PAPER

How XPath Query Minimization Impacts Query Processing Performance

Ki-Hoon LEE^{†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 provide a detailed analysis on the cases where the minimal query is slower than the original query.
- We carefully design experiments that investigate the effect of XPath query minimization. These experiments can be reproduced by other researchers for other engines to investigate how their performance is affected by XPath query minimization.

- 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 $F \subseteq \{/, //, [], *\}$ represents the set of query features supported including child axis “/”, descendant axis “//”, nested predicates “[]”, and wildcards “*”. Wood [28] has studied on the minimization of $XP^{[/, //, [], *]}$ 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 $XP^{[/, //, [], *]}$ 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 $O(n^2)$, $O(n^4)$, 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 $XP^{[/, //, [], *]}$ whose branching nodes are not “*”-labeled. They characterized the complexity of

Manuscript received September 15, 2011.

Manuscript revised March 30, 2012.

[†]The author is with Korea Telecom (KT), Seoul, Korea.

a) E-mail: khlee@mozart.kaist.ac.kr

DOI: 10.1587/transinf.E95.D.2258

*For simplicity, we will use engine as a unified term to refer to both XML/XQuery database systems and stand-alone (file-based) XQuery processors in the remainder of this paper.

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).

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

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

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.

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 site 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`. $\square$

EXAMPLE 2: Let the original query $QD_{1o} = \text{/dblp/book/}$ author and the minimal query $QD_{1m} = \text{/book/author}$ 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.

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 $QD_{2o} = /dblp/book[cite][author][year][isbn][publisher][title]$ and the minimal query $QD_{2m} = /dblp/book[cite]$. QD_{2o} 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_{2o} .

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

Table 1 XQuery engines used in the experiments.

Engine	Version
XML/XQuery database systems	
BaseX [3]	6.5
Berkeley DB XML [10]	2.5.16
MonetDB/XQuery [23]	0.40.3
Stand-alone (file-based) XQuery processors	
Saxon-EE [25]	9.3.0.4N
Zorba [31]	1.4.0
XQilla [30]	2.2.4

Table 2 Data sets used in the experiments.

Data set	Size	# of elements	# of attributes
DBLP	131 MB	3,332,130	404,276
NASA	24 MB	476,646	56,317
XMark	111 MB	1,666,315	381,878

compare the execution time T_{org} of the original query and the execution time T_{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} (QN_{im} , QX_{im}) is the minimal query of the original query QD_{io} (QN_{io} , QX_{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. $T_{org}/T_{min} > 1.2$ means that the minimal query is over 1.2 times faster than the original query. $T_{min}/T_{org} > 1.2$ means that the original query is over 1.2 times faster than the minimal query.

[†]Certain join methods (e.g., indexed nested-loop join) do not require temporary results.

Table 3 Query sets for the DBLP dataset.

ID	Query
QD_{1o}	/dblp/book/author
QD_{1m}	//book/author
QD_{2o}	//dblp/book[cite][author][year][isbn][publisher][title]
QD_{2m}	//book[cite]
QD_{3o}	//dblp/inproceedings[booktitle][year][author][url] [crossref][number][pages]/title
QD_{3m}	//inproceedings[number]/title
QD_{4o}	//dblp/article[volume][author][url][title][year] [journal]/cite
QD_{4m}	//article/cite
QD_{5o}	/dblp/inproceedings/title
QD_{5m}	//inproceedings/title
QD_{6o}	/dblp/article/cite
QD_{6m}	//article/cite
QD_{7o}	/dblp/proceedings/title
QD_{7m}	//proceedings/title
QD_{8o}	/dblp/inproceedings/crossref
QD_{8m}	//inproceedings/crossref
QD_{9o}	/dblp/incollection/booktitle
QD_{9m}	//incollection/booktitle
QD_{10o}	/dblp[./title/sub]/article[title/sub]
QD_{10m}	//article[./sub]
QD_{11o}	/dblp[./cite]/book[./@label]
QD_{11m}	//book[./@label]

Table 4 Query sets for the NASA dataset.

ID	Query
QN_{1o}	//fitsFile[description/para]/footnote
QN_{1m}	//fitsFile/footnote
QN_{2o}	/datasets/dataset/history/ingest/date/year
QN_{2m}	//ingest/year
QN_{3o}	/datasets/dataset/reference/source/other/author
QN_{3m}	//other/author
QN_{4o}	/datasets/dataset/description/footnote/para
QN_{4m}	//para/para
QN_{5o}	/datasets/dataset/reference/source/journal/ author/lastName
QN_{5m}	//journal/lastName
QN_{6o}	/datasets/dataset/history/ingest/creator/lastName
QN_{6m}	//ingest/lastName
QN_{7o}	//datasets/dataset/reference/source/other/ author/suffix
QN_{7m}	//other/suffix
QN_{8o}	//field[definition/footnote]/para
QN_{8m}	//field/para
QN_{9o}	//dataset[reference/author]/source/lastName
QN_{9m}	//source/lastName

$T_{min} \approx T_{org}$ denotes $T_{org}/T_{min} \leq 1.2 \wedge T_{min}/T_{org} \leq 1.2$ 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, QN_9 (4.60) means that $T_{org}/T_{min} = 4.60$ since QN_9 belongs to the case where $T_{org}/T_{min} > 1.2$. QX_1 (10.79, Case 1) means that (1) $T_{min}/T_{org} = 10.79$ since QX_1 belongs to the case where $T_{min}/T_{org} > 1.2$, and (2) QX_1 belongs to Case 1 in Sect. 3.1.

We first analyze some cases (QX_1 , QD_1) 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 QX_1 verifies Case 1 in

Table 5 Query sets for the XMark dataset.

ID	Query
QX_{1o}	/site/regions/europe/item
QX_{1m}	//europe/item
QX_{2o}	//site/people/person[name]/profile/education
QX_{2m}	//education
QX_{3o}	//site/open_auctions/open_auction/bidder/increase
QX_{3m}	//increase
QX_{4o}	//site/open_auctions/open_auction/reserve
QX_{4m}	//reserve
QX_{5o}	/site/people/person/name
QX_{5m}	//person/name
QX_{6o}	/site/regions/australia/item/name
QX_{6m}	//australia/name
QX_{7o}	/site/people/person[@id]/name
QX_{7m}	//person[@id]/name
QX_{8o}	/site/open_auctions/open_auction[bidder]/reserve
QX_{8m}	//open_auction[bidder]/reserve
QX_{9o}	/site/people/person[profile]/name
QX_{9m}	//person[profile]/name

Table 6 Performance comparison for Berkeley DB XML.

Case	Queries
DBLP data set	
$T_{org}/T_{min} > 1.2$	QD_3 (4.13), QD_4 (2.42), QD_8 (1.41), QD_{11} (1.30)
$T_{min} \approx T_{org}$	QD_5 , QD_{10}
$T_{min}/T_{org} > 1.2$	QD_1 (9.76, Case 2), QD_2 (9.71, Case 4.1), QD_7 (4.02, Case 2), QD_9 (2.98, Case 2), QD_6 (1.27, Case 4.2)
NASA data set	
$T_{org}/T_{min} > 1.2$	QN_9 (4.60), QN_4 (4.42)
$T_{min} \approx T_{org}$	QN_2 , QN_5 , QN_7 , QN_8
$T_{min}/T_{org} > 1.2$	QN_3 (7.10, Case 1), QN_1 (3.15, Case 3), QN_6 (1.74, Case 2)
XMark data set	
$T_{org}/T_{min} > 1.2$	QX_2 (1.89), QX_3 (1.62), QX_4 (1.26)
$T_{min} \approx T_{org}$	QX_8
$T_{min}/T_{org} > 1.2$	QX_1 (10.79, Case 1), QX_6 (6.01, Case 2), QX_9 (1.49, Case 4.2), QX_5 (1.39, Case 4.2), QX_7 (1.36, Case 4.2)

Sect. 3.1. Figures 2 and 3 show the query plans of QX_{1o} and QX_{1m} , which are obtained using the `queryPlan` command of the `dbxml` shell of Berkeley DB XML. According to the query plans, QX_{1o} and QX_{1m} use the navigation-based method[†]. Since QX_{1o} scales down the traversal of the XML document tree to specific paths while QX_{1m} traverses the whole XML document tree, QX_{1o} is 10.79 times faster than QX_{1m} .

Figure 4 shows the execution time for the DBLP data set. The experimental result of QD_1 verifies Case 2 in Sect. 3.1. According to Figs. 5 and 6, QD_{1o} uses the navigation-based method and QD_{1m} the index-based method. Since there are many nodes matching the indexed tag, `author`, of QD_{1m} , QD_{1o} is 9.76 times faster than QD_{1m} .

We now analyze a case where the minimal query is

[†]Since the syntax of a query plan is not completely documented anywhere other than in the source code [5], we manually verified the evaluation of each query using a debugger and a profiler.

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

```

<XQuery>
  <QueryPlanToAST>
    <StepQP axis="child" name="item" nodeType="element">
      <StepQP axis="child" name="europe" nodeType="element">
        <StepQP axis="child" name="regions" nodeType="element">
          <StepQP axis="child" name="site" nodeType="element">
            <SequentialScanQP container="xmark.dbxml" nodeType="document"/>
          </StepQP>
        </StepQP>
      </StepQP>
    </StepQP>
  </QueryPlanToAST>
</XQuery>

```

Fig. 2 The query plan of $QX_{1o} = //site/regions/europe/item$.

```

<XQuery>
  <QueryPlanToAST>
    <StepQP axis="child" name="item" nodeType="element">
      <StepQP axis="descendant" name="europe" nodeType="element">
        <SequentialScanQP container="xmark.dbxml" nodeType="document"/>
      </StepQP>
    </StepQP>
  </QueryPlanToAST>
</XQuery>

```

Fig. 3 The query plan of $QX_{1m} = //europe/item$.

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

faster than the original query. Figure 7 shows the NASA data set. According to Figs. 8 and 9, QN_{9o} and QN_{9m} use the index-based method using the same indexed tag lastName, but QN_{9o} requires a lot of tree traversal due to the

```

<XQuery>
  <QueryPlanToAST>
    <StepQP axis="child" name="author" nodeType="element">
      <StepQP axis="child" name="book" nodeType="element">
        <StepQP axis="child" name="dblp" nodeType="element">
          <SequentialScanQP container="dblp.dbxml" nodeType="document"/>
        </StepQP>
      </StepQP>
    </StepQP>
  </QueryPlanToAST>
</XQuery>

```

Fig. 5 The query plan of $QD_{1o} = /dblp/book/author$.

```

<XQuery>
  <QueryPlanToAST>
    <NodePredicateFilterQP uri="" name="#tmp1">
      <PresenceQP container="dblp.dbxml" index="node-element-equality-string"
        operation="prefix" child="author"/>
      <StepQP axis="parent-of-child" name="book" nodeType="element">
        <VariableQP name="#tmp1"/>
      </StepQP>
    </NodePredicateFilterQP>
  </QueryPlanToAST>
</XQuery>

```

Fig. 6 The query plan of $QD_{1m} = //book/author$.

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

```

<XQuery>
  <QueryPlanToAST>
    <NodePredicateFilterQP uri="" name="#tmp20">
      <PresenceQP container="nasa.dbxml" index="node-element-equality-string"
        operation="prefix" child="lastName"/>
      <StepQP axis="descendant" name="author" nodeType="element">
        <StepQP axis="child" name="reference" nodeType="element">
          <StepQP axis="ancestor" name="dataset" nodeType="element">
            <StepQP axis="ancestor" name="source" nodeType="element">
              <VariableQP name="#tmp20"/>
            </StepQP>
          </StepQP>
        </StepQP>
      </StepQP>
    </NodePredicateFilterQP>
  </QueryPlanToAST>
</XQuery>

```

Fig. 8 The query plan of $QN_{9o} = //dataset[reference//author]//source/lastName$.

```

<XQuery>
  <QueryPlanToAST>
    <NodePredicateFilterQP uri="" name="#tmp1">
      <PresenceQP container="nasa.dbxml" index="node-element-equality-string"
        operation="prefix" child="lastName"/>
      <StepQP axis="ancestor" name="source" nodeType="element">
        <VariableQP name="#tmp1"/>
      </StepQP>
    </NodePredicateFilterQP>
  </QueryPlanToAST>
</XQuery>

```

Fig. 9 The query plan of $QN_{9m} = //source/lastName$.

Table 7 Performance comparison for BaseX.

Case	Queries
DBLP data set	
$T_{org}/T_{min} > 1.2$	QD_3 (2.59), QD_{11} (2.31), QD_4 (2.29), QD_{10} (1.97)
$T_{min} \approx T_{org}$	$QD_1, QD_2, QD_5, QD_6, QD_7, QD_8, QD_9$
$T_{min}/T_{org} > 1.2$	
NASA data set	
$T_{org}/T_{min} > 1.2$	QN_7 (1.55), QN_9 (1.27)
$T_{min} \approx T_{org}$	QN_1, QN_4, QN_8
$T_{min}/T_{org} > 1.2$	QN_3 (1.28, Case 1), QN_6 (1.28, Case 1), QN_2 (1.27, Case 1), QN_5 (1.26, Case 1)
XMark data set	
$T_{org}/T_{min} > 1.2$	QX_2 (1.21)
$T_{min} \approx T_{org}$	$QX_1, QX_3, QX_4, QX_5, QX_6, QX_7, QX_8, QX_9$
$T_{min}/T_{org} > 1.2$	

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. $T_{min} \approx T_{org}$ 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, $QD_{5m} = //inproceedings/title$ is converted to $QD_{5o} = /dblp/inproceedings/title$. 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_3 , the original query $QN_{3o} = /datasets/dataset/reference/source/other/author$ is 1.28 times faster than the minimal query $QN_{3m} = //other/author$ since QN_{3m} basically scans the whole database to process the descendant-or-self step //other. It belongs to Case 1 in Sect. 3.1.

For QD_3 , the minimal query QD_{3m} is 2.59 times faster than the original query QD_{3o} since QD_{3o} has more branching predicates than QD_{3m} .

4.2.3 MonetDB/XQuery

Table 8 summarizes the performance comparison for MonetDB/XQuery. The minimal query is faster than (or at least

Table 8 Performance comparison for MonetDB/XQuery.

Case	Queries
DBLP data set	
$T_{org}/T_{min} > 1.2$	QD_3 (1.37), QD_4 (1.24)
$T_{min} \approx T_{org}$	$QD_1, QD_2, QD_5, QD_6, QD_7, QD_8, QD_9, QD_{10}, QD_{11}$
$T_{min}/T_{org} > 1.2$	
NASA data set	
$T_{org}/T_{min} > 1.2$	
$T_{min} \approx T_{org}$	$QN_1, QN_2, QN_3, QN_4, QN_5, QN_6, QN_7, QN_8, QN_9$
$T_{min}/T_{org} > 1.2$	
XMark data set	
$T_{org}/T_{min} > 1.2$	
$T_{min} \approx T_{org}$	$QX_1, QX_2, QX_3, QX_4, QX_5, QX_6, QX_7, QX_8, QX_9$
$T_{min}/T_{org} > 1.2$	

Table 9 Performance comparison for Saxon-EE, Zorba, and XQilla.

Case	Queries
DBLP data set	
$T_{org}/T_{min} > 1.2$	
$T_{min} \approx T_{org}$	$QD_1, QD_2, QD_3, QD_4, QD_5, QD_6, QD_7, QD_8, QD_9, QD_{10}, QD_{11}$
$T_{min}/T_{org} > 1.2$	
NASA data set	
$T_{org}/T_{min} > 1.2$	
$T_{min} \approx T_{org}$	$QN_1, QN_2, QN_3, QN_4, QN_5, QN_6, QN_7, QN_8, QN_9$
$T_{min}/T_{org} > 1.2$	
XMark data set	
$T_{org}/T_{min} > 1.2$	
$T_{min} \approx T_{org}$	$QX_1, QX_2, QX_3, QX_4, QX_5, QX_6, QX_7, QX_8, QX_9$
$T_{min}/T_{org} > 1.2$	

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://basex.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 guarantee," *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] Saxonica, <http://www.saxonica.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>

Ki-Hoon Lee received B.S. (2000), M.S. (2002), and Ph.D. (2009) degrees in Computer Science from Korea Advanced Institute of Science and Technology (KAIST). He is currently a research engineer in Korea Telecom (KT). His research interests include XML and web databases, IR and search engines, query optimization, object-relational database systems, spatial databases and GIS, and multimedia.