# Querying and Ranking XML Documents Based on Data Synopses

Weimin He[1], Teng Lv[2]
[1]Department of Computing and New Media Technologies
University of Wisconsin-Stevens Point
Stevens Point, Wisconsin 54481, USA
whe@uwsp.edu

[2]Teaching and Research Section of Computer
Artillery Academy, Hefei
Anhui, China 230031
lt0410@163.com

*Journal of Digital Information Management*

***ABSTRACT:*** *There is an increasing interest in recent years for querying and ranking XML documents. In this paper, we present a new framework for querying and ranking schema-less XML documents based on concise summaries of their structural and textual content. We introduce a novel data synopsis structure to summarize the textual content of an XML document for efficient indexing. More importantly, we extend the traditional vector space model to effectively rank XML documents over the proposed data synopses. We conduct extensive experiments over XML benchmark data to demonstrate the advantages of the indexing scheme and the effectiveness of our ranking scheme. We also compare our framework with Lucene to demonstrate our extended TF*IDF scoring function is effective.*

**Categories and Subject Descriptors**
**D.3.2 [Language Classifications]:** Data-flow languages; **H.3.1 [Content Analysis and Indexing]: I.7 [Document and Text Processing]:** Markup languages; **H.3.3 [Information Search and Retrieval]:** Query Formulation

**General Terms:** XML, Information Retrieval, Data Processing

**Keywords:** XML, Query processing, Document ranking, Query synopses, Document ranking

**Received:** 11 June 2011, Revised 12 August 2011, Accepted 19 August 2011

## 1. Introduction

As XML has become the *de facto* form for representing and exchanging data on the web, there is an increasing interest in querying and ranking text-centric XML documents. Although approximate matching with relevance ranking for plain documents has long been the focus of Information Retrieval (IR), the hierarchical nature of XML data has brought new challenges to both the IR and database communities. Extensive work has been motivated on developing efficient indexing and query evaluation algorithms, and proposing effective ranking schemes over XML data [1], [2], [3], [4], [15], [16], [17].

In this paper, we present a new framework for efficient querying and ranking schema-less XML documents based on condensed summaries extracted from the structural and textual content of the documents. Instead of indexing each single element or term in a document, we extract a structural summary and a small number of data synopses from the document, which are indexed in a way suitable for query evaluation. The result of the query evaluation is a ranked list of XML documents that best match the query. In order to rank XML documents, we extend the traditional vector space model and develop an effective ranking scheme based on *TF*IDF* scoring. We believe that our framework can serve as an infrastructure for a wide range of web applications, such as online interactive exploration of massive XML data stores and XML search engines in peer-to-peer environment.

In summary, we address the following issues in this paper:

- We present a novel framework for indexing, querying, and ranking XML documents based on content and structure synopses.
- We propose an efficient XML meta-data indexing scheme that is suitable for full-text XML query evaluation.
- We extend the traditional vector space model to effectively rank XML documents based on meta-data.
- We experimentally validate the advantages of our indexing scheme and the effectiveness of our ranking scheme.

## 2. Meta-Data Indexing

As our query language, we extend the XPath syntax with a full-text search predicate $e \sim S$, where $e$ is an arbitrary XPath expression. This predicate returns true if at least one element from the sequence returned by $e$ matches the *search specification*, $S$. A search specification is a simple IR-style boolean keyword search that takes the form

$$\text{“term”} \mid S_1 \text{ and } S_2 \mid S_1 \text{ or } S_2 \mid (S)$$

where $S$, $S_1$, and $S_2$ are search specifications. A term is an indexed term that must be present in the text of an element returned by the expression $e$. As a running example used throughout the paper, the following query, $Q$:

```
//auction//item[location ~ "Dallas"]
[description ~ "mountain" and "bicycle"]/price
```

searches for the prices of all auction items located in Dallas that contain the words “mountain” and “bicycle” in their description.

An example XML document is shown in Figure 1. The document represents the auction information from an auction web site. When an XML document is indexed, only a structural summary and a small number of concise data synopses are extracted and indexed.

```
<auction>
  <sponsor>
    <name> ebay </name>
    <address> 1040 W. Abram Dr. San Jose, CA </address>
  </sponsor>
  <item id="01">
    <name> bicycle </name>
    <description> a mountain bicycle used for 2 years </description>
    <payment> Credit Card, Cash, Money Order </payment>
    <location> Dallas, TX </location>
    <price> 30 </price>
  </item>
  <item id="02">
    <name> bicycle </name>
    <description> a brand new 24-inch bicycle for girls </description>
    <payment> Credit Card, Cash, Check </payment>
    <location> Stevens Point, WI </location>
    <price> 60 </price>
  </item>
  <item id="03">
    <name> car </name>
    <description> 1999 Toyota Camery LE, mileage 80K </description>
    <payment> Credit Card, Check </payment>
    <location> Arlington, TX </location>
    <price> 5000 </price>
  </item>
</auction>
```

Figure 1. Example XML Document

### 2.1 Structural Summary

To match the structural components in the query during the query evaluation, we construct a type of data synopses, called **Structural Summary**(SS), which is a structural markup that captures all the unique paths in the document. The structural summary of the example XML document is shown in Figure 2. Each node in a structural summary has a tagname and a unique id.

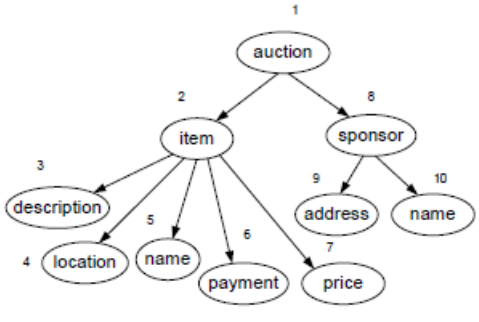



Figure 2. Structural Summary of Example XML Document

### 2.2 Content Synopses

To capture the textual content of a document, for each text node $k$ in the structural summary $S$ of the document $D$ (an $SS$ node that corresponds to at least one document element that contains text), we construct a *content synopsis* (*CS*) $H_p^D$ to summarize the textual data associated with $k$, where the path p is the unique simple path from the root of $S$ to the node $k$ in $S$. $H_p^D$ is a bit matrix of size $L \times W$, where $W$ is the number of term buckets and $L$ is the number of positional ranges in the document. Here, $W$ is proportional to the number of elements in $D$ reachable by the text label path p. $L$ is proportional to the number of begin/end tags in the document. The positional information is represented by the document order of the begin/end tags of the elements. More specifically, for each term t directly inside an element associated with the node $k$ whose begin/end position is $b/e$, we set all matrix values $H_p^D[i, \text{hash}(t)$ mod$W]$ to one, for all $\lfloor b \times L/|D| \rfloor \le i \le \lfloor e \times L//|D| \rfloor$, where 'hash' is a string hashing function and $|D|$ is the document size. That is, the $[0, |D|]$ range of tag positions in the document is compressed into the range $[0, L]$. $H_p^D$ is implemented as a $B^+$-tree index with index key $p$, because, during the query processing, we need to retrieve the content synopses of all documents for a given path $p$. For example, the content synopsis for the $SS$ node **description** ($k = 3$) is illustrated on the right of Figure 3.

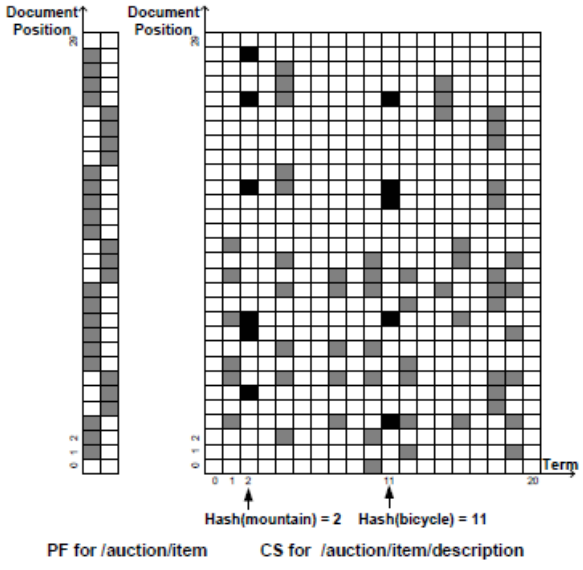



Figure 3. Data Synopses Example

### 2.3 Positional Filters

For each non-text node $n$ in the structural summary of a document, we construct another type of data synopsis, called *Positional Filter* (*PF*), denoted by $F_p^D$. As we did for $H_p^D$, $F_p^D$ is also implemented as a B-tree with index key p. $F_p^D$ is a bit matrix of size $L \times M$, where $L$ is the document positional ranges of the elements associated with node $n$ that is reachable by the label path p, and $M$ is the number of bit vectors in $F_p^D$. The positional filter for $SS$ node **item** is demonstrated on the left in Figure 3. The 7 one-bit ranges indicate there are 7 **item** elements in the document.

### 2.4 Containment Filtering

Using the above data synopses, we can enforce the element containment constraints in the query using an operation called *Containment Filtering*. Let $F$ be a positional filter of size $L \times M$ and $V$ be a bit vector extracted from a content synopsis whose size is $L \times W$. The *Containment Filtering* $CF(F, V)$ returns a new positional filter $F'$. The bit $F'[i, m]$ is on iff:

$$\exists k \in [0, L] : V[k] = 1 \wedge \forall j \in [i, k] : F[j, m] = 1$$

Basically, the *Containment Filtering* copies a continuous range of one-bits from $F$ to $F'$ if there is at least one position within this range in which the corresponding bit in $V$ is one. Figure 4 shows how the data synopses are used to determine whether a document is likely to satisfy the query $Q$ (here $M = 2$). First, we do a containment filtering between the initial positional filter $F_2$ and the bit vector $H_4$["Dallas"]. In the resulting positional filter A, only 5 one-bit ranges out of 7 in $F2$ are left. Counting from bottom to top, the 2nd and 4th one-bit ranges in $F2$ are discarded in A because there is no any onebit range in $H_4$["Dallas"] that intersects with the $2nd$ or $4th$ range, which means that neither the $2nd$ nor $4th$ **item** element contains a **location** element that contains the term "Dallas". Similarly, we can do containment filtering between A and the resulting bit vector derived from the

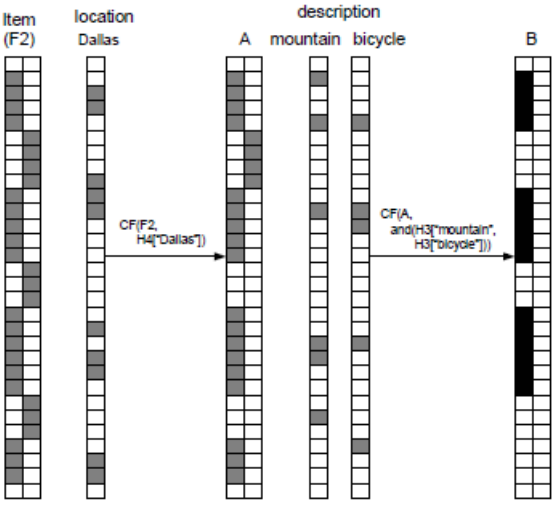



Figure 4. Testing query Q Using Data Synopses

bitwise *anding* between $H_3$["mountain"] and $H_3$["bicycle"]. The 3 one-bit ranges in $B$ indicate that 3 items out of 7 in $F_2$ satisfy all element containment constraints in the query. Thus, the document is considered to satisfy the query.

## 3. Query Processing

Due to the the space limitation, we only briefly describe the query processing in our framework in this section. As a XPath query is evaluated, a query footprint is derived from the query. A query footprint($QF$) captures the essential structural components and all the *entry points* associated with the search predicates. For example, the query footprint of Q is:

```
//auction//item:1[location:2]
                    [description:3]/price
```

The numbers 1, 2, and 3 are the numbers of entry points in $QF$ that indicate the places where data synopses are needed for query evaluation.

Based on the query footprint, all the matching structural summaries and distinct label path combinations corresponding to the query search predicates are retrieved. Based on the retrieved label path combinations, documents that have data synopses associated with those label paths are found and qualified documents are filtered out using containment filtering. The scores of documents are also calculated during the query evaluation and the ranked document list is returned to the client.

## 4. Extending Vector Space Model For Ranking

Since a query footprint may match a large number of structural summaries and there may be a large number of documents that match each structural summary, it is desirable to rank all the qualified documents using a scoring function.

### 4.1 Extended Vector Space Model

In the traditional vector space model used in IR systems, the *TF*IDF* ranking for keyword queries is defined over a document collection [19]. This ranking mechanism considers two factors: (1) $TF$, the term frequency, that measures the relative importance of a query keyword in the candidate document, and (2) $IDF$, the inverse document frequency, that measures the global importance of a query keyword in the entire collection of documents. We extend the vector space model to measure the content similarity of an XML document $D$ to a query $Q$. Due to the inherent hierarchical structure of XML data, instead of calculating the weight of a term, we consider a path-term pair as the unit for content scoring.

*1) TF*IDF Calculation*: We first give the definitions of $TF$ and $IDF$ scores of a path-term pair, and then use an example to illustrate their calculations.

**Definition 1: TF Score of a Path-Term Pair**. Let $D$ be an XML document associated with the pair $(p, t)$, where p is a full text label path from its structural summary and t is a term. Let *paths*($D$) be the set of tuples $(t_x, p_x, b_x, e_x, i_x)$ for all terms $t_x$ in $D$, where $b_x/e_x$ is the begin/end position of the element that directly contains $t_x$, $p_x$ is a full label path that reaches $t_x$, and $i_x$ is the document position of $t_x$ in $D$. The $TF$ score of $(p, t)$ relevant to $D$ is defined as:

$$TF^D(p, t) = |\{i_x | (t_x, p_x, b_x, e_x, i_x) \in paths(D) \wedge p = p_x \wedge t = t_x\}| \quad (1)$$

Basically, $TF^D(p, t)$ counts the number of $(p, t)$ pairs in the document $D$. Notice that, if t occurs n times in the same element reachable by $p$ in $D$, then $(p, t)$ will be counted $n$ times.

**Definition 2: IDF Score of a Path-Term Pair**. Let $N$ be the total number of documents in the corpus. Let $D_j$ $1 \le j \le N$, be an XML document associated with the pair $(p, t)$, where $p$ is a full text label path derived from structural summary matching and $t$ is the corresponding term in the query. The $IDF$ score of $(p, t)$ is defined as:

$$IDF(p,t) = log\left(\frac{N_p}{N(p,t)}\right) \quad (2)$$

where $N_p$ and $N(p, t)$ are calculated as follows:

$$N_p = \sum_{j=1}^{N} \left| \{ j \,|\, (t_x, p_x, b_x, e_x, i_x) \in paths(D_j) \wedge p = p_x \} \right| \quad (3)$$

$$N(p,t) = \sum_{j=1}^{N} \left| \{ (t_x, p_x) \,|\, (t_x, p_x, b_x, e_x, i_x) \in paths(D_j) \wedge p = p_x \wedge t = t_x \} \right| \quad (4)$$

Basically, $N_p$ counts the total number of documents that contain path $p$ and $N(p, t)$ counts the total number of documents that contain the path-term pair $(p, t)$ in the corpus.

We now walk through an example to illustrate $TF$ and $IDF$ score calculation. Suppose we have three documents in the corpus only, $D_1$, $D_2$, and $D_3$. The path-term pair $(p, t)$ is (/auction/item/location,"Dallas"). Suppose $D_1$ contains 30 paths /auction/item/location that can reach the term "Dallas" and $D_1$ contains 100 paths /auction/item/description that can reach the term "bicycle". Suppose that all the three documents contain the path /auction/item/location, but only the paths /auction/item/location in $D_1$ and $D_2$ can reach the term "Dallas". Then the $TF$ and $IDF$ scores can be calculated as follows:

$TF^{D_1}$(/auction/item/location,"Dallas")=30

$IDF$(/auction/item/location,"Dallas")=log(3/2)=log1.5

### 4.2 Enhanced Scoring with Positional Weight

A path-term pair $(p,t)$ corresponds to a positional bit vector $V$ in the content synopsis associated with $p$. A one-bit range in $V$ represents an element that contains $t$ and is reachable by $p$. For instance, in Figure 4, the bit vector $H_3$["mountain"] corresponds to the pair (/auction/item/description,"mountain") and it contains 5 bit-on ranges. The number of one-bit ranges in the vector reflects the $TF$ score of the pair (/auction/ item/ description,"mountain"). However, after the bitwise *anding*

operation, only 3 one-bit ranges are left in the resulting bit vector, which indicates that among those 5 **description** elements, only 3 of them contain both "mountain" and "bicycle". Similarly, after the containment filtering between the positional filter of item($F_2$) and $H_4$["Dallas"], only 5 **item** elements are left out of 7 in the resulting positional filter($A$). Thus, it is desirable to take into account this percentage of qualified elements as we calculate the weight of a path-term pair. To make the weight calculation of a path-term pair more accurate, we introduce the *positional weight*, which is the percentage of qualified path-term pairs found during the containment filtering or bitwise *anding* operation, and is used to calculate the weight of the path-term pair.

**Definition 3: Positional Weight of a Path-Term Pair.** Let $D$ be an XML document, $PF_0^D(p, t)$ be the positional filter associated with $(p, t)$, and $PF^D(p, t)$ be the result from containment filtering or bitwise anding operation. In addition, let $N_{PF_0^D(p,t)}$ be the number of one-bit ranges in $PF_0^D(p, t)$ and $PF^D(p, t)$ be the number of one-bit ranges in $PF^D(p, t)$. The positional weight of $(p, t)$ in $D$ is defined as:

$$PW^D(p,t) = \frac{N_{PF^D(p,t)}}{N_{PF_0^D(p,t)}} \quad (5)$$

To illustrate the calculation of the positional weight of a pathterm pair, we refer to our running example. In Figure 4, before the containment filtering, there are 7 one-bit ranges in $F_2$. After the containment filtering, 5 one-bit ranges remain in A. Similarly, before the bitwise anding operation, there are 5 and 4 one-bit ranges in $H_3$["mountain"] and $H_3$["bicycle"] respectively. After this operation, 3 one-bit ranges are left. Thus, the positional weight of each path-term pair is calculated as follows:

$PW^D$(/auction/item/location,"Dallas")=5/7=0.71
$PW^D$ (/auction/item/description,"mountain")=3/5=0.6
$PW^D$ (/auction/item/description,"bicycle")=3/4=0.75

Combining the $TF$ score, $IDF$ score, and the positional weight, the definition of the weight of $(p,t)$ in $D$ is determined by the following equation:

$$W^D(p,t) = PW^D(p,t) \times TF^D(p,t) \times IDF(p,t) \quad (6)$$

Finally, we give the definition of the enhanced content score *of D relevant to Q.*

**Definition 4: Enhanced Content Score of a Document.** Let $Q$ be the query and $D$ be an XML document. Let $W_i^Q(p_i^Q, t_i^Q)$ be the weight of the path-term pair $(p_i^Q, t_i^Q)$ in $Q$ and $W_i^D(p_i^D, t_i^D)$ be the weight of the corresponding pathterm pair $(p_i^D, t_i^D)$ in the document $D$. The content score of $D$ relevant to $Q$ is defined as

$$ECS(D,Q) = \frac{\sum_{i=1}^{n} W_i^Q(p_i^Q,t_i^Q)^2 \times W_i^D(p_i^D,t_i^D)^2}{\sqrt{\sum_{i=1}^{n} W_i^Q(p_i^Q,t_i^Q)^2} \times \sqrt{\sum_{i=1}^{n} W^D(p_i^D,t_i^D)_i^2}} \quad (7)$$

where $n$ is the number of path-term pairs.

In order to calculate the $TF * IDF$ score of a pathterm pair, we construct two additional synopses named **TF Synopsis** and **IDF Synopsis**. When a document $D$ is indexed, for each text label path $p$ in the structural summary of $D$, a *TF synopsis* is constructed to summarize the frequencies of all the terms associated with $p$ in $D$. More specifically, a *TF synopsis* is a hash table that maps a term to a pair consisting of **DistinctTerms** that indicates the number of distinct terms in $D$ reachable by $p$, and **Frequencies** that counts the total number of term occurrences reachable by $p$ in $D$. In our implementation, a *TF synopsis* is attached to the content synopsis associated with $p$. To obtain the $TF$ score of the path-term pair $(p, t)$ during query evaluation, path $p$ is used as the key to retrieve the corresponding content synopsis. Then term $t$ is hashed into some bucket over the attached *TF synopsis*, using the value of **Frequencies/DistinctTerms** as the term frequency of $(p, t)$. To achieve IDF score of a pathterm pair, for each text label path $p$, we construct an *IDF synopsis*, which contains two members named **Documents** and **Synopsis**, where **Documents** counts the total number of documents containing the path $p$, and **Synopsis** is a hash table that maps the term $t$ to the total number of documents containing $(p, t)$.

## 5. Experiments

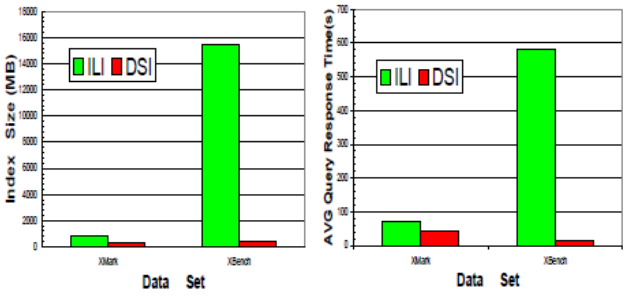
We have build a prototype system named XQR using Java (J2SE 6.0) and Berkeley DB Java Edition 3.2.13 [13] was employed as the storage manager. We conducted extensive experiments to evaluate the efficiency of our indexing scheme and the effectiveness of our ranking scheme. We used two XML benchmark datasets XMark [22] and XBench [21] as our datasets. The main characteristics of our datasets and data synopses size are summarized in Table 1. For each dataset, our query workload is 10 full-text XPath queries exhibiting different sizes, query structures and search predicates.

| Data Set | Data Size (MB) | Files | Avg. File Size (KB) | Avg. SS Size (KB) | Avg. CS Size (KB) | Avg. PF Size (KB) |
|---|---|---|---|---|---|---|
| XMark1 | 5.63 | 1150 | 5 | 0.408 | 0.292 | 0.015 |
| XMark2 | 55.8 | 11500 | 5 | 0.413 | 0.305 | 0.016 |
| XBench1 | 95.2 | 266 | 358 | 0.419 | 0.950 | 0.095 |
| XBench2 | 1050 | 2666 | 394 | 0.427 | 2.012 | 0.174 |

Table 1. Data set characteristics and data synopses size

### 5.1 Evaluation of Indexing Scheme

To demonstrate the efficiency of our Data Synopses Indexing( DSI) scheme, we implemented the traditional Inverted List Indexing(ILI) scheme in Berkeley DB and compared it with our indexing scheme. We chose XMark2 and XBench2 as datasets for this experiment because the dataset size is the key factor for the indexing scheme comparison experiment. Figure 5 shows that for XMark dataset, ILI consumes over 2 times disk space and query response time than DSI. For XBench dataset, DSI is much more efficient than ILI because XBench data is text-centric data generated from Text-CentricMultiple Document(TC/MD) class, which is more suitable for the index comparison experiments. In fact, the index size of DSI is about 3% of that of ILI. The query response time of DSI is over 40 times faster than ILI.



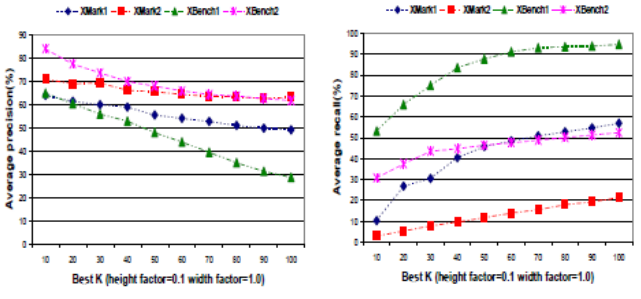

(a) Index Size (b) Query Response Time

Figure 5. Comparison between ILI and DSI

### 5.2 Evaluation of Ranking Scheme

To examine the effectiveness of our ranking function, we used two widely accepted metrics, precision and recall. We evaluated all the XMark(XBench) queries over each XMark(XBench) dataset to measure the average precision and recall. To construct the accurate relevant set for each query, we exploited Qizx/open [20] to evaluate the query over each dataset to obtain the strict relevant set.

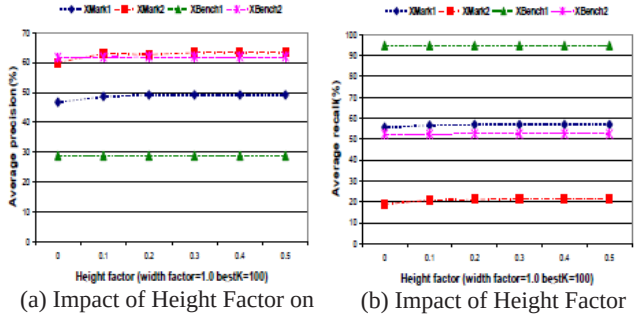
*1) Varying bestK value:* We first fixed the height and width of data synopses and varied the bestK value, i.e., the number of documents returned, to measure the average precision and recall of a query. Note that the meaning of bestK value is different from that in the classical topK algorithms, such as Fagin algorithm [18]. The bestK value here is just the number of returned documents for relevance ranking. The results in Figure 6(a) show that as the bestK value varies from 10 to 100, the average precision of a query over each dataset drops smoothly, which indicates that our scoring function can effectively rank the relevant documents on the top of the ranked list so that the precision does not drop too much when the number of returned documents increases. Since the average size of the relevant set for a query over each XBench dataset is relatively small, the average precision of a query over each XBench dataset is a little lower. Figure 6(b) shows the recall over XMark1 is greater than that over XMark2 and the recall over XBench1 is larger than that over XBench2. The reason is that when the number of documents increases, the size of relevant set increases, which leads to a lower recall.




(a) Average Precision (b) Average Recall

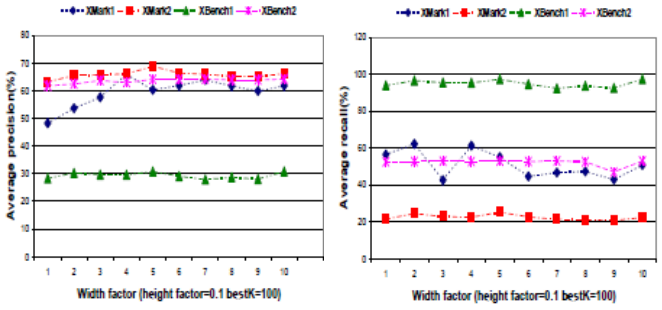
Figure 6. Varying BestK Value

*2) Impact of Height Factor:* In the second set of experiments, we fixed the bestK value and the width of data synopses to see the impact of different height factors on precision and recall. As we can see from Figure 7(a) and Figure 7(b), as the height factor varies from 0.1 to 0.5, the precision and recall almost remain at the same value for each dataset. This was expected because when the height of data synopses is reduced, a query may get more false positives, but our ranking function can effectively rank the most relevant documents close to the top, while moving false positives near the bottom of the answer set so that the precision and recall almost do not change.




(a) Impact of Height Factor on Precision (b) Impact of Height Factor on Recall

Figure 7. Impact of Height Factor




(a) Impact of Width Factor on Precision (b) Impact of Width Factor on Recall

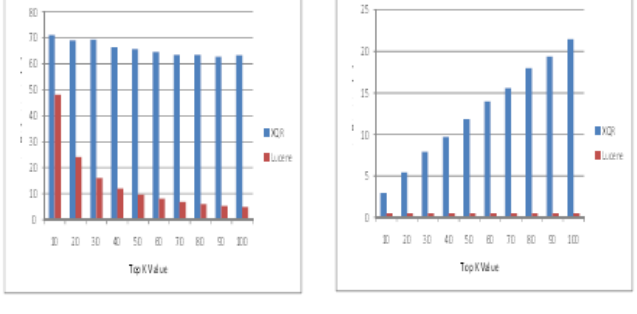
Figure 8. Impact of Width Factor

*3) Impact of Width Factor:* Finally, we fixed the size of the bestK value and the height of data synopses to see the impact of different width factors on precision and recall. The results are shown in Figure 8(a) and Figure 8(b). As we can see, the precision and recall change a little more than those in Figure 7(a) and Figure 7(b). This result implies that if we want to decrease the size of data synopses to reduce the data storage overhead but still keep high precision, the height factor should be adjusted first.

### 5.3 Experimental Comparison with Lucene

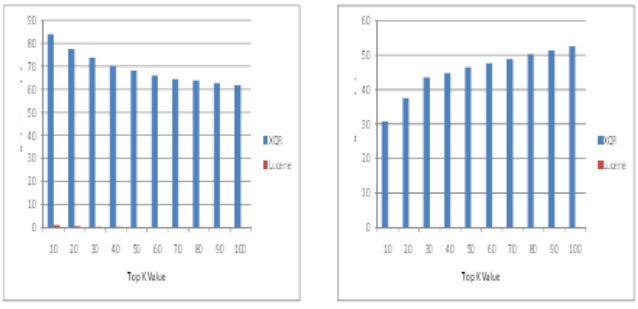
We also compared our framework with **Lucene** [14], which is a full-featured text search engine library written entirely in Java. **Lucene** is the closest to our framework because it returns ranked XML document hits as query answers and it supports boolean queries that may contain **field-term** pairs such as **title:" XML" AND author:"Suciu"**. We conducted the comparative experiments over XMark2 and XBench2 datasets. Since **Lucene** does not directly support XPath queries, we converted the XPath queries over two data sets to the corresponding queries that are supported in **Lucene**.

We first compared our ranking scheme with **Lucene** over XMark data. We varied the top K value and measured the average precision and recall. The result is shown in Figure 9. Figure 9(a) shows that for the same top K value, the average precision of **XQR** is much higher than that of **Lucene**. In addition, the average precision of **Lucene** drops rapidly when the top K value is increased because more false positives are returned in the answer set. On the contrary, the average precision of XQR does not decrease sharply when the top K value is increased because our ranking scheme can effectively rank the relevant documents on the top of the ranked list. From Figure 9(b), we can see that the average recall of **Lucene** is not only much lower than that of **XQR**, but also is lower than 2%. The reason is that the size of a relevant set is relatively larger for XMark data and there are more false positives in the answer set of **Lucene**, which leads to a very low average recall.




(a) Average Precision (b) Average Recall

Figure 9. Ranking Comparison with Lucene over XMark Data




(a) Average Precision (b) Average Recall

Figure 10. Ranking Comparison with Lucene over XBench Data

We then compared our ranking scheme with **Lucene** over XBench data. We varied the top K value and measured the average precision and recall. The result is shown in Figure 10. From Figure 10, we can see that both average precision and recall of **Lucene** are much lower than those of **XQR** and are close to 0. The is because the queries over XBench data are more selective and **Lucene** returns very few relevant document hits in the top K answer set for most queries, thus leads to close-to-zero average precision and recall.

## 6. Related Work

With the popularity of XML as the data format for a wide variety of web data repositories, extensive work has been motivated on designing powerful query languages, developing efficient indexing and query evaluation algorithms, and proposing effective ranking schemes over XML data [1], [2], [3], [15], [17].

The related work can be classified into three categories. Approaches in the first category are focused on extending existing complex XML query languages, such as XQuery, with IR search predicates and incorporating simple IR scoring methods into the query evaluation. Khalifa *et al* [1] proposes bulk algebra called TIX, which integrates simple IR scoring schemes into a traditional pipelined query evaluator for an XML database. More specifically, new operators and efficient access methods are proposed to generate and manipulate scores of XML fragments during query evaluation. TeXQuery [2] supports a powerful set of fully composable full-text search primitives, which can be seamlessly integrated into the XQuery language. It mostly focuses on the TeXQuery language design and the underlying formal model, but also provides a costing model [5]. Strategies in the second category are dedicated to extending traditional IR models for scoring simple XPath-like queries with full-text extension. In [6], the authors present a framework that relaxes a full-text XPath query by dropping some predicates from its closure and scoring the approximate answers using predicate penalties. They further propose novel scoring methods by extending *TF*IDF* ranking to account for both structure and content while considering query relaxations [3]. Sigurbjornsson *et al*, [7] propose a framework that decomposes the query into several pathterm pairs, evaluates these pairs separately and produces a final ranking that takes into account the scores of different sources of evidence. In the third category, efforts are made to address the problem of efficiently producing ranked results for keyword search queries over XML documents. XRank [16] extends Google-like keyword search to XML. The authors propose an algorithm for scoring XML elements that takes into account both hyperlink and containment edges. New inverted list index structures based on Dewey IDs and associated query processing algorithms are also presented. XSEarch [15] is a semantic search engine that extends simple keyword search by incorporating keyword context information into the query, i.e., each query term is a keyword-label pair instead of a single keyword. XSEarch extends the *TF*IDF* ranking scheme to rank the results. A recent work, XKSearch [8], introduces the concept of *smallest lowest common ancestors* (SLCAs) and proposes two efficient algorithms, Indexed Lookup Eager and Scan Eager, for keyword search in XML documents according to the SLCA semantics. Note that all the above proposals consider querying and ranking original XML documents, and returning XML fragments as query answers. None of them evaluates a full-text query over XML meta-data to return a ranked list of document locations to the client.

Since our querying and ranking schemes are based on XML meta-data, our work is also related to XML summarization techniques in the literature. Polyzotis et al [9] propose an XSKETCH synopsis model that exploits localized stability and value-distribution summaries to accurately capture the complex correlation patterns that exist between and across path structure and element values in the XML data graph. In [10], the authors further propose a novel class of XML synopses, termed XCLUSTERs, that provides a unified summarization framework to address the key problem of XML summarization in the context of heterogeneous value content. Similarly, [11] presents a novel data structure, the *bloom histogram*, to approximate XML path frequency distribution within a small space budget and to accurately estimate the path selectivity. A *dynamic summary* layer is used to keep exact or more detailed XML path information to accommodate frequently changed data. However, all these proposed XML synopses and summaries are mainly used for selectivity estimation, rather than for locating and ranking XML documents, as is in our framework. A recent work by Cho *et al* [12] addresses the meta-data indexing problem of efficiently identifying XML elements along each location step in an XPath query, that satisfy range constraints on the ordered meta-data. the authors develop a meta-data index structure named *full meta-data index*(FMI) by enhancing the R-tree index proposed for XPath location steps. The FMI can quickly identify XML elements that are reachable from a given element using a specified XPath axis and satisfy the meta-data range constraint.To reduce the metadata update overhead, another R-tree-based index structure named *inheritance meta-data index*(IMI) is also proposed to enable efficient lookup in the index structure, while keeping the update cost manageable.

## 7. Conclusion

We presented a framework for indexing and ranking XML documents based on concise data synopses extracted from original XML documents. We first proposed a novel meta-data indexing scheme suitable for full-text XPath query evaluation. We then extended vector space model to rank the query results over the indexed meta-data. The experiments demonstrate that our indexing scheme is much efficient than traditional invertedlist based indexing scheme. Our ranking scheme is effective in terms of precision and recall and superior than the existing XML search engine **Lucene**.

## 8. Acknowledgment

This work is supported by the Faculty & Program Development Grant from the University of Wisconsin-Stevens Point, USA, 2011.

## References


[1] Al-Khalifa, S., Yu, C., Jagadish, H. V. (2003). Querying Structured Text in an XML Database, *In*: *SIGMOD'03*.

[2] Amer-Yahia, S., Botev, C., Shanmugasundaram, S. (2004). TeXQuery: A Full-Text Search Extension to XQuery, *In*: *WWW'04*.

[3] Amer-Yahia, S., Koudas, N., Marian, A., Srivastava, D., Toman, D. (2005). Structure and Content Scoring for XML, *In*: *VLDB'05*.

[4] Amer-Yahia, S., Curtmola, E., Deutsch, A. (2006). Flexible and Efficient XML Search with Complex Full-Text Predicates, *In:* Proceedings of SIGMOD'06.

[5] Marian, A., Amer-Yahia, S., Koudas, N., Srivastava, D. (2005). Adaptive Processing of Top-K Queries in XML, *In*: *Proceedings of ICDE'05*.

[6] Amer-Yahia, S., Lakshmanan, L. V.S., Pandit, S. (2004). FleXPath: Flexible Structure and Full-Text Querying for XML, *In*: Proceedings of SIGMOD' 04.

[7] Sigurbjornsson, B., Kamps, J., Rijke, M. D. (2004). Processing Content- Oriented XPath Queries, *In*: Proceedings of CIKM'04.

[8] Xu, Y., Papakonstantinou, Y. (2005). Efficient Keyword Search for Smallest LCAs in XML Databases. SIGMOD 2005.

[9] Polyzotis, N., Garofalakis, M. (2002). Structure and Value Synopses for XML Data Graphs, *In*: Proceedings of VLDB'02.

[10] Polyzotis, N., Garofalakis, M. (2006). XCluster Synopses for Structured XML Content, *In:* Proceedings of ICDE'06.

[11] Wang, W., Jiang, H., Lu, H., Yu, J.H. (2004). Bloom Histogram: Path Selectivity Estimation for XML Data with Updates, *In:* Proceedings of VLDB'04.

[12]. Cho, S., Koudas, N., Srivastava, D. (2006). Meta-data Indexing for XPath Location Steps, *In:* Proceedings of SIGMOD'06.

[13] Berkeley DB. http://www.sleepycat.com/

[14] Lucene. http://lucene.apache.org/

[15] Cohen, S., Mamou, J., Kanza, Y., Sagiv, Y. (2003). XSEarch: A Semantic Search Engine for XML, *In: VLDB'03*.

[16] Guo, L., Shao, F., Botev, C., Shanmugasundaram, J. (2003). XRANK: Ranked Keyword Search over XML Documents, *In: SIGMOD'03*.

[17] Chen, L., Papakonstantinoul Y. (2010). Supporting Top-K Keyword Search in XML Databases, *In:* Proceedings of ICDE'10.

[18] Fagin, R., Lotem, A., Naor, M. (2003). Optimal aggregation algorithms for middleware, *In: Journal of Computer Systems and Science*, 66 (4) 614-656.

[19] Yates, R. B., Neto, B.R. (1999). Modern Information Retrieval, ACM Press.

[20] Qizx/open. http://www.axyana.com/qizxopen/

[21] XBench. http://se.uwaterloo.ca/»ddbms/projects/xbench/

[22] XMark. http://www.xml-benchmark.org/

[23] XML Path Language (XPath) 2.0. http://www.w3.org/TR/xpath20/