How XPath Query Minimization Impacts Query Processing Performance

Ki-Hoon LEE$^\mathrm{a)}$, Member

SUMMARY Considerable effort has been devoted to minimizing XPath queries under the assumption that the minimal query is faster than the original query. However, little attention has been paid to the validity of the assumption. In this paper, we provide a detailed analysis on the effectiveness of XPath query minimization and present an extensive experimental evaluation on the effectiveness using six publicly available XQuery engines. To the best of our knowledge, this is the first work done towards this objective. Experiments on real and synthetic data sets show that although the assumption is valid for some cases, the performance of the minimal query is often lower than or almost equal to that of the original query.

key words: XPath query minimization, query optimization

1. Introduction

XML has emerged as a standard for representing, storing, and exchanging data on the Internet, and XPath is widely used for querying XML documents [16], [18]–[21], [27]. In general, the efficiency of XPath query processing depends on the size of the query [1], [11], [13], [21], [24]. Thus, there has been significant research [1], [7], [8], [11], [17], [21], [24], [28] on finding a minimal query, i.e., a smallest equivalent one, for a given XPath query.

Although there has been considerable research on the minimization of XPath queries, little attention has been paid to the effectiveness of the XPath query minimization. In this paper, we answer the question: “Is the minimal query really faster than the original query?” To the best of our knowledge, this is the first work done towards this objective. We provide a detailed analysis on the effectiveness of XPath query minimization and present an extensive experimental evaluation on the effectiveness of XPath query minimization using six publicly available XQuery engines—three XML/XQuery database systems (Berkeley DB XML, BaseX, and MonetDB/XQuery) and three stand-alone (file-based) XQuery processors (Saxon-EE, Zorba, and XQilla).

The contributions of this paper are as follows:

- We perform extensive experiments using real and synthetic data sets. The results show that although the assumption that the minimal query is faster than the original query is valid for some cases, the performance of the minimal query is often lower than or almost equal to that of the original query depending on query evaluation strategy and engine used.

The rest of this paper is organized as follows. Section 2 reviews existing work. Section 3 provides an analysis on the effectiveness of XPath query minimization. Section 4 presents the experimental results. Finally, Sect. 5 presents our conclusions.

2. Related Work

Query minimization is a fundamental problem in query optimization [1], [24]. One of the first results in this area was that of Chandra and Merlin [6] who showed that the minimization problem for conjunctive relational queries is NP-Complete. Recently, much work has been done on XPath query minimization and can be characterized along two main dimensions: the class of queries being supported and the types of constraints being considered [8]. The various fragments of XPath queries explored so far can be denoted by $XP^F$ [11], where $F \subseteq \{/,//,\cdot,*,\}$ represents the set of query features supported including child axis "/", descendant axis "/*", nested predicates “[]”, and wildcards “*” [8]. Wood [28] has studied on the minimization of $XP^F$ and showed that, in the absence of constraints, the minimal query can be found in polynomial time. Amer-Yahia et al. [1] presented an $O(n^4)$ algorithm with respect to the query size $n$ for minimizing $XP^F$ in the absence of constraints (Case 1). They also considered the minimization problem in the presence of three kinds of constraints: required children, required descendants, and required co-occurrences (Case 2). They presented an $O(n^6)$ algorithm in the presence of only required children and required descendants constraints (Case 3). Ramanan [24] has proposed $O(n^5)$, $O(n^4)$, and $O(n^2)$ algorithms for Case 1, 2, and 3, respectively, based on the concept of graph simulation. Flesca et al. [11] have investigated the minimization problem in the absence of constraints for the queries in $XP^F$ whose branching nodes are not “*”-labeled. They characterized the complexity of...
the minimization problem and identified specific forms of XPath expressions that can be minimized in polynomial time. Kimelfeld and Sagiv [17] have explored the notions of redundancy, minimization and the connection between them in $XP^{(//\text{A})\ast}$ in the absence of constraints. Chen et al. [9] has studied the minimization problem for $XP^{(//\text{A})\ast}$ under forward and path constraints; Chen and Chan [8] under forward, subtype, backward, and sibling constraints; and Che [7] under forward, subtype, backward, sibling, and path constraints. Arion et al. [2] have proposed a method that minimizes queries in $XP^{(//\text{A})\ast}$ under summary constraints [2], but this method can fail to find a minimal query since it minimizes a query by merely erasing labels from the original query whereas a minimal query could include labels that are not present in the original query. Lee et al. [21] have formally analyzed the relationship between the original query and its minimal query in $XP^{(//\text{A})\ast}$ and proposed a method that guarantees finding a minimal query using the DataGuide [12] in the absence of constraints.

Although considerable effort has been devoted to XPath query minimization algorithms, little attention has been paid to the effectiveness of the XPath query minimization. Flesca et al. [11], Kimelfeld and Sagiv [17], Ramanan [24], and Wood [28] have not performed any analysis or experiments. Amer-Yahia et al. [1], Che [7], and Lee et al. [21] have only measured the time to generate a minimal query for an original query. Chen and Chan [8] have compared the performance of the minimal query and that of the original query using only one XQuery engine called GCX [26], which is not available anymore, and using only six test queries. In this paper, we perform a detailed analysis and extensive experiments on the effectiveness of the XPath query minimization using six XQuery engines and 29 queries on two real data sets (DBLP and NASA) and one synthetic data set (XMark).

3. Analysis on the Effectiveness of XPath Query Minimization

In this section, we provide an analysis on how XPath query minimization impacts query processing performance in various XQuery engines.

3.1 Berkeley DB XML

The query evaluation method of Berkeley DB XML can be classified into two categories according to the access method used: the navigation-based method and the index-based method. The navigation-based method traverses the XML document tree following the parent-child relationship between the nodes and is usually used when there is no available index. To support efficient navigation, every node contains pointers to its parent and children. If the query is too general (e.g., //A), the navigation-based method might be expensive since it traverses the whole XML document tree to find all A elements anywhere in the XML document. The index-based method evaluates a query using an index that provides fast access to a set of XML document nodes that match a tag (i.e., element name) of the query. It might be expensive if there are many nodes matching the indexed tag.

Since query minimization removes redundant nodes and branching predicates in the original query, the minimal query might be faster than the original query, but there are cases where the minimal query is slower than the original query as follows.

Case 1: Berkeley DB XML uses the navigation-based method for both the original query and the minimal query, and the original query consists of only child axis steps. In this case, the original query scale down the traversal of the XML document tree to specific paths, but the minimal query could fully traverse the tree due to descendant-or-self axis steps.

Example 1: Let the original query $QX_{1o} =$ /site/regions/europe/item and the minimal query $QX_{1m} =$ //europe/item on the XMark data set [29]. Berkeley DB XML uses the navigation-based method for $QX_{1o}$ and $QX_{1m}$. The original query $QX_{1o}$ traverses the child nodes of the item node to find the regions node, and the child nodes of the regions node to find the europe node, and so on. In contrast, the minimal query $QX_{1m}$ traverses the whole XML document tree due to the descendant-or-self axis step //europe. □

Case 2: Berkeley DB XML uses the navigation-based method for the original query and the index-based method for the minimal query, and the original query consists of only child axis steps. In this case, if there are many nodes matching the indexed tag of the minimal query, the minimal query could be slower than the original query.

Example 2: Let the original query $QD_{1o} =$ /dblp/book/author and the minimal query $QD_{1m} =$ //book/author on the DBLP data set [22]. $QD_{1m}$ looks up author nodes using an index and then examines the parent node of each to satisfy //book. $QD_{1m}$ is slower than $QD_{1o}$ since the number of author nodes is very large, but the number of author nodes satisfying //book is small.

Case 3: Berkeley DB XML uses the index-based method for the original query and the navigation-based method for the minimal query. In this case, the minimal query could be slower than the original query since it fully traverses the XML document tree in the worst case.

Example 3: Let the original query $QN_{1o} =$ //fitsFile [description/para]/footnote and the minimal query $QN_{1m} =$ //fitsFile/footnote on the NASA data set [22]. $QN_{1o}$ speeds up query evaluation by using an index on para nodes, but $QN_{1m}$ should fully traverse the XML document tree since there is no available index for it.

Case 4: Berkeley DB XML uses the index-based method for both the original query and the minimal query. In this case, there are two subcases where the minimal query could
be slower than the original query: (1) the number of nodes matching the indexed tag of the minimal query is larger than that of the original query, (2) the indexed tag of the minimal query is the same as that of the original query, and the original query consists of only child axis steps.

Example 4: Let the original query \( QD_{2m} = /\text{dblp/book\[cite\]} [\text{author}][\text{year}][\text{isbn}][\text{publisher}][\text{title}] \) and the minimal query \( QD_{2m} = /\text{dblp/book\[cite\]} \). \( QD_{2m} \) starts query evaluation by looking up isbn nodes, and \( QD_{2m} \) cite nodes. Since the number of cite nodes is much larger than that of isbn nodes, \( QD_{2m} \) is slower than \( QD_{2m} \).

Example 5: Let the original query \( QX_{2o} = /\text{site/people/ person\[name\]} \) and the minimal query \( QX_{2m} = /\text{person/\[name\]} \). \( QX_{2o} \) evaluates part of the query, /\text{site/people/person, by traversing the XML document tree, creating a temporary result of satisfying nodes, and joins the temporary result with name nodes using an index}\(^{4}\). \( QX_{2m} \) looks up name nodes using an index and then examines the parent node of each to satisfy /\text{person/\[name\]}. \( QX_{2o} \) is more efficient than \( QX_{2m} \) since \( QX_{2m} \) accesses many name nodes that do not satisfy /\text{person/\[name\]} while \( QX_{2o} \) does not.

We note that, for Case 1, 2, and 4.2, if the original query contains descendant-or-self axis steps, the performance of the original query could be lower than or almost the same as that of the minimal query.

3.2 BaseX

The query evaluation method of BaseX is very similar to that of Berkeley DB XML. However, in contrast to Berkeley DB XML, BaseX converts descendant-or-self steps in the input query to child steps if possible. As a result, the minimal query is often internally converted to the original query and thus shows almost the same performance.

3.3 MonetDB/XQuery

MonetDB/XQuery consists of the Pathfinder XQuery compiler [14] on top of the MonetDB RDBMS [4]. In MonetDB/XQuery, a k-step XPath query is compiled into a k-way self-join at the relational end [15]. Thus, the minimal query is generally faster than the original query since it has less steps than the original query.

3.4 Stand-alone XQuery Processors: Saxon-EE, Zorba, and XQilla

Stand-alone XQuery processors have to scan the whole XML document to process a query, and the scanning takes most of query processing time. Thus, the minimal query and the original query show almost the same performance.

4. Experimental Evaluation

4.1 Experimental Setup

To verify the effectiveness of XPath query minimization, we compare the execution time \( T_{\text{org}} \) of the original query and the execution time \( T_{\text{min}} \) of the minimal query. Each individual experiment is repeated \( n + 1 \) times, and the execution time of the first run is neglected (“warm-up”). We use \( n = 10 \), i.e., execution times are averaged over 10 runs. We use the wall clock time as the measure.

We have performed experiments using six publicly available XQuery engines in Table 1. The first three are XML/XQuery database systems, and the last three are stand-alone (file-based) XQuery processors.

We have performed experiments using two real data sets and one synthetic data set: the real DBLP data set [22], the real NASA data set [22], and the synthetic XMark benchmark data set [29]. Table 2 shows statistics of these data sets.

We have used the queries in Tables 3 ∼ 5 for the data sets in Table 2. The queries were used in the experiments of Chen and Chan [8] and Lee et al. [21]. The query \( QD_{im} (Q_{N_{im}}, Q_{X_{im}}) \) is the minimal query of the original query \( QD_{io} (Q_{N_{io}}, Q_{X_{io}}) \).

Our experimentation platform is a 3.0 GHz AMD Athlon II X2 250 (2 MB L2 cache) dual-core processor with 4 GB of main memory and a Western Digital Caviar Black WD1002FAEX disk (1 TB, 7200 RPM, 64 MB Cache). The operating system is Microsoft Windows 7 64 bit.

4.2 Experimental Results

In this section, we experimentally verify the analysis presented in Sect. 3 for each XQuery engines.

4.2.1 Berkeley DB XML

Table 6 summarizes the performance comparison between the minimal query and the original query. \( T_{\text{org}}/T_{\text{min}} > 1.2 \) means that the minimal query is over 1.2 times faster than the original query. \( T_{\text{min}}/T_{\text{org}} > 1.2 \) means that the original query is over 1.2 times faster than the minimal query.

\(^{4}\)Certain join methods (e.g., indexed nested-loop join) do not require temporary results.
$T_{\min}$ $\approx T_{\org}$ denotes $T_{\org}/T_{\min} \leq 1.2 \land T_{\min}/T_{\org} \leq 1.2$ and means that the minimal query and the original query show almost the same performance. The number in a parenthesis is $T_{\org}/T_{\min}$ or $T_{\min}/T_{\org}$ for the query. For example, $Q_{N9}$ (4.60) means that $T_{\org}/T_{\min} = 4.60$ since $Q_{N9}$ belongs to the case where $T_{\org}/T_{\min} > 1.2$. $Q_{X1}$ (10.79, Case 1) means that (1) $T_{\min}/T_{\org} = 10.79$ since $Q_{X1}$ belongs to the case where $T_{\min}/T_{\org} > 1.2$, and (2) $Q_{X1}$ belongs to Case 1 in Sect. 3.1.

We first analyze some cases ($Q_{X1}$, $Q_{D1}$) where the original query is faster than the minimal query.

Table 3 Query sets for the DBLP dataset.

| ID       | Query                                                                 |
|----------|------------------------------------------------------------------------|
| $Q_{D1}$ | /dblp/book/author                                                      |
| $Q_{D11}$| /book/author                                                           |
| $Q_{D2}$ | /dblp/book[cite][author][year][isbn][publisher][title]                |
| $Q_{D12}$| /book[cite]                                                            |
| $Q_{D3}$ | /dblp/inproceedings[booktitle][year][author][url]                     |
| $Q_{D4}$ | /inproceedings[number][title]                                          |
| $Q_{D5}$ | /article/cite                                                          |
| $Q_{D6}$ | /article/cite                                                          |
| $Q_{D7}$ | /proceedings/title                                                     |
| $Q_{D8}$ | /proceedings/title                                                     |
| $Q_{D9}$ | /inproceedings/crossref                                                |
| $Q_{D10}$| /inproceedings/booktitle                                              |
| $Q_{D11}$| /incollection/booktitle                                               |
| $Q_{D12}$| /incollection/booktitle                                               |
| $Q_{D13}$| /incollection/booktitle                                               |

Table 4 Query sets for the NASA dataset.

| ID       | Query                                                                 |
|----------|------------------------------------------------------------------------|
| $Q_{N1}$ | //itsFile[description/para]/footnote                                   |
| $Q_{N11}$| //itsFile/footnote                                                      |
| $Q_{N2}$ | //datasets/dataset/history/ingest/date/year                            |
| $Q_{N21}$| /ingest/year                                                           |
| $Q_{N3}$ | //datasets/dataset/reference/source/other/author                       |
| $Q_{N31}$| /other/author                                                          |
| $Q_{N4}$ | //datasets/dataset(descriptor)/footnote/para                            |
| $Q_{N41}$| /para/para                                                             |
| $Q_{N5}$ | //datasets/dataset/reference/source/journal/author/lastName            |
| $Q_{N51}$| /journal/lastName                                                      |
| $Q_{N6}$ | //datasets/dataset/history/ingest/creator/lastName                     |
| $Q_{N61}$| /ingest/lastName                                                       |
| $Q_{N7}$ | //datasets/dataset/reference/source/other/author/author/suffix          |
| $Q_{N71}$| /other/suffix                                                          |
| $Q_{N8}$ | //field[definition/footnote]/para                                       |
| $Q_{N81}$| /field/para                                                            |
| $Q_{N9}$ | //dataset[reference/author]/source/last/Name                           |
| $Q_{N91}$| /source/last/Name                                                      |

$T_{\min}$ $\approx T_{\org}$ denotes $T_{\org}/T_{\min} \leq 1.2 \land T_{\min}/T_{\org} \leq 1.2$ and means that the minimal query and the original query show almost the same performance. The number in a parenthesis is $T_{\org}/T_{\min}$ or $T_{\min}/T_{\org}$ for the query. For example, $Q_{N9}$ (4.60) means that $T_{\org}/T_{\min} = 4.60$ since $Q_{N9}$ belongs to the case where $T_{\org}/T_{\min} > 1.2$. $Q_{X1}$ (10.79, Case 1) means that (1) $T_{\min}/T_{\org} = 10.79$ since $Q_{X1}$ belongs to the case where $T_{\min}/T_{\org} > 1.2$, and (2) $Q_{X1}$ belongs to Case 1 in Sect. 3.1.

We first analyze some cases ($Q_{X1}$, $Q_{D1}$) where the original query is faster than the minimal query.

Figure 1 shows the execution time for the XMark data set. The experimental result of $Q_{X1}$ verifies Case 1 in Sect. 3.1. Figures 2 and 3 show the query plans of $Q_{X1o}$ and $Q_{X1m}$, which are obtained using the queryPlan command of the dbxml shell of Berkeley DB XML. According to the query plans, $Q_{X1o}$ and $Q_{X1m}$ use the navigation-based method. Since $Q_{X1o}$ scales down the traversal of the XML document tree to specific paths while $Q_{X1m}$ traverses the whole XML document tree, $Q_{X1o}$ is 10.79 times faster than $Q_{X1m}$.

Figure 4 shows the execution time for the DBLP data set. The experimental result of $Q_{D1}$ verifies Case 2 in Sect. 3.1. According to Figs. 5 and 6, $Q_{D1o}$ uses the navigation-based method and $Q_{D1m}$ the index-based method. Since there are many nodes matching the indexed tag, author, of $Q_{D1m}$, $Q_{D1o}$ is 9.76 times faster than $Q_{D1m}$.

We now analyze a case where the minimal query is...
faster than the original query. Figure 7 shows the NASA data set. According to Figs. 8 and 9, \( QN_{90} \) and \( QN_{900} \) use the index-based method using the same indexed tag last-Name, but \( QN_{90} \) requires a lot of tree traversal due to the

Fig. 1 The execution time for the XMark data set.

Fig. 2 The query plan of \( QX_{10} = \text{/site/regions/europe/item} \).

Fig. 3 The query plan of \( QX_{100} = \text{/europe/item} \).

Fig. 4 The execution time for the DBLP data set.

Fig. 5 The query plan of \( QD_{10} = \text{/dblp/book/author} \).

Fig. 6 The query plan of \( QD_{100} = \text{/book/author} \).

Fig. 7 The execution time for the NASA data set.

Fig. 8 The query plan of \( QN_{900} = \text{/dataset[reference//author]/source/lastName} \).
descendant-or-self axis step in the branching predicate [reference/author]. Thus, the minimal query $QN_{9m}$ is 4.60 times faster than the original query $QN_{9o}$.

### 4.2.2 BaseX

Table 7 summarizes the performance comparison for BaseX. $T_{\min} \approx T_{\text{org}}$ for more than half queries (62%), which is in contrast to Berkeley DB XML (24%). This is because the minimal query is often internally converted to the original query. For example, $QD_{3m} = /\text{inproceedings}/\text{title}$ is converted to $QD_{3o} = /\text{dblp}/\text{inproceedings}/\text{title}$. We note that the optimization technique is not able to be applied to the queries on the NASA data set, which has a complex and recursive schema.

For $QN$, the original query $QN_{5o} = /\text{datasets}/\text{dataset}/\text{reference}/\text{source}/\text{other}/\text{author}$ is 1.28 times faster than the minimal query $QN_{5m} = /\text{other}/\text{author}$ since $QN_{5m}$ basically scans the whole database to process the descendant-or-self step //other. It belongs to Case 1 in Sect. 3.1.

For $QD$, the minimal query $QD_{3m}$ is 2.59 times faster than the original query $QD_{3o}$ since $QD_{3m}$ has more branching predicates than $QD_{3o}$.

### 4.2.3 MonetDB/XQuery

Table 8 summarizes the performance comparison for MonetDB/XQuery. The minimal query is faster than (or at least as fast as) the original query since the efficiency of query processing is generally proportional to the number of steps in the query.

### 4.2.4 Stand-alone XQuery Processors: Saxon-EE, Zorba, and XQilla

For Saxon-EE, Zorba, and XQilla, the minimal query and the original query show almost the same performance as in Table 9 since stand-alone XQuery processors have to scan the whole XML document to process a query whether it is the minimal query or the original query.

### 5. Conclusions

In this paper, we introduced a new observation for XPath query minimization, the performance may be not always improved by employing the minimal query compared with the original query. We provided a detailed analysis and performed extensive experiments using six XQuery engines. Experimental results show that the performance of the minimal query could be lower than or almost equal to that of the
original query depending on query execution strategy and engine used. Our work can provide users with insights on the impacts of query minimization on XML query performance.

References

[1] S. Amer-Yahia, S. Cho, L.V.S. Lakshmanan, and D. Srivastava, “Tree pattern query minimization,” The VLDB Journal, vol.11, no.4, pp.315–331, Dec. 2002.
[2] A. Arion, A. Bonifati, I. Manolescu, and A. Pugliese, “Path summaries and path partitioning in modern XML databases,” The World Wide Web Journal, vol.11, no.1, pp.117–151, March 2008.
[3] BaseX. The XML Database, http://basetx.org
[4] P. Boncz, Monet: A Next-Generation DBMS Kernel for Query-Intensive Applications, PhD Thesis, UVA, 2002.
[5] D. Brian, “Indexes,” in The Definitive Guide to Berkeley DB XML, p.71, Apress, 2006.
[6] A.K. Chandra and P.M. Merlin, “Optimal implementation of conjunctive queries in relational data bases,” Proc. 9th Annual ACM Symposium on Theory of Computing (STOC), pp.77–90, May 1977.
[7] D. Che, “An efficient algorithm for tree pattern query minimization under broad integrity constraints,” Int. J. Web Inf. Syst., vol.3, no.3, pp.231–256, 2007.
[8] D. Chen and C. Chan, “Minimization of tree pattern queries with constraints,” Proc. Int’l Conf. on Management of Data, ACM SIGMOD, pp.609–622, June 2008.
[9] Z. Chen, H.V. Jagadish, L.V.S. Lakshmanan, and S. Paparizos, “From Tree Patterns to Generalized Tree Patterns: On Efficient Evaluation of XQuery,” Proc. 29th Int’l Conf. on Very Large Data Bases (VLDB), pp.237–248, Sept. 2003.
[10] Oracle Berkeley DB XML, http://www.oracle.com
[11] S. Flesca, F. Furfaro, and E. Masciari, “On the minimization of XPath queries,” J. ACM, vol.55, no.1, pp.1–46, Feb. 2008.
[12] R. Goldman and J. Widom, “DataGuides: Enabling query formulation and optimization in semistructured databases,” Proc. 23rd Int’l Conf. on Very Large Data Bases (VLDB), pp.436–445, Aug. 1997.
[13] G. Gottlob, C. Koch, and R. Pichler, “Efficient algorithms for processing XPath queries,” ACM Trans. Database Systems, vol.30, no.2, pp.444–491, June 2005.
[14] T. Grust, S. Sakr, and J. Teubner, “XQuery on SQL hosts,” Proc. 30th Int’l Conf. on Very Large Data Bases (VLDB), pp.252–263, Aug. 2004.
[15] T. Grust, J. Rittinger, and J. Teubner, “Pathfinder: XQuery off the relational shelf,” IEEE Data Eng. Bull., vol.31, no.4, pp.7–14, Dec. 2008.
[16] W.-S. Han, K.-H. Lee, and B.-S. Lee, “An XML storage system for object-oriented/object-relational DBMSs,” J. Object Technology, vol.2, no.3, pp.113–126, May 2003.
[17] B. Kimelfeld and Y. Sagiv, “Revisiting redundancy and minimization in an XPath fragment,” Proc. 11th Int’l Conf. on Extending Database Technology (EDBT), pp.61–72, March 2008.
[18] K.-H. Lee and Y.-H. Park, “Revisiting source-level XQuery normalization,” IEICE Trans. Inf. Syst., vol.E94-D, no.3, pp.622–631, March 2011.
[19] K.-H. Lee, S.-Y. Kim, E. Whang, and J.-G. Lee, “A practitioner’s approach to normalizing XQuery expressions,” Proc. 11th Int’l Conf. on Database Systems for Advanced Applications (DASFAA), LNCS 3882, pp.437–453, Singapore, April 2006.
[20] K.-H. Lee, K.-Y. Whang, and W.-S. Han, “Structural consistency: Enabling XML keyword search to eliminate spurious results consistently,” VLDB Journal, vol.19, no.4, pp.503–529, Aug. 2010.
[21] K.-H. Lee, K.-Y. Whang, and W.-S. Han, “XMin: Minimizing tree pattern queries with minimality guarantees,” World Wide Web J., vol.13, no.3, pp.343–371, Sept. 2010.
[22] G. Miklau, The XML Data Repository,

http://www.cs.washington.edu/research/xmldatasets
[23] MonetDB database system with XQuery front-end: MonetDB/XQuery, http://monetdb.cwi.nl/XQuery/index.html
[24] P. Ramanan, “Efficient algorithms for minimizing tree pattern queries,” Proc. Int’l Conf. on Management of Data, ACM SIGMOD, pp.299–309, June 2002.
[25] Saxonia, http://www.saxonia.com
[26] M. Schmidt, S. Scherzinger, and C. Koch, “Combined static and dynamic analysis for effective buffer minimization in streaming XQuery evaluation,” Proc. 23rd IEEE Int’l Conf. on Data Engineering (ICDE), pp.236–245, April 2007.
[27] K.-Y. Whang, I.-Y. Song, T.-Y. Kim, and K.-H. Lee, “The ubiquitous DBMS,” SIGMOD Record, vol.38, no.4, pp.14–22, Dec. 2009.
[28] P.T. Wood, “Minimising simple XPath expressions,” Proc. Fourth Int’l Workshop on the Web and Databases (WebDB), pp.13–18, May 2001.
[29] XMark — An XML Benchmark Project, http://monetdb.cwi.nl/xml
[30] XQilla, http://xqilla.sourceforge.net/HomePage
[31] Zorba XQuery Processor, http://www.zorba-xquery.com