An Evaluation Study of Search Algorithms for XML Streams

Evandrino G. Barros[1,2], Mirella M. Moro[1], Alberto H. F. Laender[1]

[1] Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
31270-901 –Belo Horizonte – Brasil
{ebarros,mirella,laender}@dcc.ufmg.br
[2] Departamento de Computação
Centro Federal de Educação Tecnológica de Minas Gerais
30421-169 – Belo Horizonte – Brasil
ebarros@decom.cefetmg.br

Abstract. Keyword-based searching services over XML streams are essential for widely used streaming applications, such as dissemination services, sensor networks and stock market quotes. However, XML stream keyword search algorithms are usually schema dependent and do not allow pure keyword queries. Furthermore, ranking methods are still relatively unexploited in such algorithms. This paper presents an accuracy and performance study of two keyword-based search algorithms for XML streams. Our study provides a comparison of these two algorithms by using an XPath benchmark as source of data and queries. Moreover, we also consider a large collection of XML documents and a large set of random queries, both based on DBLP dataset. Finally, we propose a strategy that combines both algorithms and ranks the keyword-based search results.

Categories and Subject Descriptors: H.2.4 [**Database Management**]: query processing and textual database

Keywords: keyword search, ranking algorithm, XML streams

1. INTRODUCTION

XML has become the most successful and ubiquitous technology for Web data exchange [Wilde and Glushko 2008]. So much has already been done on XML research that DBLP[1] had over 5500 papers on XML in the beginning of 2009 [Moro et al. 2009]. The question then becomes: what else can be done? And the answer, that keeps us moving, is simple: there is still interesting issues to be solved. One of those was identified by Vagena and Moro [2008] as *semantic search over XML streams*, i.e., keyword-based search over XML streams. This issue is motivated by three problems of using structured XML query languages (e.g., XPath and XQuery) on stream environments: (i) these languages require knowledge of the data structure in order to formulate meaningful queries; (ii) Web XML documents have large amounts of plain text, which makes it hard for processing value-based predicates with exact semantics; and (iii) those two issues are aggravated when we consider a Web scale, distributed system with different data sources, since multiple data sources require structured query rewriting. Furthermore, keyword-based search over XML streams is essential for widely used streaming applications, such as dissemination services, sensor networks and stock market quotes.

[1]http://www.informatik.uni-trier.de/~ley/db/

This research is partially supported by InWeb - National Institute of Science and Technology for the Web (MCT/CNPq grant no. 573871/2008-6) and by project Amanajé (MCT/CNPq grant no. 479541/2008-6), and by the authors' individual grants from CNPq and FAPEMIG.

The work by Vagena and Moro [2008] proposes a query language and two algorithms for semantic search over XML streams. The query language allows users to specify a set of query terms that can inform labels and keywords together (*label*::*keyword*) or individually (*label*::, ::*keyword*, *keyword*). The two algorithms are based on node relatedness heuristics for searching over XML documents. Note that there are other approaches for searching over *stored* XML documents (XRANK [Guo et al. 2003] and SLCA [Xu and Papakonstantinou 2005]). Such approaches rely heavily on full-text indexes, which do not exist on XML stream environments. Although the authors propose a language and two algorithms for keyword-based search over XML streams, they do not present any kind of evaluation of such algorithms in terms of effectiveness and performance.

Moreover, once there is a keyword-based search algorithm for XML streams, a natural further step is to present the results to the user ordered by their relevance. However, ranking has not been exploited in such algorithms. Currently, keyword-based search algorithms that allow ranking use disk-based structures, such as inverted lists or indexes to produce ranked results [Bao et al. 2009; Cohen et al. 2003; Guo et al. 2003; Li et al. 2010; Liu and Chen 2008; Zhou et al. 2010]. Thus, those algorithms are not suitable for the stream environment that we consider.

This paper presents an evaluation study of two keyword-based search algorithms for XML streams that employ the lowest common ancestor (LCA[2]) semantics (which were introduced in [Vagena and Moro 2008]). This study is divided in two parts. The first one uses queries and dataset from XPath-Mark [Franceschet 2005], a well known XPath benchmark, in order to fairly evaluate those algorithms in terms of precision and recall. The second one simulates a real stream environment as it considers a large collection of documents and a large set of random queries, both taken from DBLP.

Overall, our results show the effectiveness of those two algorithms for supporting XML keyword-based queries. Moreover, based on such evaluation, we also propose a ranking strategy that combines the two algorithms in order to improve query results. Our contributions can be summarized as follows:

(1) We present an effective and efficient implementation of the XRANK and SLCA algorithms for keyword searching over XML streams.
(2) We perform an accuracy and performance study of these algorithms by using an XPath benchmark as source of data and queries. We perform an accuracy and scalability study by using a large collection of XML documents and a large set of random queries, both based on DBLP.
(3) We propose an algorithm, called LCARank, a simple, efficient and effective strategy for ranking keyword-based search results.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 presents background information on the XRANK and SLCA algorithms. Section 4 describes our empirical study and discusses its results. Section 5 presents LCARank, our ranking algorithm for keyword search results. Finally, Section 6 concludes the paper.

2. RELATED WORK

Our work addresses three topics: traditional XML search, XML search over streams, and XML searching and ranking algorithms. This section summarizes related work on such topics and emphasizes how our paper contributes to the state-of-the-art.

Traditional XML Search. XML search algorithms employ heuristics to decide whether a node[3] is an answer to a query. There are many keyword-based algorithms for traditional XML search, which considers persistent environments where XML documents can be permanently stored and indexed.

[2]The LCA of nodes u and v in a tree is the shared ancestor of u and v that is located farthest from the root.
[3]Node in an XML document refers to both elements and attributes.

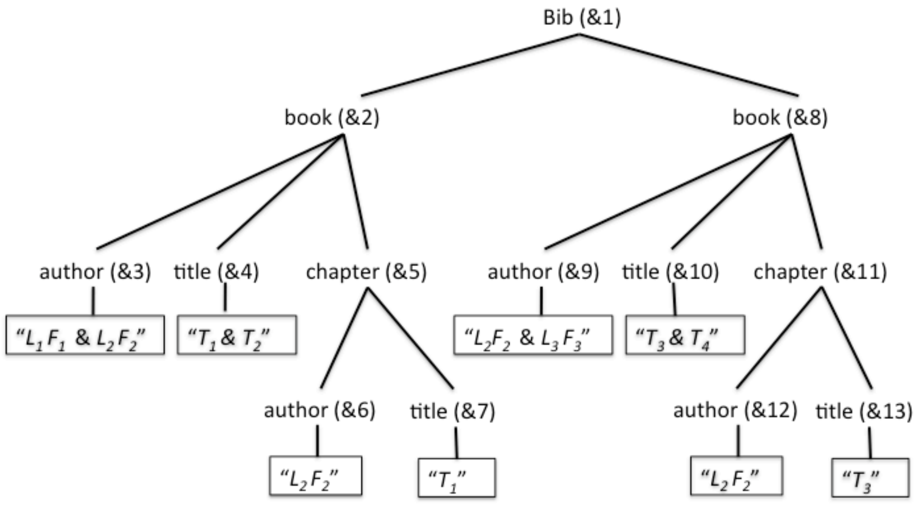

Fig. 1. XML document example: books, their chapters and authors.

Usually, the best answers for such queries are the most specific set of nodes containing the query terms [Termehchy and Winslett 2009]. After applying some heuristic, the search algorithms return either the identified sets or the document fragments containing such sets. The most employed heuristics are based on the LCA semantics [Liu and Chen 2008; Sun et al. 2007; Vagena et al. 2007; Xu and Papakonstantinou 2008; Zhou et al. 2010]. Current work mainly focuses on improving the precision and performance of existing keyword search algorithms such as XRANK and SLCA. Xu and Papakonstantinou [2008] propose the Indexed Stack algorithm, which presents precision and performance improvements over XRANK. Zhou et al. [2010] present Hash Count, an algorithm that outperforms Indexed Stack. Li and Chen [2008] present the Max-Match algorithm, which improves SLCA precision. Although these algorithms enhance existing results, they do not provide any ranking. Moreover, they use auxiliary structures such as inverted lists or B-tree indexes to improve performance and precision. However, these structures are not supported when searching over XML streams.

XML Search over Streams. Different papers have proposed strategies to evaluate queries over XML streams and can be divided in tow groups. The first one mainly addresses XPath processors for XML streams [Chen et al. 2006; Onizuka 2010; Peng and Chawathe 2005; Ramanan 2009], where a set of XPath queries is evaluated against a stream of documents to verify which documents satisfy the queries. However, these papers do not address keyword queries and ranking strategies.

The second group focuses on keyword-based queries, such as Li et al. [2008]. Their strategy ranks XML nodes whose leaves match keywords by using the distance between leaves as a ranking criterion. However, it does not handle labels as keywords and, in some situations, produces unexpected results. For example, consider the following user query based on Figure 1. Suppose a user wants an author L_2 who wrote a book or book chapter titled T_2. Thus, according to their strategy, node $book_{\&2}$ will be returned, since the leaves of $title_{\&4}$ and $author_{\&6}$ match the keywords. However, this result is not correct because author L_2 wrote a book chapter titled T_1, not T_2. Our work avoids this type of problem, as we shall see later.

XML Search and Ranking Heuristics. There are many specialized XML search engines that implement keyword searching with ranking heuristics [Bao et al. 2009; Cohen et al. 2003; Guo et al. 2003; Li et al. 2010]. Bao et al. [2009] present the XReal algorithm, which extends the TF-IDF similarity scheme to handle XML documents, thus providing a relevance ranking for search results. Cohen et al. [2003] present the XSearch algorithm that uses a simple TF-IDF ranking, involving tree sizes and node relationships. XRANK [Guo et al. 2003] extends Google's PageRank [Brin and Page 1998] to XML element level to produce ranked LCA results. Li et al. [2010] propose a ranking function that uses statistic information based on XML content and structure distributions. These distributions help to evaluate correlations between possible results and keyword queries.

Inverted lists, indexes and statistic information cannot be used when searching over XML streams because they are not supported in continuous streaming environments. Our work addresses this problem by proposing a ranking strategy that combines the SLCA and XRANK algorithms, prioritizing those nodes on the lowest subtrees.

3. KEYWORD SEARCHING CONCEPTS

This section summarizes the keyword query language and the search algorithms XRANK and SLCA. It also describes a combination of these search algorithms, which will be used as a ranking strategy.

3.1 Search Language

Vagena and Moro [2008] propose a semantic query language (inspired by [Cohen et al. 2003] and [Vagena et al. 2007]) and two algorithms for keyword search over XML streams. Specifically, a user specifies a query as a set of terms that inform labels (l) and keywords (k) in two ways: (i) together as l::k, or (ii) individually as l::, ::k and k. A node n within a document satisfies a query term when its label is equal to l and its tokenized textual content contains the word k. Also, n satisfies a query term of the form k if either its label is k or its tokenized textual content contains the word k. In other words, this last form of query allows the user to match its keywords to the node structure or to its value. Each query term results in a separate stream (S) that contains all the nodes that satisfy that term. Since a query may be composed of many query terms, its complete result is formed by the document fragments that contain nodes from S with the additional constraint that such nodes are meaningfully related. This last constraint is processed by the relatedness heuristics, as explained next.

3.2 Search Algorithms and Heuristics

After presenting the query language, the work in [Vagena and Moro 2008] introduced two algorithms for keyword search over streams based on node relatedness heuristics: XRANK [Guo et al. 2003] and SLCA [Xu and Papakonstantinou 2005]. Note that there are other approaches for searching over *stored* XML documents, as mentioned in Section 2. Such approaches rely heavily on full-text indexes, which do not exist on an XML stream environment. Furthermore, these approaches have employed many different heuristics to decide when a group of nodes contains meaningfully related nodes. A comparison survey of such heuristics [Vagena et al. 2007] has concluded that the heuristics described in [Guo et al. 2003] and [Xu and Papakonstantinou 2005] (and their variants) are the ones that return better quality results (less false positives and negatives). Those two heuristics are based on the LCA semantics. They are defined in terms of two nodes, n_1 and n_2, as follows. Assume that the XML semantic query $Q=\{Q_1,Q_2\}$ has produced two streams S_1 and S_2, for query terms Q_1 and Q_2 respectively (note that the following definitions involve only two nodes, but they can handle any number of nodes).

Definition 1. *(XRANK Heuristic [Guo et al. 2003]) The XRANK heuristic asserts that two nodes n_1 and n_2 in an XML document d_i, with $n_1 \in S_1$ and $n_2 \in S_2$, are meaningfully related if there are no nodes $n_3 \in d_i$ and $n_4 \in d_i$ such that $LCA(n_1, n_3)$ is descendant of $LCA(n_1, n_2)$ or $LCA(n_2, n_4)$ is descendant of $LCA(n_1, n_2)$.*

To better illustrate the XRANK heuristic, we present Example 1, which is based on Figure 1.

Example 1. *Suppose a user wants to retrieve titles written by author L_2. To do so, she issues the keyword query Q={author::L_2, tittle}, which has two terms, Q_1={author::L_2} and Q_2={title}. Initially, the set of nodes that match the two query terms are S_1={author$_{\&3}$, author$_{\&6}$, author$_{\&9}$, author$_{\&12}$} and S_2={title$_{\&4}$, title$_{\&7}$, title$_{\&10}$, title$_{\&13}$}, respectively. The XRANK heuristic will accepted the pairs (author$_{\&3}$, title$_{\&4}$), (author$_{\&6}$, title$_{\&7}$), (author$_{\&9}$, title$_{\&10}$), (author$_{\&12}$, title$_{\&13}$). Particularly, (author$_{\&3}$, title$_{\&4}$) will be accepted because LCA(author$_{\&3}$, title$_{\&4}$) = book$_{\&2}$ has no LCA descendant with author$_{\&3}$ or title$_{\&4}$. For instance, LCA(author$_{\&3}$, title$_{\&7}$) = book$_{\&2}$ is not descendant of LCA(author$_{\&3}$, title$_{\&4}$) = book$_{\&2}$. Therefore, the pair (author$_{\&3}$, title$_{\&4}$) will be accepted and (author$_{\&3}$, title$_{\&7}$) will not, even though author$_{\&3}$ $\in S_1$ and title$_{\&7}$ $\in S_2$. This explanation applies to accepted and rejected pairs. As final result, the XRANK heuristic will return nodes book$_{\&2}$, chapter$_{\&5}$, book$_{\&8}$and chapter$_{\&11}$, because they are the respective LCA nodes for the accepted pairs.*

Definition 2. *(SLCA Heuristic [Xu and Papakonstantinou 2005]) The SLCA heuristic states that two nodes n_1 and n_2 that belong to the same document d_i, where $n_1 \in S_1$ and $n_2 \in S_2$, are meaningfully related unless there exist two other nodes n_3 and n_4 that also belong to d_i such that the $LCA(n_3, n_4)$ is a descendant of the $LCA(n_1, n_2)$.*

Likewise, Example 2 illustrates the SLCA heuristic according to Figure 1.

Example 2. *For the same query Q={author::L_2, tittle}, presented in Example 1, the SLCA heuristic works as XRANK. However, it also rejects the pairs (author$_{\&3}$, title$_{\&4}$) and (author$_{\&9}$, title$_{\&10}$), because LCA(author$_{\&6}$, title$_{\&7}$) = chapter$_{\&5}$ is descendant of LCA(author$_{\&3}$,title$_{\&4}$) = book$_{\&2}$. Similarly, LCA(author$_{\&12}$, title$_{\&13}$) = chapter$_{\&11}$ is descendant of LCA(author$_{\&9}$,title$_{\&10}$) = book$_{\&8}$. Thus, it returns as result the document fragments rooted at chapter$_{\&5}$ and chapter$_{\&11}$, which are the LCA nodes for the accepted pairs.*

3.3 XRANK versus SLCA

The intuition behind the SLCA and XRANK heuristics is that smaller trees contain more meaningfully related nodes. The difference between these two heuristics is that the first one disqualifies a pair of nodes n_1 and n_2 if there exists any other pair (n_3, n_4) with $LCA(n_3, n_4)$ being a descendant of $LCA(n_1, n_2)$. Consequentially, SLCA returns the smallest LCA nodes whose elements match the query terms. Furthermore, SLCA establishes that appropriate result nodes are on the lowest XML subtrees.

In practice, XRANK includes SLCA result nodes and their ancestors that match the query terms. Particularly, only XRANK handles the ambiguity that might exist among XML labels, which has been identified as a semantic problem in XML keyword-based queries [Bao et al. 2009]. For example, in Figure 1, the keyword *title* occurs as *book* child and *chapter* child. Therefore, XRANK heuristic returns *book* and *chapter* nodes for the query in Example 1. SLCA, on the other hand, ignores this semantic aspect and returns only titles that appear as *chapter* child.

Therefore, by combining the XRANK and SLCA algorithms, we are able to handle any label ambiguity that might exist in the XML tree. In this way, because SLCA returns the smallest XML subtree results that present no keyword ambiguity, they are delivered first. The remaining non-SLCA results, generated exclusively by XRANK, are delivered subsequently. Our ranking algorithm, LCARank presented in Section 5, is based on this strategy.

4. EXPERIMENTAL EVALUATION

This section presents our experimental study for comparing the accuracy of XRANK and SLCA. Specifically, Section 4.1 compares the accuracy of XRANK and SLCA considering queries from an

XPath benchmark and its dataset. Then, Section 4.2 extends the accuracy study by considering a larger amount of documents and queries, both based on the DBLP dataset.

4.1 Accuracy on Benchmark Queries

The queries considered are the keyword versions of a subset of the XPath queries defined for the XPathMark benchmark [Franceschet 2005]. This way, we were able to use the corresponding results returned by a commercial XPath parser (Saxon [Kay 2010]) as a baseline. The queries were evaluated over the XMark auction dataset [Schmidt et al. 2002], a well-known synthetic XML dataset. XMark has important characteristics for our evaluation, since its documents present repetitive, recursive elements and deep XML trees. These features define a complex, more consistent dataset for recall and precision evaluations. In addition, its document generator allows creating XML documents of different sizes, which is very important for performance analyses. As we shall see, our experimental results show that XRANK and SLCA are suitable algorithms for keyword-based search over XML streams.

The experimental evaluation considered the following steps:

Step 1: We analyzed the 47 XPath queries from the XPathMark benchmark and identified those that could be expressed in the semantic query language described in Section 3. Notice that XPathMark is an XPath benchmark and, therefore, its queries have been specifically designed to assess XPath features.

Step 2: We processed the keyword version of these queries using the XRANK and SLCA algorithms over the XMark dataset. Thus, we simulated a stream environment by processing each document as an XML stream unit. The algorithms received these documents as input and processed them sequentially without any temporary storage.

Step 3: We processed the corresponding XPath queries by using the Saxon parser. This way, we were able to identify the relevant results that form our baseline, against which the keyword-based algorithms may be evaluated for recall and precision.

Step 4: We compared the accuracy of the algorithms in terms of recall, precision and F1 measures.

Next, we present the experimental setup, the queries and the results of the accuracy evaluation.

Experimental Setup. We created a dataset with five XML documents by using the XMark generator. The size of these documents in mega bytes were 0.116, 0.212, 0.468, 0.909 and 1.891. It is important to notice that the sizes of these documents are close to the sizes of real documents usually processed in XML stream environments [Lenkov 2003]. Each document was randomly generated representing a single auction XML dataset. We implemented the XRANK and SLCA algorithms in Java, by using Sun JDK version 1.6.0 and Saxon parser Java version 9.2. The experiments were conducted on an Intel Core 2 Duo workstation with 2.53 GHz and 4GB memory.

Queries. We used 18 of the 47 XPath queries defined by the XPathMark benchmark. These 18 queries are those whose semantics could be expressed by the keyword query language we adopted. Table I describes these 18 XPath queries and shows their corresponding keyword-based versions. For instance, the XPath query Q1 (*/site/regions/*/item*) returns elements *item* that are descendent of an element *regions*, child of *site*. In the keyword query language, the corresponding semantic is expressed by the query *regions*:: *item*::, which returns subtrees with elements *item* that are children, parents or brothers of elements *regions*. Hence, the keyword query language is more simple and flexible but cannot always express the exact semantics of XPath.

As Table I shows, all 18 XPath queries are expressed in keywords, despite the use of different axes, Boolean expressions and relational operators. Specifically, XPath queries with Boolean operations (Q7, Q8 and Q18) are mapped to keywords without using any logical operator; queries with matching

Table I. Selected XPath queries and their equivalent keywords.

Description	XPath queries	Keyword queries
Q1) All the items	/site/regions/*/item	regions:: item::
Q2) Keywords in annotations of closed auctions	/site/closed_auctions/closed_auction/annotation/description/parlist/listitem/text/keyword	closed_auction:: annotation:: description:: keyword::
Q3) All the keywords	//keyword	keyword::
Q4) Keywords in a paragraph item	/descendant-or-self::listitem/descendant-or-self::keyword	listitem:: keyword::
Q5) Paragraph items containing a keyword	//keyword/ancestor::listitem	listitem:: keyword::
Q6) Mails containing a keyword	//keyword/ancestor-or-self::mail	mail:: keyword::
Q7) North or South American items	/site/regions/namerica/item | /site/regions/samerica/item	namerica:: item:: samerica::
Q8) People having address and either phone or homepage	/site/people/person[address and (phone or homepage)]	person:: address:: phone:: homepage::
Q9) Initial and last bidder of all open auctions	/site/open_auctions/open_auction/bidder[position()=1 and position()=last()]	open_auction:: bidder::
Q10) Items whose description contains 'gold'	/site/regions/*/item[contains(description,'gold')]	item:: description:: ::gold
Q11) People names starting with 'Ed'	/site/people/person[starts-with(name,'Ed')]	person:: name::Ed
Q12) Mails sent on the 10th	/site/regions/*/item/mailbox/mail[substring-before(date,'/')='10']	mail:: date::10
Q13) Mails sent in September	/site/regions/*/item/mailbox/mail[substring-before(substring-after(date,'/'),'/')='09']	mail date::09
Q14) Mails sent in 1998	/site/regions/*/item/mailbox/mail[substring-after(substring-after(date,'/'),'/')='1998']	mail:: date::1998
Q15) Mails sent in the 21st century	/site/regions/*/item/mailbox/mail[substring(date,7,2)='20']	mail:: date::20
Q16) Items with descriptions longer than 1000 characters	/site/regions/*/item[string-length(normalize-space(string(description))) > 1000]	item:: description::
Q17) Person address longer than 30 characters	/site/people/person[string-length(translate(concat(address/street, address/city, address/country, address/zipcode)," ","")) > 30]	person:: address:: street:: city:: country::
Q18) People with both an email and a homepage	/site/people/person[boolean(emailaddress) = true() and not(boolean(homepage)) = false()]	person:: emailaddress:: homepage::

string functions (Q10 to Q15) are mapped to *label*::*keyword* (label and content together); and queries with XPath relational filter operators (Q9, Q16 and Q17) are mapped without their filters. As the keyword query language lacks support for all these operators, we adopted an expression as close as possible to the semantics of the corresponding XPath query. Despite that, queries Q1, Q2, Q3, Q4, Q5 and Q6 are fully expressed by our query language.

These 18 queries form a very consistent query set for our experimental evaluation. Previous work evaluated simple keyword queries [Bao et al. 2008; Cohen et al. 2003; Ramanan 2009; Yang and Shi 2007; Zhou et al. 2010]. Comparing the power of XPath against keyword-based query languages is beyond the scope of our study. Nonetheless, we employed an XPath benchmark to supply a set of expressive queries and provide a baseline for a consistent accuracy evaluation.

Accuracy Evaluation. We evaluated the accuracy of the XRANK and SLCA algorithms in terms of recall, precision and F1 measures, which are widely used in the information retrieval realm [Baeza-Yates and Ribeiro-Neto 1999]. In our context, given the result of a query, *precision* measures the percentage of the resulting nodes that are relevant, whereas *recall* measures the percentage of the relevant nodes that are in the result. F1 is the weighted harmonic mean of precision and recall. We used the default balanced F1 measure, which equally weights precision and recall.

In our evaluation, we used the set of nodes returned by the Saxon parser (for the XPath queries) to determine the relevance of the nodes returned by the keyword queries. Thus, for calculating precision, if the nodes returned by XRANK and SLCA were in the set of relevant nodes or in the set of their ancestors, we considered them relevant nodes. If a relevant node is descendant of, or equal to, the XRANK or SLCA nodes, we considered it for calculating the recall measure.

Table II presents precision, recall and F1 measures for each processed query using both algorithms. The values correspond to averages and standard deviations of these measures considering the results over the XMark dataset, which was composed of documents of different sizes.

Looking at the results, only very few queries present low precision. For example, queries Q9 and Q16 are very selective ("Initial and last bidder of all open auction" and "Items with description longer than 1000 characters"). Thus, when expressed in XPath, they return few nodes. On the other hand, their respective keyword versions are unable to express the same restrictions and, therefore, return

Table II. Average Measures for the Accuracy Evaluation

Query	XRANK precision	SLCA precision	XRANK recall	SLCA recall	XRANK F1 measure	SLCA F1 measure
1	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
2	0.48 ± 0.18	0.44 ± 0.17	1.00 ± 0.00	1.00 ± 0.00	0.63 ± 0.17	0.60 ± 0.17
3	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
4	1.00 ± 0.01	0.99 ± 0.01	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
5	1.00 ± 0.01	0.99 ± 0.01	1.00 ± 0.00	0.79 ± 0.03	1.00 ± 0.00	0.88 ± 0.02
6	0.79 ± 0.07	0.73 ± 0.11	1.00 ± 0.00	1.00 ± 0.00	0.88 ± 0.04	0.84 ± 0.07
7	1.00 ± 0.00	1.00 ± 0.00	0.91 ± 0.00	0.91 ± 0.00	0.95 ± 0.00	0.95 ± 0.00
8	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	0.78 ± 0.08	1.00 ± 0.00	0.87 ± 0.05
9	0.23 ± 0.08	0.23 ± 0.08	1.00 ± 0.00	1.00 ± 0.00	0.37 ± 0.11	0.37 ± 0.11
10	0.75 ± 0.23	0.73 ± 0.24	1.00 ± 0.00	1.00 ± 0.00	0.84 ± 0.16	0.82 ± 0.17
11	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
12	0.72 ± 0.25	0.67 ± 0.31	1.00 ± 0.00	1.00 ± 0.00	0.82 ± 0.18	0.76 ± 0.26
13	0.58 ± 0.37	0.55 ± 0.39	1.00 ± 0.00	1.00 ± 0.00	0.69 ± 0.27	0.66 ± 0.30
14	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
15	0.97 ± 0.03	0.97 ± 0.04	1.00 ± 0.00	1.00 ± 0.00	0.98 ± 0.02	0.98 ± 0.02
16	0.24 ± 0.01	0.24 ± 0.01	1.00 ± 0.00	1.00 ± 0.00	0.39 ± 0.02	0.39 ± 0.02
17	0.54 ± 0.15	0.54 ± 0.15	1.00 ± 0.00	1.00 ± 0.00	0.69 ± 0.14	0.69 ± 0.14
18	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
Average	0.79 ± 0.27	0.78 ± 0.28	1.00 ± 0.00	0.97 ± 0.07	0.85 ± 0.21	0.82 ± 0.21

many more nodes. Regarding recall, both algorithms returned 100% of the relevant nodes for almost all queries. This was due to the semantics of these algorithms that include in their results more nodes than those for the XPath queries.

In Table II, an impact on accuracy evaluation would be expected if we used raw keywords instead of explicit labels in queries. For example, Q14 could be "mailbox mail 1998", instead of "mailbox:: mail::1998". To address this issue, we run the SLCA algorithm with keyword queries without labels. The accuracy results of this experiment, however, showed no difference for this type of query, because the XMark dataset presents no ambiguity between labels and content.

4.2 Accuracy and Scalability on Random Queries

This section presents an accuracy study of XRANK and SLCA involving a dataset taken from DBLP, which is quite different from the XMark dataset. The DBLP dataset has many repetitive elements with no recursion, which is a key factor in XML schemas found on the Web [Laender et al. 2009]. We adapted the DBLP XML records for a stream environment by breaking it in a set of documents, where each record is an individual XML document. We then performed different queries over the DBLP dataset, considering one, two and three query terms, and different selectivity[4] ranges. All queries employed the keyword-based language presented in Section 3. The main goal of this study is to validate the accuracy of both algorithms considering a huge quantity of documents and random queries. Next, we present the experimental setup, the query definition process and the experimental results.

Experimental Setup. We generated the DBLP dataset with 100,000 XML documents taken from the DBLP XML records. Each document represents one single record, with 4KB on average, and was extracted randomly. We used the same XRANK and SLCA implementations of the Section 4.1 and the same runtime environment.

Query Definition. We analyzed the DBLP dataset and verified the kind of queries we could use. For accuracy and scalability evaluation, we considered five ranges of query selectivity. Each range has

[4]By query selectivity we mean how many XML documents are expected to be returned as a result of a query.

Table III. Selectivity Ranges for Queries Terms

Range	Initial to Final (%)	Min to Max Quant.	Terms Quant.
1	0.01 – 0.20	10 – 200	14651
2	0.21 – 0.40	201 – 400	500
3	0.41 – 0.60	401 – 600	170
4	0.61 – 0.80	601 – 800	92
5	0.81 – 1.00	801 – 1000	52

50 queries with 1, 2 and 3 query terms. The query terms were randomly determined, but all involved the form *label*::*keyword*.

In order to determine the ranges, we analyzed the selectivity of the single term queries and chose those that could involve a reasonable number of documents. Specifically, during setup time, we built inverted lists for single term queries and verified their sizes. This analysis demonstrated that the six most frequent single term queries are meaningless (with non-expressive keywords: *of*, *for*, *a*, *and*, *the*, and *in*). Moreover, most single term queries (approximately 96%) return no more than 10 documents. This same behavior was observed in the original collection of DBLP XML records.

From this analysis and considering the usual high query selectivity submitted by users, we employed selectivity ranges varying from 0.01% to 0.10%, as shown in Table III. According to the first selectivity range, each one of the 14,651 single term queries returns from 0.01% to 0.20% of the dataset documents. Likewise, each of the 52 single term queries in the fifth range returns from 0.81% to 1.00% of the documents. Consequentially, for fair comparison, every selectivity range involved only 50 randomly chosen terms.

Experimental Results. Initially, we evaluated the five selectivity ranges using five groups of 50 single term queries, one group for each range. We executed the five groups against the dataset with 100,000 randomly selected documents. Each query returned the correct set of documents, as found in the inverted list from Query Definition. This fact confirms the correct execution of both algorithms. In this evaluation, we also calculated the mean response time per each selectivity range, which were practically the same for both the XRANK and SLCA algorithms. Due to lack of space and the triviality of such results, we do not show them. We also evaluated queries with two terms for both algorithms, using the same five selectivity ranges. We generated a two-term query group for each selectivity range, resulting in five groups (each one with 50 queries). Again, both algorithms returned the correct answers. Finally, we evaluated three-term queries only for the first selectivity range (0.01% to 0.20%). For other selectivity ranges, obtaining 50 three-term queries proved unfeasible. Thus, we built a single group of 50 three-term randomly selected queries and processed them by using the XRANK and SLCA algorithms. Both algorithms returned the correct answer again, which was confirmed by matching their results to the inverted list entries.

Regarding scalability, we compared the mean response time for both algorithms when using one, two, and three-term queries. In this evaluation, we only considered the first selectivity range. Figure 2 shows these results. Note that both algorithms present practically the same mean time regardless the number of terms. This corroborates our claim that in keyword search over streams, performance time is not an issue when using both algorithms. In fact, what really matters is the result accuracy.

5. LCARANK ALGORITHM

As discussed in Section 3.3, SLCA nodes and their ancestors are within XRANK result nodes, which we call non-SLCA nodes. However, the XRANK algorithm returns SLCA and non-SLCA results without any order. Hence, our algorithm LCARank suggests a simple ranking strategy by combining XRANK and SLCA algorithms. LCARank works with two result lists, one for SLCA nodes and another one for non-SLCA nodes. When LCARank identifies a result node as SLCA, the node must stay in the

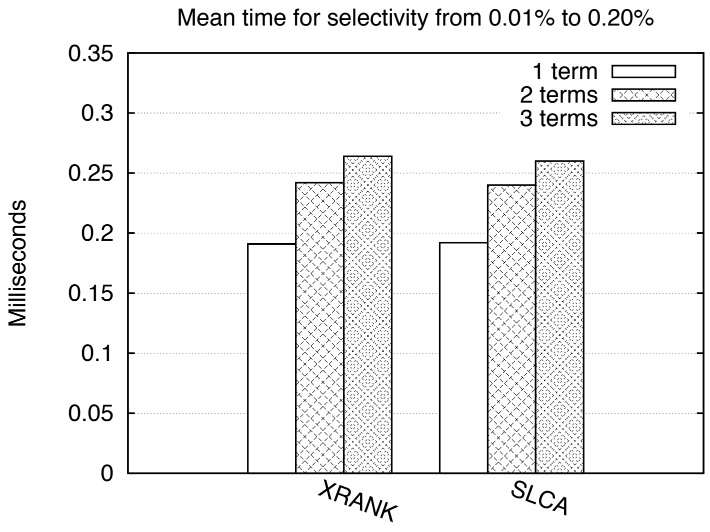

Fig. 2. Mean time for searching queries with 1, 2 and 3-term for the 0.01%–0.20% selectivity range.

Algorithm 1 LCARank - main()

```
1: {**Input**: Set of Query Terms $Q_t$}
2: {          Stream of XML documents $S_d$}
3: {**Output**: Set of result XML nodes $R_t$}
4: $R_t := \emptyset$ {initialize result set}
5: {process each document $d$ in $S_d$}
6: **for** $d \in S_d$ **do**
7:     $s.clear()$ {initialize node stack}
8:     $R_t := R_t \cup SAX\_Parse(d, Q_t, s, R_t)$ {merge new results with previous ones}
9: **end for**
10: *return* $R_t$
```

SLCA result list, otherwise it goes to the non-SLCA result list. As final result, LCARank returns the SLCA nodes first and then the non-SLCA ones. The LCARank intuition is that the algorithm now returns the smallest common lowest result elements and then their ancestors, in this order.

5.1 The Algorithm.

The general operation of LCARank is described by Algorithm 1. It evaluates a query Q_t over a stream of documents S_d. It uses internally a global node stack s (explained later) and a list R_t. Each document in S_d is sequentially processed by a SAX parser (line 8). This parser traverses an XML document tree and generates three events for each visited node, which are *startElement*(), *characters*() and *endElement*(). When the parser opens an XML node tag, it generates the event *startElement*(), which triggers Algorithm 2. Likewise, when the parser closes an XML node tag, it generates the event *endElement*(), triggering Algorithm 4. For XML text contents (enclosed by the opening and closing node tags) the SAX parser generates the *characters*() event, which triggers Algorithm 3. As an example, suppose the parser visits the leaf node $author_{\&3}$ in Figure 1. In this situation, it generates sequentially the events *startElement*(), *characters*() and *endElement*(), which then trigger respectively Algorithms 2, 3 and 4.

Algorithm 2 LCARank - startElement()

```
{Input: Accessed Document Node $n$}
{         Set of Query Terms $Q_t$}
{         Node Stack $s$}
{create new stack node, whose label is equal to the started new XML node}
$s_n.label := n.label$
{consider the current node potentially as SLCA}
$s_n.CAN\_BE\_SLCA := TRUE$
{set its query term bitmap to 0 (each bit maps a query term)}
$s_n.term\_instances := \emptyset$
{test if current node matches query terms with $l$:: or $k$}
if $\exists$ $l$:: $\in Q_t$ : $l == n.label$ then
   {current node matches query term $l$::, set query term bit to 1}
   $s_n.term\_instances(l::) := FOUND$
else if $\exists$ $k \in Q_t$ : $k == n.label$ then
   {current node matches query term $k$, set query term bit to 1}
   $s_n.term\_instances(k) := FOUND$
end if
s.push($s_n$)
```

Algorithm 3 LCARank - characters()

```
{Input: Textual Content $text$}
{         Node Stack $s$}
{process each word(token) in current text node}
for $word \in$ tokenize($text$) do
   {test if current word matches query terms with $l$::$k$ or ::$k$ or $k$}
   if $\exists$ ::$k \in Q_t$ : $k == word$ then
      {current word matches query term ::$k$, set query term bit to 1}
      $s.top().term\_instances(::k) := FOUND$
   else if $\exists$ $k \in Q_t$ : $k == word$ then
      {current word matches query term $k$, set query term bit to 1}
      $s.top().term\_instances(k) := FOUND$
   else if $\exists$ $l$::$k \in Q_t$ : ($l == s.top().label$ $AND$ $k == word$) then
      {current word matches query term $l$::$k$, set query term bit to 1}
      $s.top().term\_instances(l::k) := FOUND$
   end if
end for
```

Algorithms 2, 3 and 4 access the global node stack s, which maintains the opened nodes during the document processing. The top of the stack maintains the most recent node. Thus, each input stack corresponds to a node and has its label, its attributes and its bitmap for the query terms, whose bits individually map each user query term.

Algorithm 2 pushes the current node into the stack (line 5). It also verifies if the current node label matches a keyword with k or k:: (lines 11 to 17). In this case, the respective query term bit in its bitmap is set to 1 (line 13 or 16). For simplicity, we omitted the code that handles XML attributes, as the corresponding process is identical to visiting a leaf node. In this case, the node's label and its textual values are equivalent to the attribute name and value, respectively.

Algorithm 3 accesses the top node on the stack and verifies if its text tokens match query terms with k, ::k or l::k (lines 4 to 15). Similarly, the matched query terms have their respective bits set to

1 in the current bitmap (lines 8, 11 or 14).

Algorithm 4 LCARank - endElement()

```
1: {Input: Set of result XML nodes $R_t$}
2: {get the top node of the stack, which is being finalized}.
3: $r_n := s.pop()$
4: {test if current node matches all query terms}
5: if $r_n.term\_instances == COMPLETE$ then
6:    {test if current node adheres to SLCA node}
7:    if $s_n.CAN\_BE\_SLCA == TRUE$ then
8:       {current node matches all query terms and adheres to SLCA}
9:       {semantics, then it is included in the SLCA result list}
10:      $R_{t.slca} := R_{t.slca} \cup \{r_n\}$
11:   else
12:      {current node matches all query terms, but does not adhere to SLCA}
13:      {semantics, then it is included in the non-SLCA result list}
14:      $R_{t.no\_slca} := R_{t.no\_slca} \cup \{r_n\}$
15:   end if
16:   {since current node matches all query terms,}
17:   {its parent does not adhere to SLCA semantics}
18:   $s.top().CAN\_BE\_SLCA := FALSE$
19: else
20:   {test if current node does not adhere to SLCA node}
21:   if $s_n.CAN\_BE\_SLCA == FALSE$ then
22:      {since current node does not match all query terms and}
23:      {does not adhere to SLCA semantics, its parent also does not}
24:      {adhere to SLCA semantics}
25:      $s.top().CAN\_BE\_SLCA := FALSE$
26:   end if
27:   {process each bit in current node bitmap}
28:   {since current node does not match all query terms,}
29:   {1 bits of node are copied to the its parent node}
30:   for $tk \in r_n.term\_instances.keys$ do
31:      {test if current bit, corresponding to a query term, is true}
32:      if $r_n.term\_instances(tk) == FOUND$ then
33:         {copy current bit 1 to respective bit in parent node}
34:         $s.top().term\_instances(tk) := FOUND$
35:      end if
36:   end for
37: end if
```

Finally, Algorithm 4 finishes the current node and removes it from the node stack (line 3). If its bitmap has only $1's$, then that node satisfies all query terms and therefore is a result node, SLCA or non-SLCA (lines 5 to 19). Otherwise, its bit $1's$ are copied to the node on top of the stack, which is the parent of the popped node in the XML tree (lines 30 to 36). Algorithm 4 ranks the result nodes by using two internal lists in R_t: $R_{t.slca}$ that groups SLCA results and $R_{t.non\_slca}$ that groups other results, the non-SLCA ones (lines 7 to 15).

When Algorithm 4 finds a result node, if its SLCA flag (CAN_BE_SLCA) is set to TRUE (line 7), it is an SLCA result node (line 10). According to SLCA semantics, its parent is therefore non-SLCA. Algorithm 4 then sets the SLCA flag of its XML parent node (the next on top the stack) to FALSE (line 18). Nodes with incomplete bitmap and FALSE SLCA flag have the SLCA flags of their parent

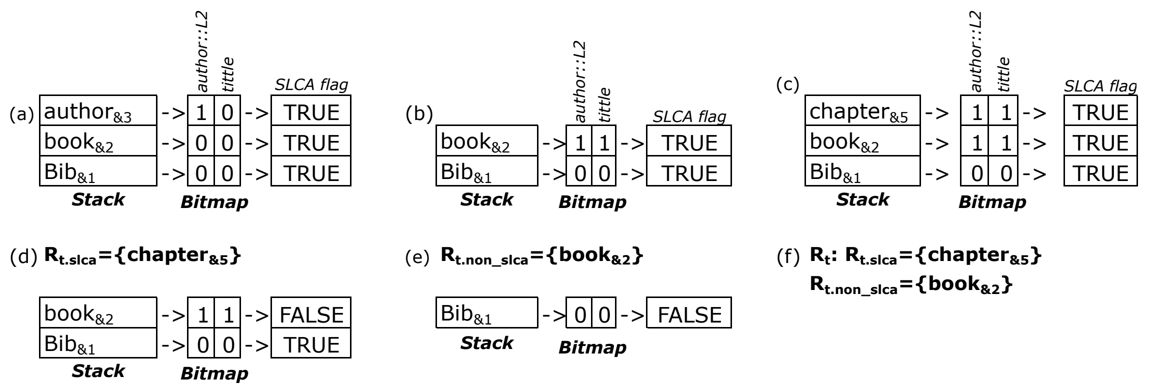

Fig. 3. LCARank algorithm example

nodes set to FALSE (lines 21 to 26). Every new visited node is potentially an SLCA node, thus Algorithm 2 always sets the new node SLCA flag to TRUE (line 9). The following example illustrates how these algorithms work.

Example 3. *We use again the query "author$::L_2$ tittle" and the XML document in Figure 1. Upon receiving the input document, the search procedure invokes the SAX parser and starts scanning the XML document. Figure 3a presents the stack contents and the bitmaps after the $Bib_{\&1}$, $book_{\&2}$ and $author_{\&3}$ start tags nodes have been visited and the textual content of author has been processed. These three nodes currently reside in the stack and have their SLCA flags as TRUE. In addition, the bit corresponding to the author element has been updated, reflecting that the token L_2 satisfies the term author::L_2. In the next step, the $author_{\&3}$ end tag is visited, so that element is popped from the stack. As its bitmap is incomplete with $1's$, the $1's$ in the bitmap are copied to their respective places in the $book_{\&2}$ node bitmap. The same happens when the node $title_{\&4}$ is popped. The bitmap and stack current configurations are illustrated in Figure 3b. At this point, node $book_{\&2}$ is a query result whose bitmap is composed only of $1's$. A similar copy of $1's$ also happens in node $chapter_{\&5}$, after node $title_{\&7}$ is popped, as shown in Figure 3c. Node $chapter_{\&5}$ is a query result as its bitmap is complete with $1's$. In the next step, node $chapter_{\&5}$ is popped and included as an SLCA result node in list $R_{t.slca}$ because its SLCA flag is TRUE. The new top, $book_{\&2}$ node, has its SLCA flag set to FALSE (Figure 3d). When this node is popped, it is included in list $R_{t.non\_slca}$ as its bitmap is complete with $1's$ and its SLCA flag is FALSE. At this point, the new top $Bib_{\&1}$ has its SLCA flag set to FALSE, (Figure 3e). As result, the query answers are $R_{t.slca} = \{chapter_{\&5}\}$ and $R_{t.non\_slca} = \{book_{\&2}\}$ (Figure 3f). The algorithm continues operating in this fashion until all tags within the document have been visited. Finally, we have $R_{t.slca} = \{chapter_{\&5}, chapter_{\&11}\}$ and $R_{t.non\_slca} = \{book_{\&2}, book_{\&8}\}$. Considering original and separated implementations of XRANK and SLCA in [Vagena and Moro 2008], the final result would be $R_t = \{chapter_{\&5}, book_{\&2}, chapter_{\&11}, book_{\&8}\}$, without ranking, and $R_t = \{chapter_{\&5}, chapter_{\&11}\}$, respectively.*

5.2 Discussion on Precision

We run the LCARank algorithm for processing the keyword queries described in Table I. As expected, we obtained the same precision, recall and F1 values shown for the XRANK algorithm. SLCA nodes involve smaller result subtrees than non-SLCA nodes and present no keyword ambiguity. Hence, they have relevance weight greater than non-SLCA ones. LCARank improves the original XRANK algorithm ranking as their internal SLCA nodes are always returned before the others, the non-SLCA.

To demonstrate this improvement, we calculated the normalized discounted cumulative gain (DCG)

Table IV. Mean Normalized DCG per file

Document Size (MB)	Mean XRANK nDCG
0.12	0.98
0.21	0.95
0.47	0.96
0.91	0.96
1.89	0.97

[Järvelin and Kekäläinen 2002] between XRANK and LCARank algorithms, considering LCARank ranking as ideal. Particularly, DCG evaluates a ranking metric for a result list, summing up the document relevance values in a result list. Each document in the result list has a relevance value based on its position in the list. The premise of DCG is that highly relevant documents appearing lower in a result list should be penalized as the relevance value is reduced logarithmic proportionally to the position of the result. DCG measure is calculated using Equation 1, where p is a particular ranking position, rel_i is the relevance weight for the document in the position p and rel_1 is the relevance weight for first document.

$$DCG_p = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2 i} \tag{1}$$

The DCG comparison for XRANK and LCARank algorithms has clear interpretation if we use the normalized DGC, which is done assuming one of these algorithms as ideal. In this case, LCARank is ideal as their smaller subtrees are returned first. For a query, the normalized discounted cumulative gain ($nDCG$) is calculated using the Equation 2, where $IDCG_p$ is LCARank DCG, given as ideal.

$$nDCG_p = \frac{DCG_p}{IDCG_p} \tag{2}$$

In our context, we adapted the DCG metric to consider retrieved nodes in a single XML document instead of a document list. For each query, we calculated its DCG based on retrieved nodes in a single XML document. As relevance weights we used 1.0 for SLCA nodes, 0.5 for non-SCLA nodes and zero for irrelevant nodes. Even if XRANK returned an SCLA node as non-SLCA, its relevance weight was set to 1.0, in order to keep the results fair.

Table IV presents the mean normalized DCG for XRANK and LCARank algorithm considering the five documents used in the first accuracy study, presented in Section 4.1. For all documents and regardless of their sizes, XRANK accuracy is inferior to LCARank accuracy, considered in this experiment as ideal, which means LCARank DCG is 1.0. Thus, we demonstrated experimentally that LCARank and its simple strategy improves the raking when compared to the original XRANK algorithm. As future work, we plan to develop a more fine-grained ranking strategy for SLCA nodes that are internally returned by the LCARank algorithm.

5.3 Discussion on Performance.

To evaluate the performance of the LCARank algorithm, we compared their mean elapsed time with that of the XRANK and SLCA algorithms. Although we used the Saxon parser to build our baseline for the first accuracy study (Section 4.1), it cannot be considered for evaluating the performance of these algorithms since they involve different semantics. Furthermore, the Saxon parser probably has been optimized for performance tuning.

Regarding performance, we evaluated the 18 keyword-based queries against the same XMark dataset and measured the mean elapsed time for each document. Figure 4 presents the respective mean elapsed

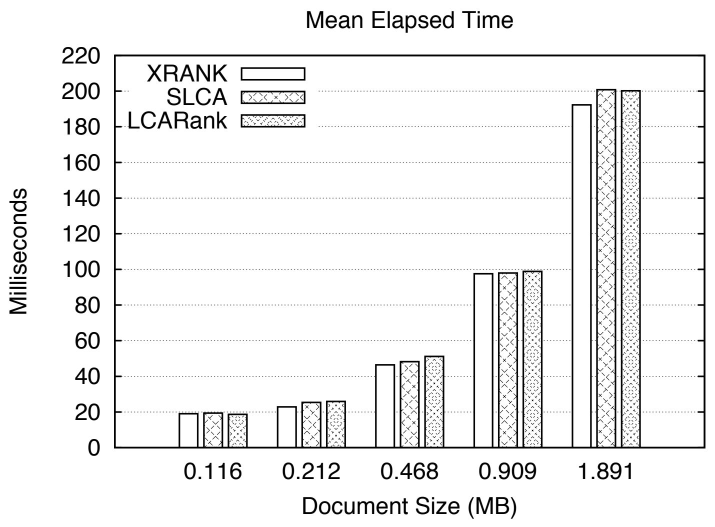

Fig. 4. Mean query time per document size

times, which involve the parsing and matching processes. The results show that XRANK, SLCA and LCARank algorithms have similar response time, however only LCARank implements a ranking strategy. This result is acceptable considering that typical XML streaming applications manipulate small documents.

6. CONCLUDING REMARKS

This paper presented an evaluation study of keyword-based search algorithms for XML streams. Specifically, it addressed two keyword search algorithms based on related node heuristics, XRANK and SLCA, and adopted a semantic keyword search language. The study consisted of an accuracy and performance evaluation for these algorithms and was divided in two parts. The first part used the queries and a dataset provided by the XPathMark benchmark, in order to fairly evaluate those algorithms in terms of precision and recall. The second part simulated a real stream environment as it considered a large collection of documents and a large set of random queries, taken from DBLP. The results showed that both algorithms are effective for implementing searching services over XML streams. The results of the study also led us to propose a simple but effective ranking algorithm (LCARank), that combines the two search algorithms. Its performance is equivalent to that of the original algorithms (do not consider any ranking strategy). For future work, we plan to deploy the LCARank algorithm in a real streaming application and develop more fine-grained ranking heuristics for XML stream environments, aiming at superior accuracy results.

REFERENCES

Baeza-Yates, R. A. and Ribeiro-Neto, B. A. *Modern Information Retrieval.* Addison-Wesley, 1999.

Bao, Z., Ling, T. W., Chen, B., and Lu, J. Effective XML Keyword Search with Relevance Oriented Ranking. In *Proceedings of IEEE International Conference on Data Engineering.* Shangai, China, pp. 517–528, 2009.

Bao, Z., Wu, H., Chen, B., and Ling, T. W. Using Semantics in XML Query Processing. In *Proceedings of International Conference on Ubiquitous Information Management and Communication.* Suwon, Korea, pp. 157–162, 2008.

Chen, Y., Davidson, S. B., and Zheng, Y. An Efficient XPath Query Processor for XML Streams. In *Proceedings of International Conference on Data Engineering.* Atlanta, USA, pp. 79–91, 2006.

Cohen, S., Mamou, J., Kanza, Y., and Sagiv, Y. XSearch: A Semantic Search Engine for XML. In *Proceedings of International Conference on Very Large Data Bases*. Berlin, Germany, pp. 45–56, 2003.

Franceschet, M. XPathMark: An XPath Benchmark for the XMark Generated Data. In *Proceedings of International XML Database Symposium*. Trondheim, Norway, pp. 129–143, 2005.

Guo, L., Shao, F., Botev, C., and Shanmugasundaram, J. XRANK: Ranked Keyword Search over XML Documents. In *Proceedings of ACM SIGMOD International Conference on Management of Data*. San Diego, California, pp. 16–27, 2003.

Järvelin, K. and Kekäläinen, J. Cumulated Gain-based Evaluation of IR Techniques. *ACM Transactions on Information Systems* 20 (4): 422–446, 2002.

Kay, M. H. Saxon 9.2. The XSLT and XQuery Processor. Website, 2010. `http://www.saxonica.com/contact.html`.

Laender, A. H., Moro, M. M., Nascimento, C., and Martins, P. An X-Ray on Web-Available XML Schemas. *ACM SIGMOD Record* 38 (1): 37–42, 2009.

Lenkov, D. Binary XML - Position paper for The W3C Workshop on Binary Interchange of XML Information Item Sets. Website, 2003. `http://www.w3.org/2003/08/binary-interchangeworkshop/31-oracle-BinaryXML_pos.htm`.

Li, J., Liu, C., Zhou, R., and Wang, W. Suggestion of Promising Result Types for XML Keyword Search. In *Proceedings of International Conference on Extending Database Technology*. Lausanne, Switzerland, pp. 561–572, 2010.

Liu, Z. and Chen, Y. Reasoning and Identifying Relevant Matches for XML Keyword Search. *Proceedings of the Very Large Database Endowment* 1 (1): 921–932, 2008.

Moro, M. M., Braganholo, V., Dorneles, C. F., Duarte, D., Galante, R., and Mello, R. S. XML: Some Papers in a Haystack. *SIGMOD Record* 38 (2): 29–34, 2009.

Onizuka, M. Processing XPath Queries with Forward and Downward Axes over XML Streams. In *Proceedings of International Conference on Extending Database Technology*. Lausanne, Switzerland, pp. 27–38, 2010.

Peng, F. and Chawathe, S. S. XSQ: A Streaming XPath Engine. *ACM Transactions Database Systems* 30 (2): 577–623, 2005.

Ramanan, P. Worst-case optimal algorithm for xpath evaluation over xml streams. *Journal of Computer and System Sciences* 75 (8): 465 – 485, 2009.

Schmidt, A., Waas, F., Kersten, M., Carey, M. J., Manolescu, I., and Busse, R. XMark: A Benchmark for XML Data Management. In *Proceedings of International Conference on Very Large Data Bases*. Hong Kong, China, pp. 974–985, 2002.

Sun, C., Chan, C.-Y., and Goenka, A. K. Multiway SLCA-based Keyword Search in XML Data. In *Proceedings of International World Wide Web Conference*. Alberta, Canada, pp. 1043–1052, 2007.

Termehchy, A. and Winslett, M. Effective, Design-Independent XML Keyword Search. In *Proceedings of ACM Conference on Information and Knowledge Management*. Hong Kong, China, pp. 107–116, 2009.

Vagena, Z., Colby, L. S., Özcan, F., Balmin, A., and Li, Q. On the Effectiveness of Flexible Querying Heuristics for XML Data. In *Procedings of International XML Database Symposium*. Viena, Austria, pp. 77–91, 2007.

Vagena, Z. and Moro, M. M. Semantic Search over XML Document Streams. In *Proceedings of International Workshop on Database Technologies for Handling XML Information on the Web*. Nantes, France, 2008.

Wilde, E. and Glushko, R. J. XML Fever. *Commun. ACM* 51 (7): 40–46, 2008.

Xu, Y. and Papakonstantinou, Y. Efficient Keyword Search for Smallest LCAs in XML Databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data*. Maryland, USA, pp. 537–538, 2005.

Xu, Y. and Papakonstantinou, Y. Efficient LCA-based Keyword Search in XML Data. In *Proceedings of International Conference on Extending Database Technology*. pp. 535–546, 2008.

Yang, W. and Shi, B. Schema-Aware Keyword Search over XML Streams. In *Proceedings of IEEE International Conference on Computer and Information Technology*. Fukushima, Japan, pp. 29–34, 2007.

Zhou, R., Liu, C., and Li, J. Fast ELCA Computation for Keyword Queries on XML Data. In *Proceedings of International Conference on Extending Database Technology*. Lausanne, Switzerland, pp. 549–560, 2010.