Effective Evaluation of XML Queries in a Visual Environment

Weimin He
Department of Computing and New Media Technologies
University of Wisconsin-Stevens Point
Stevens Point, WI, USA 54481
whe@uwsp.edu

Teng Lv
School of Information Engineering
Anhui Xinhua University
Hefei, Anhui, China 230031
lt0410@163.com

Ping Yan
School of Science
Anhui Agricultural University
Hefei, Anhui, China 230031
want2y2002@163.com

Journal of Digital Information Management

ABSTRACT: *Since its emergence in 1997, Extensible Markup Language (XML) has become a standard data format for data exchange among a variety of web applications. In this paper, we present a framework to effectively evaluate XML queries in a visual environment. We have developed a system termed VXPath, which is a visual XPath query evaluator that allows the user to evaluate various XPath queries by clicking the nodes in an expanding tree instead of typing the whole XPath queries manually. Our system GUI is built on top of Microsoft .NET Windows Forms Applications. We have implemented most XPath axes in our system, including child, descendant, self, parent, ancestor, following-sibling, preceding-sibling, predicate and so on. In order to handle large size XML documents, instead of loading the whole XML document into memory, we developed an effective algorithm to extract a concise data synopsis called structural summary from the original XML document to avoid the loading overhead for large XML documents. We evaluated our system over the data from XMark and DBLP and our system can handle large XML documents up to gigabytes.*

Subject Categories and Descriptors: [H.2.3 Languages]; Query languages: **[H.2.4 Systems]**; Query processing

General Terms: Query languages, XML, Databases

Keywords: Database, XML, XPath, Query Processing

Received: 17 October 2018, Revised 3 January 2019, Accepted 16 January 2019

Review Metrics: Review Scale: 0/6, Review Score: 4.8/6, Inter-reviewer consistency: 82.5%

DOI: 10.6025/jdim/2019/17/3/113-121

1. Introduction

Since its emergence in 1997, Extensible Markup Language (XML) has become a standard data format for data exchange among a variety of web applications. A volume number of web data sources are represented as XML documents or fragments on the Internet. A large number of web services are also encoded in XML format. XML has undoubtedly become the new standard for data representation and data exchange on the web. Therefore, querying XML data on the web has attracted much attention in the database literature.

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 primi-

tives, 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]. In [3], 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 [6]. Sigurbjornsson *et al*, [22] propose a framework that decomposes the query into several path-term pairs, evaluates these pairs separately and produces a final ranking that takes into account the scores of different sources of evidence. XRank [15] 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 [12] 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. XKSearch [23] 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.

In this paper, we propose a framework to evaluate XPath queries over XML documents in a user-friendly visual manner. we developed a prototype software system named VXPath to facilitate the efficient evaluation of XPath queries over XML data. VXPath is a visual XPath query evaluator that allows the user to evaluate an XPath query by clicking the nodes in an expanding tree instead of typing the whole XPath query by hand. Our system supports a variety of XPath axes, including child, descendant, self, parent, ancestor, following sibling, preceding-sibling, predicate and so on. In order to handle XML documents in very large size, instead of loading the whole XML document into memory, we extracted a concise data synopsis termed structural summary from the original XML document to avoid the loading overhead for large XML document. By looking at the structural summary in the graphical user interface, the user can obtain essential schema knowledge and issue an effective XPath query. This can greatly facilitate the learning and evaluation of XPath queries.

The rest of the paper is organized as follows. In section II, the algorithms for tree view binding is presented. Then we present the graphical user interface of VXPath and XPath evaluation algorithm in section III. In section IV, we illustrate an example XML document, the structural summary extracted from the document in order to handle XML documents in very large size, and the structural summary extraction algorithm. We report experimental results in section V. Finally, we conclude the paper and present the future work in section VI.

2. Binding DOM Tree to Windows Treeview

In order to display the content of an XML document in a visual manner, we leverage the DOM parser to create a DOM tree in memory and develop an algorithm to bind the DOM tree to a TreeView control in Windows Forms application. The key method for tree view binding written in C# is shown in Figure 1. The first parameter of the method represents a node in the DOM tree. The second parameter represents a node in the TreeView. If the current XML node has children nodes, the program loops through the XML nodes until the leaf is reached and add the nodes to the TreeView during the looping process. If the current node has no children nodes, it can be a text node or empty element. Either case is processed separately in the algorithm.

3. VXPath Graphical User Interface and XPath Query Evaluation

The graphical user interface (GUI) of VXPath is shown in Figure 2. The GUI of VXPath consists of four main components. On the left panel, the user can specify the type of XPath queries and choose whether to match attributes in the query. In the middle section of the left panel, the user can choose the XPath axis he/she wants to evaluate when the user clicks a node in the tree view located in the central panel. The variety of types of axis supported in the our system are listed in Table I. When the user chooses an axis and clicks a node in the tree view in the central panel, the visual query result will be displayed as a tree view located at the bottom section of the left panel. For example, if the user chooses the descendant axis and clicks the node book in the tree view of the central panel, the descendant list of node book is listed as a tree view on the left. The user can further expand a node in the result list. In the central panel, there are two tabs. One tab named XML shows the structural summary of an XML document is displayed as an expanded tree. The user can click the nodes in the tree to form an XPath query. Another tab displays the query result of an XPath query. On the top panel is the XPath query expression created from the user clicks. On the top right is a button to run the XPath query. By clicking the File menu on the top left, the user can choose the XML document he/she wants to process.

In order to avoid the memory overhead to evaluate XPath queries over large XML documents in gigabytes, we exploited the XML SAX parser named XPathReader [32] from Microsoft to read XML data in a streaming manner instead of loading the whole document as a DOM tree. The XPathReader provides the ability to filter and process large XML documents in an efficient manner using an XPath-aware XmlReader. With the XPathReader, one can sequentially process a large document and extract an identified sub-tree matched by an XPath expression [32]. Our algorithm to evaluate XPath queries is shown in Figure 3.

```
private void AddNode(XmlNode inXmlNode, TreeNode inTreeNode)
{
XmlNode xNode;
TreeNode tNode;
XmlNodeList nodeList;
int i;
//Loop through the XML nodes until the leaf is reached.
//Add the nodes to the TreeView during the looping process.
if(inXmlNode.HaschildNodes)
{
        nodeList = inXmlNode.ChildNodes;
        for(i = 0; i < = nodeList.Count-1; i++)
        {
                xNode = inXmlNode.childNodes;
                String atts ="";
                if(xNode.Attributes!=null)
        {
                //how to get attributes as elements!
                for(int j = 0; j < xNode.Attributes.Count; j++)
                {
        atts+= "@ + xNode.Attribute[j].Name+
                "=\"" + xNode.Attributes.[j].Value + "=\" ";

                }
}
                inTreeNode.Nodes.Add(newTreeNode(xNode.Name + atts));
                tNode = inTreeNode.Nodes [i];
                AddNode(xNode, tNode);
                }
}
else// no children, usually a text node, but could be an empty element
{
        String atts = "";
        if(inXmlNode.Attributes!= null)
{
          for (int j = 0; j < inXmlNode.Attributes.Count; j++)
        {
                  atts + = " @" + inXmlNode.Attributes.[j].Name+
          "=\"" +  inXmlNode.Attributes.[j].Value + "\" ";
        }
        inTreeNode.Text = (inXmlNode.Name + atts).Trim ();
        if(inXmlNode.InnerText ==" ")
           inTreeNode.Nodes.Add(" ")
}
else
{
                inTreeNode.Text = (inXmlNode.OuterXml).Trim();
                if (inXmlNode.InnerText =="")
                  inTreeNode.Nodes.Add("");
                }
        }
}
```

Figure 1. Algorithm for Tree View Binding

Types Of Xpath QueryAxis	Meaning of XPath Query Axis
Child	Selects nodes that are immedite child nodes of the Context Node
Descendant	Selects all the descendant nodes of the context node
Self	Selects the context node
parent	Select the parent Node of the context node
Ancestor	Select all the ancestor nodes of the context node
Following- Siblings	Selects the next immediate sibling node of thew the context node
Preceding-Sibling	Selects the previous immediate sibling node of the context node

Table 1. Types of Supported Xpath Axis

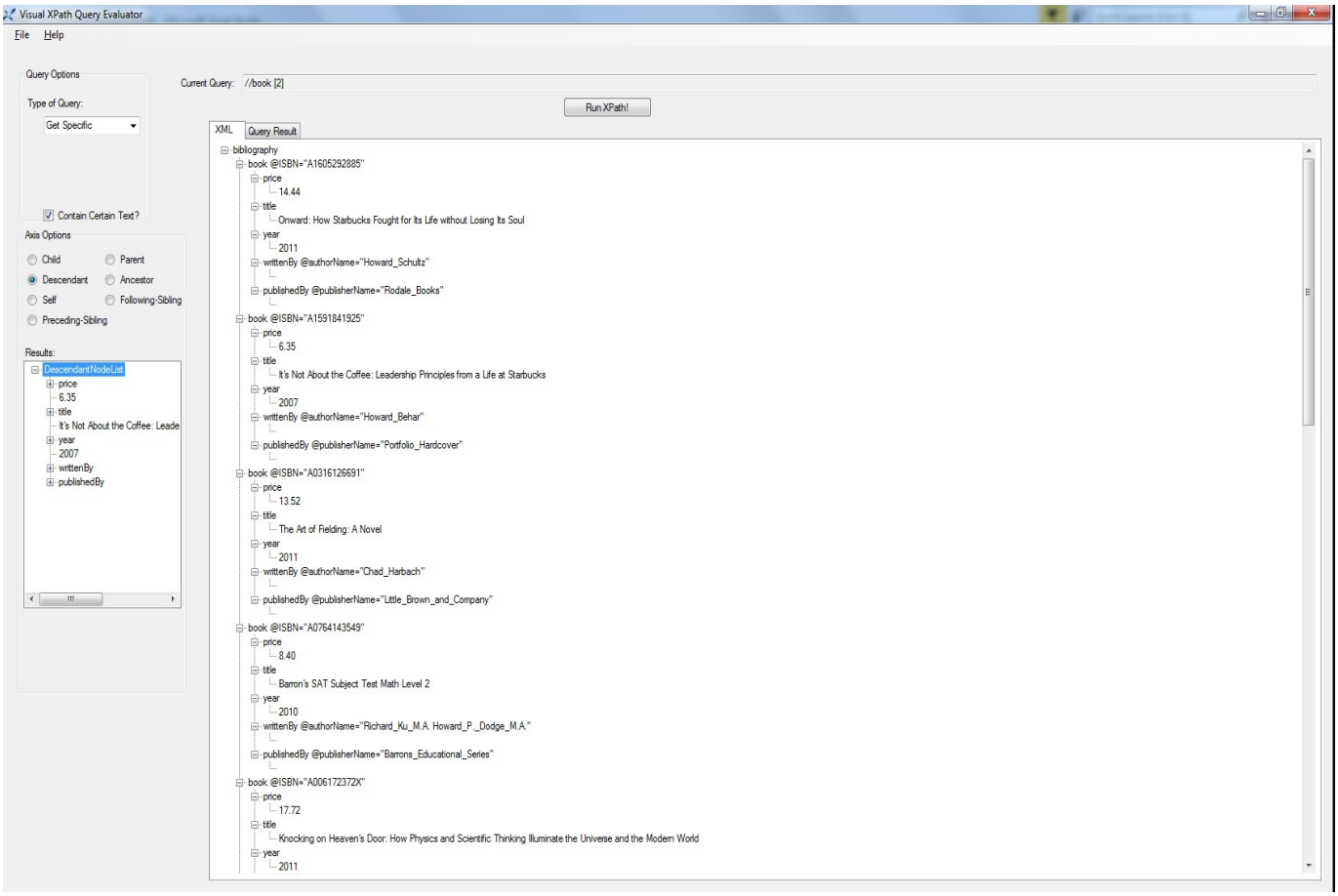

Figure 2. VXPath Graphical User Interface

4. Handling of Large Size XML Documents

To present an XML document to the user as a tree view, we exploit the DOM API in the .NET library to parse the document and construct a DOM tree in memory. However, this approach has a limitation because it can only handle smaller XML documents due to the fact that the size of a DOM tree grows very fast when the size of the XML document increases. In order to handle an XML document in large size, we extract a concise data synopsis called Structural Summary from an XML document, which is a structural markup that captures all the unique paths in the document. Instead of loading the original XML document into memory, we load and present the structural summary to the user. To achieve this goal, we leverage a third party SAX parser named SAX for .NET [31] to parse an XML document. Figure 4 gives an example XML document that represents the excerpted book information from Amazon. The structural summary of the example XML document is shown in Figure 5. Each node in an *SS* has a tagname and a unique Id. Note that one *SS* node may correspond to more than one elements in the original XML document. For example, the node book in Figure 5 corresponds to three book elements in Figure 4. We devel-

```
private String evaluateXPath(String xpath, String path)
{
if (xmlLarge)   //Handling large  xml documents
{
   FileStream strmTemp = new FileStream(path, FileMode.Open, FileAccess.Read);

        XmlTextReader xtr = new XmlTextReader(strmTemp);
        xtr.DtdProcessing = Dtdprocessing.Ignore;
        xtr.XmlResolver = null;

        XPathCollection xc = new XPathCollection();
        xc.Add(xpath);

        XpathReader xpr = new XPathReader(xtr, xc);

        return EvaluateLarge(xpr);

}
else
{
        XmlDocument doc = new XmlDocument();

        doc.Load(path);

        XPathNavigator navigator = doc.CreateNavigator();
        XPathExpression myXpath = XPathExpression.Compile(xpath);

        return Evaluate(myXPath, navigator);
        }
}
// Evaluate XPath Queries over Large XML Documents
private static String EvaluateLarge(XPathReader xpr)
{

        StringBuilder results = new StringBuilder();

    while(xpr.ReadUntilMatch())
{
        results.Append(xpr.ReadOuterXml());
}
        String fResults = "<results>" + results.ToString()+ "</results>";

        return fResults;
}
Private static String Evaluate(XPathExpression expression, XPathNavigator navigator)
{
        String results = "";
        XPathNodeIterator values = (XPath NodeIterator)navigator.Evaluate(expression);
        while(values.MoveNext())
{
        result+=(String)values.Current.OuterXml;
}
        result = "<results>" + results + "</results>";
        return results;
}
```

Figure 3. Algorithm for XPath Query Evaluation

XPath Query Expression	Query Evaluation Time (s)
//site/regions	16.66
//site/regions/samerica/item	7.58
//site/regions/samerica/item/description/parlist	3.78
//site/catgraph	3.47
//site/regions/samerica/item/mailbox	4.13

Table 2. Experimental Evaluation Over Xmark Data

XPath Query Expression	Query Evaluation Time (s)
//dblp/article	34.12
//dblp/book	9.76
//dblp/article/editor	8.15
//dblp/article/title	13.78
//dblp/book/publisher	5.32

Table 3. Experimental Evaluation Over DBLP Data

oped a novel algorithm to extract the structural summary from the document. The excepted C# code for structural summary extraction is shown in Figure 6.

5. Experimental Result

We conducted some experiments to evaluate our system over the XML data from XMark [26] and DBLP [30]. We first evaluated 5 XPath queries shown in Table 2 over the XML document from XMark. The file size is 113MB. The file parsing time is 3.65 seconds. The XPath queries and the query evaluation time are shown in Table 2. We then evaluated our system against the XML document from DBLP whose size is around 1GB. VXPath can handle this large XML document smoothly and the file parsing time is 58.42 seconds. The XPath queries and query evaluation time are shown in Table 3. As we can see, although the file size of DBLP is about 10 times of the file size of XMark, the query evaluation time does not grow linearly when the file size is larger.

```
<amazon>
<book>
        <title> Beginning XML </title>
        <year> 2003 </year>
        <authors>
                <author> Dave Gibbons </author>
                <author> David Hunter </author>
        </authors>
                <edition> 4 </edition>
                <isbn> 9780470114872 </isbn>
                </book>
<book>
        <title> Starting Out with Java: Early Objects </title>
        <year> 2007 </year>
        <authors>
```

```
                <author> Tony Gaddis </author>
        </authors>
                <edition> 3 </edition>
                <isbn> 9780321497680 </isbn>
</book>
<book>
        <title> Pro XML Development with Java Technology </title>
        <year> 2006 </year>
        <authors>
                <author> Ajay Vohra </author>
                <author> Deepak Vohra </author>
        </authors>
        <edition> 1 </edition>
        <isbn> 9781590597064 </isbn>
</book>
</amazon>
```

Figure 4. Example XML Document

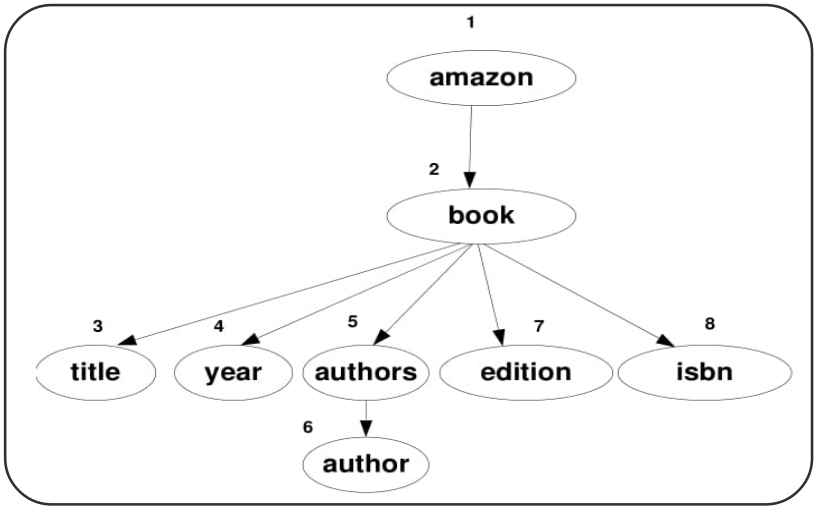

Figure 5. Structural Summary of Example XML Document

```
public class StructuralSummaryHandler : DefaultHandler {
  private int level;
  public StructuralSummary[] summary;
 int nodeId;

public StructuralSummaryHandler();
{
        level = 0;
        summary = new StructuralSummary[100];
        nodeId =0;
        summary[0] = new StructuralSummary(0, “”);
}
public override void StartElement(String Uri, StringLocalName, String qName, IAttributes atts){
Boolean found = false;
```

```
List<StructuralSummary> cs = summary[level].children;
for(int i = 0;!found && i<cs.Count; i++){
  StructuralSummary c = cs.ElementAt(i)
if(c.tagname.Equals(localName)){
        summary[++level] = c;
        found = true;
} else if (c.tagname.CompareTo(localName)>0){
        StructuralSummary current = new
        StructuralSummary(++nodeId, localName);
        cs.insert(i, current);
        summary(++level] = current
        found = true;
        }
}
if(!found){
        StructuralSummary current = new
                StructuralSummary(++nodeId, localName);
                cs.Add(current);
                summary(++level] = current;
        }
}
for( int k = 0; k < atts.getLength(); k++){
/*Proceeding current attributes in the attribute list The code here is ommitted*/
        }
}
public override void EndElement (String uri,  String localName, String qName) {
level--;
}
}
```

Figure 6. Extraction of Structural Summary

6. Conclusion and Future Work

In this paper, we proposed a new framework that can both evaluate XPath queries over XML data in a user-friendly visual manner and process XML documents in large size. Our system can support the visual evaluation of most XPath queries including forward and backward axes. This research work serves several purposes. First, The prototype system can serve as the foundation for the construction of the query engine in native XML databases because it can handle XML documents in any size. Secondly, the final completed software system can be used as a teaching/learning tool for the instructors/students in XML-related courses. The students can click around an XML file and learn how to construct an XPath expression in real time. This is much easier for the students beginning XML. As our future work, we plan to evaluate our system over some XML data sets from real world. We also plan to improve our software system to make it be able to handle XML document in larger size.

References

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

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

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

[4] Amer-Yahia, S., Curtmola, E., Deutsch, A. (2006). Flexible and Efficient XML Search with Complex Full-Text Predi

cates, *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, 2005.

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

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

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

[9] Bloom, B. (1970). Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communications of the ACM*. 13, July, p. 422-426.

[10] Grothoff , C. (2004). A Quick Introduction to Bloom Filters. *Technical Report*. Department of Computer Science, Purdue University, 2004.

[11] Bonifati, A., Matrangolo, U., Cuzzocrea, A., Jain, M. (2004). XPath Lookup Queries in P2P Networks, *In*: Proc. of WIDM 2004, p. 48-55, 2004.

[12] Cohen, S., Mamou, J., Kanza, Y., Sagiv, Y. (2003). XSEarch: A Semantic Search Engine for XML, *In*: Proc. of VLDB 2003, p. 45-56, 2003.

[13] Galanis, L., Wang, Y., Jeffery, S. R., and DeWitt, D. J. (2003). Locating Data Sources in Large Distributed Systems, *In*: Proc. of VLDB 2003, p. 874- 885, 2003.

[14] Graefe, G. (1993). Query Evaluation Techniques for Large Databases, *ACM Computing Surveys*. 25 (2) 73-170, 1993.

[15] Guo, L., Shao, F., Botev, C., Shanmugasundaram, J. (2003). XRANK: Ranked Keyword Search over XML Documents, *In*: Proc. of SIGMOD 2003, p. 16-27, 2003.

[16] Chen, L., PapakonstantinouL, Y. (2012). Supporting Top-K Keyword Search in XML Databases, *In*: Proceedings of ICDE'10, 2010.

[17] Liu, S., Zou, Q., Chu, W. (2004). Configurable indexing and ranking for XML information retrieva, *In*: Proceedings of SIGIR'04, 2004.

[18] Han, Z., Le, J., Shen, B. (2006). Effectively Scoring for XML IR Queries, *In*: Proceedings of DEXA'06, 2006.

[19] Kaushik, R., Bohannon, P., Naughton, J. F., Shenoy, P. (2002). Updates for Structure Indexes, *In*: Proceedings of VLDB'02, 2002.

[20] Liu, F., Yu, C. T., Meng, W., Chowdhury, A. (2006). Effective Keyword Search in Relational Databases, *In*: Proceedings of SIGMOD'06.

[21] Buneman, P., Choi, B., Fan, W., Hutchison, R., Mann, R., Viglas, S. (2005). Vectorizing and Querying Large XML Repositories, *In*: Proceedings of ICDE'05, 2005.

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

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

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

[25] Zhang, C., Naughton, J., DeWitt, D., Luo, Q., Lohman, G. (2001).On Supporting Containment Queries in Relational Database Management Systems, *In*: Proceedings of SIGMOD'01, 2001.

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

[27] XBench. http://se.uwaterloo.ca/'ddbms/projects/xbench/

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

[29] XQuery 1.0: An XML Query Language. http://www.w3.org/TR/xquery/

[30] DBLP. https://dblp.uni-trier.de/.

[31] SAX FOR .NET. http://saxdotnet.sourceforge.net/.

[32] XPathReader. https://www.microsoft.com/enus/ download/details.aspx?id=22677.