance compared with other existing approaches in different data sets and evaluation metrics.

K. Nguyen · J. Cao (✉)
Department of Computer Science and Computer Engineering,
La Trobe University, Melbourne VIC 3086, Australia
e-mail: jinli@cs.latrobe.edu.au

K. Nguyen
e-mail: tk11nguyen@students.latrobe.edu.au

Keywords XML data · keyword search · top-k answers · mutual information · skyline queries

1 Introduction

Extensible Markup Language (XML) has become a *de facto* standard for representing and exchanging data, resulting in the proliferation of XML documents distributed over the internet. Traditionally, XML data are retrieved using query languages such as XPath [39] and XQuery [40] or twig pattern queries [18, 34–36], where users have to learn both data schemas and the query languages in order to effectively issue queries. Since the data schemas and the query languages may be complex, searching XML data using the XPath/XQuery languages is usually limited to advanced users.

Keyword searches have been widely accepted as a friendly mechanism to search flat documents [13, 21] on the Internet and has been hugely successful with the popularity of several web search engines such as Google, Yahoo!, Bing, etc,... Using keyword searches for querying XML data also has attracted the attention of the research community from both the fields of information retrieval (IR) and databases in recent years because it can liberate users from the steep learning curve of query languages and the schemas of XML data.

As a result, several approaches have been proposed to identify the relevant results of keyword queries over XML data [7, 8, 12, 19, 23, 25, 26, 31, 32, 37, 38, 41]. The most basic approach (or baseline approach) uses the lowest common ancestor (LCA) semantics from graph theory [4] to identify the results of a given keyword query. This approach returns search results consisting of all candidates (i.e., subtrees) which contain at least one instance of the query keywords. However, the returned LCA candidates are potentially numerous and of low relevance to the user's information need. Thus, how to identify the most relevant results from this potentially large number of LCA candidates becomes a challenging and unresolved issue.

Recently, many proposals [8, 12, 19, 37] have been made to improve the precision of the baseline approach. Most of them focus on identifying a set of meaningful candidates from the LCA candidates using heuristic-based rules. The common idea behind these proposals is to define a set of heuristic-based rules which a relevant result has to satisfy. Then, the candidates which satisfy the heuristic-based rules will be returned as the results of the query, while those candidates which violate the the rules are filtered. Their approaches are intuitivel but very ad-hoc. It has been experimentally proved by [24] that these approaches not only miss relevant results (known as *false negatives*) but also return irrelevant results (known as *false positives*).

To improve the result quality of XML keyword queries, we believe that the following requirements need to be addressed: (R_1) effectively measuring the degree of relevance of a candidate; (R_2) providing an effective mechanism to identify the most relevant results from a large number of candidates; and (R_3) effectively ranking the returned results in descending order of the degree of relevance to the query. In this article, we introduce a novel approach to quantify the relevance of a candidate by using the concept of *mutual information* for fulfilling requirement R_1. We introduce a new algorithm for selecting the most relevant results of a given keyword query using the semantics of *skyline queries* [5, 6], which has been proven as an effective mechanism to select the most relevant results from a large number of candidates, to satisfy requirement R_2. Finally, we propose our three ranking criteria which can effectively rank the returned results as the requirement of R_3. Overall, the contributions of this article are summarized as follows.

- Proposing a novel approach to evaluate how strong the relationship is between the query keywords in a candidate, based on the mutual information concept from information theory.
- Presenting a new semantics called *dominance lowest common ancestor* (DLCA) of relevant results. An efficient algorithm is proposed for selecting DLCA answers for a given keyword query from a potentially large number of query candidates using the skyline semantics.
- Introducing different promising ranking criteria to recommend the top-k relevant results and exploiting some important skyline properties to accelerate our top-k ranking algorithms.
- Conducting extensive experiments were conducted to evaluate our approach. The experimental results show that our approach outperforms other existing approaches in a wide range of data sets and evaluation metrics.

The rest of this article is organized as follows. Section 2 presents preliminaries and the related work. Section 3 introduces a novel approach for evaluating the relationship between query keywords in a candidate using the concept of mutual information from information theory. Section 4 defines DLCA semantics to identify relevant results using skyline semantics. Section 5 introduces our three ranking criteria for retrieving the top-k results. Efficient algorithms are developed in Section 6. Section 7 discusses experimental results and finally, the conclusions are given in Section 8.

2 Preliminaries and related work

2.1 Data model and query

In this article, we use the conventional labeled directed tree notation to represent XML documents, which is formally defined as follows.

Definition 1 (XML data tree) An XML data tree is defined as $T =< V_T, E_T >$ where V_T is a finite set of nodes, representing elements and attributes in the corresponding XML document. Each node $v \in V_T$ is labeled with a tag $\lambda(v)$. If v is a leaf node, it has a content value $val(v)$ that contains a list of keywords. We assume that each node v has a unique id $id(v)$; E_T is set of directed edges where each edge $e(v_1, v_2)$ represents the parent-child relationship between two nodes $v_1, v_2 \in V_T$.

Each subtree $S = (V_S, E_S)$ of T is a tree such that $V_S \subseteq V_T$ and $E_S \subseteq E_T$. For example, Figure 1 illustrates an XML data tree that represents an excerpt of a bibliographical database.

Definition 2 (Keyword query) A keyword query is a set of different terms, denoted by $Q = \{k_1, k_2, \ldots, k_q\}$.

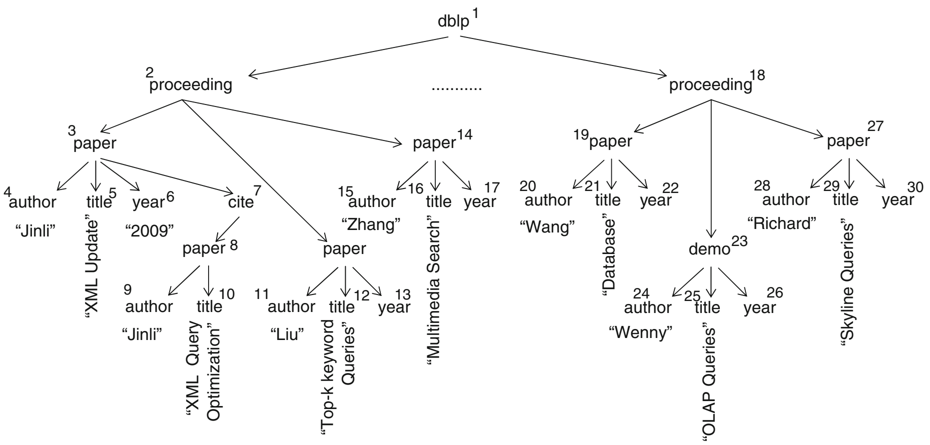


Figure 1 Example XML data tree T.

In this article, we consider the AND-semantics for the query, which is popularly used in the literature [7, 8, 12, 19, 23]. As a requirement of this semantics, a query result must contain at least one occurrence of each term in the query.

Consider a keyword query $Q = \{k_1, \ldots, k_q\}$ consisting of q keywords and a labeled XML tree $T =< V_T, E_T >$ of an XML document D. We say that node $v \in V$ directly contains a keyword k_i, denoted as *contains*(v, k_i) if the keyword appears as the label of the element or in the content of the element. We denote $S_1, \ldots, S_q$ as the sets of nodes such that $S_i = \{v | v \in V \text{ and } contains(v, k_i)\}$, $1 \leq i \leq q$.

Intuitively, the results of query Q contain a set of subtrees in T whose nodes contain all the keywords of Q. The roots of the subtrees are identified using the semantics of the lowest common ancestor. The lowest common ancestor of a set of nodes $V' = \{v_1, \ldots, v_q\} (V' \subset V)$ is defined as the deepest node v in T which is an ancestor of all nodes in V' and evaluated by a function $lca(v_1, \ldots, v_q)$.

We refer to such subtrees as the candidates of the query which is formally defined as follows.

Definition 3 (Query candidates) A subtree S_T is a candidate of query Q in the data tree T if its leaf nodes contain at least one instance of each keyword in query Q. The root of the candidate answer is the lowest common ancestor of its leaf nodes.

For the keyword query $Q = \{k_1, \ldots, k_q\}$, we can see that the number of candidates of Q equals to $|S_1| \times \cdots \times |S_q|$, where $|S_i|$ the number of nodes in S_i.

2.2 Identifying results for XML keyword queries

Based on the notion of lowest common ancestor (LCA) from graph theory [4, 14, 30], there have been several approaches proposed to identify meaningful and relevant results to XML keyword searches [8, 12, 19, 37].

The baseline approach is to return a search result consisting of all candidates. However, the candidates of a given keyword query are potentially numerous and of low relevance to the user's information need. In order to identify the most relevant results from that potentially large number of LCA candidates, many proposals [8, 12, 19, 37] have been put forward to boost the precision of the baseline approach. These approaches rely on heuristic-based approaches to identify a set of meaningful candidates from the candidate set. The common idea behind these approaches is to define a set of heuristic-based rules which a relevant result has to satisfy and to prune those candidates which do not satisfy the defined rules.

Exclusive LCA (ELCA) [12] This semantics is very close to the semantics of the lowest common ancestor. The set of ELCA nodes is the set of LCA nodes after excluding any node v that is LCA of a set of nodes $V' = \{v_1, \ldots, v_q\}$ such that $\exists v_i \in V'$ that also belongs to the subtree of another node u that is also a descendant of v. This semantics has a high recall but very low precision, which is a similar problem to the LCA semantics.

Smallest LCA (SLCA) [37] The set of SLCA nodes, for instance, is the set of LCA nodes if we exclude any LCA nodes that are ancestors of other LCA nodes. The drawback of this approach is that the relationships between the query keywords in the subtrees rooted at SLCA nodes were ignored. As a results, it can return irrelevant results. In addition, this approach relies on the hierarchical (i.e., ancestor/descendant) relationship to filter out more general candidates (i.e., candidates which are the ancestors of other candidates). Thus it may unsuccessfully prune irrelevant results which do not have the ancestor/descendant relationship with the other candidates.

XSEarch [8] is the first work which evaluates the relationship between the query keywords in a candidate. Two nodes u and v containing the query keywords are meaningfully related if their shortest connecting path does not have two distinct nodes with the same label, except u and v. A subtree S is meaningful if it satisfies the two following conditions: (i) S contains matches to all the query keywords; and (ii) every two matches in S have to be meaningfully related (all-pair semantics) or there exists a node in S such that all the other nodes in S must be meaningfully related to it (star-semantics). However, this approach fails to recognize a weak (or even meaningless) relationship between two nodes which do not have two distinct nodes with the same label on their connecting path (i.e., node 20 and node 25 in the data tree T illustrated in Figure 1).

In [19], a semantics called *Compact Valuable LCA (CVLCA)* has been introduced to identify the results of keyword queries. A node u is called VLCA if the subtree S rooted at u satisfies the star-semantics of XSEarch [8]. A node u is a CVLCA if it is a VLCA node and u dominates every nodes in S, where u dominates a node v in S if there does not exist another LCA node u' which is a descendant of u and the subtree rooted at u' contains v. This approach faces the same problem as XSEarch [8].

Recently, Li et al. [22] proposed an approach called *Meaningful LCA (MLCA)* for pruning irrelevant subtrees based on the structure of XML data, rather than using node labels as XSEarch [8] and CVLCA [19]. Two nodes u and v are meaningfully related if there does not exist node u' (or v') which has the same label with u (or v) and $lca(u', v)$ (or $lca(u, v')$ is a descendant of $lca(u, v)$).

In summary, all of these approaches evaluate the relevance of a candidate in a boolean manner. It means that a candidate is considered a relevant result if it satisfies a set of pre-defined heuristics rules. Otherwise, it is considered an irrelevant result. However, keyword queries are frequently short and ambiguous in terms of expressing the user's search intention, thus it is difficult (sometimes impossible) to exactly conclude if a candidate is a relevant result or an irrelevant result. With that observation, we believe that a quantitative approach for quantitatively evaluating the relevance degree of a candidate is more appropriate.

Some approaches [20] have been proposed to quantitatively measure the relevance level of a candidate by using the length of the paths connecting its leaf nodes. However, there are many cases where the correlation between two pairs of nodes is significantly different while their path length is the same and vice versa. To more effectively measure the correlation between two nodes in a query candidate, we adopt the concept of *mutual information* from information theory [9] which has been successfully used in mining the meaningful correlation between the attributes of a relation. The details of the approach will be introduced in Section 3.

In addition, in order to effectively identify a set of relevant results from a potentially large number of candidates of a given keyword query, we employ the semantics of skyline queries.

2.3 Skyline semantics

Multi-object optimization has been widely studied in the literature, initially as the *maximum vector problem* [17] and more recently as *skyline queries* [5]. Given a set of points in a d-dimension space. The skyline is defined as the subset containing those points that are not dominated by any other points, whereas point p_i dominates point p_j if p_i is better than or equal to p_j in all the dimensions and strictly better in at least one dimension. Thus, the best answer for such a query exists in the skyline. However, how to apply the skyline semantics to the scenario of keyword queries on XML database is an interesting and unresolved problem.

Skyline queries have received much attention in recent years. Consequently, several algorithms have been proposed [3, 5, 6, 16, 33]. BNL [5] is a straightforward skyline algorithm. This algorithm iterates over the data set to compare each point with every other point, and reports the points that are not dominated by any other points. SFS [6] improves the efficiency of BNL by pre-sorting the input according to a monotone ranking function to reduce the number of dominance tests required. SaLSa [3] proposes an additional modification so that the computation may terminate before scanning the whole data set. These algorithms are generic, in the sense that they do not require any specialized data indexes to compute the skyline, and can therefore be applied even when the points are the results of some other operations. Although our work adapts the basic techniques underlying these methods, they are not directly applicable to our problem, as these approaches do not deal with ranking issues.

Some other algorithms rely on the existence of appropriate indexes, such as B^+-tree or R-tree to speed-up skyline computations [16, 33]. Note that these approaches only apply to static data, where the overhead for building the indexes is amortized across multiple queries. In our setting, the underlying data (or returned candidates) are highly dependent on the submitted query. In this case, building indexes at query time is very expensive, thus these approaches are not applicable to our problem.

3 Measuring the relationship between two nodes in a data tree

In this section, we review the concept of mutual information (MI) and its related concepts. The adaptation of this concept for measuring the meaningful relationship between two nodes in an XML data tree will be discussed in detail.

3.1 Concepts of mutual information

Entropy and *mutual information* are two of the central concepts in information theory [9]. Entropy is a measure of the uncertainty of a random variable, while MI quantify the mutual dependence of two random variables.

Entropy Let x be a discrete random variable that takes value v_x from the set $dom(x)$ with a probability distribution function $p(v_x)$. The entropy of x is defined as follows.

$$H(x) = -\sum_{v_x \in dom(x)} p(v_x) \log p(v_x)$$

The *conditional entropy* of a random variable y given another variable x, referred to as the entropy of y conditional on x, denoted as $H(y|x)$, is defined as follows.

$$H(y|x) = -\sum_{v_y \in dom(y)} \sum_{v_x \in dom(x)} p(v_x, v_y) \log p(v_y|v_x)$$

where $p(v_x, v_y)$ is the joint probability of $(x = v_x)$ and $(y = v_y)$; $p(v_y|v_x)$ is the conditional probability of $(y = v_y)$ given that $(x = v_x)$.

Mutual information The mutual information of two random variables is a quantity that measures the mutual dependence of the two variables. Formally, given two discrete random variables x and y, their mutual information can be defined as follows.

$$I(x; y) = \sum_{v_y \in dom(y)} \sum_{v_x \in dom(x)} p(v_x, v_y) \log \frac{p(v_x, v_y)}{p(v_x) p(v_y)}$$

where $p(v_x, v_y)$ is the joint probability of $(x = v_x)$ and $(y = v_y)$; $p(v_x)$ and $p(v_y)$ are the probability of $(x = v_x)$ and $(y = v_y)$ respectively.

The mutual information has some properties. Detailed proofs of the properties can be found in [9].

Property 1 $I(x; y) = H(x) - H(x|y) = H(y) - H(y|x)$.

Property 1 gives an important interpretation of the mutual information. It indicates that the information that y tells us about x is the reduction in the uncertainty of x given the knowledge of y, and similarly, for the information that x tells us about y. The greater the value of the mutual information of the two variables x and y, the more information x and y tell us about each other.

Property 2 $I(x; y) = I(y; x)$.

Property 2 suggests that the mutual information is symmetric, which means the amount of information x tells about y is the same as y tells about x.

Property 3 $I(x; y) \geq 0$.

Property 3 gives the lower bound for the mutual information. When $I(x; y) = 0$, we have $p(v_x, v_y) = p(v_x)p(v_y)$ for every possible value of x and y, which means that x and y are independent, then knowing x does not give any information about y and vice versa, so their mutual information is zero.

Property 4 $I(x; x) = H(x)$.

Property 4 states that the mutual information of x with itself is the entropy of x. Thus, entropy is also called self-information.

Property 5 $I(x; y) \leq H(x)$ *and* $I(x; y) \leq H(y)$

Property 5 indicates that the mutual information of two variables is bounded by the minimum of their entropy.

3.2 Evaluating node relationship

Different from the existing work which evaluates the relationship between two nodes using some intuitively heuristics-based rules, in this article, the relationship between two nodes in a data tree is measured by adapting the mutual information concept which has been successfully used in mining the correlation between the attributes of a database relation.

Since it is common for XML data tree to contain nodes with the same label in different contexts, we use prefix label paths to denote node types. A prefix label path is a sequence of element names that appear in the path from the root to the node in question. We identify each node in the data tree by its node type. For example, the prefix label path of node 4 in data tree T shown in Figure 1 is `dblp/proceeding/paper/author`. A prefix label path can have many occurrences in the XML data tree. We called all occurrences of a prefix label path in the data tree "node instances". It is obvious that all instances of a node type have the same prefix label path. Each instance has a value which is a set of keywords directly contained in that instance. Consider data tree T in Figure 1, for instance, prefix label path `dblp/proceeding/paper/author` has five instances 4, 11, 15, 20 and 28 which have values "Jinli", "Liu", "Zhang", "Wang" and "Richard", respectively. We call a set of distinct values of all instances of prefix label path u the value domain of u, denoted as $dom(u)$. We have, for instance dom(`dblp/proceeding/paper/author`)={"Jinli", "Liu", "Zhang", "Wang", "Richard"}. Each prefix label path can take a value from its value domain with a specific probability.

Probability The probability of a node u getting value v_u from $dom(u)$ in the data tree T can be defined as the number of instances with the value v_u over the total number of instances of u in the data tree T. For example, node `dblp/proceeding/paper/author` takes value "Jinli" with the probability $p($`dblp/proceeding/paper/author`="Jinli"$) = 1/5$.

Join probability Due to the hierarchical structure of XML data, any two nodes in an XML tree can be joined at different node levels which leads to different joint distribution between the two nodes. Considering the data tree T shown in Figure 1, for instance, two nodes `dblp/proceeding/paper/author` and `dblp/proceeding/paper/title` can be joined at node `dblp/proceeding/paper`, through their instances 4 and 5, but they also can be joined at node `dblp/proceeding`, through their instances 4 and 12. In this article, we refer to a node c at which two nodes u and v joined as the context node of u and v. We define $p(v_u, v_v|c)$ is the join probability of node u getting the value v_u and node v getting the value v_v when they join at the context node c. For example, $p($`dblp/proceeding/paper/author`="Jinli", `dblp/proceeding/paper/title`="XML update" | `dblp/proceeding/paper`$) = 1/5$. Similarly, $p($`dblp/proceeding/paper/author`="Jinli", `dblp/proceeding/paper/title`="multimedia search" | `dblp/proceeding`$) = 1/8$. The join probability of nodes `dblp/proceeding/paper/author` and `dblp/proceeding/paper/title` at the context nodes `dblp/proceeding/paper`) and `dblp/proceeding` in XML data tree T in Figure 1 is shown in Tables 1 and 2 respectively, where the first column of the tables represent value domain of node `dblp/proceeding/paper/author` and the first row of the tables represent the value domain of node `dblp/proceeding/paper/title`. We use the '–' sign to indicate that the join probability is equal to zero.

From the definitions of probability and join probability, we adaptively define the mutual information of two nodes u and v at the context node c in an XML data tree as follows.

Definition 4 (Mutual information of two nodes) Let u and v be two nodes which join at the context node c in an XML data tree. The mutual information of u and v at the context node c is defined as:

$$I(u; v|c) = \sum_{v_u \in dom(u)} \sum_{v_v \in dom(v)} p(v_u, v_v|c) \log \frac{p(v_u, v_v|c)}{p(v_u)p(v_v)}$$

Table 1 Join probability of two nodes `dblp/proceeding/paper/author` and `dblp/proceeding/paper/title` at their context node `dblp/proceeding/paper`.

	XML update	Top-K keyword queries	Multimedia search	Databases	Skyline queries
Jinli	1/5	–	–	–	–
Liu		1/5	–	–	–
Zhang	–	–	1/5	–	–
Wang	–	–	–	1/5	–
Richard	–	–	–	–	1/5

Table 2 Join probability of two nodes `dblp/proceeding/paper/author` and `dblp/proceeding/paper/title` at their context node `dblp/proceeding`.

	XML update	Top-K keyword queries	Multimedia search	Databases	Skyline queries
Jinli	–	1/8	1/8	–	–
Liu	1/8	–	1/8	–	–
Zhang	1/8	1/8	–	–	–
Wang	–	–	–	–	1/8
Richard	–	–	–	1/8	–

where $p(v_u)$ and $p(v_v)$ are the probability of $(u = v_u)$ and $(v = v_v)$ respectively; $p(v_u, v_v|c)$ is the join probability of $u = v_u$ and $v = v_v$ at the context node c.

The higher value of the mutual information between two nodes in a data tree indicates their stronger relationship. For example, from the join probability shown in Table 1, we can calculate the mutual information of nodes `dblp/proceeding/paper/author` and `dblp/proceeding/paper/title` at their context node `dblp/proceeding/paper` as

$$I(\texttt{dblp/proceeding/paper/author}; \texttt{dblp/proceeding/paper/title} \\ |\texttt{dblp/proceeding/paper}) \\ = (1/5)\log\frac{1/5}{(1/5)(1/5)} + (1/5)\log\frac{1/5}{(1/5)(1/5)} + (1/5)\log\frac{1/5}{(1/5)(1/5)} \\ + (1/5)\log\frac{1/5}{(1/5)(1/5)} + (1/5)\log\frac{1/5}{(1/5)(1/5)} = \log\frac{1/5}{(1/5)(1/5)} = \log 5 = 0.70$$

Similarly, from the join probability shown in Table 2, we have the mutual information of nodes `dblp/proceeding/paper/author` and `dblp/proceeding/paper/title` at the context node `dblp/proceeding` calculated as

$$I(\texttt{dblp/proceeding/paper/author}; \texttt{dblp/proceeding/paper/title} \\ |\texttt{dblp/proceeding}) \\ = (1/8)\log\frac{1/8}{(1/5)(1/5)} + (1/8)\log\frac{1/8}{(1/5)(1/5)} + (1/8)\log\frac{1/8}{(1/5)(1/5)} \\ + (1/8)\log\frac{1/8}{(1/5)(1/5)} + (1/8)\log\frac{1/8}{(1/5)(1/5)} + (1/8)\log\frac{1/8}{(1/5)(1/5)} \\ + (1/8)\log\frac{1/8}{(1/5)(1/5)} + (1/8)\log\frac{1/8}{(1/5)(1/5)} = \log\frac{1/8}{(1/5)(1/5)} = \log\frac{25}{8} = 0.49$$

This example indicates that the mutual information of two nodes in different contexts can be varied. More specifically, we can see that the mutual information between the title and author(s) in the context of a paper is much higher than the mutual information between the title and author(s) of two different papers (or in the context of a proceeding).

Although the MI serves as a good measure to quantify how closely two nodes are related to each other, the scale of the MI values does not fall in a unique range, as shown by Property 5. Property 5 indicates that the MI of two nodes can be bounded by the minimum of their entropy. Since the entropy of different nodes varies greatly, the value of MI also varies for different pairs of nodes. To apply MI to our problem, we require a unified scale for measuring MI among a global set of nodes. For this purpose, we measure the relationship between two nodes as their normalized mutual information, which is formally defined as follows.

Definition 5 (Node relationship) Let u and v be two nodes which join at the context node c in an XML data tree. The relationship between two nodes u and v at the context node c is defined as:

$$\texttt{rel}(u; v|c) = \frac{I(u; v|c)}{\max\{H(u), H(v)\}} \tag{1}$$

where $H(u)$ and $H(v)$ are the entropy of two nodes u and v respectively, and they are calculated in the same way as the entropy of a random variable.

The higher value of $\texttt{rel}(u; v|c)$ indicates a stronger relationship between the two nodes u and v at their context node c. For example, the entropy of two nodes `dblp/proceeding/paper/author` and `dblp/proceeding/paper/title` is calculated as follows:

$$\begin{aligned} &H(\texttt{dblp/proceeding/paper/author}) \\ &\quad = -[(1/5)\log(1/5) + (1/5)\log(1/5) \\ &\quad\ (1/5)\log(1/5) + (1/5)\log(1/5) + (1/5)\log(1/5)] \\ &\quad = -\log(1/5) = \log 5 = 0.70 \end{aligned}$$

$$\begin{aligned} &H(\texttt{dblp/proceeding/paper/title}) \\ &\quad = -[(1/5)\log(1/5) + (1/5)\log(1/5) \\ &\quad\ (1/5)\log(1/5) + (1/5)\log(1/5) + (1/5)\log(1/5)] \\ &\quad = -\log(1/5) = \log 5 = 0.70 \end{aligned}$$

Then, the relationship between nodes `dblp/proceeding/paper/author` and `dblp/proceeding/paper/title` at the context node `dblp/proceeding/paper` is

$$\begin{aligned} &\texttt{rel(dblp/proceeding/paper/author; dblp/proceeding/paper/title} \\ &\quad |\texttt{dblp/proceeding/paper}) = \frac{0.70}{0.70} = 1.0 \end{aligned}$$

Similarly,

$$\begin{aligned} &\texttt{rel(dblp/proceeding/paper/author; dblp/proceeding/paper/tile} \\ &\quad |\texttt{dblp/proceeding}) = \frac{0.49}{0.70} = 0.7 \end{aligned}$$

Lemma 1 *The relationship between any two nodes u and v at any context node c in an XML data tree is always in range of* [0, 1].

$$0 \le rel(u; v|c) \le 1$$

Proof From Property 3 we have $I(u; v|c) \geq 0$, thus $rel(u; v|c) = \frac{I(u;v|c)}{\max\{H(u),H(v)\}} \geq 0$. Furthermore, Property 5 gives us $I(x; y|c) \leq H(x)$ and $I(x; y|c) \leq H(y)$. Thus, we have $rel(u; v|c) = \frac{I(u;v|c)}{\max\{H(u),H(v)\}} \leq 1$ □

4 Dominance lowest common ancestor (DLCA)

To select relevant answers from a potentially huge number of LCA-based candidates, we propose a new semantics called DLCA (Dominance LCA). Before giving the formal definition of DLCA semantics, we introduce the dominance relationship between LCA-based candidates.

4.1 Dominance relationship

Rather than identifying query candidates using their root nodes, we represent those candidates as subtrees, as in [15]. Specifically, given a keyword query $Q = \{k_1, \ldots, k_q\}$, a candidate S of Q is represented as $S(n_{lca}, \{n_1; \ldots; n_q\})$ where each n_i is a leaf node which contains the term k_i and n_{lca} is the lowest common ancestor of $\{n_1; \ldots; n_m\}$, bearing in mind that each node in the candidate is identified by its identifier which is encoded as a Dewey code in this article.

Dewey coding is based on the Dewey Decimal Classification developed for general knowledge classification [28]. With Dewey coding, each node is assigned a vector that represents the path from the trees root to the node. Each component of the path represents the local order of an ancestor node, as illustrated in Figure 2. We selected to encode the nodes' identifiers using the Dewey code because it is very useful in representing the hierarchical relationships between tree nodes which are a key to calculate the lowest common ancestor between any two nodes. In addition, we can easily find the corresponding label path of node from its Dewey code. For example, we consider the sample data tree T_2 in Figure 2 where each node is identified by its Dewey code. Given a node's id [0.1.0.0], we can easily find the corresponding label path of this node which is $n_1/n_3/n_4/n_5$. We name `ID2LP`(*id*) to be a function which takes a Dewey code *id* as an input and returns its corresponding label path.

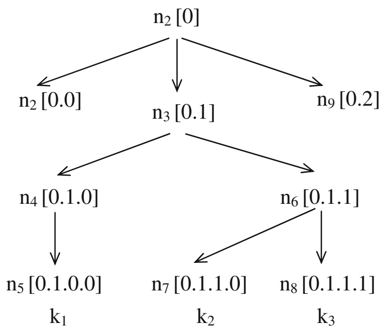


Figure 2 Sampe data tree T_2.

Since it is possible that each keyword of the query can have more than one occurrence in a subtree candidate $S(n_{lca}, \{n_1; \ldots; n_m\})$, for every keyword k_i in the query, we calculate the set $L_i = \{n_i | val(n_i)$ contains keyword k_i $(1 \leq i \leq m)\}$.

We calculate the relationship of any two keywords k_i and k_j in the candidate subtree S as follows.

$$\texttt{rel}(k_i, k_j) = \max_{n_i \in L_i, n_j \in L_j \wedge i<j} \{\texttt{rel}(\texttt{ID2LP}(n_i); \texttt{ID2LP}(n_j) | \texttt{ID2LP}(\text{lca}(n_i, n_j)))\} \quad (2)$$

where $\texttt{ID2LP}(n_i)$ is a function which returns the corresponding type of node having Dewey code n_i and $\texttt{rel}(\texttt{ID2LP}(n_i); \texttt{ID2LP}(n_j) | \texttt{ID2LP}(\text{lca}(n_i, n_j)))$ is calculated by Formula (1) which measures the relationship between nodes having Dewey codes n_i and n_j at their lowest common ancestor.

In other words, the relationship between two keywords k_i and k_j in a candidate is evaluated as the maximum relationship between two nodes containing the two keywords in that candidate.

For example, we consider a query $Q = \{k_1, k_2, k_3\}$ and a data tree T_2. There is only one candidate subtree of Q rooted at $n_3[0.1]$ in the tree T_2 and it can be represented as $S(0.1, \{0.1.0.0; 0.1.1.0; 0.1.1.1\})$. We can calculate the relationship between the query keywords in candidate S as,

$$\begin{aligned}
\texttt{rel}(k_1, k_2) &= \texttt{rel}(\texttt{ID2LP}(0.1.0.0); \texttt{ID2LP}(0.1.1.0) | \texttt{ID2LP}(0.1)) \\
&= \texttt{rel}(n_1/n_3/n_4/n_5; n_1/n_3/n_6/n_7 | n_1/n_3) \\
\texttt{rel}(k_1, k_3) &= \texttt{rel}(\texttt{ID2LP}(0.1.0.0); \texttt{ID2LP}(0.1.1.1) | \texttt{ID2LP}(0.1)) \\
&= \texttt{rel}(n_1/n_3/n_4/n_5; n_1/n_3/n_6/n_8 | n_1/n_3) \\
\texttt{rel}(k_2, k_3) &= \texttt{rel}(\texttt{ID2LP}(0.1.1.0); \texttt{ID2LP}(0.1.1.1) | \texttt{ID2LP}(0.1.1)) \\
&= \texttt{rel}(n_1/n_3/n_6/n_7; n_1/n_3/n_6/n_8 | n_1/n_3/n_6)
\end{aligned}$$

Given a keyword query $Q = \{k_1, \ldots, k_q\}$ we calculate the relationship of each pair of the query keywords in a candidate subtree S and store them in a vector D_S which we refer to as the *keyword relationship vector* in this article.

$$D_S = [\texttt{rel}(k_i, k_j) | k_i, k_j \in Q \wedge (i < j)]$$

Since there are a total of C_q^2 two-keyword combinations from a set of q keywords $\{k_1, \ldots, k_q\}$, vector D_S contains C_q^2 elements, denoted as $|D_S| = C_q^2$. For example, the keyword relationship vector corresponding to candidate $S(0.1, \{0.1.0.0, 0.1.1.0, 0.1.1.1\})$ of the query $Q = \{k_1, k_2, k_3\}$ contains $C_3^2 = \frac{3!}{2!(3-1)!} = 3$ elements: $D_S = [\texttt{rel}(k_1, k_2), \texttt{rel}(k_1, k_3), \texttt{rel}(k_2, k_3)]$.

Let D_S and D'_S be two keyword relationship vectors of two candidates S and S', respectively. We define the *dominance* relationship between candidates S and S' as follows.

Definition 6 (Dominance) Let S and S' be two candidate answers of Q over an XML database T. S' dominates S, denoted as $S' \succ S$ if the two following conditions hold:

– $\forall i (1 \leq i \leq d) D_S[i] \leq D_{S'}[i]$, and
– $\exists j (1 \leq j \leq d) D_S[j] < D_{S'}[j]$

where d is the length of keyword relationship vectors of S and S' ($d = |D_S| = |D_{S'}| = C_q^2$). The $D_S[i]$ is the i-th element in vector D_S.

In words, candidate S' dominates candidate S if the relationship between every pair of query keywords in candidate S' is at least as strong as the relationship between that pair of query keywords in candidate S. Consequently, candidate S' is more relevant to query Q than candidate S if $S' \succ S$.

4.2 Retrieving DLCA answers

From the definition of the dominance relationship defined in Definition 6, we define a set of DLCA answers for a given keyword query as follows.

Definition 7 (Dominant LCA) Given a set of candidates $\mathscr{C}(Q, T)$ of a query Q in an XML data tree T, candidate $S \in \mathscr{C}(Q, T)$ is called a DLCA candidate if there does not exist any other candidate $S' \in \mathscr{C}(Q, T)$ such that S' dominates S.

Definition 8 (Query result) The results of a keyword query Q in a data tree T is a set of all DLCA candidates of Q in the data tree T.

In the next section, we will propose three different ranking scores for identifying the top-k most meaningful candidates of Q which rely on our defined *dominance* relationship.

5 Top-*k* answers

We observe that the DLCA answers can vary with different queries. However, when searching for information, users are usually interested in the top-k answers which should be sorted in the descending order of their relevance degrees to the users' information need. In this section, we define three ranking functions that will be used to identify the top-k results for a keyword-based search over XML data. Our ranking functions exploit several different aspects of the dominance relationship between query candidates to rank their relevance degree to the query.

Given $\mathscr{C}(Q, T)$ is a set of candidates of query Q in an XML database T, we measure the degree of relevance of a candidate based on the three following ranking scores.

5.1 Dominating score

Given a candidate answer S, we define the dominating score of S as follows.

$$score_{dg}(S) = |\{S' \in \mathscr{C}(Q, T) | S \succ S'\}| \tag{3}$$

The dominating score of a candidate $score_{dg}(S)$ indicates the number of other candidates that S dominates. A candidate is more relevant if it dominates as many other candidates as possible. Thus, a higher dominating score of candidate S indicates S is more relevant to the query.

Lemma 2 *Let $S \in \mathscr{C}(Q,T)$ and $S' \in \mathscr{C}(Q,T)$ be two candidates of query Q in an XML data tree T. If $S \succ S'$, then $score_{dg}(S) \geq score_{dg}(S')$.*

Proof This lemma can be proved by using the transitive property of the dominance relationship. Specifically, for any two candidates $S \in \mathscr{C}(Q,T)$ and $S \in \mathscr{C}(Q,T)$, if $S \succ S'$, then $\forall S_i \in \mathscr{C}(Q,T) | S' \succ S_i$, we have $S \succ S_i$. Therefore, $|\{S_i \in \mathscr{C}(Q,T) | S \succ S_i\}| \geq |\{S_i \in \mathscr{C}(Q,T) | S' \succ S_i\}|$, or $score_{dg}(S) \geq score_{dg}(S')$ □

Lemma 2 guarantees that if candidate S dominates candidate S', then S will be ranked higher than S' in the returned top k results.

5.2 Dominated score

Given a candidate answer S, we define the dominated score of S as follows.

$$score_{dd}(S) = |\{S' \in \mathscr{C}(Q,T) | S' \succ S\}| \tag{4}$$

The dominated score of candidate S, $score_{dd}(S)$, indicates the number of other candidates can dominate S. Thus, the lower the dominated score of candidate S, the more meaningful to the query it is. In other words, candidate S is more relevant if it is dominated by as few other candidates as possible.

Lemma 3 *Let $S \in \mathscr{C}(Q,T)$ and $S' \in \mathscr{C}(Q,T)$ be two candidates of query Q in an XML data tree T. If $S \succ S'$, then $score_{dd}(S) \leq score_{dd}(S')$.*

Proof This lemma can be proved in a similar manner as Lemma 3. For any two candidates $S \in \mathscr{C}(Q,T)$ and $S' \in \mathscr{C}(Q,T)$, if $S \succ S'$, then $\forall S_i \in \mathscr{C}(Q,T) | S_i \succ S$, we have $S_i \succ S'$. Therefore, $|\{S_i \in \mathscr{C}(Q,T) | S_i \succ S\}| \leq |\{S_i \in \mathscr{C}(Q,T) | S_i \succ S'\}|$, or $score_{dd}(S) \leq score_{dd}(S')$ □

Lemma 3 guarantees that if candidate S dominates candidate S', then S should be ranked higher than S' in the returned top k results.

5.3 Dominance score

Given a candidate answer S, we define the dominance score of S as:

$$score_d(S) = \alpha score_{dg}(S) - (1-\alpha) score_{dd}(S), \tag{5}$$

where $\alpha (0 \leq \alpha \leq 1)$ is a tunable parameter. We set $\alpha = 0.5$ if we weight $score_{dg}(S)$ as important as $score_{dd}(S)$; $\alpha > 0.5$ if we consider $score_{dg}(S)$ as more important; otherwise $\alpha < 0.5$ we weigh $score_{dg}(S)$ as less important.

The dominance score $score_d(S)$ measures the relevance degree of a candidate answer S by considering both dominating and dominated scores. A candidate S is more relevant if it dominates as many other candidates as possible and is dominated by as few other candidates as possible. Thus, a higher dominance score of a candidate indicates that it is a more relevant answer.

Lemma 4 *Let $S \in \mathscr{C}(Q,T)$ and $S' \in \mathscr{C}(Q,T)$ be two candidates of query Q in an XML data tree T. If $S \succ S'$, then $score_d(S) \geq score_d(S')$.*

Proof Since $S \succ S'$, from Lemma 2 and 3, we have $score_{dg}(S) \geq score_{dg} S'$ and $score_{dd}(S) \leq score_d S'$. Thus, for $\forall \alpha \in [0,1]$, we have $\big(\alpha score_{dg}(S) - (1-\alpha) score_{dd}(S)\big) \geq \big(\alpha score_{dg}(S') - (1-\alpha) score_{dd}(S')\big)$, or $score_d(S) \geq score_d(S')$ □

Lemma 4 guarantees that if candidate S dominates candidate S', then S should be ranked higher than S' in the returned top k results.

6 Algorithms for retrieving top-*k* results

In this section, we introduce our algorithms to identify relevant results and the top-k answers, based on skyline semantics according to the aforementioned ranking criteria. To obtain the set of LCA-based candidates of a given keyword query, as other approaches in the literature [37, 38, 41], we adopt the inverted indexes. These indexes are built offline at the time we parsed the XML database. Specifically, let $Q = \{k_1, \ldots, k_q\}$ be a given keyword query and IL_i be the inverted list of keyword k_i. Each entry in the inverted list IL_i is the Dewey code of the node containing the keyword k_i. The set $\mathscr{C}$ of candidates of query Q can be defined as $\mathscr{C} = \{lca(n_1, n_q) | n_1 \in IL_1, \ldots, n_q \in IL_q\}$, where $lca(n_1, \ldots, n_q)$ is an operation which returns the *lowest common ancestor* of $\{n_1, \ldots, n_q\}$. The keyword relationship vector of each candidate is concurrently computed during the process of candidate generation. The generated candidates are stored in a list ordered by the values of their keyword relationship vectors. The detailed explanations will be in the following subsections.

Algorithm 1 Naïve algorithm for selecting top-*k* answers

```
input  : A set of candidates $\mathscr{C}$; the number $k$ of answers to return
output: The top-$k$ answers w.r.t score$_{dd}$ in a sorted set $\mathscr{R}$
1  $\mathscr{R} \leftarrow \{\}$;
2  foreach candidate $S$ in $\mathscr{C}$ do
3      score$_{dd}(S) \leftarrow 0$;
4      foreach other candidate $S'$ in $\mathscr{C}$ do
5          if $S' \succ S$ then score$_{dd} \leftarrow$ score$_{dd}(S) + 1$;
6      end
7      if $|\mathscr{R} = k$ then
8          $S_k \leftarrow$ the $k$-candidate in $\mathscr{R}$;
9          if score$_{dd}(S) <$ score$_{dd}(S_k)$ then
10             remove the tail result from $\mathscr{R}$;
11             $\mathscr{R} \leftarrow$ insert $S$ ordered by score$_{dd}$ asc.;
12         end
13     else $\mathscr{R} \leftarrow$ insert $S$ ordered by score$_{dd}$ asc.;
14 end
15 return $\mathscr{R}$;
```

The naïve algorithm for identifying the top-k desired results corresponding to their dominated scores (similarly, dominating and dominance scores) is illustrated in Algorithm 1. This algorithm iterates through each candidate in the candidate set and calculates its score by performing pairwise dominance checks between that candidate and all other candidates in the set (*lines 2–6*). The result set is updated depending on the result of the score comparison between the new candidate and the current $k-$th candidate in the current top-k results (*lines 7–13*).

The drawback of this algorithm is that its computational cost is extremely high because regardless the value of k, it needs to iterates through each candidate in the candidate set and calculates the score of each candidate by performing the pairwise dominance checks between the candidate with all other candidates in the set. This means that no matter what the value of k is, it exhaustively performs all pairwise dominance tests amongst candidates. For example, given a set of candidates in Table 3, in order to identify the top-3 results, we need to calculate the score of each candidate $S_i (1 \leq i \leq 10)$ by iterating over 9 other candidates and doing pairwise dominance checks. Therefore, it takes $10 \times 9 = 90$ pairwise dominance checks. In general, in order to calculate the score of a candidate in a set of n candidates, we need to do pairwise dominance checks between that candidate and $(n - 1)$ other candidates in the set.

Definition 9 (Search space) Let $\mathscr{C}(Q, T)$ be a set of candidates of query Q in an XML database T. The search space of candidate $S \in \mathscr{C}(Q, T)$ can be defined as the set of candidates in $\mathscr{C}(Q, T)$ on which pairwise comparisons need to be performed to calculate the score of candidate S.

From the naïve algorithm, we can see that the number of candidates in the search space to calculate the score of one candidate is $(n - 1)$. Now, we delve into some useful properties of dominance relationships in order to (i) reduce the number of pairwise comparisons between candidates which need to be performed to calculate a dominated (*reps.* dominating) score of a candidate and (ii) design a stopping condition so that the algorithm can stop earlier without exhaustively scanning all candidates to retrieve the top-k answers.

Property 1 Let $F()$ be a function of any candidate S, $F(S) = \sum_{i=1}^{d} D_S[i]$, where $d = |D_S|$. If $S' \succ S$, then $F(S') > F(S)$.

Proof We have $S' \succ S$, thus (i) $\forall i (1 \leq i \leq d) D_S[i] \leq D_{S'}[i]$, and (ii) $\exists j (1 \leq j \leq d) D_S[j] < D_{S'}[j]$ (from Definition 6). From (i) and (ii), we have $\sum_{i=1}^{d} D'_S[i] > \sum_{i=1}^{d} D_S[i]$, which in turn gives us $F(S') > F(S)$ □

This property indicates that we can reduce the search space for computing dominated (or dominating) scores by using the sorted list L in descending order of

Table 3 A sample set of 2-dimension candidates.

	D_1	D_2
S_1	0.95	0.9
S_2	0.15	0.5
S_3	0.1	0.95
S_4	0.5	0.4
S_5	0.8	0.8
S_6	0.9	0.4
S_7	0.4	0.4
S_8	0.3	0.2
S_9	0.7	0.6
S_{10}	0.3	0.3

$F()$ values, as shown in Figure 3. That is, search space for $score_{dd}(S_i)$ can be limited to the candidates in the sublist L_B (i.e., a set of candidates which have smaller $F()$ values than of the value of $F(S_i)$). Similarly, the search space for computing $score_{dg}(S_i)$ can be limited to the candidates in L_A (i.e., a set of candidates which have smaller $F()$ values than the $F(S_i)$ value in the list L).

For example, given a list of candidates which are sorted in descending order of their $F()$ values as shown in Figure 4, the number of dominance tests to calculate the dominated scores of candidates $S_i (1 \leq S_i \leq 10)$ is given in Figure 5. We can see that the total number of dominance checks being performed to calculate the dominated sores of all the candidates is $0 + 7 + 4 + 5 + 1 + 2 + 6 + 7 + 3 + 9 = 44$. Thus, we can save up to $90 - 44 = 46$ dominance checks to calculate the dominated scores of the candidates by using Property 1.

Property 2 Let $M()$ be a function of any candidate S, $M(S) = \max_{i=1}^{d}\{D_S[i]\}$, where $d = |D_S|$. If $\min_{i=1}^{d}\{D_{S'}[i]\} > M(S)$ for two candidates S and S', then $S' \succ S$ as well as all candidates with $M()$ values smaller than $M()$ value of S.

Proof We have $\min_{i=1}^{d}\{D_{S'}[i]\} > \max_{i=1}^{d}\{D_S[i]\}$, thus $\forall i \in [1..d]$, $D'_S[i] > D_S[i]$ which gives us $S' \succ S$. □

Property 2 in combination with Lemmas 3 and 2 provide a termination condition. Assuming S_k is the current k-th candidate and S is the next candidate being processed and we select $M(S) = \max_{i=1}^{d}\{D_S[i]\}$, if $\min_{i=1}^{d}\{D_{S_k}[i]\} \geq M(S)\}$, then the algorithm can be safely terminated.

For example, given a set of candidates shown in Figure 7, we assume that candidate S_5 is the current k-th candidate and S_9 is the next candidate being processed. We have $\min_{i=1}^{2}\{D_{S_5}[i]\} = 0.8$, while $\max_{i=1}^{2}\{D_{S_9}[i]\} = 0.7$ (Figure 6). Thus, the algorithm can be safely terminated without the need to consider the other candidates $S_2, S_4, S_7, S_8, S_{10}$.

6.1 Top-k dominated algorithm (TKDD)

The goal of TKDD is to efficiently find, for each candidate, the number of other candidates which dominate it, avoiding exhaustive pairwise comparisons between the candidates. After k results have been retrieved, we use the score of the k-th result as a maximum threshold. The candidates whose dominated scores exceed the threshold are pruned. In addition, it guarantees the safe termination of the algorithm if the scores of all remaining candidates exceed the threshold.

More specifically, the TKDD proceeds in the following four steps: (i) *Initialization (line 1)*: the result set $\mathscr{R}$ and `minValue` are initialized; (ii) *Termination condition*

$$L = \underbrace{\{S_1, \ldots, S_{i-1}\}}_{L_B} S_i \underbrace{\{S_{i+1}, \ldots, S_n\}}_{L_A}$$

Figure 3 Search spaces L_A and L_B to calculate $score_{dd}(S_i)$ and $score_{dg}(S_i)$ respectively from a list L of candidates sorted in descending order of $F()$ values.

Candidate	S_1	S_5	S_6	S_9	S_3	S_4	S_7	S_2	S_8	S_{10}
$F() = \sum_{i=1}^{2} D_i$	1.85	1.6	1.3	1.3	1.05	0.9	0.8	0.65	0.5	0.5

Figure 4 A list of candidates sorted in descending order of $F()$ values.

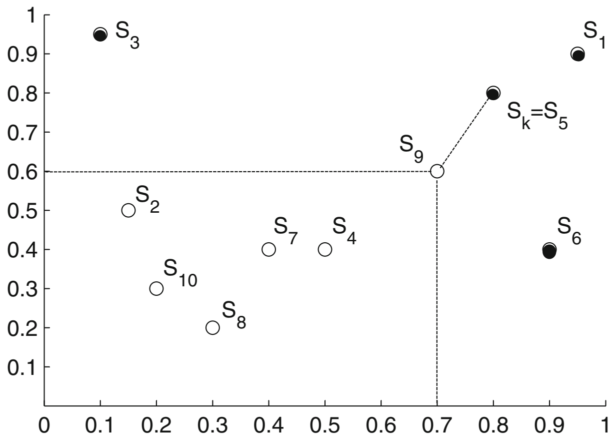
Candidate	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
No. of dominance checks	0	7	4	5	1	2	6	7	3	9

Figure 5 The number of dominance checks to calculate the dominated scores of the candidates.

Candidate	S_1	S_3	S_6	S_5	S_9	S_4	S_2	S_7	S_8	S_{10}
$M() = \max_{i=1}^{2}\{D_i\}$	0.95	0.95	0.9	0.8	0.7	0.5	0.5	0.4	0.3	0.3

Figure 6 A list of candidates sorted in descending order of their $M()$ values.

Figure 7 Example of a stopping condition.



Algorithm 2 TKDD

```
input : $L_{max}$: a list of candidates sorted in desc order of M(); $L$: a list of candidates sorted in
        desc order of F(); the number $k$ of answers to return
output: The top-$k$ answers w.r.t $\mathsf{score}_{dd}$ in a sorted set $\mathscr{R}$
1  $\mathscr{R} \leftarrow \{\}$;
2  minVal $\leftarrow (-1)$;
3  foreach candidate $S \in L_{max}$ do
4      if $|\mathscr{R}| = k$ then
5          if M($S$) $\leq$ minVal then return $\mathscr{R}$;
6      end
7      $\mathsf{score}_{dd}(S) \leftarrow 0$;
8      foreach candidate $S'$ in the search space $L_B \subset L$ of $S$ do
9          if $S' \succ S$ then $\mathsf{score}_{dd}(S) \leftarrow \mathsf{score}_{dd}(S) + 1$;
10     end
11     if $|\mathscr{R}| = k$ then
12         $S_k \leftarrow k$-th candidate in $\mathscr{R}$;
13         if $\mathsf{score}_{dd}(S) < \mathsf{score}_{dd}(S_k)$ then
14             remove the tail result from $\mathscr{R}$;
15             $\mathscr{R} \leftarrow$ insert $S$ ordered by $\mathsf{score}_{dd}$ asc.;
16         end
17     else $\mathscr{R} \leftarrow$ insert $S$ ordered by $\mathsf{score}_{dd}$ asc.;
18     if $|\mathscr{R}| = k$ then
19         $S_k \leftarrow k$-th candidate in $\mathscr{R}$;
20         minVal $\leftarrow \min_{i=1}^{|D_{S_k}|}\{D_{S_k}[i]\}$;
21     end
22 end
23 return $\mathscr{R}$;
```

(lines 4–6): If the M() value of the current candidate S does not exceed the minimum value of the current k-th candidate in $\mathscr{R}$, the result set $\mathscr{R}$ is returned and the algorithm terminates; (iii) *Dominance checks (lines 7–10)*: the pairwise dominance check between S and every other candidate S' in the search space of S takes place. The dominated score of S is increased by 1 every time it is dominated by another candidate; (iv) *Result updates (lines 11–17)*: if k results exist and the dominated score of the k-th candidate is larger than the current candidate's score, the k-th candidate is removed and the current candidate is inserted into $\mathscr{R}$; otherwise if less than k results exist in $\mathscr{R}$, it inserts the current candidate into $\mathscr{R}$. Finally, if the size of $\mathscr{R}$ is k, the threshold *minValue* is updated *(lines 18–21)*.

6.2 Top-k dominating algorithm (TKDG)

The aim of the TKDG algorithm is to retrieve the top-k results that dominate the larger number of other candidates. This is a more challenging task compared to that of TKDD, for the following reason. Let pos(S) be the position of the currently examined candidate S in the list L of candidates sorted in descending order of F() values. To calculate $\mathtt{score}_{dd}(S)$, TKDD may perform a number of pos(S) dominance tests on those candidates whose F() values $\leq$ F(S). However, to calculate $\mathtt{score}_{dg}(S)$, TKDG may perform $(|L| -$ pos $(S))$ tests on those candidates whose F() values $\geq$ F (S). As the top relevant results are usually located in the top of L (i.e., pos(S) $\ll |L|$), it implies that pos$S \ll (|L| -$ pos$(S))$. Thus, the search space of the TKDG algorithm is significantly larger than that of TKDD.

Algorithm 3 TKDG

```
input : A list $L$ of candidates sorted in desc. order of F(); the number $k$ of answers to return
output: The top-$k$ answers w.r.t $\mathsf{score}_{dg}$ in a sorted set $\mathscr{R}$
1  $\mathscr{R} \leftarrow \{\}$;
2  foreach candidate $S \in L$ do
3      if $|\mathscr{R}| = k$ then
4          $S_k \leftarrow k$-th candidate in $\mathscr{R}$;
5          if $|\mathscr{C}| - \texttt{pos}(S) \leq \mathsf{score}_{dg}(S_k)$ then
6              return $\mathscr{R}$;
7          end
8      end
9      $\mathsf{score}_{dg}(S) \leftarrow 0$;
10     foreach candidate $S'$ in the search space $L_A$ of $S$ do
11         if $S' \prec S$ then
12             $\mathsf{score}_{dg}(S) \leftarrow \mathsf{score}_{dg}(S) + 1$;
13         end
14     end
15     if $|\mathscr{R}| = k$ then
16         $S_k \leftarrow k$-th candidate in $\mathscr{R}$;
17         if $\mathsf{score}_{dg}(S) > \mathsf{score}_{dg}(S_k)$ then
18             remove the tail result from $\mathscr{R}$;
19             $\mathscr{R} \leftarrow$ insert $S$ ordered by $\mathsf{score}_{dg}$ desc.;
20         end
21     else $\mathscr{R} \leftarrow$ insert $S$ ordered by $\mathsf{score}_{dg}$ desc.;
22 end
23 return $\mathscr{R}$;
```

Let S' be a currently examined candidate. S' can dominate at most ($|L|-$ $\texttt{pos}(S)$) candidates. Thus, if ($|L|-$ $\texttt{pos}(S)$) $\leq$ $\texttt{score}_{dg}(S_k)$, where S_k is the k-th result in $\mathscr{R}$, the algorithm terminates.

Specifically, the TKDG algorithm (Algorithm 3) proceeds in the following steps: (i) *Initialization (line 1):* the result set is initialized; (ii) *Stopping condition (lines 3–8):* if there are already k candidates in the result set and the dominating score of the k-th candidate is equal or greater than $(|L| - pos(S))$, the algorithm terminates; (iii) *Dominance tests (lines 9–14):* the pairwise dominance test between the currently examined candidate and each other candidate in the search space will be performed. Concurrently, the dominating score of the candidate is calculated; (iv) *Updating the result set (lines 15–23):* if there are already k results in the result set, the k-th candidate from the result set is replaced by the new candidate if its dominating score is less than the current candidate's score. Otherwise, if less than k results exist in $\mathscr{R}$, it inserts the new candidate into $\mathscr{R}$.

6.3 Top-k dominance algorithm (TKD)

A naive approach can be processed in two steps: (i) calculating the dominating and dominated scores of each candidate and (ii) then applying the Threshold Algorithm (TA) [11] over these two ranking dimensions to select the top-k candidates. However, this approach may be not efficient because it needs to calculate the dominating and dominated scores for all candidates. Let $\mathscr{R}_{dg}$ and $\mathscr{R}_{dd}$ be the lists of candidates which are sorted in the descending order of dominating and dominated scores, respectively. Let U_i and V_i be two candidates at position i in $\mathscr{R}_{dg}$ and $\mathscr{R}_{dd}$

Figure 8 An illustration of the stopping point for the TA algorithm.

Top-k	**TKDG**	**TKDD**
1	U_1	V_1
2	U_2	V_2
…	…	…
i	$\mathbf{U_i}$	$\mathbf{V_i}$
…	…	…
n	U_n	V_n

respectively. Assume S_k is the current top-k result, it has been proved [11] that the TA algorithm can be safely terminated if,

$$\texttt{score}_d(S_k) \geq \alpha\texttt{score}_{dg}(U_i) - (1-\alpha)\texttt{score}_{dd}(V_i)$$

Thus, rather than calculating the dominating and dominated scores of all n candidates in advance, we can progressively calculate these scores of candidates until we find the safe stopping point for the TA algorithm, as demonstrated in Figure 8.

7 Experimental evaluation

We have designed and performed a set of experiments to evaluate the search performance of our approach. In this section, we analyze the results of our experiments to compare the search quality and efficiency of our approach and some existing approaches.

7.1 Experimental setup

The experiments were performed on a Pentium 4 3.2GHz computer running Windows XP Professional, with 2GB of main memory. All approaches were implemented in Java. We used Oracle Berkeley DB 11*g* [29] as a tool for storing and managing the data indexes used in the experiments.

Data sets We tested three XML data sets: DBLP Computer Science Bibliography (877 MB) [10], Mondial (1 MB) [27], and Auction (36.7 MB) [1]. DBLP Computer Science Bibliography is a list of bibliographic information on major computer science journals and proceedings. Mondial is a world geographic database integrated from the CIA World Factbook, the International Atlas, and the TERRA database among other sources. Auction is a synthetic benchmark data set generated by the XML Generator from XMark using the default DTD.

Query sets We asked a group of students to submit 50 different keyword queries to search on each data set. Each query contains a set of search keywords and a short description which indicates the user's search intension corresponding to the query. The required short description of each query is necessary to identify the user's intension of the query. We observed that even searching on a specific domain (i.e., on each of the three specific data sets in our case), keyword queries are sometimes ambiguous in terms of expressing the user's search intention. As a result, it is sometimes hard to identify the relevant results of these queries, which is a prerequisite for us to evaluate the performance of our approach and other approaches.

7.2 Search quality

We compared the search quality of our proposed DLCA approach with other existing approaches, including ELCA [12], SLCA [37], XSEarch [8], CVLCA [19], MLCA [22] and XReal [23]. The quality was measured in three popular metrics in information retrieval [2]: *precision (P), recall (R) and F-measure*.

To compute precision and recall, we manually reformulated the keyword queries into schema-aware XQuery queries, based on the schemas of data sets and the descriptions of the keyword queries. We took the results of these corresponding transformed queries as a baseline, then computed the precision and recall of the given queries according to the baseline as follows. Given a keyword query Q and its corresponding transformed XQuery X_Q, the accurate result set of Q, i.e., the result of X_Q, is denoted as *relevant results*, and the approximate result set, i.e., the result of a specified algorithm on Q, is denoted as *retrieved results*. Accordingly, we can define the precision and recall of this algorithm as follows.

The *precision* is a fraction of retrieved results that are relevant to the search:

$$P = \frac{|\{\texttt{relevant results}\} \cap \{\texttt{retrieved results}\}|}{|\{\texttt{retrieved results}\}|}$$

The *recall* is a fraction of the relevant results that are successfully retrieved by the search system:

$$R = \frac{|\{\texttt{relevant results}\} \cap \{\texttt{retrieved results}\}|}{|\{\texttt{relevant results}\}|}$$

The *F-measure* shows the trade-off between the precision and recall and is computed as:

$$F\text{-}measure = \frac{(1+\beta^2)PR}{\beta^2 P + R}$$

where $\beta = 1$ weights precision and recall equally; $\beta < 1$ emphasizes precision, while $\beta > 1$ focuses on recall.

We can see that the relevant results of each keyword query need to be obtained before we can calculate the above evaluation metrics. To obtain these relevant results of the tested queries, we manually formed the corresponding schema-aware XQuery [40] queries of the keyword queries, with the help of the users' described search intention accompanied by the queries. The relevant results of each query were then used as the ground truth for evaluating the performance of our approach and other existing approaches.

We conducted experiments with the same set of 50 keyword queries by employing different approaches and we measured the precision and recall of each approach as the average of precision and recall values of all the tested queries.

The comparisons of precision and recall of our approach with other approaches in the three different data sets are shown in Tables 4, 5 and 6.

Table 4 Precision and recall of queries on DBLP data.

	ELCA	SLCA	XSEarch	CVLCA	MLCA	XReal	DLCA
Precision	0.523	0.733	0.640	0.688	0.720	0.733	0.934
Recall	1	0.647	0.941	0.911	0.923	0.647	0.941

Precision We can see that our approach outperforms all the other approaches in terms of precision on all the tested data sets. More importantly, the precision of DLCA is over 90% in all the data sets, which indicates that the results returned by our approach are highly relevant to the users' queries. Specifically, Table 4 illustrates that DLCA returns results with 93% of precision, which is over 20% higher than XReal when tested on DBLP data set. In the Mondial and Auction data sets, our approach has precision of 95% and 90% respectively (as shown in Tables 5 and 6).

Recall The results from Tables 4–6 indicate that DLCA can achieve high recall from 93% to 94% in all three tested data sets. The ELCA approach has very high recall in all data sets, however it has very low precision. This is because as discussed in Section 2, ELCA employs a similar semantics with the baseline approach which returns almost all candidate results without evaluating the relationship between query keywords. Other three approaches XSEarch, CVLCA and MLCA have similar recall values to our approach, but these approaches achieve much lower precision.

F-measure The overall performance of DLCA and the other approaches are measured by the F-measure metric which is a trade-off between the precision and recall metrics. In our experiments, we measured the F-measure with $\beta = 0.5$, 1.0 and 2.0. From Figures 9, 10 and 11, we can see that the overall performance of DLCA outperforms the other approaches in all tested data sets.

7.3 Quality of top-k answers

To evaluate the effectiveness of our ranking criteria, we employ the following standard evaluation metrics from the field of information retrieval [2].

– Mean Average Precision (MAP): for a single information need, Average Precision is the average of the precision value obtained for the set of top documents existing after each relevant document is retrieved, and this value is then averaged over information needs. That is, if the set of relevant documents for an information need $Q \in \mathcal{Q}$ is $d_1, \ldots, d_{mj}$ and R_{jk} is the set of ranked retrieval results from the top result until you get to document d_k, then

$$\mathtt{MAP}(\mathcal{Q}) = \frac{1}{|\mathcal{Q}|} \sum_{j=1}^{|\mathcal{Q}|} \frac{1}{m_j} \sum_{k=1}^{m_j} \mathtt{Precision}(R_{jk})$$

Table 5 Precision and recall of queries on Mondial data.

	ELCA	SLCA	XSEarch	CVLCA	MLCA	XReal	DLCA
Precision	0.503	0.712	0.635	0.670	0.712	0.721	0.922
Recall	1	0.624	0.943	0.910	0.903	0.647	0.939

Table 6 Precision and recall of queries on Auction data.

	ELCA	SLCA	XSEarch	CVLCA	MLCA	XReal	DLCA
Precision	0.478	0.706	0.623	0.64	0.699	0.703	0.901
Recall	1	0.650	0.931	0.920	0.907	0.650	0.931

– R-Precision (R-prec): R-Precision is the precision after R documents have been retrieved, where R is the number of relevant documents for the query.
– bpref: bpref computes a preference relation of whether the judged relevant documents are retrieved ahead of the judged irrelevant documents. Thus, it is based on the relative ranks of judged documents only. The bpref measure is defined as:

$$bpref = \frac{1}{R}\sum_{r}\left(1 - \frac{|n \text{ ranked higher than } r|}{min(R, N)}\right)$$

where R is the number of judged relevant documents, N is the number of judged irrelevant documents, r is a relevant retrieved document, and n is a member of the first R irrelevant retrieved documents.
– Reciprocal Rank (R-rank): measures (the inverse of) the rank of the top relevant document.
– Precision at N (P@N): measures the precision after N documents have been retrieved.

We compared the performance of our three ranking algorithms TKDD, TKDG and TKD corresponding to our three proposed ranking criteria. Regarding the TKD algorithm, we studied the effect of parameter α (see Section 5) considering three variations, denoted TKD-α, for $\alpha = 0.5$, 1.0 and 2.0. We evaluated a set of 20 randomly selected queries with a various number of keywords in each of the three data sets described in Section 7.1. In order to obtain ranked lists of relevant results of these queries, we manually reformulated the keyword queries into schema-aware

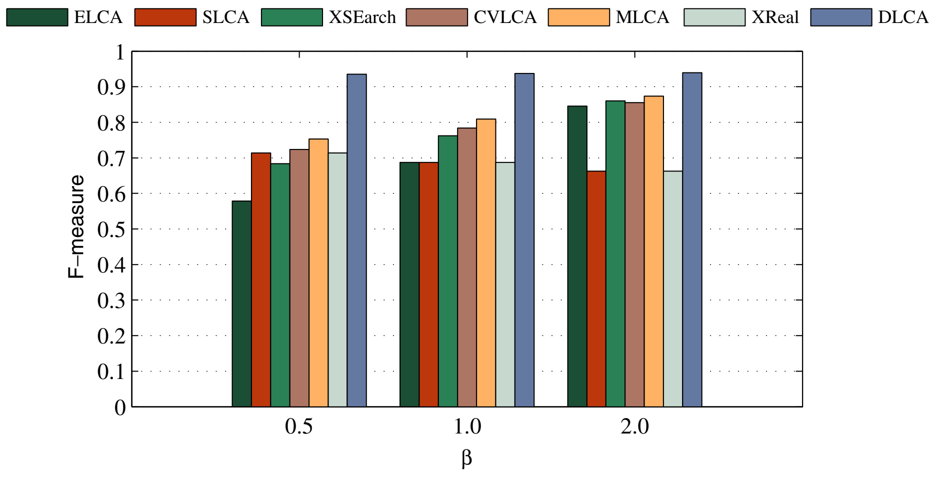


Figure 9 Overall result quality on DBLP data.

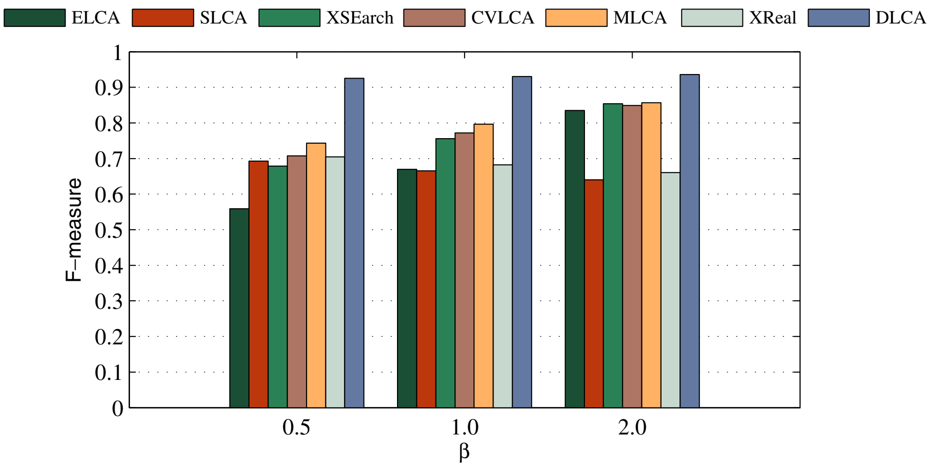


Figure 10 Overall result quality on Mondial data.

XQuery queries basing on the schemas of data sets. We took the results of each transformed query as its relevant results, then these results are ranked using user studies. The ranked lists of the relevant results are used as ground truths for evaluating the ranking quality of the approaches.

The average values of the evaluation metrics are recorded in Table 7. The experimental results in Table 7 show that TKDG has better performance than TKDD in most of the evaluation metrics. We observed that some candidates, even though dominated by a small number of other candidates, were not evaluated as relevant results in the returned results. The performance of TKD is dependent on the selected values of α parameter. In our experiments, we found that TKD-0.25 outperforms the other variations in most of evaluation metrics.

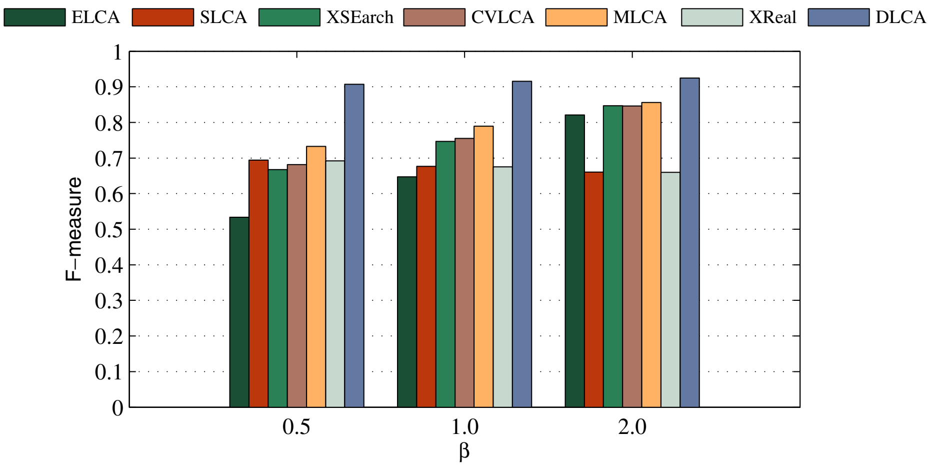


Figure 11 Overall result quality on Auction data.

Table 7 Comparisons on ranking effectiveness of the algorithms.

	MAP	R-prec	bpref	R-rank	P@1	P@5	P@10
TKDD	0.87	0.83	0.75	0.86	0.89	0.87	0.81
TKDG	0.85	0.82	0.79	0.87	0.92	0.89	0.84
TKD-0.25	0.84	0.80	0.79	0.86	0.91	0.91	0.86
TKD-0.5	0.86	0.82	0.76	0.86	0.90	0.87	0.82
TKD-0.75	0.87	0.85	0.73	0.88	0.88	0.85	0.80
XRank	0.67	0.75	0.61	0.71	0.69	0.68	0.65
XSEarch	0.70	0.77	0.63	0.68	0.73	0.68	0.66

More importantly, all our ranking algorithms can identify the top-10 results at a precision from 80% to 85%. The mean average precision of these algorithms is approximately 85% and we can even achieve higher precision if we select a suitable α value which maximizes the balance between the two dominated and dominating scores. Proposing a method which can automatically find a suitable value of α parameter to maximize the precision of TKD is interesting and is proposed as our future work.

7.4 Query processing time for identifying DLCA answers

We compared the query processing time of our approach with the existing approaches using a different number of keywords. The number of keywords ranged from two to ten keywords. For each query length, we selected 10 different queries with various keyword frequencies. The query processing time corresponding to each query length is measured as the average processing time of the queries with that length. Figure 12 illustrates the processing time of our approach in comparison to the other approaches. We can see that the processing time of the DLCA approach is higher than that of the other approaches. This is because the DLCA approach performs the skyline semantics over the query candidates to identify relevant results. This causes a higher processing time of the DLCA semantics, compared with the heuristics-based semantics of the other approaches. In contrast, we can see from the previous section that our approach outperforms the other approaches in terms of the result quality.

7.5 Efficiency of Top-*k* algorithms

We evaluate the cost for returning top-*k* results of our three ranking functions. We tested the effects of two parameters: the number of candidates being processed and the number *k* of returned results.

7.5.1 Effectiveness of stopping conditions

The aim of this experiment is to evaluate the effectiveness of the stopping conditions on the reduction in the number of pairwise comparisons between candidates that need to be performed for calculating a dominated (or dominating) score of a candidate. We compared our top-*k* algorithms with the "improved" naive algorithms which are the variants of the top-*k* algorithms but without using the stopping conditions. We tested ten queries with various keyword lengths in each data set. For

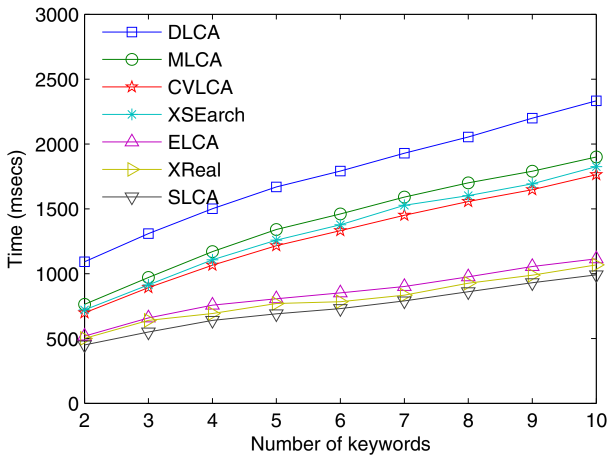


Figure 12 Processing time for retrieving results of the approaches vs. different number of keywords.

each query, we selected a set of 5,000 candidates. For those queries which have less than that number of candidate results, we repeatedly made a replica of the candidates until we obtained at least the required number of candidates. Then, we randomly selected 5,000 candidates from the duplicated set. The reduction in the number of pairwise tests is measured as the average reduction of all the tested queries.

Figure 13 shows the effectiveness of the stopping conditions on the reduction in the pairwise tests that need to be performed for calculating the score of a candidate, compared with the "improved" naive algorithm. The result indicates that the TKDD has the highest reduction in the pairwise comparisons. The main reason is because the top relevant results mostly locate at the top of the $F()$ list, which results in a low number of comparison for computing the dominated score of a candidate. In contrast, this can cause a higher number of pairwise tests for computing the corresponding dominating score of a candidate. Consequently, the TKDG has a lower reduction.

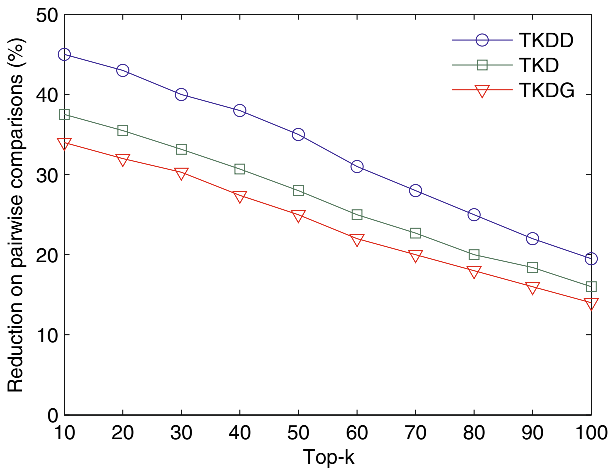
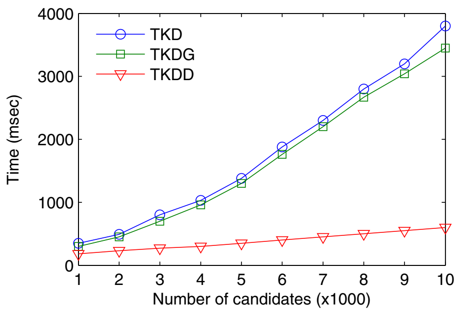
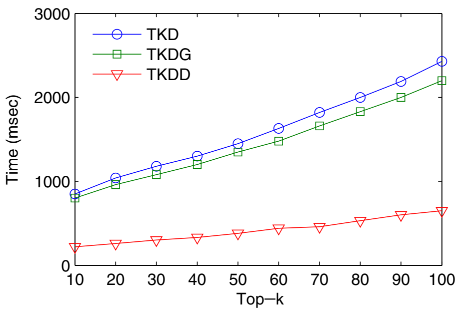


Figure 13 Effectiveness of the stopping conditions on the reduction in the number of pairwise comparisons.



(a) Effect of number of candidates



(b) Effect of number of selected results

Figure 14 Ranking efficiency of TKDD, TKDG and TKD algorithms.

This reduction of the TKD is approximately the average on the reduction of the TKDD and TKDG algorithms.

7.5.2 Efficiency and scalability of top-k algorithms

We tested 10 queries with various keyword lengths in each data set. In default scenarios, we tested on the set of 5,000 candidates and the number of results returned was 30. For those queries which have less than the required number of candidate results, we repeatedly made a replica of their candidates until we obtained at least the required number of candidates. Then, we randomly selected the required number of candidates from the duplicated set. The computational costs of our algorithms are shown in Figure 14a and b. From Figure 14a we can see that when the number of candidates increases, the processing time of all algorithms increases, but at different trends. Specifically, TKDD is the most efficient method and it is less effected by the increase of the number of candidates. As already discussed, TKDD is interested in results that are dominated by as few other candidates as possible and these results are usually located at the top of the sorted list of candidates; hence, it searches a relatively small portion of the candidate list. However, the search space for TKDG is much larger, so its delay is expected. Similarly, the lower performance of TKD is mainly due to the impact of dominating score; therefore, it is reasonable that its processing time rises at the similar trend as TKDG's, with a small additional overhead for calculating dominated score.

The result from Figure 14b shows that the processing time of TKDD is very slightly affected by the increase of the number k of results being returned and it can return 10 to 100 results from a set of 5,000 candidates in less than 1 s. The processing time of TKDG algorithm is more strongly affected by the change of this parameter, however it takes less than 2.5 s to return the top-100 results from a set of 5,000 candidates.

8 Conclusions

In this article, we have studied the problem of identifying the most relevant results and the top-k relevant results for XML keyword queries. More specifically, we have addressed the three crucial requirements for effective XML keyword searches. We have introduced a new method for evaluating the relationship between the query keywords in a candidate using the mutual information concept and propose a new DLCA semantics of keyword queries; we have proposed an approach for selecting DLCA results from numerous candidates and three ranking criteria for selecting the top-k relevant results based on the semantics of skyline queries. Some proven properties have been obtained to accelerate our proposed algorithms. The experiments were conducted to evaluate our approach and the experimental results show that our approach outperforms the existing approaches in the tested data sets and evaluation metrics.

References

1. Auction. http://monetdb.cwi.nl/xmark/auctions.xml
2. Baeza-Yates, R.A., Ribeiro-Neto, B.: Modern Information Retrieval. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1999)
3. Bartolini, I., Ciaccia, P., Patella, M.: Efficient sort-based skyline evaluation. ACM Trans. Database Syst. **33**(4), 1–49 (2008). doi:10.1145/1412331.1412343
4. Bender, M.A., Farach-Colton, M., Pemmasani, G., Skiena, S., Sumazin, P.: Lowest common ancestors in trees and directed acyclic graphs. J. Algorithms **57**, 75–94 (2005)
5. Börzsönyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: Proceedings of the 17th International Conference on Data Engineering, pp. 421–430. IEEE Computer Society, Washington, DC, USA (2001)
6. Chomicki, J., Godfrey, P., Gryz, J., Liang, D.: Skyline with presorting. In: Proceedings of 19th International Conference on Data Engineering, vol. 717 (2003)
7. Cohen, S., Kanza, Y., Kimelfeld, B., Sagiv, Y.: Interconnection semantics for keyword search in XML. In: CIKM '05: Proceedings of the 14th ACM International Conference on Information and Knowledge Management, pp. 389–396. ACM, New York, NY, USA (2005)
8. Cohen, S., Mamou, J., Kanza, Y., Sagiv, Y.: XSEarch: a semantic search engine for XML. In: Proceedings of the 29th International Conference on Very Large Data Bases, pp. 45–56. VLDB Endowment (2003)
9. Cover, T.M., Thomas, J.A.: Elements of Information Theory. Wiley-Interscience, New York, NY, USA (1991)
10. DBLP. dblp.uni-trier.de/xml/dblp.xml
11. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. In: Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 102–113. ACM, New York, NY, USA (2001)
12. Guo, L., Shao, F., Botev, C., Shanmugasundaram, J.: XRANK: ranked keyword search over XML documents. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, pp. 16–27. ACM, New York, NY, USA (2003)
13. Han, S.K., Shin, D., Jung, J.Y., Park, J.: Exploring the relationship between keywords and feed elements in blog post search. World Wide Web **12**(4), 381–398 (2009)
14. Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestors. SIAM J. Comput. **13**(2), 338–355 (1984)
15. Hristidis, V., Koudas, N., Papakonstantinou, Y., Srivastava, D.: Keyword proximity search in XML trees. IEEE Trans. Knowl. Data Eng. **18**, 525–539 (2006)
16. Kossmann, D., Ramsak, F., Rost, S.: Shooting stars in the sky: an online algorithm for skyline queries. In: Proceedings of the 28th International Conference on Very Large Data Bases, pp. 275–286. VLDB Endowment (2002)
17. Kung, H.T., Luccio, F., Preparata, F.P.: On finding the maxima of a set of vectors. Journal of the ACM, pp. 469–476. ACM, New York, NY, USA (1975)
18. Lee, K.H., Whang, K.Y., Han, W.S.: Xmin: Minimizing tree pattern queries with minimality guarantee. World Wide Web **13**(3), 343–371 (2010)
19. Li, G., Feng, J., Wang, J., Zhou, L.: Effective keyword search for valuable LCAs over XML documents. In: Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management, pp. 31–40. ACM, New York, NY, USA (2007)

20. Li, G., Ooi, B.C., Feng, J., Wang, J., Zhou, L.: Ease: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, pp. 903–914. ACM, New York, NY, USA (2008)
21. Li, L., Otsuka, S., Kitsuregawa, M.: Finding related search engine queries by web community based query enrichment. World Wide Web **13**(1), 121–142 (2010)
22. Li, Y., Yu, C., Jagadish, H.V.: Enabling schema-free XQuery with meaningful query focus. VLDB J. **17**(3), 355–377 (2008)
23. Liu, Z., Chen, Y.: Identifying meaningful return information for xml keyword search. In: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, pp. 329–340. ACM, New York, NY, USA (2007)
24. Liu, Z., Chen, Y.: Reasoning and identifying relevant matches for XML keyword search. In: Proceedings of the 34th International Conference on Very Large Data Bases, pp. 921–932 (2008)
25. Liu, Z., Chen, Y.: Processing keyword search on XML: a survey. World Wide Web **14**, 671–707 (2011)
26. Liu, Z., Walker, J., Chen, Y.: XSeek: a semantic XML search engine using keywords. In: Proceedings of the 33rd International Conference on Very Large Data Bases, pp. 1330–1333. VLDB Endowment (2007)
27. Mondial. http://www.cs.washington.edu/research/xmldatasets/data/mondial/mondial-3.0.xml
28. Online computer library center. Introduction to the Dewey Decimal Classification. http://www.oclc.org/oclc/fp/about/about the ddc.htm
29. Oracle Berkeley DB. http://www.oracle.com/technology/products/berkeley-db/index.html.
30. Schieber, B., Vishkin, U.: On finding lowest common ancestors: simplification and parallelization. SIAM J. Comput. **17**(6), 1253–1262 (1988)
31. Shao, F., Guo, L., Botev, C., Bhaskar, A., Chettiar, M., Yang, F., Shanmugasundaram, J.: Efficient keyword search over virtual xml views. In: VLDB '07: Proceedings of the 33rd International Conference on Very Large Data Bases, pp. 1057–1068. VLDB Endowment (2007)
32. Sun, C., Chan, C.Y., Goenka, A.K.: Multiway slca-based keyword search in xml data. In: Proceedings of the 16th International Conference on World Wide Web, pp. 1043–1052. ACM, New York, NY, USA (2007)
33. Tan, K.L., Eng, P.K., Ooi, B.C.: Efficient progressive skyline computation. In: VLDB '01: Proceedings of the 28th International Conference on Very Large Data Bases, pp. 301–310 (2001)
34. Wang, G., Ning, B., Yu, G.: Holistically stream-based processing xtwig queries. World Wide Web **11**(4), 407–425 (2008)
35. Wang, J., Yu, J.X., Liu, C.: Independence of containing patterns property and its application in tree pattern query rewriting using views. World Wide Web **12**(1), 87–105 (2009)
36. Wu, X., Theodoratos, D., Souldatos, S., Dalamagas, T., Sellis, T.: Evaluation techniques for generalized path pattern queries on xml data. World Wide Web **13**(4), 441–474 (2010)
37. Xu, Y., Papakonstantinou, Y.: Efficient keyword search for smallest lcas in xml databases. In: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, pp. 527–538. ACM, New York, NY, USA (2005)
38. Xu, Y., Papakonstantinou, Y.: Efficient lca based keyword search in xml data. In: Proceedings of the 11th International Conference on Extending Database Technology, pp. 535–546. ACM, New York, NY, USA (2008)
39. XML Path Language (XPath). http://www.w3.org/tr/xpath/
40. XQuery: An XML Query Language. http://www.w3.org/tr/xquery/
41. Zhou, R., Liu, C., Li, J.: Fast elca computation for keyword queries on xml data. In: Proceedings of the 13th International Conference on Extending Database Technoloy, pp. 549–560. ACM, New York, NY, USA (2010)