Algebraic-Based XQuery Cardinality Estimation

Sherif Sakr
University of Kosntanz, Germany
sherif.sakr@inf.uni-konstanz.de

Abstract. Estimating the sizes of query results and intermediate results is crucial to many aspects of query processing. All database systems rely on the use of selectivity estimates to choose the cheapest execution plan. In principle, the problem of cardinality estimation is more complicated in the XML domain than the relational domain. In this article we present a novel framework for estimating the cardinality of XQuery expressions as well as its sub-expressions. As a major innovation, we exploit the relational algebraic infrastructure to provide accurate estimation in the context of XML and XQuery domains.

Keywords
Cardinality Estimation - XQuery - XPath.

1 Introduction

Estimating the sizes of query results and intermediate results is crucial to many aspects of query processing. In particular, it is very important for an effective query optimization process. In principle, the problem of cardinality estimation is more complicated in the XML domain than the relational domain. The main reason behind this is that the XML queries involves structural conditions in addition to the value-based conditions. Therefore, any complete and accurate size estimation system for the XML queries requires maintaining statistical summary information about the structure of the source XML documents as well as statistical summary information about the distribution of the values associated with the XML nodes.

XML database management systems still lack a convincing means to estimate cardinalities for arbitrary XQuery (sub-)expressions. All of the existing work in this domain remains limited to a subset of XPath expressions. In this article we present a novel framework for estimating the cardinality of XQuery expressions as well as its sub-expressions. The work of this article was developed within the Pathfinder project [1]. The aim of the Pathfinder project is to implement the XQuery language to query XML data stored on relational database systems. The architecture of Pathfinder is designed in a front-end/back-end fashion. Pathfinder receives an XQuery expression, which is parsed, normalized, and translated into XQuery Core. The Core expression is then simplified, type checked optimized,
and translated into a relational algebraic plan. This algebraic plan thus serves as an intermediate representation for the XQuery expression which is then in our context translated into SQL code over the conventional RDBMS [15] and in other contexts translated into MIL code over MonetDB DBMS [6], [26]. Figure 1 illustrates framework of our proposed XQuery cardinality estimation. In this framework, the cardinality estimation process for XQuery expressions runs through the following steps:

1. The source XML documents have to be mapped (shredded) into a relational scheme using the XPath Accelerator encoding scheme presented by Grust in [13]. During the shredding process, we build our summarized tree structure to capture summary information about the structure of the source XML document. We name this summary structure as the Statistical Guide.
2. To support the estimation of the value based predicates, we need to build statistical histograms for capturing the distribution of the data values. As it is always a trade-off between the required estimation accuracy and the used storage space, the decision to build the histograms for capturing the distribution of the atomic values of the most frequently used guide nodes of the Statistical Guide is left to the system administrator.
3. Using the loop-lifting compilation techniques, the Pathfinder algebraic translation module translates its input XQuery expressions into their equivalent relational algebraic plans [16],[31].
4. The XQuery cardinality estimation module receives four inputs:
   (a) The summarized tree structure \textit{Statistical Guide} built in step 1.
   (b) The statistical histograms information built in step 2.
   (c) The algebraic plans annotated with its special properties from step 3.
   (d) The inference rules for the cardinality estimation of the \textit{Pathfinder} algebraic operators (Section 4.1).
   and returns back the estimated cardinality for each operator in the algebraic plan.

5. The estimated cardinality for the root operator in the \textit{Pathfinder} algebraic plan represents the estimated cardinality for the whole XQuery expression, while the estimated cardinalities for the intermediated operators represent the estimated sizes of their correspondent sub-expressions.

In particular, the main contributions of the work of this article is that it presents a novel framework for XQuery cardinality estimation with the following main advantages:

- It is able to support the cardinality estimation of a large subset of the powerful XML query language XQuery.
- It estimates not only the whole XQuery expression but also each of its sub-expressions as well as the cardinality of each iteration in the context of the FLWOR expressions.
- It introduces a novel concept that of a \textit{Guide Node} annotation which gives the framework the flexibility to integrate any XPath or predicates size estimation techniques.

The rest of this article is organized as follows. In Section 2, we give an overview of the \textit{loop-lifting} compilation technique which translates XQuery expressions into their equivalent intermediate relational algebraic plans. In Section 3, we describe the main building blocks of our proposed XQuery cardinality estimation framework. In Section 4, we describes the process for estimating the cardinality of XQuery expression as well as its sub-expressions. Our proposed benchmark for XQuery cardinality estimation system is presented in Section 5 to establish the basis of the experimental analysis introduced in Section 6. Section 7 reviews the related work before we conclude in Section 8. The appendix of this article fully describes the queries of our proposed benchmark for XQuery cardinality estimation system.

2 XQuery Algebraic Compilation

Relational algebra has been a main component in relational database systems, and has played an important role in their success for gaining widespread usage. A corresponding XML algebra would have the same importance and substantial role for XML query processing. In our context, having an adequate algebraic compilation for XQuery expressions provides us with the solid infrastructure for predicating the cardinality of the main XQuery expressions and its
sub-expressions in a very convenient and accurate way. Many proposals for an algebra for XML query processing have been introduced [7],[8],[28],[19],[23],[17]. According to [25] existing algebras for XQuery fall into two classes:

- **Tuple-based algebra:** this class of algebra tries to facilitate the use of relational optimization techniques as well as the whole relational query processing framework (theory, compilation, optimization, execution).
- **Tree-based algebra:** this class of algebra provides more natural support for novel XML-specific optimizations and manipulates XML data modelled as forests of labelled ordered trees.

The Pathfinder project has a special module for compiling XQuery expressions into its own dialect of *tuple-based algebra* producing equivalent relational query plans. The Pathfinder algebra is quite primitive such that it can efficiently fit within the capabilities of SQL-based systems. Pathfinder compiles the XQuery core dialect listed in Table 1 into relational query plans using the set of algebraic operators listed in Table 2. The Pathfinder algebraic operators are designed to receive one or more inputs and produce one or more outputs. These inputs and outputs are in the form of sets of tuples. Most of these operators are rather standard or even restricted variants of the operators found in a classical relational algebra which allow the query optimizer to employ the usual relational algebraic optimization techniques such as the pushdown of the selection operator. For example, the attachment operator (\( \theta_{a,v} \)) appends one a new attribute \( a \) to the tuples of the input relation \( R \) where the value for the new attribute is assigned via the value \( v \). The selection operator (\( \sigma_a \)) filters the tuples of the input relation \( R \) to only the tuples where the Boolean selection attribute \( a \) equals to \( \text{true} \). The Boolean selection attribute \( (a) \) are usually produced from applying the *comparison* operator (\( \equiv \)). The unsorted row numbering operator (\( \#_a \)) and the sorted row numbering operator (\( \varrho_{a}(o_1,...,o_n)/p \)) are two of the most important Pathfinder’s algebraic operators. Together, they are responsible of preserving the *order* concept defined by the XQuery/XPath specifications [9].

The unsorted row numbering operator (\( \#_a \)) receives an input relation \( R \) and returns the same relation extended with a new consecutive numbering attribute \( (a) \) from 1 to \( n \) where \( n \) is the cardinality of the input relation \( R \). The sorted row numbering operator (\( \varrho_{a}(o_1,...,o_n)/p \)) receives an input relation \( R \) an ordering attribute list \( (o_1,...,o_n) \) and an optional partitioning attribute \( p \). It returns the same relation extended with a new consecutive numbering attribute \( (a) \). The numbering attribute \( (a) \) respects the tuple sorting of relation \( R \) defined by the order specification \( (o_1,...,o_n) \) and restarts the numbering from 1 for each partition defined by the *optional* partitioning attribute \( (p) \). The XPath location step evaluator operator (\( \mathcal{E}_{\text{item}:(a,m)} \)) is responsible for evaluating XPath expressions. It returns the result nodes for each iteration of the input context relation in the *document order* and *duplicate free*. Figure 2 illustrates Examples for the behavior of some Pathfinder algebraic operators.

In this section, we give an overview of the *loop-lifting* technique, which is considered to be the heart of this compilation process. For a detailed and complete description of this technique and its translation rules we refer to [31],[16],[14].
a nutshell, loop-lifting is the key technique used to compile XQuery iterations into efficient bulk style application of algebraic operators. The principal idea behind the compilation scheme is that every XQuery expression occurs in the scope of an iteration. The iterations of each scope are encoded by a column iter in the associated relational representation. In order to clarify the loop-lifting idea, we will show in the following subsections some examples of the loop-lifted translations for some XQuery expressions into its associated relational algebraic representation.

<table>
<thead>
<tr>
<th>atomic literals</th>
<th>document order (e1 &lt;&lt; e2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequences(e1,e2)</td>
<td>node identity (e1 is e2)</td>
</tr>
<tr>
<td>variables</td>
<td>arithmetics (+,-,*,...)</td>
</tr>
<tr>
<td>let $v:=e1 return e2</td>
<td>comparisons (=, &lt;, &gt;, ...)</td>
</tr>
<tr>
<td>for $v [at$p] in e1 return e2</td>
<td>Boolean connectives (and, or)</td>
</tr>
<tr>
<td>if (e1) then e2 else e3</td>
<td>user-defined functions</td>
</tr>
<tr>
<td>e1 order by e2,...,en</td>
<td>fn:doc(), fn:root(), fn:data()</td>
</tr>
<tr>
<td>unordered {e}</td>
<td>fn:id(), fn:idref()</td>
</tr>
<tr>
<td>element {e1},{e2}</td>
<td>fn:distinct-values()</td>
</tr>
<tr>
<td>attribute {e1},{e2}</td>
<td>op:union(), op:intersect(), op:difference()</td>
</tr>
<tr>
<td>text (e)</td>
<td>fn:count(), fn:sum(), fn:max(), ...</td>
</tr>
<tr>
<td>XPath Location Steps</td>
<td>fn:position(), fn:last()</td>
</tr>
</tbody>
</table>

Table 1. Pathfinder supported XQuery dialect

<table>
<thead>
<tr>
<th>π_{a_1:b_1,...,a_n:b_n}</th>
<th>Projection</th>
</tr>
</thead>
<tbody>
<tr>
<td>@_a:v</td>
<td>Attachment</td>
</tr>
<tr>
<td>σ_{a}</td>
<td>Selection</td>
</tr>
<tr>
<td>\</td>
<td>Difference</td>
</tr>
<tr>
<td>x</td>
<td>Cartesian Product</td>
</tr>
<tr>
<td>_a=b</td>
<td>Equi-Join</td>
</tr>
<tr>
<td>δ</td>
<td>Duplicate Elimination</td>
</tr>
<tr>
<td>¬_{item:item}</td>
<td>Negation</td>
</tr>
<tr>
<td>A</td>
<td>Tables</td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
</tbody>
</table>

Table 2. Pathfinder algebraic operators

2.1 Sequences

The XQuery language is designed to operate over ordered, finite sequences of items as its principal data type. The evaluation of any XQuery expression yields
Fig. 2. Illustrating Examples for the behavior of some Pathfinder algebraic operators.
an ordered sequence of \( n \geq 0 \) items. These items can be either atomic values (integers, strings, ..., etc) or XML tree nodes. An XQuery item sequence \((x_1, ..., x_n)\) is encoded with the following relational table:

\[
\begin{array}{c|c}
\text{pos} & \text{item} \\
1 & x_1 \\
\vdots & \vdots \\
n & x_n \\
\end{array}
\]

The column \( \text{pos} \) is used to preserve the order information between the items inside the target sequence. The \( \text{item} \) column is a polymorphic column. In the case of atomic items, it stores the values of the encoded atomic items \((1, "A", ...)\) and in the other case of XML tree nodes, it stores the pre-order ranks of the encoded nodes. The RDBMS supports the representation of such polymorphic columns using the \textit{Variant} data type. The empty sequence \((\emptyset)\) is encoded with an empty table with the same schema \((\text{pos, item})\).

### 2.2 FLWOR Expressions

The FLWOR expression is one of the main features provided by the XQuery language. It is used for representing iterations and for the binding of variables to intermediate results. It is also used for computing joins between two or more sequences and for restructuring data. A loop of \( n \) iterations is represented by a relation \textit{loop} with a single column \textit{iter} of \( n \) values \((1, 2, ..., n)\).

\[
\begin{array}{c}
\text{iter} \\
1 \\
\vdots \\
n \\
\end{array}
\]

Based on the relational representation for the sequence \((x_1, ..., x_n)\) we presented in the previous section, we can now represent the compilation of variables bound in the iterations of FLWOR expression using the following XQuery \textit{for-loop} example:

\[
\text{for } \$v \text{ in } (x_1, ..., x_n) \text{ return } e.
\]

This example expression binds each \( x_i \) item to variable \$v\ and evaluates the loop body \( e \) for each iteration. The relational encoding of the variable \$v\ has the following form:

\[
\begin{array}{c|c|c}
\text{iter} & \text{pos} & \text{item} \\
1 & 1 & x_1 \\
2 & 1 & x_2 \\
\vdots & \vdots & \vdots \\
n & 1 & x_n \\
\end{array}
\]
This translation encodes all bindings of $v$ in a single relation. In general, each tuple of the encoding relation $(i, p, x)$ indicates that for the $i$-th iteration, the item at position $p$ stores the value $x$.

### 2.3 Path Steps

The compilation of path steps is represented in the algebraic plans using the XPath evaluator operator ($\varepsilon$). It takes a context relation $(\text{iter}, \text{item})$ as input, where the $\text{item}$ column stores the node identifiers of the input context nodes. It uses the XPath encoding relation of the current live nodes ($\Gamma$) to evaluate the path step $(\alpha, n)$ and returns a new relation with the same schema $(\text{iter}, \text{item})$. In the output relation, the $\text{item}$ column stores the node identifiers of the resulting nodes. Figure 3 illustrates an example for the loop-lifted compilation of the path step where Figure 3(a) represents the input context nodes, Figure 3(b) represents the sample source XML document and Figure 3(c) represents the output context nodes from applying the path step $\text{child::c}$ over input context nodes and the sample source XML document.

![Fig. 3. An illustrating example for the loop-lifted compilation of the path steps.](image)

### 2.4 Arithmetic and Comparison Expressions

Given the relational representation $R_1(\text{iter}_1, \text{item}_1)$ and $R_2(\text{iter}_2, \text{item}_2)$ of two XQuery expression $e_1$ and $e_2$, the design of the loop-lifting compilation of the arithmetic and comparison expressions requires that the schema of the relational representation ($R_i$) of each argument expression ($e_i$) must have a single node identifier column ($\text{item}_i$) for each iteration ($\text{iter}_i$) with no order preserving column ($\text{pos}$). The arithmetic expression $e_1 \circ e_2$ is evaluated by joining $R_1$ and $R_2$ over their iterations attribute $\text{iter}$, for each tuple of the result we apply the arithmetic operator ($\circ$) over the two arguments columns $\text{item}_1$ and $\text{item}_2$, and store
the result in a new column $res$. The algebraic representation of the arithmetic expression ($e_1 \odot e_2$) is defined as follows:

$$e_1 \odot e_2 \Rightarrow \pi_{iter1:iter, res}(\bigotimes_{res}(item1, item2)(R_1 \Join_{(iter1=iter2)} R_2))$$

where $R_1$ and $R_2$ are consequentially representing the relational representation of the two expressions $e_1$ and $e_2$.

Similarly, the comparison expression $e_1 \oplus e_2$ is evaluated by joining $R_1$ and $R_2$ over their iterations attribute $iter$, for each tuple of the result we apply the comparison operator ($\odot$) over the two argument attributes $item1$ and $item2$, and store the result in a new attribute $res$. The comparison operators are normally followed by a selection operator to filter the tuples satisfying the comparison condition. The algebraic representation of the comparison expression ($e_1 \oplus e_2$) is defined as follows:

$$e_1 \oplus e_2 \Rightarrow \sigma_{res}(\pi_{iter1:iter, res}(\bigotimes_{res}(item1, item2)(R_1 \Join_{(iter1=iter2)} R_2)))$$

3 Building Blocks of XQuery Cardinality Estimation

3.1 Statistical Guide

The first step in estimating the result size of an XQuery expression is to summarize the source XML document into a compact graph structure which we name the Statistical Guide. The Statistical Guide represents an implementation for the Data Guide summary tree structure presented in [12] and is very similar to the path tree summary structure presented by Aboulnaga in [3]. Every node in the Statistical Guide represents a rooted path starting from the root node of the source XML document. The root node of the Statistical Guide represents the root element of the document. The Statistical Guide has a node for every distinct rooted path in the source XML document. Each node of the Statistical Guide is labeled with the tag name of the elements reachable by the path it represents and annotated with the number of occurrences for such node (rooted path). Figure 4 presents an XML document and its associated Statistical Guide. We construct the Statistical Guide of an XML document during the normal shredding process of the XML document using the XPath accelerator storage scheme presented by Grust in [13]. It is thus constructed during the normal scan of the document using a SAX parser. To estimate the cardinality of an XPath expression using the Statistical Guide, we need to evaluate the XPath expression over the Statistical Guide. This will result in a set of nodes which correspond to the query path expression. Hence, the estimated cardinality of the XPath expression is defined by computing the sum of the number of occurrences of these resulting nodes. Due to its compact size even for larger XML documents, the Statistical Guide can easily fit into main memory and be made available for the query processor during the compilation process.
Fig. 4. An example of an XML document and its associated Statistical Guide.

<table>
<thead>
<tr>
<th>Histogram ID</th>
<th>Guide Node</th>
<th>Size</th>
<th>Unique Values</th>
<th>Min. Value</th>
<th>Max. Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(a) Histograms Information

<table>
<thead>
<tr>
<th>Histogram ID</th>
<th>Bucket ID</th>
<th>Start</th>
<th>End</th>
<th>Acc. Before</th>
<th>Acc. After</th>
<th>Size</th>
<th>Unique Val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b) Buckets Information

<table>
<thead>
<tr>
<th>Histogram ID</th>
<th>Bucket ID</th>
<th>Value</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(c) Buckets Detail Information

Fig. 5. Statistical histograms repository
3.2 Histograms

Histograms are the most popular summary data structures used for estimating query result sizes in relational systems. In our XQuery cardinality estimation system, it is used to estimate the number of nodes satisfying a specified value predicate. Although, we can build a histogram for each value node (text node - attribute) in the Statistical Guide, it is always a trade-off between the required estimation accuracy and the used storage space. In practice, we leave it to the system administrator to decide building the required histograms for the frequently used nodes in the Statistical Guide. A histogram $H$ on node $X$ is constructed by partitioning the data distribution of the node values into $N$ mutually disjoint subsets called buckets and then counting the number of values for each bucket. The histogram’s buckets can be defined according to different partitioning rules that searches for effective distribution of the values which leads to accurate estimations. The values in the range of each bucket is assumed to be uniformly distributed between the lowest and highest values in the bucket. In [24], Poosala has provided a survey on several well-known histograms such as equi-width, equi-depth and serial histograms [18] where each of them is using different partitioning rules and constraints. In particular, each of these histograms has its own advantages. For example, equi-depth histogram works well for range queries only when the data distribution has low skew while serial histogram has shown to be optimal for equality joins and selection when a list of all the attribute values in each bucket is maintained.

In fact our proposed estimation framework is not dependent on or affected by the type of the used histograms and it is flexible to integrate any of them. Currently, we are using on standard the simple equi-width histograms for collecting the statistical information of the relevant nodes however it is also open for the system administrator to use any other type of histograms without affecting the estimation process. Figure 5 illustrates the structure of our used statistical histograms repository. Remarks about this structure are given as follows:

- Each created histogram is identified by a Histogram ID attribute and is associated with a Guide Node from the Statistical Guide. The associated Guide Node represents a group of nodes on the original XML document which have the same rooted path and stores atomic values (i.e text nodes or attributes).
- For each created histogram, we need to store the following descriptive information:
  - **Size**: The total number of values in the histogram data set.
  - **Number of unique values**: The number of the unique values in the histogram data set.
  - **Minimum value**: The minimum value of the histogram data set.
  - **Maximum value**: The maximum value of the histogram data set.
- The characteristics of the distribution of the values for each histogram is defined in terms of buckets. Each bucket is defined by the Histogram ID and Bucket ID attributes. For each bucket we collect the required descriptive
information that specifies the characteristics of the values which lies between the buckets’ boundaries as defined by its Start Value and End Value attributes. These descriptive information are:

- **Size:** The count of the values of the histogram data set which are lying between the the lower boundary and the upper boundary of the bucket.
- **Number of unique values:** The count of the unique values of the histogram data set which are lying in the defined range of the bucket.
- **Accumulative before:** The count of the values in the histogram data set which are less than the lower boundary of the bucket (Start Value).
- **Accumulative after:** The count of the values in the histogram data set which are greater than the upper boundary of the bucket (End Value).

Additionally, for each defined bucket we may add detailed information concerning one or more of the atomic values which lies inside the buckets’ boundary. Assuming we have a bucket of size \((S)\), this extra information is defined for each value \((V)\) with number of occurrences \((N)\) satisfying the following conditions:

\[
\frac{N}{S} > P
\]

where the value of the percentage parameter \((P)\) is defined by the system administrator.

**Predicate Selectivity Estimation Using Histograms** A key ingredient in any cardinality estimation system is the estimation of the selectivity of value-based predicates. Hence, the estimation of value-based predicates selectivity is a well-known problem in database theory and practice. The common solutions of this problem relies on the use of histograms for capturing the distribution of values in the data.

By using the information of the repository of the statistical histograms, the task of computing the selectivity of the value-based predicates becomes a straightforward process. Given the information that the distribution of the values of the guide node \(X\) of the Statistical Guide is captured by the statistical histogram \(H\), computing the selectivity of any value-based predicate of the form \((X \theta v)\), where \(\theta\) represents the comparison operators (\(=, <, >\)) and \(v\) represents an atomic value, is achieved using the inference rules represented in Figure 6. Assuming that the selectivity estimation of \((X \theta v)\) is equal to \(S\) then the selectivity estimation of \((X \vartheta v)\) is equal to \((1 - S)\) where \(\vartheta\) is the negation of the comparison operator \(\theta\).

In practice, given the information that the distribution of the data values of the guide node \(X\) is associated with the histogram \(H\), the inference rules for the estimation of the value based predicate of the mentioned form \((X \theta v)\) are derived by the following steps:

(i) Firstly, we find the bucket \(Buck(i)\) of the histogram \(H\) where the value \(v\) lies between the boundaries of the bucket.

(ii) Using the information of the bucket \(Buck(i)\), we can compute the estimated count for the values of the histogram \(H\) which satisfies the condition of the predicate \((X \theta v)\). This computed count is represented as \(C\).
\[
X.\text{Histogram} = H \\
(H.\text{MinVal} > v) \lor (v > H.\text{MaxVal}) \\
\Rightarrow \text{Sel}(X = v) = 0 \tag{PRED-1}
\]

\[
X.\text{Histogram} = H \\
H.\text{MinVal} < v < H.\text{MaxVal} \\
H.\text{Buck}(i).\text{Start} < v < H.\text{Buck}(i).\text{End} \\
H.\text{Buck}(i).\text{Detail}(j).\text{Value} = v \\
\Rightarrow \text{Sel}(X = v) = \left( \frac{H.\text{Buck}(i).\text{Detail}(j).\text{Size}}{H.\text{Size}} \right) \tag{PRED-2}
\]

\[
X.\text{Histogram} = H \\
H.\text{MinVal} < v < H.\text{MaxVal} \\
H.\text{Buck}(i).\text{Start} < v < H.\text{Buck}(i).\text{End} \\
\Rightarrow \text{Sel}(X = v) = \left( \frac{1}{H.\text{Buck}(i).\text{UniqueValues}} \right) \times \frac{H.\text{Buck}(i).\text{Size}}{H.\text{Size}} \tag{PRED-3}
\]

\[
X.\text{Histogram} = H \\
H.\text{MaxVal} < v \\
\Rightarrow \text{Sel}(X < v) = 1 \tag{PRED-4}
\]

\[
X.\text{Histogram} = H \\
v < H.\text{MinVal} \\
\Rightarrow \text{Sel}(X < v) = 0 \tag{PRED-5}
\]

\[
X.\text{Histogram} = H \\
H.\text{MinVal} < v < H.\text{MaxVal} \\
H.\text{Buck}(i).\text{Start} < v < H.\text{Buck}(i).\text{End} \\
\Rightarrow \text{Sel}(X < v) = (H.\text{Buck}(i).\text{AccBefore} + \left( \frac{v - H.\text{Buck}(i).\text{Start}}{H.\text{Buck}(i).\text{End} - H.\text{Buck}(i).\text{Start}} \right) \times \frac{H.\text{Buck}(i).\text{Size}}{H.\text{Size}}) \tag{PRED-6}
\]

\[
X.\text{Histogram} = H \\
H.\text{MaxVal} < v \\
\Rightarrow \text{Sel}(X > v) = 0 \tag{PRED-7}
\]

\[
X.\text{Histogram} = H \\
v < H.\text{MinVal} \\
\Rightarrow \text{Sel}(X > v) = 1 \tag{PRED-8}
\]

\[
X.\text{Histogram} = H \\
H.\text{MinVal} < v < H.\text{MaxVal} \\
H.\text{Buck}(i).\text{Start} < v < H.\text{Buck}(i).\text{End} \\
\Rightarrow \text{Sel}(X > v) = (H.\text{Buck}(i).\text{AccAfter} + \left( \frac{H.\text{Buck}(i).\text{End} - v}{H.\text{Buck}(i).\text{End} - H.\text{Buck}(i).\text{Start}} \right) \times \frac{H.\text{Buck}(i).\text{Size}}{H.\text{Size}}) \tag{PRED-9}
\]

**Fig. 6.** Inference rules of predicate selectivity estimation using histograms
(iii) The estimated selectivity of \((X \theta v)\) is computed by dividing the computed count of values which satisfies the predicate condition \((C)\) by the total number of values in the data set represented by the histogram \(H\) (Histogram Size).

(iv) An exact estimation for the equality predicate can be computed in the case where the bucket detail information for the value \(v\) exists (Rule Pred-2).

(v) If the value \(v\) lies outside the boundaries of histogram \(H\) then the value of the estimated selectivity is equal to 0 or 1 based on the comparison operator \((\theta)\) and the comparison conditions between the value \(v\) and the histogram boundaries (Rules Pred-1, Pred-4, Pred-5, Pred-7 and Pred-8).

Remarks about the used notations in the inference rules of Figure 6 are given as follows:

- The notation \((X.Histogram = H)\) states that the values of the guide node \(X\) is represented by the histogram \(H\).
- The notation \((H.MinVal)\) represents the lower boundary of the histogram \(H\) while the notation \((H.MaxVal)\) represents the upper boundary of the histogram \(H\).
- The notation \((H.Buck(i).Start)\) represents the lower boundary of the bucket number \(i\) of the histogram \(H\) while the notation \((H.Buck(i).End)\) represents the upper boundary of the bucket number \(i\) of the histogram \(H\).
- The notation \((H.Size)\) represents the total number of values of the data set represented by the histogram \(H\) while the notation \((H.Buck(i).Size)\) represents the number of values that lies between the boundary of the bucket \(i\) of the histogram \(H\).
- The notation \((H.Buck(i).AccBefore)\) represents the number of values that are less than the lower boundary of the bucket \(i\) of the histogram \(H\) while the notation \((H.Buck(i).AccAfter)\) represents the number of values that are greater than the upper boundary of the bucket \(i\) of the histogram \(H\).
- The notation \((H.Buck(i).UniqueValues)\) represents the number of unique values lying between the boundaries of the bucket \(i\) of the histogram \(H\).
- The notation \((H.Buck(i).Detail(j).Value = v)\) states that the bucket \(i\) of the histogram \(H\) has detailed information for the value \(v\. The exact size of the value \(v\) is represented by the notation \((H.Buck(i).Detail(j).Size)\)

3.3 Histograms Operation

Histograms Constant Manipulation The XQuey language provides the capability of performing arithmetic operations. One variation of the instances of the arithmetic operators \(\odot_{z:(x,y)}\) inside the algebraic plan is the instance where one argument \((x)\) is represented by an item sequence of atomic values and the other argument \((y)\) is a constant value \(v\). Assuming the data distribution for the atomic values of the item sequence representing the first argument \((x)\) is captured by the histogram \((H1)\). Applying the arithmetic operation yields a resulting item sequence where the data distribution is captured by the histogram
(H2). The characteristics of the data distribution of (H2) is defined by manipulating the boundaries of the histogram (H1), its minimum value, its maximum value and the boundaries of its associated buckets with the defined arithmetic operation and constant value (V) while the distribution of the values inside each buckets remains as the same.

**Histograms Arithmetic** Another variation of the instances of the arithmetic operators $\odot_{z(x,y)}$ is the situation where both of the arguments (x, y) are represented by an item sequence of atomic values. Assume that the data distribution for the atomic values of the item sequence representing the two arguments (x, y) are captured by the two histograms (HX, HY) respectively. In this situation we rely on the histogram discretization algorithm presented by Berleant in [4] for performing arithmetic operations over histograms and computing the resulting histogram H3 which captures the distribution of the values of the resulting attribute z. The steps of performing the arithmetic operation $\odot$ over the two histograms (HX, HY) using the histogram discretization algorithm are described as follows:

1. Compute the Cartesian product of the buckets of two histograms (HX, HY).
2. For each member (HXi, HYj) in the Cartesian product, an intermediate result bucket is computed by:
   (a) Executing the arithmetic operation $\odot$ on HXi and HYj to get the intermediate resulting bucket HZij = HXi $\odot$ HYj.
   (b) Associating with the intermediate bucket HZij the size information equal to the size of the bucket HXi multiplied by the size of the bucket HYj.
3. The resulting intermediate buckets are combined to get a final result by:
   (a) Deciding on a set of intervals partitioning the domain of HZ. This partition determines the placement of the boundaries of the buckets of the resulting histogram.
   (b) Calculating the size of each bucket of the resulting histogram HZ as follows:
      i. Any intermediate result bucket HZij that falls completely within some member of the partition has its entire results added to that member.
      ii. Any intermediate result bucket that overlaps more than one member of the partition has its size divided among them proportionally to the fraction of the intermediate result bucket it overlaps.

**Histograms Comparison** One variation of the instances of the comparison operators $\sqsubseteq_{z(x,y)}$ is the instance where both of the arguments (x, y) are represented by an item sequence of atomic values. Assuming the data distribution for the atomic values of the item sequence representing the two argument (x, y) are captured by the two histograms (HX, HY) respectively. In this situation and using the same idea of the histogram discretization algorithm, the selectivity of the resulting boolean attribute z is computed by applying bucket by bucket comparison of the Cartesian product between the two histograms (HX, HY).
3.4 Special Properties for the Algebraic Plan

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{card}_0, \text{card}_1</td>
<td>associated relation with cardinality 0 or 1</td>
</tr>
<tr>
<td>\text{key} : a</td>
<td>attribute (a) is a key attribute</td>
</tr>
<tr>
<td>\text{domain} : a^{(\alpha)}</td>
<td>attribute (a) with domain identifier (\alpha)</td>
</tr>
<tr>
<td>\text{const} : a^{(v)}</td>
<td>attribute (a) is an attribute with value (v) for all tuples of the relation</td>
</tr>
</tbody>
</table>

Table 3. Special properties of Pathfinder algebraic operators (relations)

**Algebraic Compilation Based Properties** The direct translation of XQuery expression using the \textit{loop-lifting technique} results in relational algebraic plans that are relatively big. For example, the operators tree of XMark queries Q1 and Q2 consists of a total of 4349 and 1548 operators respectively. In [14], Grust has shown that further analysis on the resulting algebraic plans reveals the following two important observations:

1. The resulting algebraic plans always include several occurrences of common sub-plans. Based on this observation, it is more suitable and effective to use \textit{direct acyclic graphs (DAGs)} instead of operators tree for representing the resulting algebraic plans. Clearly, using DAGs instead of operator trees leads to a reduction on the number of the algebraic operators of the algebraic plans. For example, the number of the operators in the query plan of XMark queries Q1 and Q2 using DAGs is reduced to 131 and 98 operators. Further optimization and rewriting techniques on DAGs introduced by Grust, in [14] lead to a DAG plans consists of 39 and 41 operators for the same two XMark queries Q2 and Q8.

2. Due to the design of the \textit{loop-lifting} compilation rules, it is noticed that some of the algebraic operators have specific properties such as:
   - All join operators (\(\bowtie\)) are equi-join.
   - The projection operators (\(\pi\)) do not need to remove any duplicates.
   - All union operators (\(\cup\)) are receiving disjoint argument relations.

Based on these observations, it is possible infer a variety of properties for the algebraic operators and relations representing intermediate plan results. Table 3 list the properties we are able to infer and propagate during the plan analysis of the algebraic plan. The three properties (key, domain, constant) play a vital role in our proposed size estimation system for XQuery. Making use of these inferred properties for having accurate size estimations for the results of XQuery expressions will be described in Section 4.1. Figures 7 and 8 list the set of inference rules used for inferring and propagating these properties (key, domain, constant) through the resulting algebraic DAG plans. These inference rules are represented in the form:

\[ \frac{\text{Premise}_1 \quad \ldots \quad \text{Premise}_n}{\text{Conclusion}_1 \quad \ldots \quad \text{Conclusion}_m} \quad \text{(Inference Rule)} \]
Where each inference rule establishes a relationship between a set of premises and conclusions, whereby the conclusion is said to be inferable from the premises. In other words, whenever in the course of some logical derivation the given premises have been obtained, the specified conclusion can be taken for granted as well. Examples of the used notations for representing the special properties of the algebraic operators listed in table 3 are given as follows:

- The notation \((R.\text{card}_0)\) represents that the cardinality of the relation \(R\) is equal to 0 while the notation \((R.\text{card}_1)\) represents that the cardinality of the relation \(R\) is equal to 1.
- The notation \((R.\text{cols} : \{a, b\})\) represents that the schema of the relation \(R\) consists of the attributes \((a, b)\). While the notation \((a \in R.\text{cols})\) represents that the attribute \((a)\) belongs to the schema of the relation \(R\).
- The notation \((R.\text{key} : \{a, b\})\) represents that the attributes \((a, b)\) are the key attributes for the relation \(R\). While the notation \((a \in R.\text{key})\) represents that the attribute \((a)\) is a key attribute for the relation \(R\).
- The notation \((R.\text{domain} : \{a^{(\alpha)}, b^{(\beta)}\})\) represents that the schema of the relation \(R\) contains the attributes \((a, b)\), the domain identifier of the attribute \(a\) is \(\alpha\) and the domain identifier of the attribute \(b\) is \(\beta\). While the notation \((a^{(\alpha)} \in R.\text{domain})\) represents that the attribute \((a)\) with the domain identifier \((\alpha)\) belongs to the schema of the relation \(R\).
- The notation \((R.\text{const} : \{a^{(10)}, b^{(\text{True})}\})\) represents that the constant attributes of the relation \(R\) are the two constant attributes \((a, b)\) where the constant value of the attribute \(a\) is 10 and the constant value of the attribute \(b\) is \(\text{True}\). While the notation \((a^{(10)} \in R.\text{const})\) represents that the constant attribute \((a)\) with the value 10 belongs to the schema of the relation \(R\).

Remarks about the inference rules of the key property are given as follows:

- Generally, If applying any of the algebraic operators \((\text{OP})\) results in a relation with a cardinality property that is equal to 1, then all attributes of the resulting relation are key attributes (Rule KEY-1).
- Some of the algebraic operators are propagating all key attributes from their input relations to their output relation without any changes (Rule KEY-2). These operators are: the selection operator \((\sigma)\), the arithmetic operator \((\otimes)\), the comparison operator \((\boxtimes)\), the negation operator \((\neg)\) and the document access operator \((\Delta)\).
- The attachment operator \((\hat{\oplus})\) propagates all key attributes from its input relation to its output relation and the new attached attribute is a key if and only if the cardinality of the input relation is equal to 1 (Rule KEY-4 and Rule KEY-5).
- An attribute \((a)\) is a key attribute for the relation \((e_1 \sqcup e_2)\) if and only if \((a)\) is a key attribute in the relation \(e_1\) with a domain \((\alpha)\), \((a)\) is a key attribute in the relation \(e_2\) with a domain \((\beta)\) and \(\alpha, \beta\) are disjoint domains (Rule KEY-7).
- The difference operator \((\setminus)\) propagates all key attributes from its left hand side input relation to its output relation (Rule KEY-6).
Fig. 7. Inference rules of the Algebraic Compilation Based Properties (1)
\[
\begin{align*}
    & a^{(v)} \in e.\text{const} \quad a \in PL \\
    & a^{(v)} \in \pi_{PL}(e).\text{const} \quad \text{(CONST-1)} \\
    & \sigma_{a}(e).\text{const} : e.\text{const} \cup \{a^{(\text{True})}\} \\
    & \text{(CONST-3)} \\
    & (e_1 \setminus e_2).\text{const} : e_1.\text{const} \quad \text{(CONST-5)} \\
    & (e_1 \times e_2).\text{const} : e_1.\text{const} \cup e_2.\text{const} \quad \text{(CONST-7)} \\
    & \delta(e).\text{const} : e.\text{const} \quad \text{(CONST-8)} \\
    & \#_{a}(e).\text{const} : e.\text{const} \quad \text{(CONST-9)} \\
    & \text{e.card} \quad \text{(Const-10)} \\
    & \theta_{a(v_1,\ldots,v_n) / P(e).\text{const}} : e.\text{const} \quad \text{(CONST-11)} \\
    & \text{e.card} \quad \text{(Const-12)} \\
    & \bigotimes_{a(b,c)(e).\text{const}} : e.\text{const} \quad \text{(CONST-13)} \\
    & \text{(CONST-14)} \\
    & \text{(CONST-15)} \\
    & \text{item}^{(\text{False})} \in e.\text{const} \quad \text{(CONST-17)} \\
    & \text{item}^{(\text{True})} \in e.\text{const} \quad \text{(CONST-19)} \\
    & \Delta_{nitem} \text{item}(e).\text{const} : e.\text{const} \quad \text{(CONST-20)} \\
    & A^{(v)} \in A^{v}.\text{const} \quad \text{(CONST-21)} \\
    & A^{(v)} \in \text{const} \quad \text{(CONST-22)} \\
    & \text{Agg} \in \text{\{min, max, avg\}} \quad A^{(v)} \in e.\text{const} \quad \text{(CONST-23)} \\
    & \text{Agg}_{a(e).\text{const}} : e.\text{const} \cup \{a^{(v_1)}\} \quad \text{(CONST-24)} \\
\end{align*}
\]

Fig. 8. Inference rules of the Algebraic Compilation Based Properties (2)
– The cartesian product operator \((\times)\) propagates all key attributes of one input relation if the cardinality of the other relation is equal to 1 (Rule Key-8). In the case where the cardinalities of the two input relations are greater than one, the set of inferred key attributes is empty.

– The equi-join operator \((\bowtie)\) propagates all key attributes of the two input relations if and only if the two join attributes are key attributes in their relations (Rule Key-11).

– The equi-join operator \((\bowtie)\) propagates all key attributes of one input relation if and only if the join attribute of the other relation is a key attribute (Rule Key-12).

– If the duplicate elimination operator \((\delta)\) operates over a single attribute and this attribute is not a key then this attributes becomes a key after the application of the duplicate elimination operation (Rule Key-10).

– The unsorted row numbering operator \((\#)\) propagates all key attributes from its input relation to its output relation and the resulting row numbering attribute is a key attribute as well (Rule KEY-13).

– The sorted row numbering operator \((\varrho)\) propagates all key attributes from its input relation to its output relation and the resulting row numbering attribute is a key attribute if and only if the Row Numbering operator does not use a partitioning argument or if it uses a partitioning argument \((P)\) and this partitioning argument is a key in its relation (Rules Key-14 and Key-16).

– If the aggregation operator \((Agg)\) does not use a partitioning argument, then the resulting relation is a relation with one attribute and a single value, i.e this attribute is a key attribute (Rule Key-17).

– If the aggregation operator \((Agg)\) uses a partitioning argument \((p)\), then the resulting relation is a binary relation with two attribute \((p,v)\) where the aggregate value \((v)\) is computed for each partition value \((p)\). In this case, the resulting partitioning attribute \((p)\) is a key attribute (Rule Key-18).

– The XPath location step evaluator operator \((\mathcal{E\Delta})\) propagates all key attributes from its input relation to its output relation. The resulting node identifiers attribute is guaranteed to be unique because the result of the XPath location step evaluator operator is always ordered and duplicate-free (Rule Key-19).

Remarks about the inference rules of the domain property are given as follows:

– In the inference rules of the domain property, we use the annotation \(a^{(D+)}\) to represent the assignment of a new domain identifier \((D+)\) for the attribute \((a)\).

– Some of the algebraic operators commonly propagate all domain information of the attributes from their input relations to their output relation without any changes and additionally the operators create a new domain identifiers \((D+)\) for their resulting (extending) attributes \((r)\) (Rule Dom-1). These operators are: the attachment operator \((@)\), the unsorted row Numbering operator \((\#)\), the sorted row Numbering operator \((\varrho)\), the arithmetic operator \((\odot)\), the comparison operator \((\equiv)\), the negation operator \((\neg)\) and the document access operator \((\Delta)\).
The selection operator ($\sigma$) creates new sub-domains for all attributes of the output relation (Rule Dom-3).

The disjoint union operator ($\cup$) creates new super-domains for all attributes of the output relation (Rule Dom-4).

The difference operator ($\setminus$) creates new sub-domains for all attributes of the output relation (Rule Dom-5). If the schema of each of the two argument relations consists of only one attribute ($a$), then the domain of attribute ($a$) in the output relation is disjoint with the domain of the the same attribute in the right hand side input relation (Rule Dom-6).

The equi-join operator ($\times$) creates new sub-domains for all attributes of the output relation (Rule Dom-8) and propagates the domains of the join attributes using the rules Dom-9, Dom-10 and Dom-11.

The duplicate elimination operator ($\delta$) creates new sub-domains for all attributes of the output relation (Rule Dom-12).

If the aggregation operator ($Agg$) does not use a partitioning argument, then the resulting attribute is assigned a new domain identifier ($D+$) (Rule Dom-13).

If the aggregation operator ($Agg$) uses a partitioning argument ($p$), then the resulting partitioning attribute has the same domain identifier as the input relation and the resulting attribute is assigned a new domain identifier ($D+$) (Rule Dom-14).

The XPath location step evaluator operator ($\varnothing$) creates a new sub-domain for the iter column of the output context relation. This is because the resulting iterations from applying this operator are only the iterations which contain a context nodes satisfying the step conditions ($\alpha,n$). The XPath location step evaluator operator also creates a new domain identifier of the resulting item column of the output context relation (Rule Dom-16).

The constant property is used for identifying these attributes which store the same value ($v$) for all tuples. These attributes usually originate either from the use of literal tables or from the use of the attachment operators to attach a constant value to an intermediate result. Remarks about the inference rules of the constant property are given as follows:

- The attachment operator ($\& a:v$) propagates all constant attributes from its input relation to its output relation. The newly attached attribute(s) are always constant attributes (Rule Const-2).

- The selection operator ($\sigma a$) propagates all constant attributes from its input relation to its output relation and assigns the Boolean selection attribute ($a$) with the constant value of (True) (Rule Const-3).

- A constant attribute ($a^{(v)}$) is a constant attribute for the relation ($e_1 \cup e_2$) if and only if ($a^{(v)}$) is a constant attribute in the relation $e_1$ and ($a^{(v)}$) is a constant attribute in the relation $e_2$ (Rule Const-4).

- The row numbering operator ($\varnothing$) propagates all constant attributes from its input relation to its output relation and the resulting row numbering attribute is a constant attribute if and only if cardinality of the input relation is equal to 1 (Rules Const-11 and Const-12).
– The arithmetic operator (☉) propagates all constant attributes from its input relation to its output relation and the resulting attribute (a) is a constant attribute if the two argument attributes (b, c) are constants. In this case, the value of the resulting constant attribute is computed by applying the arithmetic operations (☉) over the values of the two argument attributes (Rules Const-13 and Const-14).

– The comparison operator (□) propagates all constant attributes from its input relation to its output relation and the resulting attribute (a) is a constant attribute if the two argument attributes (b, c) are constants. In this case, the value of the resulting constant attribute (True or False) is computed by applying the comparison operation (☉) over the values of the two argument attributes (Rules Const-15 and Const-16).

– The negation operator (¬) propagates all constant attributes from its input relation to its output relation and the resulting attribute (nitem) is a constant attribute if and only if the argument attribute (item) is constant. In this case, the value of the resulting constant attribute (True or False) is the negation for the constant Boolean value of the argument attribute (Rules Const-17, Const-18 and Const-19).

– If the aggregation operator (Agg) does not use a partitioning argument, then the resulting attribute is a constant attribute if \((\text{Agg} \in (\text{min, max, avg}))\) and the argument attribute is a constant attribute. In this case, the value of the resulting constant attribute is equal to the value of the argument attribute (Rule Const-23).

– If the aggregation operator uses a partitioning argument \((p)\), then the resulting partitioning attribute is a constant attribute if and only if it was a constant attribute in the input relation (Rule Const-24).

**Guide Node Annotation** Having an accurate estimation of XPath expression is a very crucial aspect for having accurate XQuery cardinality estimation system. In our presented algebraic compilation of XQuery expression, the XPath evaluator operator \((\mathcal{E}_{\text{Item}:(\alpha,n)})\) is used for representing the XPath steps. The XPath evaluator operator \((\mathcal{E}_{\text{Item}:(\alpha,n)})\) receives an input context relation \((ctx)\) with the schema \((\text{iter, item})\) where the item column stores the node identifiers of the input context nodes as well as an encoded XPath accelerator relation for the associated live nodes fragment \((\Gamma)\) and returns back the set of output context nodes from the evaluation the path step \((\alpha, n)\) over the input context nodes. In order to estimate the cardinality of the XPath evaluator operator \((\mathcal{E}_{\text{Item}:(\alpha,n)})\), we need to keep track of the node identifiers of the input context nodes. Hence, we represent a new annotation for the item columns storing the node identifiers of the context nodes, we name it as Guide Node annotation. An item column of the context relation \((ctx)\) annotated with the Guide Node property \(X\) indicates that the list of the node identifiers stored in the item are corresponding to the node with the pre-order rank \(X\) of the Statistical Guide. Having an input context relation where the item column is annotated by the Guide Node property \(X\). Applying the XPath evaluator operator \((\mathcal{E}_{\text{Item}:(\alpha,n)})\) over the context relation...
ctx annotates the item column of the resulting context relation with the Guide Node property equal to Y, where Y is the set of the pre-order rank(s) of the resulting node(s) from applying the path step (a,n) over the Guide Node X. A detailed explanation for the steps of estimating the cardinality of the XPath evaluator operator (⋈) will be presented in Section 4.1.

**Cardinality Related Properties** In addition to the set of the special properties that can be attached to the algebraic plans presented by Grust in [14], we extend these special properties by introducing a new set of cardinality related properties. Examples of the notations used for representing the cardinality related properties of the algebraic operators listed in Table 4 are given as follows:

- The notation (R.card : C) represents that the cardinality of the relation R is equal to C.
- The notation (a{gn} ∈ R.GuideNode) represents that the attribute (a) of the relation R stores a sequence of node identifiers associated with guide node with a pre-order rank (gn) on the statistical guide.
- The notation (a{h} ∈ R.Histogram) represents that the distribution of the values of the attribute (a) of the relation R is captured by the histogram definition (h).
- The notation (a{s} ∈ R.Selectivity) represents that the attribute (a) of the relation R is a Boolean attribute where the percentage of its true value is equal to s.
- The notation (a{n} ∈ R.DomainSize) represents that the values of the attribute (a) of the relation R are ranging between 1 and n.

Figures 9 and 10 illustrates the inference rules for inferring and propagating the values of the GuideNode, Histogram, Selectivity and DomainSize. Remarks about these rules are given as follows:

- If the attribute a in relation R is annotated with the GuideNode property \{gn1\} and the attribute a in relation S is annotated with the GuideNode property \{gn2\}, then the resulting attribute a in (R⊔S) combines the GuideNode information of both input attributes \{gn1, gn2\} (Rule GUIDE-3).

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>card : C</td>
<td>associated relation with cardinality equal to C.</td>
</tr>
<tr>
<td>GuideNode : a{gn}</td>
<td>attribute a is a node identifiers attribute for group of nodes represented by the guide node gn.</td>
</tr>
<tr>
<td>Histogram : a{h}</td>
<td>the distribution of values for the attribute a is captured by the histogram h.</td>
</tr>
<tr>
<td>Selectivity : a{s}</td>
<td>attribute a is a boolean attribute with a selectivity value equal to s.</td>
</tr>
<tr>
<td>DomainSize : a{n}</td>
<td>the values of attribute a is ranging between the values 1 and n.</td>
</tr>
</tbody>
</table>

**Table 4.** Cardinality related properties of the algebraic operators (relations)
\( a^{(gn)} \in R.GuideNode \quad \alpha \in \{ \emptyset, \sigma, \Delta, \#, \circ, \ominus, \neg, \Delta \} \quad \text{(GUIDE-1)} \)
\[
\frac{a^{(gn)} \in R.GuideNode}{a^{(gn)} \in \pi_{PL}(R).GuideNode} \quad \text{(GUIDE-2)}
\]
\[
\frac{a^{(gn)} \in R.GuideNode}{b^{(gn2)} \in S.GuideNode \quad OP \in \{ M, \times \} \quad \text{(GUIDE-4)}}
\]
\[
\frac{a^{(gn)} \in R.GuideNode}{a^{(gn)} \in (R\setminus S).GuideNode} \quad \text{(GUIDE-5)}
\]
\[
\frac{\gamma^{(h)} \in R.Histogram \quad OP \in \{ \emptyset, \sigma, \delta, \#, \neg, \ominus \}}{\gamma^{(h)} \in OP(R).Histogram} \quad \text{(HIST-1)}
\]
\[
\frac{\gamma^{(h)} \in R.Histogram}{\gamma^{(h)} \in \pi_{PL}(R).Histogram} \quad \text{(HIST-2)}
\]
\[
\gamma^{(h)} = \text{getHistogram}(\gamma^{(h)}), \quad \text{value}^{(h)} \in \Delta_{\text{value}}(\gamma^{(h)}) \quad \text{(HIST-3)}
\]
\[
\frac{x^{(v1)}, y^{(v2)} \in R.const}{x^{(v1)}, y^{(v2)}, z^{(v1,v2)} \in \otimes_{x,(x,y)}(R).Const} \quad \text{(HIST-7)}
\]
\[
\frac{x^{(H1)} \in R.Histogram \quad y^{(H2)} \in R.Histogram}{z^{(H3)} \in \otimes_{x,(x,y)}(R).Histogram} \quad \text{(HIST-8)}
\]

\( H : \{ H_1, H_2, \ldots, H_n \} \quad H_1 \in \{ H_1, H_2, \ldots, H_n \} \quad x^{(H1)} \in R.Histogram \quad \text{value}^{(v)} \in R.const \quad H_2 : \{ H_2, H_2, \ldots, H_2 \} \quad \text{H2} = \text{ArithHistConst}(\circ, H_1, v) \quad H_2.\text{weight} = H_1.\text{weight} \quad \text{H2} \in (R\setminus S).GuideNode \quad \text{(HIST-9)}
\]

Fig. 9. Inference rules of the cardinality related properties (1)
\[ a \in \{ \text{Selectivity} \} \quad \text{OP} \in \{ \oplus, \sigma, \delta, \#, \#, \#, \Delta, \oplus \} \quad a \in \OP(R).\text{Selectivity} \quad \text{SEL-1} \]

\[ a \in \pi_{PL}(R).\text{Selectivity} \quad \text{SEL-2} \]

\[ a \in \{ \text{Selectivity} \} \quad \exists \text{PL} \quad a \in \pi_{PL}(R).\text{Selectivity} \]

\[ \frac{R.\text{card} : C_1 \quad S.\text{card} : C_2 \quad a \in \{ \text{Selectivity} \} \quad \exists \text{PL} \quad a \in \pi_{PL}(R).\text{Selectivity} \quad \text{SEL-3} }{a \in \{ \text{Selectivity} \}} \]

\[ \frac{H : \{ H_1, H_2, \ldots, H_n \} \quad x^{(H)} \in \text{R.Histogram} \quad y^{(\sigma)} \in \text{R.const} \quad s_i = \text{CompareHistConst}(\sigma, H_i, \nu) \quad s = \sum_{i=1}^{n} s_i \text{weight} \quad z^{(\ast)} \in \text{R.Selectivity} \quad \text{SEL-8} } {z^{(\ast)} \in \text{R.Selectivity} \quad \text{SEL-9} } \]

\[ \frac{\text{a}^{(n)} \in \text{R.DomainSize} \quad \text{OP} \in \{ \oplus, \sigma, \delta, \#, \#, \#, \Delta \} \quad \text{a}^{(n)} \in \OP(R).\text{DomainSize} \quad \text{DOMSIZE-1} } {\text{a}^{(n)} \in \pi_{PL}(R).\text{DomainSize} \quad \text{DOMSIZE-2} } \]

\[ \frac{\text{a}^{(n_1)} \in \text{R.DomainSize} \quad \text{a}^{(n_2)} \in \text{S.DomainSize} \quad \exists \text{Max}(n_1, n_2) \quad \text{Max}(n_1, n_2) \in \text{R.\text{DomainSize}} \quad \text{DOMSIZE-3} } {\text{a}^{(n_1)} \in \pi_{PL}(R).\text{DomainSize} \quad \text{DOMSIZE-4} } \]

\[ \frac{\text{a}^{(n_1)} \in \text{R.DomainSize} \quad \text{b}^{(n_2)} \in \text{S.DomainSize} \quad \text{OP} \in \{ \oplus, \times \} \quad \text{a}^{(n_1)} \in \pi_{PL}(R).\text{DomainSize} \quad \text{DOMSIZE-5} } {\text{b}^{(n_2)} \in \pi_{PL}(R).\text{DomainSize} \quad \text{DOMSIZE-6} } \]

\[ \frac{\text{R.card} : C \quad \text{b}^{(n)} \in \text{R.DomainSize} \quad \text{C}^{(n)} \in \text{R.DomainSize} \quad \text{b}^{(n)} \in \text{R.\text{DomainSize}} \quad \text{DOMSIZE-7} } {\text{C}^{(n)} \in \text{R.\text{DomainSize}} \quad \text{DOMSIZE-8} } \]

Fig. 10. Inference rules of the cardinality related properties (2)
If the attribute \textit{item} in the input context relation \( R \) is annotated with the \textit{GuideNode} property \((\text{Ign})\) (the set of input guide nodes), then applying the XPath location step evaluator operator \((\mathcal{E}_{\text{item}(\alpha,n)})\) over the input context relation \( R \) changes the annotation of the attribute \textit{item} in the resulting relation by the set of the guide nodes \((\text{Ogn})\) resulting from applying the step \((\alpha,n)\) over the set of guide nodes \((\text{Ign})\) in the \textit{statistical guide} (Rule \text{Guide-6}).

The document access operator \((\Delta_{\text{value:item}})\) uses the guide node(s) information annotated to the \textit{item} column of the context relation to retrieve their associated histogram(s) from the statistical repository. The retrieved histogram(s) are annotated to the \textit{Histogram} property of the resulting \textit{value} attribute (Rule \text{Hist-3}). If the the \textit{item} column of the context relation is annotated with \textit{more than one} guide node information, then each of the retrieved histograms is assigned a \textit{weight} of a participation \((H_i.weight)\) which is equal to the computed percentage by dividing the size of its associated guide node \((\text{StatisticalGuide.getSize}(gns_i))\) by the total size of \textit{all} annotated guide nodes \((\text{StatisticalGuide.getSize}(gns))\). Another alternative in this situation is to \textit{merge} the retrieved histograms for the set of the guide nodes into \textit{one} histogram. However, we prefer to use and maintain the \textit{weight} of each histogram for gaining more accurate estimation of the \textit{selectivity} values by applying the comparison operators. If the \textit{item} column is annotated with \textit{only one} guide node, then the weight percentage of the retrieved histogram is naturally equal to 100\%. In fact, the document access operator is responsible for retrieving the atomic values of the context nodes. Consequently, the document access operator is the algebraic operator which is responsible for introducing the \textit{Histogram} property into the algebraic plans.

If the attribute \(a\) in relation \(R\) is annotated with the \textit{Histogram} property \(X\) and the attribute \(a\) in relation \(S\) is annotated with the \textit{Histogram} property \(Y\) (each of \(X\) and \(Y\) represents a \textit{set} of histograms which may contain one or more histogram), then the resulting attribute \(a\) in \((R\cup S)\) combines all the histogram information of both input attributes where the \textit{weight} of each element histogram in the resulting set of histograms \(Z\) \((Z.\text{Histogram}(X_i).weight\text{ and } Z.\text{Histogram}(Y_j).weight)\) is computed by dividing the computed values of associated tuples in the source relation \((X.\text{Histogram}(X_i).weight \cdot C1\text{ and } Y.\text{Histogram}(Y_j).weight \cdot C2)\) by the total number of the tuples of the resulting relation \((C1 + C2)\) where \(C1\) is the cardinality of the relation \(R\) and \(C2\) is the cardinality of the relations \(S\) (Rule \text{Hist-4}).

The instances of the arithmetic operators \((\circ_{z(x,y)})\) in the \textit{Pathfinder} algebraic plans come in one of the following three situations:

- The argument attribute \(x\) is a constant attribute with the value \(v1\) and the argument attribute \(y\) is a constant attribute with the value \(v2\). In this case the resulting attribute \(z\) is a constant attribute with the value \((v1 \circ v2)\) (Rule \text{Hist-7}).
- The argument attribute \(x\) is a constant attribute with the value \(v\) and the argument attribute \(y\) is annotated with the \textit{Histogram} property equal to \(H1\) where \(H1\) represents a set of histograms. In this case the resulting
attribute \( z \) is annotated with the \textit{Histogram} property equal to \( H_2 \). The set of histograms \( H_2 \) is a result of manipulating each element of the set of histograms \( H_1 \) with the arithmetic operation \( \circ \) and the constant value \( v \) (Rule Hist-9). In Rule Hist-9, the function \textit{ArithHistConst} represents the process of histogram manipulation using a constant value discussed in Section 3.3.

- The argument attribute \( x \) is annotated with the \textit{Histogram} property equal to the set of histograms \( H_1 \) and the argument attribute \( y \) is annotated with the \textit{Histogram} property equal to the set of histograms \( H_2 \). In this case the resulting attribute \( z \) is annotated with the \textit{Histogram} property equal to the set of histograms \( H_3 \). The set of histograms \( H_3 \) is a result of manipulating the histogram(s) of \( H_1 \) with the arithmetic operation \( \circ \) and the histogram(s) of \( H_2 \) (Rule Hist-8). In Rule Hist-8, the function \textit{ArithHistHist} represents the process of applying arithmetic operations on histograms discussed in Section 3.3.

- If the attribute \( a \) in relation \( R \) is annotated with the \textit{Selectivity} property \( s_1 \) and the attribute \( a \) in relation \( S \) is annotated with the \textit{Selectivity} property \( s_2 \), then the \textit{Selectivity} property of the resulting attribute \( a \) in \((R \cup S)\) is computed by dividing the sum of the tuples with the \textit{True} value in \( R \) and \( S \) by the total sum of all tuples in \( R \) and \( S \) (Rule Sel-3).

- The instances of the comparison operators \((\boxcircle z; (x, y))\) in the Pathfinder algebraic plans come in one of the following three situations:
  - The argument attribute \( x \) is a constant attribute with the value \( v_1 \) and the argument attribute \( y \) is a constant attribute with the value \( v_2 \). In this case the resulting Boolean attribute \( z \) is either annotated with the selectivity property equal to 1, if \((v_1 \circ v_2)\) is equal to true, or with the selectivity property equal to 0, if \((v_1 \circ v_2)\) is equal to false (Rule Sel-7). In Rule Sel-7, the function \textit{CompareConstConst} evaluates the comparison operator \((v_1 \circ v_2)\) and returns the equivalent selectivity value.
  - The argument attribute \( x \) is annotated with the \textit{Histogram} property equal to the set of histograms \( H \) and the argument attribute \( y \) is a constant attribute with the value \( v \). In this case the resulting Boolean attribute \( z \) is annotated with the \textit{Selectivity} property equal to the set of histograms \( H \) and the argument attribute \( y \) is a constant attribute with the value \( v \). In this case the resulting Boolean attribute \( z \) is annotated with the selectivity property equal to the set of histograms \( H \) and the argument attribute \( y \) is a constant attribute with the value \( v \). In this case the resulting Boolean attribute \( z \) is annotated with the \textit{Selectivity} property equal to \( (s) \) where the values of \( (s) \) is computed using the following two steps (Rule Sel-8):
    1. Computing the intermediate selectivity values \( s_i \) by comparing each element histogram \( H_i \) with the constant value \( v \) using the inference rules of predicate selectivity estimation using the histograms illustrated in Figure 6.
    2. The final resulting selectivity values \( s \) is equal to the weighted average of the intermediate selectivity values \( s_i \). It is computed by summing the values resulting from multiplying each intermediate selectivity values \( s_i \) with the weight of each associated histogram \( H_i.weight \).

In Rule Sel-8, the function \textit{CompareHistConst} evaluates the intermediate selectivity values \( s_i \) using the already mentioned inference rules (Figure 6).
The argument attribute $x$ is annotated with the *Histogram* property that is equal to the set of histograms $H_1$ and the argument attribute $y$ is annotated with the *Histogram* property that is equal to the set of histograms $H_2$. In this case the resulting Boolean attribute $z$ is annotated with the *Selectivity* property equal to $(s)$ where $(s)$ is computed using the following three steps (Rule SEL-9):

1. Computing the intermediate selectivity values $s_i$ results from applying the comparison operator $\circ$ over the values of each pair of histograms $(H_{1i}, H_{2j})$.
2. Each intermediate selectivity value $s_i$ is assigned a *weight* value equals to the resulting value from multiplying the weights of its associated pair of histograms $(s_{1i}, s_{2j})$.
3. The final resulting selectivity values $s$ is equal to the weighted average of the intermediate selectivity values $s_{ij}$. It is computed by summing the values resulting from multiplying each intermediate selectivity values $s_{ij}$ with its computed weight $w_{s_{ij}}$.

In Rule SEL-9, the function $\text{CompareHistHist}$ represents the process of histograms comparison discussed in Section 3.3.

- If the Boolean attribute $\text{item}$ in relation $R$ is annotated with the *Selectivity* property equal to $(s)$ then, applying the negation operator $(\neg)\text{item, item}(R)$ annotates the resulting Negated Boolean attribute $\neg\text{item}$ with the *Selectivity* property equal to $(1 - s)$.
- If the attribute $a$ in relation $R$ is annotated with the *DomainSize* property $(n_1)$ and the attribute $a$ in relation $S$ is annotated with the *DomainSize* property $(n_2)$ then, the resulting attribute $a$ in $(R \cup S)$ is annotated with the *DomainSize* property equal to the maximum value of $n_1$ and $n_2$ (Rule DomSize-3).
- If the relational $R$ is annotated with the cardinality property equal to $C$, then applying the the unsorted row numbering operator $(\#a)$ or the sorted row numbering operator with no partitioning argument $(\rho a; (o_1, ..., o_n))$ over the input relation $R$ annotates the resulting numbering attribute $a$ with *DomainSize* property equal to $C$ (Rules DomSize-6, DomSize-7).

### 4 XQuery Cardinality Estimation

#### 4.1 Estimation Rules

This section presents the main part of the estimation process which integrates the different building blocks of the XQuery cardinality estimation framework. In this section we present the *inference rules* for estimating the cardinality of XQuery expression and its sub-expressions using the previously mentioned building blocks. As previously mentioned, Pathfinder compiles the input XQuery expression into its equivalent algebraic plans. The XQuery expressions are compositional in nature as sub-expressions are combined with each other to form the final query. Our approach is able to estimate the cardinality of the final XQuery expression as well as its sub-expressions by estimating the cardinalities of the...
algebraic operators in the query plan representing the sub-expression in addition to the estimation of the final resulting operator. Figures 11 and 12 illustrate the inference rules for estimating the cardinalities of Pathfinder algebraic operators.

In Section 4.2 we will provide an example to describe the use of these inference rules in the estimation process.

Remarks about the estimation inference rules are given as follows:

- Since the two input relations \( R, S \) of the union operator \( (\cup) \) are guaranteed to be always disjoint, the estimated cardinality of \((R \cup S)\) is equal to the sum of the cardinality of \( R \) \((C_1)\) plus the cardinality of \( S \) \((C_2)\) (Rule Card-2).

- The cardinality of the resulting relation from applying the aggregate operator \((\text{Agg})\) is equal to 1 in the case when it does not use a partitioning attribute or if the partitioning attribute \( a \) is constant attribute (Rules Card-9 and CARD-10). The cardinality of the resulting relation from applying the aggregate operator \((\text{Agg}_{v:a/p}(R))\) is equal to the cardinality of the input relation \( R \) in the case when the partitioning attribute \( (p) \) is a key attribute (Rule Card-11). The estimated cardinality of the resulting relations is equal to \( n \) if the partitioning attribute \( (p) \) is not a key and it is annotated with a DomainSize property that is equal to \( n \). constant (Rule Card-12).

- An accurate estimation of XPath location steps is crucial for an accurate estimation of the whole XQuery expression. Assuming that the cardinality of the input relation \( R \) is equal to \( C_1 \) and the item column storing the node identifier of the input context nodes is annotated with the set of guide node(s) \((Igns)\), the estimated cardinality of the operator \( \mathcal{L}_{(\alpha,n)}(R) \) is achieved through the following steps represented in Rule CARD-15:

  (i) Using our summarized tree structure, the Statistical Guide, we can get the sum of cardinalities of the input guide node(s) \((Igns)\) represented as \((C_2)\).

  (ii) We apply the location step \((\alpha, n)\) over the statistical guide with the input context guide nodes \((Igns)\) and the resulting guide nodes \((Ogns)\) are annotated to the resulting item column.

  (iii) Once again, the Statistical Guide is used to get the sum of cardinalities of the output guide node(s) \((Ogns)\) represented as \((C_3)\).

  (iv) Using the cardinality of the input relation \( (C_1)\), the cardinality of the input guide nodes \( (C_2)\) and the cardinality of the output guide nodes \( (C_3)\), the estimated cardinality of the operator \( \mathcal{L}_{(\alpha,n)}(R) \) is calculated as follows:

\[
(\frac{(C_1)}{(C_2)}) \times (C_3)
\]

- The estimated cardinality of the resulting relation from applying the selection operator \((\sigma_a(R))\) is equal to the cardinality of the input relation \( R \) multiplied by the value of the Selectivity property of the input selection attribute \( (a) \) (Rule Card-13).

- The estimated cardinality of the resulting relation from applying the join operator \((\Join_{a=b})\) is very dependent on the key and domain properties of the join attributes of the input relations. Many different situations can occur in this context hence, four different inference rules have been defined to
\[ R.\text{card} : C \quad OP \in \{\pi, @, \delta, \#, \odot, \oslash, \neg\} \quad \text{(CARD-1)} \]

\[ R.\text{card} : C \quad (\text{R} \cup \text{S}).\text{card} : C + C2 \quad \text{(CARD-2)} \]

\[ R.\text{cols} = \text{S.cols} = \{a\} \quad \text{(CARD-3)} \]

\[ R.\text{cols} = \text{S.cols} = \{a\} \quad \text{(CARD-4)} \]

\[ R.\text{card} : C1 \quad \text{S.card} : C2 \quad \alpha \subseteq \beta \quad \text{(CARD-5)} \]

\[ R.\text{cols} = \text{S.cols} = \{a\} \quad \alpha \subseteq \beta \quad \text{(CARD-6)} \]

\[ R.\text{card} : C \quad \text{a}(\alpha) \in \text{R.key} \quad \text{a}(\beta) \in \text{S.key} \quad \beta \subseteq \alpha \quad \text{(CARD-7)} \]

\[ R.\text{card} : C \quad \text{p} \in \text{R.const} \quad \text{Agg}_{\text{a}/p}(\text{R}).\text{card} : C \quad \text{(CARD-8)} \]

\[ R.\text{card} : C \quad \text{p}(\alpha) \in \text{R.Domsizem} \quad \text{Agg}_{\text{a}/p}(\text{R}).\text{card} : n \quad \text{Agg}_{\text{a}/p}(\text{R}).\text{card} : (\text{CARD-9}) \]

\[ R.\text{card} : C \quad \text{a}(\alpha) \in \text{R.Selectivity} \quad \text{(CARD-10)} \]

\[ \sigma_{\text{a}}(\text{R}).\text{card} : C \quad \text{(CARD-11)} \]

\[ \text{e1.card} : C1 \quad \text{e2.card} : C2 \quad \text{(CARD-12)} \]

\[ R.\text{card} : C \quad \alpha \subseteq \beta \quad \text{(CARD-13)} \]

\[ \delta_{\text{a}/p}(\text{R}).\text{card} : (\text{CARD-14)} \]

\[ \text{R.card} : C1 \quad \text{S.card} : C2 \quad \text{a}(\alpha) \in \text{R.key} \quad \text{b}(\beta) \in \text{S.key} \quad \text{(CARD-15)} \]

\[ \text{(R \text{Min} a \text{b} S).card} : C1 \quad \text{(CARD-16)} \]

\[ R.\text{card} : C1 \quad \text{S.card} : C2 \quad \alpha \subseteq \beta \quad \text{(CARD-17)} \]

\[ \text{(R \text{Min} a \text{b} S).card} : \text{Min}(C1, C2) \quad \text{(CARD-18)} \]

\[ R.\text{card} : C1 \quad \text{S.card} : C2 \quad \text{a}(\alpha) \in \text{R.key} \quad \text{b}(\beta) \in \text{S.key} \quad \text{(CARD-19)} \]

\[ \text{(R \text{Min} a \text{b} S).card} : 1 \quad \text{a}(\alpha) \in \text{R.key} \quad \text{(CARD-20)} \]

\[ \text{(CARD-21)} \]

\[ \text{(CARD-22)} \]

**Fig. 11.** The inference rules for estimating the cardinalities of Pathfinder algebraic operator (1).
The inference rules for estimating the cardinalities of Pathfinder algebraic operator (2)
dead with the different possible instances of the join operators (Rules CARD-16, CARD-17, CARD-18, CARD-19 and CARD-20). Since some instances of the join operators may fit with the preconditions of more than one size estimation inference rule, the instances of the join operations are checked against the preconditions of the specified inference rules in a deterministic order as specified in Figure 12. Rule CARD-20 represents the else case of these set of the inference rules which is used when we do not have sufficient information about the domain and key properties of the join attributes. In this case, we use the System R rule which estimates the cardinality of the resulting relation to be equal to \( \frac{1}{10} \) of the Cartesian product of the input relations [30].

Generally, the cardinality estimation of the duplicate elimination operator (\( \delta \)) is very difficult and not as straightforward in the general case because it is very dependant on the values of the attributes of the tuples of the input relation. However, we have provided different estimation rules for handling the most common cases for instances of this operator. These cases are listed as follows:

(i) The estimated cardinality of the resulting relation from applying the duplicate elimination operator (\( \delta(R) \)) is equal to the cardinality of the input relation \( R \) if any of the attributes of the input relation is a key attribute (Rule CARD-21).

(ii) If the schema of the input relation \( R \) consists of only one attribute \( a \) and this attribute is a constant attribute then the cardinality of the resulting relation is equal to 1 (Rule CARD-22).

(iii) If the schema of the input relation \( R \) consists of only one attribute \( a \) and this attribute is annotated with the Histogram property that is equal
to $H$, the cardinality of the resulting relation is equal to the cardinality of the input relation $R$ multiplied by the percentage of the unique values of the data represented by the histogram $H_1$ (Rule CARD-23).

(iv) If the schema of the input relation $R$ consists of only one attribute $a$ and this attribute is annotated with a DomainSize property that is equal to $n$, assuming that the cardinality of the input relation $R$ is equal to $C_1$, the cardinality of the resulting relation would then equal to the minimum value between $C$ and $n$ (Rule CARD-24).

(v) For cases where the schema of the input relation $R$ consists of more than one attribute where all attributes are constant attributes with the exception of an attribute $a$, if the attribute $a$ is annotated with the Histogram property that is equal to $H$ then the estimation is done in the same way as in case (iii) (Rule CARD-25), if the attribute $a$ is annotated with the DomainSize property equal to $n$ then the estimation is done in the same way as described in case (iv) (Rule CARD-26).

In the case where the status of the duplicate elimination operator does not fit with the preconditions of any of the mentioned cases, then we estimate the cardinality of the resulting relation to be equal to the cardinality of the input relation.

4.2 Estimation Process

Having the algebraic compilation of the XQuery expressions, the special properties of the the algebraic plans, the statistical guide, the histogram repository and the estimation inference rues of the algebraic operators, the XQuery cardinality estimation in our framework becomes a very straightforward process. The estimation process is mainly achieved through two main steps. Firstly, the input XQuery expression is compiled into its equivalent algebraic plan. Secondly, the cardinality estimation of the input XQuery expression and its sub-expression is done by traversing the DAG of the algebraic operators in a post-order approach and then estimating the cardinality of each algebraic operator using its defined estimation rules (Section 4.1). Each sub-expression is associated with an algebraic operator in the DAG plan and the estimated cardinality of each intermediate algebraic operator is used as an input for the immediate following operator(s) in the algebraic DAG plan. To illustrate we present an example of the estimation process of an XQuery expressions using its associated compiled algebraic plan and the estimation rules. In this example, we use the XMark sample document "auction.xml" as a source XML document where Figure 13 represents its associated Statistical Guide. In the Statistical Guide figure, we used the following notations:

- The notation [...] is used to replace the sub-trees which are not relevant for our sample queries.
- All nodes of the statistical guide are annotated on their left sides with their associated pre-order ranks.
The interesting nodes for our examples are annotated on their right-hand side with their associated number of occurrences.

**Fig. 13.** The Statistical Guide for the XMark document.

**Example 1** Find all price nodes with value less than 50

```xml
Q1
for $x$ in doc("auction.xml")//price/text()
  where $x < 50$
  return $x$
```

Figure 14(a) illustrates the *Pathfinder* algebraic plan for the XQuery query Q1. Figure 15 lists the operators of the algebraic plans and the annotations of the special properties related to the estimation process. The columns with the title *GNode* represent the annotation of the *GuideNode* property while the column with the title *Hist* represents the annotation of the *Histogram* property. Figure 14(b) illustrates the distribution of the price values ($H1 - GuideNode 51$). Remarks about the inferred properties and cardinalities of the algebraic operators are given as follows:

- The *(Doc)* operator retrieves the document node from the XPath accelerator encoding relation of the "auction.xml" document. Hence, the resulting
(a) Algebraic plan for Q1.

(b) Statistical histogram for the values of the price node.

Fig. 14. Estimation components of the XQuery Q1.
<table>
<thead>
<tr>
<th>ID</th>
<th>key</th>
<th>const</th>
<th>GNode</th>
<th>Hist</th>
<th>Selectivity</th>
<th>card</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(Serialize)</td>
<td>item, pos</td>
<td>iter, item, pos</td>
<td>item (51)</td>
<td></td>
<td>90</td>
</tr>
<tr>
<td>2</td>
<td>(\π_{item, pos})</td>
<td>iter, item, pos</td>
<td>iter</td>
<td>item (51)</td>
<td></td>
<td>90</td>
</tr>
<tr>
<td>3</td>
<td>(\π_{iter})</td>
<td>iter</td>
<td>ite</td>
<td></td>
<td>1</td>
<td>90</td>
</tr>
<tr>
<td>4</td>
<td>(σ_{item})</td>
<td>iter</td>
<td>iter</td>
<td>item</td>
<td>0.9</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>(\π_{iter, item})</td>
<td>iter</td>
<td>iter</td>
<td>item</td>
<td>0.9</td>
<td>100</td>
</tr>
<tr>
<td>6</td>
<td>(\lt_{item, item1, item2})</td>
<td>iter</td>
<td>item2 (50)</td>
<td>item1 (h1)</td>
<td></td>
<td>0.9</td>
</tr>
<tr>
<td>7</td>
<td>(\lt_{item2, 50})</td>
<td>iter</td>
<td>item2 (50)</td>
<td>item1 (h1)</td>
<td></td>
<td>100</td>
</tr>
<tr>
<td>8</td>
<td>(\lt_{item1, item1})</td>
<td>iter, item</td>
<td>iter, item</td>
<td>item (51)</td>
<td></td>
<td>100</td>
</tr>
<tr>
<td>9</td>
<td>(\lt_{iter, item1})</td>
<td>iter</td>
<td>iter</td>
<td>item</td>
<td>0.9</td>
<td>100</td>
</tr>
<tr>
<td>10</td>
<td>(\lt_{iter, item})</td>
<td>iter, item</td>
<td>iter, item</td>
<td>item (51)</td>
<td></td>
<td>100</td>
</tr>
<tr>
<td>11</td>
<td>(\lt_{#iter})</td>
<td>iter, item, pos</td>
<td>iter, item, pos</td>
<td>item (51)</td>
<td></td>
<td>100</td>
</tr>
<tr>
<td>12</td>
<td>(\lt_{pos1, item})</td>
<td>iter, item, pos</td>
<td>iter, item, pos</td>
<td>item (51)</td>
<td></td>
<td>100</td>
</tr>
<tr>
<td>13</td>
<td>(\lt_{item1})</td>
<td>iter, item, pos</td>
<td>iter, item, pos</td>
<td>item (51)</td>
<td></td>
<td>100</td>
</tr>
<tr>
<td>14</td>
<td>(\lt_{item})</td>
<td>iter, item, pos</td>
<td>iter, item, pos</td>
<td>item (51)</td>
<td></td>
<td>100</td>
</tr>
<tr>
<td>15</td>
<td>(\lt_{desc, price})</td>
<td>iter, item, pos</td>
<td>iter, item, pos</td>
<td>item (51)</td>
<td></td>
<td>100</td>
</tr>
<tr>
<td>16</td>
<td>(\lt_{desc, closed_auctions})</td>
<td>iter, item, pos</td>
<td>iter, item, pos</td>
<td>item (51)</td>
<td></td>
<td>100</td>
</tr>
<tr>
<td>17</td>
<td>(doc)</td>
<td>iter, item</td>
<td>iter, item</td>
<td>item (0)</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

Fig. 15. Annotated properties of operators of algebraic plan of Q1

- The item column is annotated with the Guide Node property equal to 0 referring to the root node of the Statistical Guide of the "auction.xml" document.
- The operators \(\lt_{item1, item1}\) \(\lt_{iter, item}\) and \(\lt_{#iter}\) are instances of the XPath evaluator operator (\(\lt\)). Operator \(\lt_{item1, item1}\) represents the evaluation of the path step (descendant :: closed_auctions) while the operator \(\lt_{iter, item}\) represents the evaluation of the path step (descendant :: price). The notation desc is used as an abbreviation for the descendant axis.
- The estimated cardinalities of the XPath evaluator operators (\(\lt\)) \(\lt_{item1, item1}\) \(\lt_{iter, item}\) are done on the basis of the inference rule Card-15. For operator number \(\lt_{item1, item1}\), applying the path step (descendant :: closed_auctions) over the statistical guide with the input context guide node (GuideNode 0) yields to the guide node with the pre-order rank equal to 47 (GuideNode 47) and the number of occurrences is equal to 1. Hence, the estimated cardinality for this operator is equal to 1 and the resulting item attribute is annotated with the GuideNode property equal to 47. Similarly, for operator \(\lt_{iter, item}\) applying the path step (descendant :: price) over the statistical guide with the input context guide node (GuideNode 47) yields to the guide node with the pre-order rank equal to 51 (GuideNode 51) where the number of occurrences is equal to 100. The estimated cardinality for this operator is equal to 100 and the resulting item attribute is annotated with the GuideNode property equal to 51.
- The document access operator (\(\lt_{item1, item}\)) \(\lt_{doc}\) uses the GuideNode property of the input item attribute to annotate the resulting attribute item1 with its associated histogram H1 capturing the distribution of the values of the price values (GuideNode 51). Simultaneously, the estimated cardinality of the resulting relation is equal to the cardinality of the input relation.
The comparison operator \( <item;item1, item2> \) applies a comparison operation \( < \) between the \( item1 \) attribute annotated with the *Histogram* property equal to \( H1 \) and the \( item2 \) attribute annotated with the constant value 50 resulting from the attachment operator \( @item2;50 \). The *Selectivity* property of the resulting Boolean attribute \( item \) is computed on the basis of the inference rule \( \text{Pred-6} \). Using the following information of the histogram \( H1 \):

- The histogram size is equal to 100.
- The constant value 50 lies within the range of the third bucket \([40, 60]\)
  where the size of the bucket is equal to 20 and the accumulative size of
  the values that are less than 40 is equal to 80.

the selectivity of the resulting Boolean attribute \( item \) is computed as follows:

\[
(80 + \frac{50 - 40}{60 - 40} \times 20))/100 = 0.9
\]

The estimated cardinality of the resulting relation from applying the document access operators is equal to the estimated cardinality of the input relation.

- The estimated cardinality of the resulting relation from applying the selection operator \( \sigma_{item} \) is computed by multiplying the value of the annotated selectivity property of the selection attribute \( item \) (0.9) by the estimated cardinality of the input relation (100).

- The operator \( \mu_{iter=iter} \) applies a join operation between the resulting relations from applying the operators \( 4 \text{ and } 12 \). Each of the input relations has its join attribute \( iter \) as a key attribute. Actually, estimating the cardinality of the resulting relation from the join operators depends mainly on the information provided by the *key* and *domain* properties of the join attributes. Let us Assume that \( iter^{(\alpha)} \) belongs to the domain information \( (R.domain) \) of the resulting relations from applying the operators \( 4 \). Using the inference rules of the domain property, the domain information of the \( iter \) column \( iter^{(\alpha)} \) will be propagated up through the algebraic plan till it reaches the selection operator \( 5 \). Applying the selection operator \( \sigma_{iter} \) will annotate the resulting \( iter \) attribute with the domain property equal to \( (\beta) \) where \( \beta \subseteq \alpha \). Consequently, the \( iter \) column of the resulting relation from applying the project operator \( \pi_{iter} \) will be annotated with the domain information \( (\beta) \). In this situation, the estimated cardinality of the resulting relation from applying the join operator \( 3 \) is derived using the Rule \( \text{CARD-17} \) where the join attribute of each input relation is a key attribute and the domain of one join attribute is a subset of the domain other join attribute.

- Additionally, the cardinality estimation of the sub-expressions could be inferred by detecting the corresponding relationship between each sub-expression and its associated algebraic operator. For example, the operator \( 15 \) with estimated cardinality equal to 100 corresponds to the evaluation of the sub-expression \( doc("auction.xml")//price \) while the the operator \( 3 \) with es-
estimated cardinality equal to 90 corresponds to the evaluation of the sub-expression $x < 50$. 

– Using the loop-lifted compilation of the FLWOR expression and the Loop scope relations described in Section 2.2, we are able to estimate the cardinality of each iteration in the scope of the FLWOR expression. A description of the estimation of the cardinality of each iteration in the FLWOR expression will be presented in the experiments of Section 6.2.

5 Proposed Benchmark for XQuery Cardinality Estimation Systems

The XML research community has proposed several benchmarks such as [5], [29], [33], [22] which are very useful for their targets and perspectives. However, none of these benchmarks fits in our context. As such, in our context we need a benchmark which is designed with a main focus to measure the accuracy of our XQuery cardinality estimation systems with their different aspects. In this section we present our proposed benchmark for XML query size estimation. Unlike the usual XML database benchmarks, the aim of our proposed benchmark is to establish the basis of different estimation approaches in the XML domain in terms of the accuracy of the estimations and its completeness in handling different XML querying features. The benchmark consists of six groups of queries where each group is intended to address the challenges posed by the different aspects of XML query result size estimation. The individual queries in each covers the major possibilities of XML querying capabilities.

The queries of our proposed benchmark are based on the well-known sample XML document of the XMark benchmark "auction.xml". The structure of this XML document is described in details in [29]. We also reuse some of the XMark queries but from other perspectives serving our cardinality estimation targets. Tables 5 and 6 list the descriptions of the queries of our proposed benchmark and the specific XQuery cardinality estimation aspect handled by each query. The appendix of this article will give a complete description for the queries of our proposed benchmark.

6 Experiments

This sections presents as an assessment for the quality of the estimates produced by our proposed XQuery estimation framework. The experiments of this section have the following goals:

– To demonstrate that our proposed XQuery cardinality estimation system has a high accuracy rate.
– To demonstrate the ability of our estimation system to provide accurate cardinality estimations not only for the main XQuery expression but also for its sub-expressions as well as the estimated cardinality per each iteration in the scope of the FLWOR expression.
<table>
<thead>
<tr>
<th>ID</th>
<th>Aspect \ Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Group 1: Path Expressions</strong></td>
<td></td>
</tr>
<tr>
<td>Q1</td>
<td>Path expression with non-recursive axes</td>
</tr>
<tr>
<td></td>
<td>Find the names of all persons in the auction document</td>
</tr>
<tr>
<td>Q2</td>
<td>Path expression with recursive axes</td>
</tr>
<tr>
<td></td>
<td>Find all description nodes descendant of all item nodes in the auction document</td>
</tr>
<tr>
<td>Q3</td>
<td>Path expression with wild cards</td>
</tr>
<tr>
<td></td>
<td>Return the subtree of the africa region</td>
</tr>
<tr>
<td>Q4</td>
<td>Path expression with ordered-based axes</td>
</tr>
<tr>
<td></td>
<td>Return the description nodes following the tags with name closed_auction</td>
</tr>
<tr>
<td>Q5</td>
<td>Branching XPath Expressions</td>
</tr>
<tr>
<td></td>
<td>Return the names of all persons with id less than 'person10'</td>
</tr>
<tr>
<td><strong>Group 2: Twig Expressions</strong></td>
<td></td>
</tr>
<tr>
<td>Q6</td>
<td>Simple twig expression</td>
</tr>
<tr>
<td></td>
<td>Return the names and description of all items</td>
</tr>
<tr>
<td>Q7</td>
<td>Twig expression with element construction</td>
</tr>
<tr>
<td></td>
<td>Return the restructured result of the names and description of all items</td>
</tr>
<tr>
<td><strong>Group 3: Predicates</strong></td>
<td></td>
</tr>
<tr>
<td>Q8</td>
<td>Positional Predicates</td>
</tr>
<tr>
<td></td>
<td>Return the third bidder of each open auction</td>
</tr>
<tr>
<td>Q9</td>
<td>Equality Predicates</td>
</tr>
<tr>
<td></td>
<td>Return the closed auctions with price equal to 40</td>
</tr>
<tr>
<td>Q10</td>
<td>Range Predicates</td>
</tr>
<tr>
<td></td>
<td>Return the closed auctions with price less than 40</td>
</tr>
<tr>
<td>Q11</td>
<td>Conjunctive/Disjunctive Predicates</td>
</tr>
<tr>
<td></td>
<td>Return the closed auctions with price greater than 40 and less than 100</td>
</tr>
<tr>
<td>Q12</td>
<td>Predicates with merged nodes from different paths</td>
</tr>
<tr>
<td></td>
<td>Return the african and asian items with id value greater than 100</td>
</tr>
<tr>
<td>Q13</td>
<td>Predicates with merged nodes from different paths and hybrid nature</td>
</tr>
<tr>
<td></td>
<td>Return the price nodes and quantity nodes with value greater than 100</td>
</tr>
<tr>
<td>Q14</td>
<td>String Predicates</td>
</tr>
<tr>
<td></td>
<td>Return all persons with id value greater than &quot;person200&quot;</td>
</tr>
<tr>
<td><strong>Group 4: Values Comparisons (Theta Joins)</strong></td>
<td></td>
</tr>
<tr>
<td>Q15</td>
<td>Value comparison where the values of each operand are constructed by path expression</td>
</tr>
<tr>
<td></td>
<td>Return all pairs of increase value and price value where the increase value is greater than the price value</td>
</tr>
<tr>
<td>Q16</td>
<td>Value comparison where the values of one operand are constructed by path expression and the values of the other operand are constructed by path expression manipulated with arithmetic expression</td>
</tr>
<tr>
<td></td>
<td>Return all pairs of increase value and price value where the increase value is greater than the price value multiplied by 2</td>
</tr>
<tr>
<td>Q17</td>
<td>Values Join</td>
</tr>
<tr>
<td></td>
<td>Return all pairs of increase value and price value where the increase value is equal to the price value</td>
</tr>
<tr>
<td>Q18</td>
<td>Histograms Arithmetic 1</td>
</tr>
<tr>
<td></td>
<td>Return all pairs of increase value and price value where the sum of the increase value and the price value is greater than 100</td>
</tr>
<tr>
<td>Q19</td>
<td>Histograms Arithmetic 2</td>
</tr>
<tr>
<td></td>
<td>Return all pairs of increase value and price value where the sum of the increase value and the price value is equal to 100</td>
</tr>
<tr>
<td>Q20</td>
<td>Histograms Arithmetic 3</td>
</tr>
<tr>
<td></td>
<td>Return all triples of increase value, price value and income where the sum of the increase value and the income value is greater than the sum of the price value and the income value</td>
</tr>
</tbody>
</table>

Table 5. Benchmark Queries list (1)
<table>
<thead>
<tr>
<th>ID</th>
<th>Aspect</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q21</td>
<td>Let - Aggregates</td>
<td>Return the names of persons and the number of items that they bought</td>
</tr>
</tbody>
</table>

### Group 6: Data Dependent Estimations

<table>
<thead>
<tr>
<th>Q22</th>
<th>Predicates with values constructed by aggregate function</th>
<th>Return the open auctions with sum of bidder increases that are greater than 1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q23</td>
<td>Sub-String Matching</td>
<td>Return the names of all items whose description contains the word ‘gold’</td>
</tr>
<tr>
<td>Q24</td>
<td>Distinct</td>
<td>Return the distinct price values</td>
</tr>
<tr>
<td>Q25</td>
<td>Document Order</td>
<td>Return the open auctions where a certain person issued a bid before another person</td>
</tr>
</tbody>
</table>

#### Table 6. Benchmark Queries list (2)

### 6.1 Estimation Accuracy

This sub-section presents an assessment for the accuracy for our XQuery estimation system described in this article. In these experiments, we use the queries of our proposed XQuery estimation benchmark and an XMark document instance with a size of 10 MB. As previously mentioned, unlike usual XML database benchmarks, our proposed queries are designed with close eyes for the different aspects of testing the accuracy of XQuery cardinality estimation system.

To assess the accuracy of our system, we compare our estimated cardinalities for the proposed queries with their actual resulting cardinalities. We use the following error metric for evaluating the accuracy of our estimation system:

$$RE(Q) = \frac{|EST(Q) - ACT(Q)|}{ACT(Q)}$$

where $RE(Q)$ represents the relative error in the estimated cardinality of the query $Q$, $EST(Q)$ represents the estimated cardinality of the query $Q$ and $ACT(Q)$ represents the actual resulting cardinality of applying the query $Q$. Figure 16 illustrates the computed relative errors for our experiment. Remarks about the results of this figure are given as follows:

- Using our introduced notion of Guide Node annotations and our summary structure, Statistical Guide, our XQuery estimation system is able to produce exact estimates with no errors for different types of path expressions as well as twig expressions (Q1, Q2, Q3, Q5, Q6, Q7 and Q8). Similarly, the cardinality of nested XQuery expression (Q21) is very dependent on the cardinality of the referenced XPath expression in the scope of the outer for loop. Since we are able to produce exact estimates for the cardinality of XPath expression, we are also able to produce exact estimates for similar
Fig. 16. Estimation accuracy.

nested expressions. Q4 does not appear on the results of Figure 16 because our estimation system does not support the estimation of XPath expressions with order-based axes due to a limitation in our summary structure as it does not capture the order information of the source XML documents.

– The resulting relative errors of predicates based XQuery expressions and theta join based XQuery expressions (Q9 – Q20) are very dependent on the used statistical histogram techniques for capturing the distribution of the data values as well as the used algorithms for performing comparison and arithmetic operations over histograms. The more sophisticated histogram techniques and algorithms used, the higher the accuracy that could be achieved. Q14 does not appear on the results of Figure 16 because our estimation system does not currently support string-based histograms.

– The queries (Q22 – Q25) do not appear in the results of Figure 16 because our estimation system does not support them for the following reasons:
  - Q22, Q24 are using a form of predicates which is dependent on individual values of the source data values. We can not track or estimate these values in our algebraic plans.
  - Q23 is similar to Q14 uses string-based predicates. Our estimation system does not currently support string-based histograms.
  - Q25 is similar to Q4 uses order-based operation. Our currently used summary structure does not capture the order information of the source XML documents.
6.2 Estimating the Cardinality of XQuery Sub-Expression and Context of FLWOR Iterations

In Section 2.2, we described the loop-lifting compilation of FLWOR expression. In this compilation, a loop of \( n \) iterations is represented by a base loop relation with a single column iter of \( n \) values (1,2,...,\( n \)). This base loop relation preserves the scope of the context nodes in the FLWOR expression. In Section 4.2, we presented an example for inferring the cardinality related properties of the algebraic plans. In this section we present an example of estimating the cardinality of each sub-expression of the main XQuery expression as well as the cardinality of the context nodes for each iteration in the scope of the FLWOR expressions. In this example we use the XMark query Q8 and an XMark document instance with a size of 10 MB.

```xml
let $auction := doc("auction.xml") return
for $p in $auction/site/people/person
let $a :=
  for $t in $auction/site/closed_auctions/closed_auction
  where $t/buyer/@person = $p/@id
  return $t
return <item person="{$p/name/text()}">{count($a)}</item>
```

To illustrate the correspondence relationship between the sub-expressions and their associated algebraic operator we need to represent the algebraic plan of Q8. The algebraic plan of Q8 consists of a total of 245 operators. Due to space limitation, we are not able to present this algebraic plan. Figure 17 illustrates the estimated cardinalities of the of XQuery sub-expressions and iterations contexts for Q8. Remarks about this figure are given as follows:

- In the loop-lifting compilation of Q8, we have four different scopes described as follows:
  - S0: represents the scope of the outermost let expression.
  - S1: represents the scope of the outer for expression.
  - S2: represents the scope of the inner for expression.
  - S3: represents the scope of the then part of the conditional where expression.

  Figure 17(a) illustrates the four base loop relations, their actual cardinalities and their estimated cardinalities.

- Figure 17(b) illustrates the scope of each sub-expression of the XMark query Q8, its actual cardinality, its estimated cardinality and the estimated cardinality for each iteration in the associated scope. The estimated cardinality of each sub-expression is derived from the estimated cardinality of its corresponding algebraic operator. The estimated cardinality for each iteration in the associated scope is computed by dividing the the estimated cardinality of each sub-expression by the cardinality of the base loop relation of its associated scope.
<table>
<thead>
<tr>
<th>Scope</th>
<th>Sub-expression</th>
<th>Act.</th>
<th>Est.</th>
<th>Est./Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0</td>
<td>&quot;auction.xml&quot;</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>s0</td>
<td>doc(&quot;auction.xml&quot;)</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>s0</td>
<td>$auction</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>s0</td>
<td>$auction/site</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>s0</td>
<td>$auction/site/people</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>s0</td>
<td>$auction/site/people/person</td>
<td>2205</td>
<td>2205</td>
<td>2205</td>
</tr>
<tr>
<td>s1</td>
<td>$p</td>
<td>2205</td>
<td>2205</td>
<td>1</td>
</tr>
<tr>
<td>s1</td>
<td>$auction/site</td>
<td>2205</td>
<td>2205</td>
<td>1</td>
</tr>
<tr>
<td>s1</td>
<td>$auction/site/closed_auctions</td>
<td>2205</td>
<td>2205</td>
<td>1</td>
</tr>
<tr>
<td>s1</td>
<td>$auction/site/closed_auctions/closed_auction</td>
<td>1852200</td>
<td>1852200</td>
<td>840</td>
</tr>
<tr>
<td>s2</td>
<td>$t/buyer</td>
<td>1852200</td>
<td>1852200</td>
<td>1</td>
</tr>
<tr>
<td>s2</td>
<td>$t/buyer/@person</td>
<td>1852200</td>
<td>1852200</td>
<td>1</td>
</tr>
<tr>
<td>s2</td>
<td>$p/@id</td>
<td>1852200</td>
<td>1852200</td>
<td>1</td>
</tr>
<tr>
<td>s2</td>
<td>$t/buyer/@person = $p/id</td>
<td>1852200</td>
<td>1852200</td>
<td>1</td>
</tr>
<tr>
<td>s3</td>
<td>$t</td>
<td>840</td>
<td>3784</td>
<td>1</td>
</tr>
<tr>
<td>s2</td>
<td>for $t ... return ...</td>
<td>840</td>
<td>3784</td>
<td>1</td>
</tr>
<tr>
<td>s1</td>
<td>$a</td>
<td>840</td>
<td>3784</td>
<td>1.7</td>
</tr>
<tr>
<td>s1</td>
<td>$p/name</td>
<td>2205</td>
<td>2205</td>
<td>1</td>
</tr>
<tr>
<td>s1</td>
<td>$p/name/text()</td>
<td>2205</td>
<td>2205</td>
<td>1</td>
</tr>
<tr>
<td>s1</td>
<td>element::name</td>
<td>2205</td>
<td>2205</td>
<td>1</td>
</tr>
<tr>
<td>s1</td>
<td>count($a)</td>
<td>2205</td>
<td>2205</td>
<td>1</td>
</tr>
<tr>
<td>s1</td>
<td>element::count</td>
<td>2205</td>
<td>2205</td>
<td>1</td>
</tr>
<tr>
<td>s1</td>
<td>element::item</td>
<td>2205</td>
<td>2205</td>
<td>1</td>
</tr>
<tr>
<td>s0</td>
<td>for $p ... return ...</td>
<td>2205</td>
<td>2205</td>
<td>2205</td>
</tr>
</tbody>
</table>

a) The actual and the estimated cardinalities for the base loop relations of the algebraic plan of XMark query Q8.

b) The detailed actual and estimated cardinalities for the sub-expressions of XMark query Q8.

**Fig. 17.** The estimated cardinalities of the XQuery Sub-expressions and iterations contexts
7 Related Work

Several research efforts have been presented in the literature to propose different estimation models for XML queries. In [3], Aboulnaga et al. have presented two different techniques for capturing the structure of the XML documents and for providing accurate cardinality estimations for the path expressions. The first technique is a summarizing tree structure called a path tree. A path tree is a tree containing each distinct rooted path in the database where the nodes are labelled by the tag name of the nodes. The second technique is a statistical structure called Markov table. This table, implemented as an ordinary hash table, contains any distinct path of length up to $m$ and its selectivity. The presented techniques only support the cardinality estimations of simple path expressions that are without predicates, inline conditions, recursive axes and order-based axes. Moreover, the models cannot be applied to twigs. In [21], Lim et al. presented XPathLearner as a cardinality estimation system for XPath expressions which employs the same summarization and estimation techniques presented in [3] in a different way. Instead of scanning the XML data, XPathLearner gathers and refines the required statistical information in an on-line manner from query feedbacks.

In [2], Zhang et al. have addressed the problem of deriving cardinality estimation of XPath expressions. In this work, the authors are mainly focusing on the handling of XPath expressions which involve only structural conditions. The main idea behind the paper is to provide an efficient treatment of recursive XML documents and the accurate estimation of recursive queries.

In [20], Li et al. have described a framework for estimating the selectivity of XPath expressions with a main focus on the order-based axes (following, preceding, following-sibling, preceding-sibling). They used a path encoding scheme to aggregate the path and order information of XML data. They used a path encoding scheme to aggregate the path and order information of XML data.

In [11] Freire et al. have presented an XML Schema-based statistics collection technique called StatiX. StatiX leverages the available information in the XML Schema to capture both structural and value statistics about the source XML documents. These structural and value statistics are collected in the form of histograms.

The authors of [32] proposed a framework for XML path selectivity estimation in a dynamic context using a special histogram structure named as Bloom Histogram. The Bloom Histogram keeps a count of the statistics for paths in XML data.

8 Conclusion and Outlook

In this article we described the design and implementation of our proposed framework for estimating the cardinality of XQuery expressions. Several research literature have presented different estimation models for XML queries. However, some of these models are limited only to cardinality estimation of path expression, while others also deal with twig queries, but all of them do not address the
critical issues of XML queries such as iterators, nested queries, element construction, etc. To the best of our knowledge, our work is the first which provides a uniform framework to estimate the cardinality of more powerful XML querying capabilities using XQuery expressions as well as its sub-expressions. Moreover, the proposed framework can act as a meta-model through its ability to incorporate different summarized XML structures and different histogram techniques which allows the model designers to achieve their targets by focusing their effort on designing or selecting the adequate techniques for them. Our XQuery estimation framework is similar to the work proposed by Sartiani in [27]. However, we have the following main advantages:

– Sartiani approach is very limited to FLWOR expressions while we are supporting a much larger subset (almost complete) of the XQuery language.
– Our approach is able to estimate not only the whole XQuery expression but also each of its sub-expressions as well as the cardinality of each iteration in the context of the FLWOR expressions.
– Based on our introduced Guide Node annotation concept, our framework is more flexible as it is able to integrate any XPath or predicates size estimation techniques.
– The ability of our framework to estimate the cardinality of the sub-expression and its associated intermediate relational results as well the inferred Guide Node property of each algebraic operator provides the XQuery engine with the power to generate enhanced execution plans.

Unfortunately, the current status of the our proposed XQuery cardinality estimations framework suffers from two main limitations:

1. It does not support the estimation of the queries over the order information of the source XML documents using either ordered-based XPath axes or node ordering operators. That is because the used summary structure, Statistical Guide, does not keep this order information of the source XML document. However, currently we are working on integrating the SLT ordered-based summary structure presented by Fisher and Maneth in [10]. Integrating such XML summary structure will solve this problem and enhance our framework with the ability to support ordered-based XQuery expressions.
2. It does not support the estimation of the data dependent queries described in our proposed benchmark.

In addition, our contributions in this article include our proposed benchmark for XQuery cardinality estimation systems. Our proposed benchmark distinguishes itself from the other existing XML benchmarks in its focus on establishing the basis for comparing the different estimation approaches in the XML domain in terms of their accuracy of the estimations and their completeness in handling different XML querying features.
References


Appendix

Group 1: Path Expressions

Q1 : Path expression with non-recursive axes.

Find the names of all persons in the auction document.

for $b$ in doc("auction.xml")/site/people/person/name/text()
    return $b

where non-recursive axes are child, parent, attribute, following-sibling and preceding-sibling.

Q2 : Path expression with recursive axes.

Find all description nodes descendant of all item nodes in the auction document.

for $b$ in doc("auction.xml")/site/item//description
    return $b

where recursive axes are descendant, descendant-or-self, ancestor and ancestor-or-self.

Q3 : Path expression with wild cards.

Return the subtree of the africa region.

for $b$ in doc("auction.xml")/site/regions/africa//*
    return $b

Q4 : Path expression with ordered-based axes.

Return the description nodes following the tags with name closed_auction.

for $b$ in doc("auction.xml")/site//closed_auction/following::description
    return $b

where ordered-based axes are following, following-sibling, preceding and preceding-sibling.

Q5 : Branching XPath Expressions.

Return the names of all persons with id less than 'person10'.

for $b$ in doc("auction.xml")/site/person[@id < 'person10']/name
    return $b
Group 2: Twig Expressions

Q6 : Simple twig expression.
Return the names and description of all items.

for $b$ in doc("auction.xml")//item
return ($b/name, $b/description)

Q7 : Twig expression with element construction.
Return the restructured result of the names and description of all items.

for $b$ in doc("auction.xml")//item
return
<Result>
   <name>{$b/name}</name>
   <description>{$b/description}</description>
</Result>

Group 3: Predicates

Q8 : Positional Predicates.
Return the third bidder of each open auction.

for $b$ in doc("auction.xml")/site/open_auctions/open_auction
return $b/bidder[3]

Q9 : Equality Predicates.
Return the closed auctions with price equal to 40.

for $b$ in doc("auction.xml")/site//closed_auction
where data($b/price) = 40
return $b

Q10 : Range Predicates.
Return the closed auctions with price less than 40.

for $b$ in doc("auction.xml")/site//closed_auction
where data($b/price) < 40
return $b

where the range predicates uses any of the operators (<,≤,=,!=,>,≥)}
Q11: Conjunctive/Disjunctive Predicates.

*Return the closed auctions with price greater than 40 and less than 100.*

```xml
for $b in doc("auction.xml")/site//closed_auction
where data($b/price) > 40 and data($b/price) < 100
return $b
```

where conjunctive predicates uses the conjunctive/disjunctive operators (*AND, OR*)

Q12: Predicates with merged nodes from different paths.

*Return the african and asian items with id value greater than 100.*

```xml
for $b in (doc("auction.xml")/site/africa/item,
    doc("auction.xml")/site/asia/item)
where data($b/@id) > 100
return $b
```

An accurate estimation of such query should consider the different data distribution for the nodes resulting from each different path expression as well as the percentage of each path in constructing the nodes of the operated sequence.

Q13: Predicates with merged nodes from different paths and hybrid nature.

*Return the price nodes and quantity nodes with value greater than 100.*

```xml
for $b in (doc("auction.xml")/site//price,
    doc("auction.xml")/site//quantity)
where data($b) > 1 and data($b) > 100
return $b
```

This query is more challenging than the previous one because the resulting nodes of the sequence are representing different items (*price, quantity*).

Q14: String Predicates.

*Return all persons with id value greater than "person200".*

```xml
for $b in doc("auction.xml")/site/people/person
where $b/@id > "person200"
return $b
```
Group 4: Values Comparisons (Theta Joins)

Q15: Value comparison where the values of each operand are constructed by path expression.

Return all pairs of increase value and price value where the increase value is greater than the price value.

```xml
for $x$ in doc("auction.xml")/site//increase,
   $y$ in doc("auction.xml")/site//price
where data($x) > data($y)
return <pair>{$x,$y}<(pair>
```

Q16: Value comparison where the values of one operand are constructed by path expression and the values of the other operand are constructed by path expression manipulated with arithmetic expression.

Return all pairs of increase value and price value where the increase value is greater than the price value multiplied by 2.

```xml
for $x$ in doc("auction.xml")/site//increase,
   $y$ in doc("auction.xml")/site//price
where data($x) > data($y) * 2
return <pair>{$x,$y}<(pair>
```

Q17: Values Join.

Return all pairs of increase value and price value where the increase value is equal to the price value.

```xml
for $x$ in doc("auction.xml")/site//increase,
   $y$ in doc("auction.xml")/site//price
where data($x) = data($y)
return <pair>{$x,$y}<(pair>
```

Q18: Histograms Arithmetic 1.

Return all pairs of increase value and price value where the sum of the increase value and the price value is greater than 100.

```xml
for $x$ in doc("auction.xml")/site//increase
   $y$ in doc("auction.xml")/site//price
where data($x) + data($y) > 100
return <pair>{$x,$y}<(pair>
```
Q19 : Histograms Arithmetic 2.
Return all pairs of increase value and price value where the sum of the increase value and the price value is equal to 100.

```xml
for $x in doc("auction.xml")/site//increase
   $y in doc("auction.xml")/site//price
   where data($x) + data($y) = 100
   return <pair>{$x,$y}</pair>
```

Q20 : Histograms Arithmetic 3.
Return all triples of increase value, price value and income where the sum of the increase value and the income value is greater than the sum of the price value and the income value.

```xml
for $x in doc("auction.xml")/site//increase
   $y in doc("auction.xml")/site//price
   $z in doc("auction.xml")/site//@income
   where data($x) + data($z) > data($y) + data($z)
   return <pair>{$x,$y,$z}</pair>
```

Group 5: Nested Expressions

Q21 : Let - Aggregates.
Return the names of persons and the number of items that they bought.

```xml
let $auction := doc("auction.xml") return
for $p in $auction/site/people/person
   let $a :=
      for $t in $auction/site/closed_auctions/closed_auction
         where $t/buyer/@person = $p/@id
      return $t
   return <item>
      <person>{$p/name/text()}</person>
      <count>{count($a)}</count>
   </item>
```

Group 6: Data Dependent Estimations

Q22 : Predicates with values constructed by aggregate function.
Return the open auctions with sum of bidder increases that are greater than 1000.

```xml
for $b in doc("auction.xml")/site/open_auctions/open_auction
   where sum(data($b/bidder/increase)) > 1000
   return <increase>{$b}</increase>
```
Q23 : Sub-String Matching.

Return the names of all items whose description contains the word ‘gold’.

let $auction := doc("auction.xml")
for $i in $auction/site/item
where contains(string(exactly-one($i/description)), "gold")
return $i

Q24 : Distinct.

Return the distinct price values.

for $b in distinct-values(doc("auction.xml")//price/text())
return $b

Q25 : Document Order.

Return the open auctions where a certain person issued a bid before another person.

let $auction := doc("auction.xml")
return for $b in $auction/site/open_auctions/open_auction
where some $pr1 in $b/bidder/personref[@person = "person20"],
$pr2 in $b/bidder/personref[@person = "person51"]
satisfies $pr1 << $pr2
return <history>{$b}</history>