LCA-based Selection for XML Document Collections

Georgia Koloniari
Department of Computer Science
University of Ioannina, Greece
kgeorgia@cs.uoi.gr

Evaggelia Pitoura
Department of Computer Science
University of Ioannina, Greece
pitoura@cs.uoi.gr

ABSTRACT

In this paper, we address the problem of database selection for XML document collections, that is, given a set of collections and a user query, how to rank the collections based on their goodness to the query. Goodness is determined by the relevance of the documents in the collection to the query. We consider keyword queries and support Lowest Common Ancestor (LCA) semantics for defining query results, where the relevance of each document to a query is determined by properties of the LCA of those nodes in the XML document that contain the query keywords. To avoid evaluating queries against each document in a collection, we propose maintaining in a preprocessing phase, information about the LCAs of all pairs of keywords in a document and use it to approximate the properties of the LCA-based results of a query. To improve storage and processing efficiency, we use appropriate summaries of the LCA information based on Bloom filters. We address both a boolean and a weighted version of the database selection problem. Our experimental results show that our approach incurs low errors in the estimation of the goodness of a collection and provides rankings that are very close to the actual ones.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms

Algorithms, Experimentation, Performance

1. INTRODUCTION

As the number of available data sources increases, the problem of database selection, that is, determining which from a set of available databases are the most relevant to a given user query, is attracting considerable attention [9, 4, 5, 26, 20, 22]. The relevance or goodness of a database to a query is derived from the relevance of the data (or documents) in the database to the query. Most current research has considered the selection of text [9, 4] or relational [26, 20, 22] databases. Since XML is the de facto standard for data representation and exchange on the web and a large amount of XML document collections are available, in this paper, we address the problem of database selection for XML document collections.

Keyword queries offer a popular, easy and intuitive way of searching. The main difference between keyword search for XML and text documents is that, for XML documents, the relevance of a document to a query also depends on the relative position of the query keywords in the XML tree. We support keyword queries and rank their results by adopting *lowest common ancestor* (LCA) query semantics [10, 6, 11, 14, 13, 24, 15, 21, 25, 17] based on which the result of a keyword query is defined by the LCA node of the keywords. We then determine the relevance (similarity) of a document to a query based on the LCA semantics and derive the goodness of a collection of documents by aggregating over the similarities of each of its documents to the query.

To avoid the processing overhead entailed in evaluating a query against each document, we propose an approximate approach that estimates the similarity of a document to a query by exploiting appropriate information about the lowest common ancestors of all pairs of keywords that appear in a document. That is, it estimates the LCA of the query result based on the LCA of each pair of keywords in the query. In addition, we present a novel data structure based on Bloom filters to efficiently summarize this LCA pairwise information. We study both a boolean and a weighted selection problem. Our experimental results both on real and synthetic datasets show that our approximations are sufficiently accurate, while requiring significantly less space and processing overheads.

The rest of the paper is organized as follows. In Section 2, we define the lowest common ancestor semantics, and the selection problem. Section 3 describes our pairwise approximation approach, while Section 4 introduces the Bloom filter based structures. Section 5 includes our experimental evaluation and Section 6 presents related work. We conclude in Section 7.

2. PROBLEM DEFINITION

We first describe the keyword query model and the lowest common ancestor semantics deployed for query evaluation. Then, we formally define the problem of database selection over XML document collections by defining similarity and goodness measures under both a boolean and a weighted model.

2.1 Lowest Common Ancestor Semantics

Keyword queries present an easy and intuitive way for querying XML data without requiring from the user to write

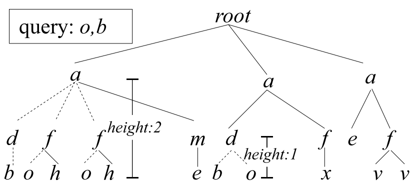


Figure 1: An example XML tree

complex queries or be aware of the data schema. To capture the structural information that is inherent in XML documents, keyword queries are usually interpreted through lowest common ancestor (LCA) semantics [10, 6, 11, 14, 13, 24, 15, 21, 25, 17].

Consider a conjunctive keyword query $q = \{w_1, \ldots, w_k\}$ consisting of k keywords $w_1, \ldots, w_k$ and an unordered labeled XML tree $T = (V, E)$ of an XML document d. Each node $v_i \in V$ corresponds to an XML element and edges $e(v_i, v_j) \in E$ capture that v_j is a subelement of v_i. We say that an element (node) $v \in V$ directly contains a keyword w_i ($contains(v, w_i)$), if the keyword appears as the label of the element, the label of one of its attributes, the value for one of its attributes, or in the content of the element. We denote as S_1 to S_k the sets of nodes such that $S_i = \{v | v \in V \text{ and } contains(v, w_i)\}$, $1 \leq i \leq k$.

Intuitively, the result set for q consists of the subtrees in T whose nodes contain all the keywords of q. We refer to such subtrees (query results) by their root node. More specifically, all approaches supporting LCA semantics define the results of q based on some variation of the lowest common ancestor of the nodes in d that contain the query keywords. The Lowest Common Ancestor (LCA) of a set of nodes $V' = \{v_1, \ldots, v_k\}$ ($V' \subseteq V$) is defined as the deepest node v in T which is an ancestor of all nodes in V' and is evaluated by a function $lca(v_1, \ldots, v_k)$. A node v of tree $T(V, E)$ of document d belongs to the result set of query $q = \{w_1, \ldots, w_k\}$, denoted $Result(q)$, if $v = lca(v_1, \ldots, v_k)$, where $v_1 \in S_1, \ldots, v_k \in S_k$. Similarly to $lca(v_1, \ldots, v_k)$, we define a function $lca(S_1, \ldots, S_k)$ that evaluates the set of LCA nodes V'', such that $v' \in V''$ if $v' = lca(v_1, \ldots, v_k)$ and $v_1 \in S_1, \ldots, v_k \in S_k$. It is obvious that $lca(S_1, \ldots, S_k)$ computes the result set for q.

We discern between two groups of LCA-based approaches: (i) the Smallest LCA [24] and the Exclusive LCA [25] with their variations [10, 16, 21], and (ii) the Meaningful LCA [14] and the Valuable LCA [13]. The first group defines query results solely on the structural relationships between the nodes in an XML document, while the second group also takes into account node types derived from the schema of the documents. Table 1 briefly presents each approach.

A common property of all the variations of the LCA is that for any query q and document d the set of the LCA nodes of the keywords in q as we have defined them above, referred to as basic LCA nodes in the following, is a superset of any type of LCA nodes we mentioned. The set of SLCA nodes is the set of basic LCA nodes if we exclude nodes that contain other LCA nodes for the query keywords in their subtree. The set of ELCA nodes is the set of basic LCA nodes if we exclude any node v that is a basic LCA of a set of nodes $V' = \{v_1, v_2, \ldots, v_k\}$ such that $\exists v_i \in V'$ that also belongs to the subtree of another node u that is also a descendant of v. Similarly for the MLCA and VLCA, for a node to be an MLCA or VLCA, it also needs to be a basic LCA by definition (Table 1). This means that if for any query q, we evaluate its result set against a document d using basic LCA semantics, the results for any of the other LCA types are included in this set. Thus, in our approach, for the document-query similarity evaluation, we do not focus on a specific type of LCA nodes, but instead, adopt a more general approach by supporting basic LCA semantics. Since basic LCA-based results are a superset of any of the other types of results, our approach can be used to approximate the results attained by any of the LCA-based approaches.

LCA semantics ensure that all query keywords appear in the result. However, intuitively, we would like to favor results where keywords appear close to each other in the tree, since for example two keywords that appear in the same *chapter* of a *book* are more related than two keywords that appear in the same *book*, but in different *chapters*. A common approach for quantifying the distance among the keywords in a result is the one presented in XRank [10] that uses Exclusive LCA semantics. In this case, the evaluation of the rank for a result takes into account the distance of each keyword from their Exclusive LCA by including a decay factor whose value increases with this distance. Here, we take the maximum of all such distances which is the one that determines the height of the subtree that has the LCA node as a root and the nodes directly containing the keywords as its leaves. In particular, we define the height, h, of each node $v \in Result(q)$ as:

$$h(v) = \max_i dist(v, v_i), \tag{1}$$

where v_i ($1 \leq i \leq k$) is a node that directly contains keyword w_i, and $dist$ is the length of the shortest path between v and v_i in the XML tree. The lower the height, the more closely connected the keywords in the XML tree, that is, they are contained into a more specific element. For example, in Fig. 1, for query (b,o), we have two results, one with height 2 (located under the left most a element) and one with height 1 (located under the second from the left a element).

In deep XML trees, i.e. trees with a large depth, a result with great height may still represent only a small percentage of the XML tree, while in shallow trees, even a result with low height may represent a large portion of the tree. Depending on the semantics one wants to convey, normalizing the height with respect to the tree depth may be required.

2.2 Problem Definition

The goal of database selection is to determine which databases or document collections are more useful (relevant) to a user query and rank them accordingly. More formally, we define the problem of database selection over collections of XML documents as:

Given N collections of XML documents $(D_1, D_2, \ldots, D_N)$ *and a keyword query* q*, rank the collections according to their goodness to* q.

For a document collection to be useful (good) for a query, the documents that are contained in the collection should provide relevant results for the query. Thus, the goodness of a collection D for query q is determined by the relevance of its documents to q. Similarly to [9], we consider that no user feedback is available to determine such relevance, and thus, define the relevance of a document $d \in D$ to q based solely on the similarity of d to q. Furthermore, we consider that a document is relevant only if this similarity measure exceeds a user-specified *similarity threshold* l. This threshold is used

Table 1: Lowest Common Ancestor Semantics

LCA-type	Description	Definition
Smallest LCA (SLCA) [24]	v is an SLCA if all keywords of q appear in the subtree rooted at v and none of its descendants has such a subtree containing all keywords.	$v \in slca(S_1, S_2, \ldots, S_k)$ if $v \in lca(S_1, \ldots, S_k)$ and $\forall u \in lca(S_1, S_2, \ldots, S_k)$ v not an ancestor of u.
Exclusive LCA (ELCA) [25]	v is an ELCA if it contains at least one occurrence of each keyword in the subtree rooted at v, excluding the occurrences of the keywords in subtrees of its descendants already containing all the keywords	$v \in elca(S_1, S_2, \ldots, S_k)$ iff $\exists v_1 \in S_1, \ldots, v_k \in S_k$: $v = lca(v_1, \ldots, v_k)$ and $\forall v_i (1 \leq i \leq k)$ the child of v in the path from v to v_i is not an LCA of $S_1, \ldots, S_k$ itself or an ancestor of such an LCA.
Meaningful LCA (MLCA) [14]	v is an MLCA if in the subtree rooted at v, the nodes containing the keywords are pairwise meaningfully related	v is an MLCA, if all pairs of nodes (v_i, v_j) in the subtree rooted at v that contain the keywords of q are such that $\nexists\ v'_i, v'_j$ containing the same keywords such that lca(v_i, v_j) is an ancestor of lca(v'_i, v'_j)
Valuable LCA (VLCA) [13]	v is a VLCA, iff for the nodes v_i, v_j, containing keywords (w_i, w_j), in the subtree rooted at v, there are no other two nodes of the same label/tag except v_i, v_j.	For $v = lca(v_1, \ldots, v_k)$, v is the VLCA of $v_1, \ldots, v_k$ iff $\forall v_i, v_j$ there are no other two nodes of the same label/tag.

to enable the user to specify what she considers as relevant for her query. We estimate the goodness of D by aggregating the similarity scores of all relevant documents. In particular:

Definition 1. *Given a user specified similarity threshold l and a query q, the goodness of an XML document collection D to q is defined as:*

$$Goodness(q, D, l) = \sum_{d \in D} sim(q, d, l),$$

where $sim(q, d, l)$ determines the similarity of a single document to a query.

Thus, we rank highly (estimate high goodness values for) both collections that have a large number of documents with a relatively small similarity score, as well as collections that contain less documents but with higher similarity scores. By using the similarity threshold, we somewhat limit the tendency of the model to favor large collections and consider only documents that are considered relevant by the user. Alternatively, we can consider for the goodness evaluation of D, instead of the entire collection, only its top-K most relevant documents to q, where K is, similarly to l, user-defined.

We distinguish between a *boolean* and a *weighted* model for the selection problem, which differ on the definition of the similarity of a document to a query.

Boolean model. For a given similarity threshold l, a document d is considered to *match* query q if there exists at least one result $v \in Result(q)$ such that $h(v) \leq l$.

$$sim(q, d, l) = \begin{cases} 1, & \text{if } \min_{u \in Result(q)} h(u) \leq l \\ 0, & \text{otherwise} \end{cases}$$

The goodness of a document collection is then defined according to Def. 1 as the number of matching documents $d \in D$.

Note that our model is not purely boolean, as the similarity threshold is used as a means to discard the irrelevant documents from the goodness estimation. If l has a high enough value, i.e., if l is greater than the depth of a document d, then the similarity measure becomes purely boolean as it only checks for the existence of the keywords in d.

Weighted model. In the weighted model, besides determining whether a document d matches a query q, i.e., $\exists v \in Result(q)$ such that $h(v) \leq l$, we also take into account the height of the result into the computation of the query similarity to d. We define a function F of the height h of a result node v such that the similarity of d to q is greater when $h(v)$ is small. For example, F can be defined as the inverse of the height of the node, $1/h(v)$. With this similarity measure, we obtain greater similarity scores for documents that contain all query keywords within more specific elements.

Since a document d may contain many results for a query q, to determine its similarity to q, we consider in the similarity computation the height of the most specific element node in the result set, i.e., the node in $Result(q)$ with the smallest height.

$$sim(q, d, l) = \begin{cases} F(\min_{v \in Result(q)} h(v)), \\ \quad \text{if } \min_{v \in Result(q)} h(v) \leq l \\ 0, \text{otherwise} \end{cases}$$

The goodness of a collection is evaluated this time by accumulating the similarity scores of all documents that have at least one result node with height smaller than l.

3. PAIRWISE LCA ESTIMATION

To evaluate a keyword query q against a document d, the straightforward approach is to apply on d an algorithm for finding the LCA nodes of the k keywords that appear in q, which constitute $Result(q)$. Since there may be multiple occurrences of each keyword w in d, there may also exist more than one such LCA nodes which we need to compute.

For the selection problem, after the evaluation of the LCAs, the LCA node $v \in Result(q)$ with the minimum height is selected and if $h(v) \leq l$, the boolean model returns a match for d, while the weighted model computes the similarity based on function F.

Thus, to estimate the goodness of an XML document collection D for a query q, we need to compute the LCA nodes of the keywords in q, for each document $d \in D$. This evaluation is expensive processing-wise and leads to low response times especially for large collections with many documents. For instance, if the Exclusive LCA (ELCA) is used, then, for the evaluation of the ELCA nodes a brute force algorithm has a $O((k)(depth)|S_1| \ldots |S_k|)$ processing cost, for a tree with depth *depth* and k keywords in q. Even the state-of-the-art algorithms for ELCA, only reduce the complexity to $O((k)(depth)|S|log|S'|)$ [25], where $|S|$ ($|S'|$) is the cardinality of the set with the nodes that directly contain the least (most) frequent keyword in the query.

To avoid this step at execution time, a preprocessing phase may be deployed. In this phase, for each document $d \in D$, we compute the LCA nodes for all possible combinations of keywords that appear in D and maintain their heights. Based on this information, we may evaluate the similarity of any query q to d very efficiently.

However, the number of all possible combinations of keywords is very large and its precomputation imposes a large overhead both on processing cost and on storage, since this information needs to be maintained. In particular, the number of the LCAs that need to be computed for an XML document with n (non-distinct) keywords is: $\sum_{i=2}^{n} \binom{n}{i} = O(2^n)$.

Algorithm 1 Boolean Similarity Evaluation

Input: q: keyword query, $Htab(d)$: keyword pair table for document d, l: similarity threshold
Output: sim: similarity, $fpFlg$: false positive flag

```
 1: Extract all possible keyword pairs $(w_s, w_t)$ from $q$
 2: Lookup all $(w_s, w_t)$ in $Htab(d)$
 3: if a pair does not appear in the table then
 4:    $sim(q, d, l) = 0$
 5: else
 6:    $sim(q, d, l) = 1$
 7:    if for any $(w_s, w_t)$, $h_{max}(s, t) = NULL$ then
 8:       $fpFlg$=1
 9:    end if
10: end if
11: RETURN($sim, fpFlg$)
```

3.1 Pairwise Estimation

We claim that one does not need to compute and maintain the LCA nodes of all the possible combinations of keywords in a document d to evaluate the document similarity to a query q. Instead, we rely on pairwise LCAs to estimate the height of the LCA for any set of keywords.

PROPOSITION 1. *Let $G(V, E)$ be an acyclic directed graph, and $V' = \{v_1, \ldots, v_M\}$ any subset of M nodes in G, $V' \subseteq V$. Then, $h(lca(v_1, \ldots, v_M)) = \max_{v_i, v_j \in V'} h(lca(v_i, v_j))$.*
Proof. Let $u \in V$ and $u = lca(v_1, \ldots, v_M)$ and there is also $u' \in V$ and $u' = lca(v_l, v_m)$ where $v_l, v_m \in V'$, such that: $h(u') > h(u)$. However, u is a common ancestor of v_l,v_m and since $h(u') > h(u)$, then, u cannot be the LCA of all $v \in V'$. Thus, $h(lca(v_1, \ldots, v_M)) \geq \max_{v_i, v_j \in V'} h(lca(v_i, v_j))$.

Let us now assume there is a $u'' = lca(v_l, v_m)$, $v_l, v_m \in V'$, such that $h(u'') = \max_{v_i, v_j \in V'} h(lca(v_i, v_j))$, and $h(u) > h(u'')$. Then, there is at least one pair of nodes (v_s, v_t), $v_s, v_t \in V'$, such that $lca(v_s, v_t) = u''$. But since the LCA with the greatest height is u, there cannot be such a node. Thus, $h(lca(v_1, \ldots, v_M))$=$\max_{v_i, v_j \in V'} h(lca(v_i, v_j))$.□

Based on Prop. 1, we can estimate the height for the result of any query q against an XML document d. If the keywords in d are distinct (i.e., each keyword appears only once), then any query q has only a single result in d, i.e., the keywords of q have a single LCA node and we can provide an exact estimation of its height. If we maintain all possible pairs of keywords that appear in d along with the corresponding height of their LCA node, the maximum height of all the height values of the LCAs that are recorded for any pair of keywords in q corresponds to the height of the result of q against d.

However, in most cases, there are multiple occurrences of each keyword in an XML document and each pair of keywords may have multiple LCAs of different heights. Thus, for each distinct pair of keywords (w_i, w_j) in a document d, we maintain two values: (i) the height $h_{min}(i, j)$ of the LCA node $v \in lca(S_i, S_j)$ with $h(v) \leq h(u), \forall u \in lca(S_i, S_j)$ and (ii) the height $h_{max}(i, j)$ of the LCA node $v' \in lca(S_i, S_j)$ with $h(v') \geq h(u), \forall u \in lca(S_i, S_j)$. Let us denote with $Htab(d)$, the three column table in which we maintain this information for document d. In the first column of $Htab(d)$, we store the pair of keywords (w_i, w_j), in the second column, the $h_{min}(i, j)$ value, and in the third, the $h_{max}(i, j)$ value.

For a query q with k keywords, we look up each pair of keywords (w_s, w_t) of q in $Htab(d)$ to determine if these keywords appear in d. If they do, we also derive from the information maintained for the matching pair of keywords in $Htab(d)$, the minimum and maximum value height $(h_{min}(s, t), h_{max}(s, t))$ of the LCA nodes under which the pair appears in d. After looking up all pairs of keywords of q in $Htab(d)$, let $(w_{s'}, w_{t'})$ be the pair which has the maximum $h_{min}(s', t')$ value. We denote this value as $H_{min}(d, q)$. Similarly, we consider the pair $(w_{s''}, w_{t''})$ which has the maximum $h_{max}(s'', t'')$ value among all pairs and denote it as $H_{max}(d, q)$.

THEOREM 1. *Given a keyword query q and a document d, the height of any $v \in Result(q)$ is such that: $H_{min}(d, q) \leq h(v) \leq H_{max}(d, q)$.*
Proof. We first prove that any $v \in Result(q)$ has a height $h(v) \geq H_{min}(d, q)$. Let us assume that there exists $u \in Result(q)$, such that $h(u) < H_{min}(d, q)$. Then, for all keyword pairs (w_s, w_t) in table $Htab(d)$ that correspond to pairs of keywords in q, $h_{min}(s, t)$ should also be lower or equal to $h(u)$, since due to Prop. 1, the height of a result is determined by the maximum height value of all the node pairs. But $H_{min}(d, q)$ is defined as the maximum value of the $h_{min}(s, t)$ of all (w_s, w_t) in q, therefore we have at least one keyword pair $(w_{s'}, w_{t'})$ in q such that $h_{min}(s', t') > h(u)$. Thus, there cannot be a result node with height lower to $H_{min}(d, q)$.

Let us now prove that $v \in Result(q)$ has a height $h(v) \leq H_{max}(d, q)$. Let us assume that there exists $u \in Result(q)$ such that $h(u) > H_{max}(d, q)$. Then, according to Prop. 1, there is at least one pair of keywords associated with an LCA node with a height value greater than $H_{max}(d, q)$, which is impossible since $H_{max}(d, q)$ is defined as the maximum value of all height values associated with any pair of keywords corresponding to q. Thus, there cannot be a result node u with $h(u) > H_{max}(d, q)$.□

According to Th. 1, we can bound the height of any result of q in d between $H_{min}(d, q)$ and $H_{max}(d, q)$. Thus, if we maintain the appropriate information for all distinct pairs of keywords (w_i, w_j) in d in the table $Htab(d)$, we can provide an estimation for the height of the result of any keyword query q against d. That is, we determine that the height of any result cannot be lower than $H_{min}(d, q)$ and higher than $H_{max}(d, q)$. For example, in Fig. 1, the height of the result for query (o,b) is bounded between 1 and 2.

Given a query q and a similarity threshold l, if for document d, $H_{min}(d, q) > l$, then we can safely deduce that there are no results in d that exceed the similarity threshold the user has set. Theorem 1 guarantees, that we have no false negatives, there are no results with lower height than $H_{min}(d,q)$. If $H_{min}(d,q) \leq l$ and $H_{max}(d,q) \leq l$, d is surely relevant to q. False positives may appear only for documents for which $H_{min}(d,q) \leq l$ and $H_{max}(d,q) > l$, in which case pairs of nodes belonging to different subtrees may be combined to give a false $H_{min}(d,q)$ estimation.

```
Algorithm 2 Weighted Similarity Evaluation
Input: $q$: keyword query, $Htab(d)$: keyword pair table for document $d$, $l$: similarity threshold
Output: $sim$: similarity, $bsim$: lower bound for similarity
1: Extract all possible keyword pairs $(w_s, w_t)$ from $q$
2: Lookup all $(w_s, w_t)$ in $Htab(d)$
3: if a pair does not appear in the table then
4:     $sim(q,d,l) = 0$
5: else
6:     $sim(q,d,l) = F(H_{min}(d,q))$
7:     $bsim(q,d,l) = F(H_{max}(d,q))$
8: end if
9: RETURN$(sim, bsim)$
```

3.2 Goodness Estimation

Based on the above observations, we define appropriate algorithms for estimating the similarity of a document to a query under both the boolean and weighted models based on the pairwise LCA nodes. Thus, we reduce the complexity of the preprocessing phase from O(2^n) to $\binom{n}{2} = O(n^2)$, both processing and storage-wise.

For each document $d \in D$, we maintain the table $Htab(d)$, in which we insert information for the LCA with the minimum and maximum height value ($h_{min}(i,j)$, $h_{max}(i,j)$) for all distinct keyword pairs (w_i, w_j) that appear in d. Since we know that any pair of keywords (w_i, w_j) with $h_{min}(i,j)$ greater than l does not contribute in the similarity evaluation, we can safely omit such pairs from $Htab(d)$ for space and processing efficiency. Furthermore, we do not need to maintain the value of $h_{max}(i,j)$ for any keyword pair (w_i, w_j) for which $h_{max}(i,j) > l$ and simply set that value to $NULL$ in $Htab(d)$.

Given $Htab(d)$, we describe how any keyword query q is processed against document d under the boolean model. All keyword pairs from q are extracted and looked up in $Htab(d)$. If any of the pairs is not found, then we set the similarity of the document to the query to 0 (q does not match d). Otherwise, we set the similarity equal to 1. We also check the $h_{max}(s,t)$ values for all the keyword pairs (w_s, w_t) of q to determine whether there is a possibility for a false positive and set a corresponding flag ($fpFlg$) accordingly. The algorithm is detailed in Alg. 1.

To compute the goodness of a collection D of XML documents, the above algorithm is applied for all documents d in the collection. The goodness of the collection based on Def. 1 is estimated as the sum of the similarity values for each relevant document in the collection. We can derive a lower bound for the goodness of D, by counting the number of documents in D that have set their false positive flags and subtracting it from the estimated goodness value.

For the weighted version of the problem, we do not only determine whether a document matches or not a query, but we also evaluate a measure for its similarity as a function of the height of the LCA node. As in the boolean version, since

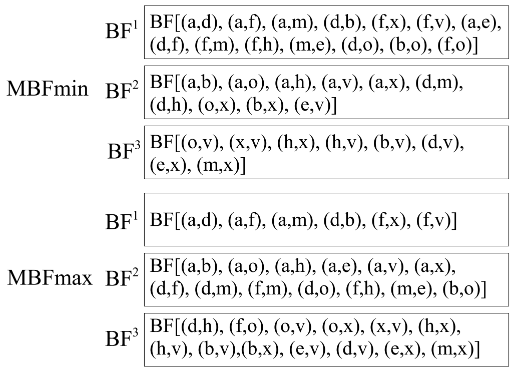

BF¹ BF[(a,d), (a,f), (a,m), (d,b), (f,x), (f,v), (a,e), (d,f), (f,m), (f,h), (m,e), (d,o), (b,o), (f,o)]
MBFmin
BF² BF[(a,b), (a,o), (a,h), (a,v), (a,x), (d,m), (d,h), (o,x), (b,x), (e,v)]
BF³ BF[(o,v), (x,v), (h,x), (h,v), (b,v), (d,v), (e,x), (m,x)]
BF¹ BF[(a,d), (a,f), (a,m), (d,b), (f,x), (f,v)]
MBFmax
BF² BF[(a,b), (a,o), (a,h), (a,e), (a,v), (a,x), (d,f), (d,m), (f,m), (d,o), (f,h), (m,e), (b,o)]
BF³ BF[(d,h), (f,o), (o,v), (o,x), (x,v), (h,x), (h,v), (b,v),(b,x), (e,v), (d,v), (e,x), (m,x)]

Figure 2: The MBFs for the XML tree in Fig. 1.

there may be several occurrences of the query keywords in a document to determine the similarity value, we rely again on the height of the LCA with the minimum height and the height of the LCA with the maximum height among all the LCA nodes in our result set.

The table $Htab(d)$ for a document d still suffices for determining the similarity value for the weighted model. While it can also provide an estimation on the number of false positives, $Htab(d)$ cannot be used for providing an estimation on the maximum height of the LCA node that may contain our query keywords, i.e., a lower bound for the similarity value. For the evaluation of a lower bound, we insert in the table for all keyword pairs (w_i, w_j) with $h_{min}(i,j) < l$ their $h_{max}(i,j)$ value even if $h_{max}(i,j) > l$. Then, we can derive a lower bound for the similarity ($bsim$) of the document d to query q based on $H_{max}(d,q)$. The detailed procedure for the evaluation is presented in Alg. 2.

To evaluate the goodness of the entire collection D, we evaluate the similarity score for each document based on $H_{min}(d,q)$ and sum these scores; thus, acquiring an upper bound for the goodness value. Furthermore, by summing the lower bound similarity estimations based on $H_{max}(d,q)$, we also provide a lower bound for the goodness estimation of the collection.

4. BLOOM-BASED SUMMARIES

Instead of maintaining the information about the keyword pairs for every document in a table, for space efficiency, we summarize this information using a well known hash-based structure, the Bloom filters [3]. Bloom filters are compact data structures for the probabilistic representation of a set of elements that support membership queries. Consider a set $A = \{a_1, ..., a_n\}$ of n elements. The idea is to allocate a vector x of s bits, initially all set to 0, and then choose m independent hash functions, $h_1, \ldots, h_m$, each with range 1 to s. For each element $a \in A$, the bits at positions $h_1(a), \ldots, h_m(a)$ in x are set to 1. Given a membership query for b, the bits at positions $h_1(b), \ldots, h_m(b)$ are checked. If any of them is 0, then certainly $b \notin A$. Otherwise, we conjecture that b is in the set, although there is a probability that we are wrong (false positive). To support updates, we maintain for each location i in the bit vector a counter of the number of times the corresponding bit is set to 1.

For the boolean problem, we replace the table $Htab(d)$, for each document d in the collection, with two Bloom filters, $BFmin(d)$ and $BFmax(d)$ corresponding to the second and third columns of the $Htab(d)$ table respectively, that maintained the $h_{min}(i,j)$ and $h_{max}(i,j)$ values for each distinct keyword pair (w_i, w_j) in d. Given a similarity threshold l, any keyword pair (w_i, w_j) in d for which $h_{min}(i,j) \leq l$ is hashed as one key and inserted into $BFmin(d)$. All keyword pairs (w_i, w_j) inserted in $BFmin(d)$ for which $h_{max}(i,j) \leq l$ are also inserted into $BFmax(d)$.

Algorithm 3 Bloom-Based Boolean Similarity Evaluation

Input: q: query, $BFmin(d)$, $BFmax(d)$: Bloom filters for document d, l: similarity threshold
Output: sim: similarity, $fpFlg$: false positive flag

```
 1: Extract all possible keyword pairs $(w_s, w_t)$ from $q$
 2: for all $(w_s, w_t) \in q$ do
 3:    Apply the hash functions to $(w_s, w_t)$
 4:    Lookup the output of the hash functions for $(w_s, w_t)$
       in $BFmin(d)$
 5:    if there is a miss then
 6:       $sim(q, d, l) = 0$
 7:       RETURN($sim$)
 8:    end if
 9: end for
10: $sim(q, d, l) = 1$
11: for all $(w_s, w_t) \in q$ do
12:    Lookup the output of the hash functions for $(w_s, w_t)$
       in $BFmax(d)$
13:    if there is a miss then
14:       $fpFlg = 1$
15:       RETURN($sim, fpFlg$)
16:    end if
17: end for
18: $fpFlg = 0$
19: RETURN($sim, fpFlg$)
```

To evaluate the similarity of d to q, first every pair of keywords of q is checked against $BFmin(d)$ and if there are no misses, the similarity is set to 1. Then $BFmax(d)$ is also checked to identify any possible false positives (Alg. 3).

The goodness estimation proceeds as when $Htab(d)$ is used. Note that this time the sum of false positive flags over all the collection only gives an estimation of the false positives present and not an upper bound. This is because the use of the Bloom filters introduces additional false positives due to the hash function collisions.

Bloom filters cannot be directly used for the weighted problem since they cannot maintain the value of the height of the LCA nodes which is required to determine the similarity value. Instead, we deploy a variation, called *Multi-Level Bloom filters*.

Instead of inserting all pairs of keywords (w_i, w_j) in d with $h_{min}(i,j) \leq l$ in a single Bloom filter $BFmin(d)$, we group the keyword pairs according to their $h_{min}(i,j)$ value and we use a separate Bloom filter for each such group. Thus, we construct a multi-level Bloom filter $MBFmin(d)$, which is defined as a set of l simple Bloom filters $BF^1, BF^2, \ldots, BF^l$ such that all pairs of keywords (w_i, w_j) in d with $h_{min}(i,j) = 1$ are hashed into BF^1, with $h_{min}(i,j) = 2$ into BF^2 and so on until $h_{min}(i,j) = l$. Again, we are not interested in any keyword pair for which $h_{min}(i,j) > l$, since such pairs do not contribute to the similarity of d to q.

Similarly to the $MBFmin(d)$, we define $MBFmax(d)$ instead of $BFmax(d)$. However, if we want to provide a lower bound for the similarity value of document d to q, as explained, maintaining only l levels in our filter is not enough. In this case, the number of levels in the filter is at most equal to the depth of the XML tree T of document d, since the highest value for the height of any LCA node in the tree is at most equal to the XML tree height. Figure 2 shows the $MBFmin(d)$ and $MBFmax(d)$ for the XML tree of Fig. 1 and $l = 3$.

To provide a lower bound for the similarity value, for each document $d \in D$, we maintain both the $MBFmin(d)$ with l levels and the $MBFmax(d)$ with *depth* levels, where *depth* is the depth of the XML tree corresponding to d. For each query q, we first examine the $MBFmin(d)$ and maintain for each pair of keywords (w_s, w_t) of q the highest level at which we found a match $(y'_{s,t})$. If a pair cannot be found in any level, then we set similarity to 0. Otherwise, the highest level y'', such that: $y'' > y'_{s,t}, \forall (w_s, w_t) \in q$, is used to estimate the similarity. A similar procedure is followed then against the $MBFmax(d)$ to estimate the lower bound for the similarity. By adding the similarity estimation and the lower bound for the similarity for all the documents in the collection, we determine an estimation (upper bound) and a lower bound for the goodness of the collection.

5. EXPERIMENTAL EVALUATION

We assume one of the popular LCA-type semantics, and in particular, exclusive lowest common ancestor (ELCA) semantics [10, 25], and examine whether our approximation approach can be efficiently used to estimate the goodness of a collection. That is, we compare our approach with an algorithm, we refer to as the *tree-based* (*tree*) approach, that estimates the goodness of a collection by evaluating a query against each document to retrieve the ELCA-based results, and returns a similarity measure based on the height of these results. Since the evaluation of the results and thus, the similarity of each document to the query is exact, we consider that the tree-based approach provides also the exact value for goodness.

We include in our comparison a *keyword-based* (*keyword*) approach, which ignores the structure of the XML data and evaluates the goodness of a collection by considering a document as relevant to a query q based solely on the appearance of the k keywords in the document, returning similarity 1 if all keywords exist, and 0 otherwise. We evaluate both a *pair-based* (*pair*) approach that uses the *Htab*(d) table, and a *bloom-based* (*bloom*) approach that uses the Bloom-based summaries for maintaining the LCA information.

We first perform a set of experiments on synthetic data to examine the performance of our approach under different settings and also explore the influence of the different parameters. In a second set of experiments, we evaluate our approach against a real data set to demonstrate how it works in real conditions.

5.1 Synthetic Data Sets

For data generation, we use the Niagara generator [18]. The accuracy of our pairwise estimation of goodness depends mainly on the structure of documents and in particular on the relative position of the query keywords. Thus, to generate documents, we keep the number of elements fixed and vary the percentage of distinct elements and the tree height. The generated queries consist of keywords of which 90% be-

Table 2: Input Parameters

Parameter	Range	Default
# of documents per collection ($\|D\|$)	20-200	100
# of elements per document (n)	-	50000
depth of XML tree ($depth$)	4-20	12
% of repeating element names (r)	0-0.6	0.3
query length (k)	1-6	4
similarity threshold (l)	1-12	4
number of collections (N)		12
number of Bloom filter hash functions		4
size of Bloom filter		996bits

long to the documents and 10% are other random keywords. Table 2 summarizes our parameters.

Goodness Estimation. We first evaluate the quality of the goodness estimation for a single document collection.

COLLECTION SIZE. We vary the number of documents in the collection from 20 to 200, and measure the goodness estimation for all four approaches and the lower bound estimation for our two approaches (*lower-pair* and *lower-bloom*) for the boolean (Fig. 3c) and the weighted model (Fig. 3a).

Our approaches provide a very accurate estimation for both the boolean and weighted models having only a small difference from the actual value of the goodness as given by the tree-based approach. The keyword-based approach, in most cases, overestimates the goodness, and only provides accurate estimations for 10% of the queries, which include keywords that do not appear in the documents. While, our approaches also overestimate the goodness, the estimation error is significantly smaller. For the weighted model, we observe lower goodness values than the boolean one because of the use of function F. The keyword-based approach is not considered in this case since it ignores structure.

The bloom-based approach trades off accuracy for storage and processing efficiency. While it increases the estimation error compared to the pair-based approach because of the false positives caused by the Bloom filters, it also reduces the required storage to about 8% of the storage occupied by the $Htab(d)$ table. It also reduces the processing cost by relying on hash-based lookups instead of table scans.

If we also evaluate the lower bounds for the goodness estimation, we provide tight bounds in which the actual goodness value for a collection is always included. The bloom-based approach usually presents a more optimistic lower bound estimation due to false positives introduced by the structure, which are around 10%.

Finally, we observe that as the number of documents in a collection increases, the estimation error of our approaches scales gracefully (around 20%) at the most. The bloom-based approach behaves a bit worse for larger collections also due to an increase of the false positives as the size of the Bloom filters remains fixed in the experiment. Allocating larger sizes for the Bloom filters alleviates this problem.

SIMILARITY THRESHOLD. We maintain the number of documents fixed, and vary the similarity threshold l (Fig. 3d-Fig. 3b). The similarity threshold affects mostly our approach with respect to the depth of the XML documents. When l is small, then our approaches provide better estimations (Fig. 3d). As the value of l increases estimation errors in the height of the LCA nodes cause larger estimation errors in both the goodness estimation and the lower bound estimation. For values closer to the tree depth our estimation again improves since most documents are considered matches. In particular, for the boolean model, if l becomes equal to the tree depth, then our estimation is similar to the one of the keyword-based approach that only checks for keyword containment.

DOCUMENT AND QUERY STRUCTURE. To examine the influence of the document structure and the size of the query, we measure the absolute estimation error of our approaches which is defined as the absolute value of the difference between the actual goodness value as provided by the tree-based approach and the value provided by each approximation approach. We observe similar results under both the boolean and the weighted model (Fig. 4).

XML tree depth. All approaches provide the best estimations for tree depths close to the similarity threshold (Fig. 4a). The estimations of our approaches are still reasonable for larger depths, where in the worst case, for $depth = 16$, the pair-based approach overestimates goodness by 27% and the bloom-based one by 34%. The keyword-based approach is not affected by the depth of the tree. If the tree depth increases even further, since we keep the number of elements fixed, due to the small fan-out of the tree, our approach again behaves better.

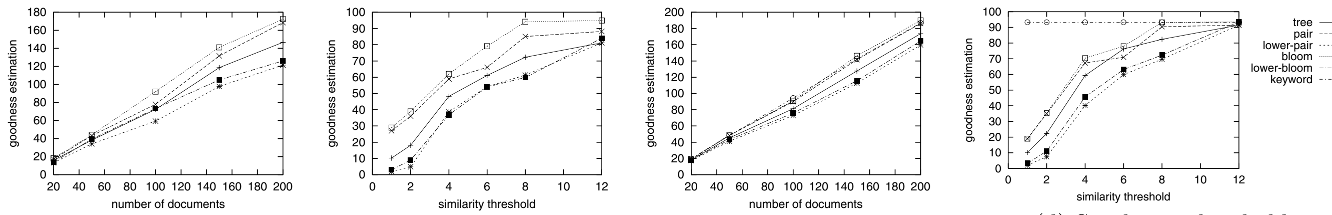
Percentage of repeating elements. Our approach is 100% accurate when we have no repeating element names and its accuracy decreases as the percentage of repeating elements increases (Fig. 4b). After a point the accuracy starts to improve, since the probability that two keywords appear close to each other at least once increases. Again, the keyword-based approach is not significantly affected.

Query length. As the query length increases, our approach behaves better since it considers more pairs for the ELCA evaluation (Fig. 4c). For a single-keyword query, all approaches except the bloom-based one provide accurate estimations.

XML Document Selection. We evaluate the quality of the rankings we derive based on our goodness estimation. We consider 12 document collections and compare both boolean and weighted versions of the keyword-based, the pair-based and the bloom-based approach.

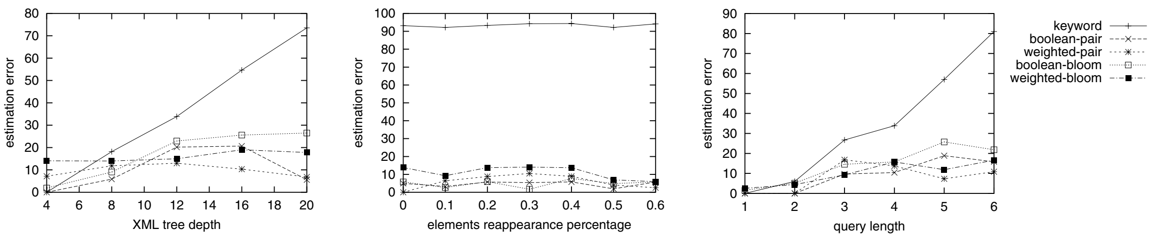
We measure for each of the approaches, the *Spearman Footrule distance* to the actual ranking for varying similarity thresholds. The Spearman Footrule (SF) distance between two ranked lists is defined as the absolute difference of their pairwise elements and is normalized by dividing with $1/2(S)$, where S is the number of elements in each list. We also evaluate the MAP, *mean average precision* of each approach, which is defined for a set of different queries as the average of the precision value (percentage of relevant collections) attained after each different query, divided by the number of queries. While the SF distance focuses on comparing the respective rank of each collection in two rankings, MAP is more precision oriented.

The document collections are constructed as follows: for a given set of queries, we produced a collection in which each document has at least one result with height 1, another collection in which each document has no results with height 1 but at least one result with height 2, and so on. Each document is constructed with the default parameter values. We consider three different sets of document collections. The first set contains collections of equal size, i.e., collections having the same number of documents. The sec-



(a) Number of documents (b) Similarity threshold (c) Number of documents (d) Similarity threshold

Figure 3: Goodness estimation for (a-b) the weighted and (c-d) the boolean problem.



(a) XML tree depth (b) Repeated elements (c) Query length

Figure 4: Goodness estimation for different document and query structures.

ond set contains collections of different size. In particular, we set the collection with the greatest ELCA height result we constructed (the 12th collection) as the one with the most documents, and decrease the size of the collection as the ELCA height decreases. That is, the collection with the results with ELCA height equal to 1, thus, the most relevant collection, is the one with the smallest size. Finally, we include a third set of random size collections as constructed in the first set of experiments.

The keyword-based approach has the worst overall performance with regards to the SF distance (Fig. 5a), since all collections are regarded as equally relevant despite their different structural properties. The approach approximates the real ranking best for collections of different sizes (Fig. 5b), since both the actual ranking in the boolean model and the keyword-based approach favor collections with large size.

The pair-based and the bloom-based approaches provide a ranking very close to the real one (maximum SF distance 0.3). The ranking of the pair-based approach has the smallest SF distance to the real ranking, in general, but the bloom-based approach sometimes outperforms it because of the more optimistic estimations it provides (Fig. 5). In the boolean model, similar to the actual ranking, our approaches also order all the collections with ELCA lower than l according to their size, and give a 0 goodness estimation to the others. Their SF distance is in most cases lower than 0.15.

For the weighted problem, for different size collections, the performance is slightly worse than for equal size collections. In this case, large collections are not favored against smaller ones with more specific ELCA-based results. For $l = 12$, for example, the 5-6th collections, that have ELCA with height 5 and 6, and are about average size among all, are the ones ranked the highest. Finally, the random collections induce the largest estimation errors, since there are more errors in the ELCA height estimation, and the weighted model is more sensitive to such errors than the boolean one.

With regards to the MAP measure, we obtain similar results. For the boolean model, the keyword-based approach has the worst precision since it considers any document that contains the query keywords as relevant (Fig. 6). Both our approaches behave well with a MAP that does not drop below 0.67, while it is in most cases around 0.75 to 0.85 (Fig. 6). The worst behavior is observed for collections of different size and when l takes values from 2 to 6. For larger values, the precision increases, since the percentage of irrelevant collections in our data set is reduced. The bloom-based approach is less precise than the pair-based one, because of the errors caused by the false positives in the structure.

5.2 Real Datasets

To evaluate our approach under a realistic setting, we use the DBLP bibliographic data collection. We split the DBLP data set into different documents so as to have one document for all the publication in one conference for each year, i.e., we have a document with the publications for "VLDB 2009", one for "VLDB 2008", etc. Journal articles are omitted. We then form two sets of collections, one set by grouping the documents based on their year of publication and one based on their conference. For instance, in the first set, we have collections such as "2009" which include documents "VLDB 2009", "ICDE 2009", and so on, while in the second set we have collections such as "VLDB" with documents corresponding to "VLDB 2009", "VLDB 2008", etc.

We pose queries using author names as our keywords and report our results. Note that according to the format of the DBLP data, if author "X" is a coauthor with author "Y" in an article, then the minimum height of the ELCA-based result for query "X and Y" is 1, while if they are authors in different articles then the minimum height is 2. We evaluate our approach under the boolean model and set l equal to 1 in order to retrieve documents including publications cowritten by "X" and "Y". Next, we report some indicating results.

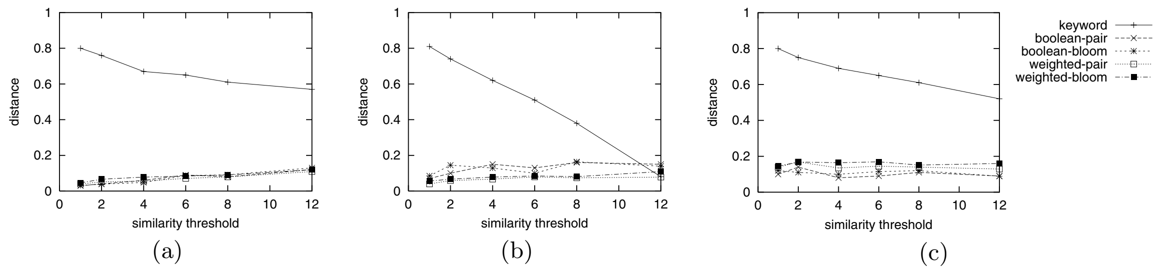


Figure 5: Database selection for (a) equal size collections, (b) different size collections and (c) random collections.

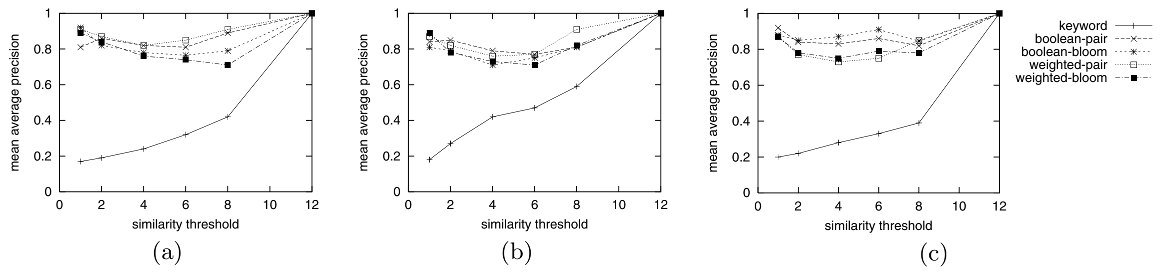


Figure 6: MAP for (a) equal size collections, (b) different size collections and (c) random collections.

We posed the following query: "Omar Benjelloun and Serge Abiteboul" against the document collections split by year. According to the DBLP data, the two authors had the most publications together (excluding journals) in 2004 with 4, followed by 2002 with 3, 2003 with 3, and 2005 with 1. Thus, the correct order for the collection selection problem is 2004, 2002, 2003 and 2005, while all other collections contain no results of interest. By applying the pair-based approach, we retrieved the collections in this exact order, i.e., with 0 SF distance from the actual ranking and precision equal to 1. For the bloom-based approach, the SF distance becomes 0.2 while the precision drops to 5/6 because of the false positives caused by the Bloom filters. Finally, for the keyword-based approach, the SF distance dropped to 0.46 and the precision became 1/2. In particular, the first two top-ranked collections were the ones corresponding to years 2006 and 2007 in which both authors had a lot of publications and many of them in common conferences, but not as coauthors.

We performed a second experiment against the collections split by conference, with query "Alon Y. Halevy and Zachary G. Ives". The authors have the most articles together in SIGMOD (6 articles) followed by several venues such as WebDB, WWW, CIDR, ICDE where they have one cowritten article. Both our approaches and the keyword based approach are able to identify SIGMOD as the collection with the most articles of interest. The pair-based approach again returns exactly the actual ranking, while the bloom-based one has an SF of 0.75 and a precision of 6/8. The keyword-based approach has the worst behavior.

6. RELATED WORK

Keyword-based XML Queries: Most approaches to keyword query evaluation are based on some type of LCA semantics [10, 24, 13, 14]. All such approaches can be approximated by our pairwise LCA estimations and can exploit the proposed compact statistics to efficiently support database selection. We have also shown experimentally the accuracy of such approximations for the ELCA [10, 25] semantics. Other related approaches include variations of the basic types of LCAs that we have presented, such as the Multi-Way Smallest LCA [21], which extends the Smallest LCA [24]. There is also related work towards more efficient evaluation of the LCA-based results, such as [25] that proposes efficient stack-based algorithms for the evaluation of the ELCAs, and [16] that uses materialized views to evaluate SLCAs. Recently, MaxMatch [17] extends previous LCA-based semantics to compute results that follow more strict querying properties, such as consistency and monotonicity. XKeyword [11] is a different approach that builds appropriate path indices summarizing the data, which however requires schema information. IR techniques have also been deployed for keyword queries processing over XML data such as [8, 6].

Selectivity Estimation for XML Documents: Summaries for XML documents have also been used to provide selectivity estimations for queries against XML documents. For this problem, most approaches rely on the use of path indexes [2] or summary graphs [19] and are designed to support queries on structure (such as twigs) instead of keyword queries. Also, [7] presents a tool for extracting statistics from XML schema to construct compact and accurate structural summaries. The approach requires schema information. Bloom filters were also used for summarizing the path expressions in an XML document in the form of Bloom histograms [23]. A histogram based on the frequencies of the paths in an XML tree is built, and the paths that fall in each bucket are summarized by a Bloom filter. This structure supports efficient matching similar to ours, but as [2] supports path and not keyword queries. Bloom-based summaries were also used for XML (not keyword-based) query processing in peer-to-peer networks [12, 1]. The focus there is on using such summaries for indexing collections as opposed to ranking collections.

Database Selection: The problem of database selection has been mainly addressed for text and relational databases. Most approaches rely on summaries of the database content. Previous research has not addressed what should such summaries for XML contain. This paper proposes maintaining information about the LCA of pairs of document keywords. In [9], the vector-space model is deployed for maintaining statistical summaries of each document of a text collection that contain information such as the tf/idf frequencies of the terms in the collection. Similarly in [4], inference networks based on information retrieval principles are used to estimate the goodness of document collections.

In Kite [20], keyword queries are deployed against relational data. Similarly, in [26] and [22], database selection is supported based on keyword queries. In [26, 22] meaningful relationships between keywords are defined against summaries built on top of relational databases. The relationships are defined based on the number of joins required to combine the tuples that hold the respective keywords and evaluated based on the tuple query tree that contains all query keywords and connecting tuples. Our approach uses a similar measure to define the importance of a result based on the height of the respective result tree that contains all keyword queries. A matrix summarizing the relationships of each keyword pair in a database is constructed as a summary in [26], while the same information is maintained in a more compact graph-based structure in [22]. Instead, we summarize the respective information in the Bloom filter structures. While both we and [26] rely on keyword pairs in the query to determine our results, [22] treats queries holistically. Finally, both [26, 22] deploy frequency statistics appropriate for relational databases for measuring the importance of each result. While these approaches could be applied for XML document collections if they were transfered into appropriate relational databases, they cannot be directly applied to the document collections. Furthermore, the main difference between our work and keyword queries on relational data is that in our case similarly to [9, 4] and based on IR principles when dealing with text document collections, the basic unit of information is a document, while in relational databases the basic unit is a tuple.

7. CONCLUSIONS

In this paper, we dealt with the problem of selection for XML document collections. We considered keyword queries and evaluated the similarity of a document to a query based on the height of the LCA node defining the query result. We introduced an efficient approach for approximating the height of the LCA node of any keyword query by maintaining information about the LCAs of the pairs of keywords that appear in a document. Furthermore, we presented a compact index structure for maintaining this information and exploited these approximations to estimate the goodness of an XML collection under both a boolean and a weighted model. Through our experimental evaluation on both real and synthetic data, we evaluated our approach with respect to approximating exclusive LCA [10, 25] semantics and showed that it is accurate, efficient and provides rankings close to the actual one.

We plan to experimentally study how well our approach can approximate other types of LCA semantics, such as smallest LCA [24] semantics. Furthermore, we plan to examine other forms of statistical information about the pairwise LCAs for the goodness estimation for specific types of LCA semantics.

8. REFERENCES

[1] S. Abiteboul, I. Manolescu, N. Polyzotis, N. Preda, and C. Sun. XML processing in DHT networks. In *ICDE*, 2008.

[2] A. Aboulnaga, A. R. Alameldeen, and J. F. Naughton. Estimating the selectivity of xml path expressions for internet scale applications. In *VLDB*, 2001.

[3] B. Bloom. Space/time trade-offs in hash coding with allowable errors. *CACM*, 13(7), 1970.

[4] J. P. Callan, Z. Lu, and W. B. Croft. Searching distributed collections with inference networks. In *SIGIR*, 1995.

[5] S. Chernov, P. Serdyukov, M. Bender, S. Michel, G. Weikum, and C. Zimmer. Database selection and result merging in p2p web search. In *DBISP2P*, 2005/2006.

[6] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. Xsearch: A semantic search engine for xml. In *VLDB*, 2003.

[7] J. Freire, J. R. Haritsa, M. Ramanath, P. Roy, and J. Simeon. Statix: making xml count. In *SIGMOD*, 2002.

[8] N. Fuhr and K. Groβjohann. Xirql: A query language for information retrieval in xml documents. In *SIGIR*, 2001.

[9] L. Gravano, H. Garcia-Molina, and A. Tomasic. Gloss: text-source discovery over the internet. *ACM Trans. on Database Systems*, 24(2):229–264, 1999.

[10] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. Xrank: Ranked keyword search over xml documents. In *SIGMOD*, 2003.

[11] V. Hristidis, Y. Papakonstantinou, and A. Balmin. Keyword proximity search on xml graphs. In *ICDE*, 2003.

[12] G. Koloniari and E. Pitoura. Content-based routing of path queries in peer-to-peer systems. In *EDBT*, 2004.

[13] G. Li, J. Feng, J. Wang, and L. Zhou. Effective keyword search for valuable lcas over xml documents. In *CIKM*, 2007.

[14] Y. Li, C. Yu, and H. V. Jagadish. Schema-free xquery. In *VLDB*, 2004.

[15] Z. Liu and Y. Chen. Identifying meaningful return information for xml keyword search. In *SIGMOD*, 2007.

[16] Z. Liu and Y. Chen. Answering keyword queries on xml using materialized views. In *ICDE (Poster)*, 2008.

[17] Z. Liu and Y. Chen. Reasoning and identifying relevant matches for xml keyword search. *PVLDB*, 1(1):921–932, 2008.

[18] The niagara generator. In *http://www.cs.wisc.edu/niagara*.

[19] N. Polyzotis, M. Garofalakis, and Y. Ioannidis. Selectivity estimation for xml twigs. In *ICDE*, 2004.

[20] M. Sayyadian, H. LeKhac, A. Doan, and L. Gravano. Efficient keyword search across heterogeneous relational databases. In *ICDE*, 2007.

[21] C. Sun, C. Chan, and A. Goenka. Multiway slca-based keyword search in xml data. In *WWW*, 2007.

[22] Q. H. Vu, B. C. Ooi, D. Papadias, and A. K. H. Tung. A graph method for keyword-based selection of the top-k databases. In *SIGMOD*, 2008.

[23] W. Wang, H. Jiang, H. Lu, and J. X. Yu. Bloom histogram: Path selectivity estimation for xml data with updates. In *VLDB*, 2004.

[24] Y. Xu and Y. Papakonstantinou. Efficient keyword search for smallest lcas in xml databases. In *SIGMOD*, 2005.

[25] Y. Xu and Y. Papakonstantinou. Efficient lca based keyword search in xml data. In *EDBT*, 2008.

[26] B. Yu, G. Li, K. Sollins, and A. K. H. Tung. Effective keyword-based selection of relational databases. In *SIGMOD*, 2007.