Dependable Cardinality Forecasts for XQuery

Jens Teubner
IBM Watson
Hawthorne, NY, USA
teubner@us.ibm.com

Torsten Grust
TU München
Munich, Germany
grust@in.tum.de

Sebastian Maneth   Sherif Sakr
NICTA and UNSW
Sydney, Australia
{firstname.lastname}@nicta.com.au

ABSTRACT

Though inevitable for effective cost-based query rewriting, the derivation of meaningful cardinality estimates has remained a notoriously hard problem in the context of XQuery. By basing the estimation on a relational representation of the XQuery syntax, we show how existing cardinality estimation techniques for XPath and proven relational estimation machinery can play together to yield dependable forecasts for arbitrary XQuery (sub)expressions. Our approach benefits from a light-weight form of data flow analysis. *Abstract domain identifiers* guide our query analyzer through the estimation process and allow for informed decisions even in case of deeply nested XQuery expressions. A variant of *projection paths* [15] provides a versatile interface into which existing techniques for XPath cardinality estimation can be plugged in seamlessly. We demonstrate an implementation of this interface based on *data guides*. Experiments show how our approach can equally cope with both, structure- and value-based queries. It is robust with respect to intermediate estimation errors, from which we typically found our implementation to recover gracefully.

1. INTRODUCTION

Modern database implementations derive much of their performance from sophisticated optimizer components that transform incoming queries into efficient execution plans. To properly select access paths, join order, or materialization strategies, optimizers heavily depend on accurate predictions of the value distribution and cardinality of individual query sub-results.

Such cardinality forecasts have been notoriously hard to make in the context of XQuery, where the absence of a strict database schema, the expressiveness of the XQuery language, and the dualism between structural and value-based querying all add to the complexity of the estimation problem. In this work, we show how relational plan equivalents for XQuery—originally developed to enable scalable XQuery processing on relational back-ends—can be used to determine dependable cardinality forecasts for XQuery. By faithfully keeping the connection between the relational plan and the original query, we make sure these forecasts are valuable even if the evaluation strategy of the actual back-end is not relational.

PVLDB '08, August 23-28, 2008, Auckland, New Zealand

Consider the following XQuery query against `weather.com` forecast data for New Zealand (which we hope takes the `else` branch more often than the `then` branch during VLDB 2008):

```
                                                     ①
for $d in doc("forecast.xml")/descendant::day
let $day := $d/@t
let $ppcp := data($d/descendant::ppcp)
return                                               ②      (Q1)
  if ($ppcp > 50)                                    ③
    then ("rain likely on", $day,
          "chance of precipitation:", $ppcp)
    else ("no rain on", $day)
                        ④
```

Although existing techniques (*e.g.*, [1, 8, 16]) could well estimate the cardinality of the rooted path expression `doc(·)/descendant::day`, the remaining expression kinds in this query (`for` loops, conditionals, sequence construction, and implicit existential quantification), let alone the arbitrary nesting of such clauses, are beyond the capabilities of existing work. It is our goal to derive accurate cardinality estimates for *any* subexpression in this query, and we will illustrate our approach for the ones marked ① to ④.

Interlude: The importance of cardinality forecasts. To pinpoint the impact of such fine-grained cardinality forecasts, we used the Pathfinder relational XQuery compiler [11] to generate a SQL formulation of the XQuery expression

`doc("forecast···")/descendant::day/descendant::ppcp`

For two predicates $p_{1,2}$ in this SQL query we injected annotations `SELECTIVITY` s that forced IBM DB2 to assume that the p_i have selectivity $0\,\% \leqslant s \leqslant 5\,\%$:

p_1, a predicate emitted by the compiler to extract the document node of the particular document `forecast.xml` from Pathfinder's tabular XML node encoding, and

p_2, a predicate that selects the XML elements that are reachable by the subsequent `descendant::day` XPath location step.

The assumed selectivities led DB2 to yield nine different execution plans as documented by Figure 1 (inspired by Haritsa's Picasso optimizer visualizer [17]). The actual selectivities of p_1 and p_2 are about $0\,\%$ and $1.1\,\%$, respectively. Equipped with this information, DB2 finds an execution plan that runs in three orders of magnitude less the exe-

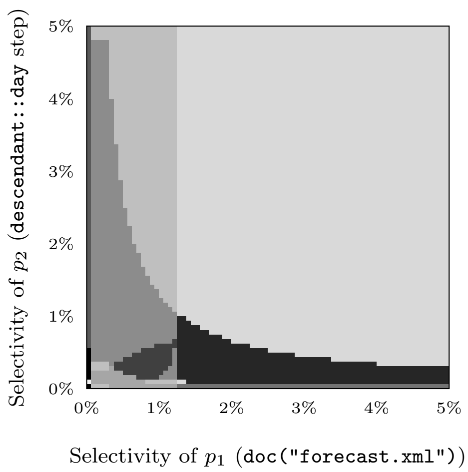


Figure 1: Impact of (assumed) selectivities on DB2's choice of execution plan: the optimizer proposes a variety of nine different plans (gray shades).

cution time of the worst plan. Clearly, there is something to be gained from cardinality forecasts at the XQuery subexpression level.

Expressions like Query Q_1 are easily handled by the approach we pursue here. We compile the input query into an equivalent relational representation. While this provides an immediately optimizable and executable specification of the XQuery semantics (a route taken by the *Pathfinder* XQuery compiler [5, 10]), note that execution is *not* our original intent here. Instead, we build on the equivalence of original input XQuery expression and its associated relational plan and call upon existing techniques for relational plan analysis to derive cardinality estimates for XQuery. To this end, we use plan variants that maintain a direct correspondence between row count and number of XQuery items. The procedure is carefully designed to interoperate with a wide range of the existing estimation techniques for XPath and, conversely, to make the outcome of the plan analysis accessible to any XQuery processor, whether it is based on a relational back-end or not.

The most effective component of our inference procedure turns out to be the introduction of *abstract domain identifiers* as an approximation of the value space that individual query subexpressions take at runtime. An inferred inclusion property between value domains, together with an approximation of the size of each domain capture enough information to estimate the cardinality of relational XQuery plans.

We will present our approach as follows. Section 2 recapitulates all relevant aspects of relational XQuery processing, followed by the principles of relational XQuery cardinality estimation in Section 3. In Section 4, we add support for structural (XPath-) and value-based queries and illustrate the cardinality inference process for Query Q_1 in Section 5. Sections 6 and 7 discuss related work and wrap up.

2. RELATIONAL XQUERY

We map XQuery syntax to relational plans via *loop lifting* [12], a compilation technique—originally developed for the XQuery compiler Pathfinder [11]—that derives algebraic queries from the compositional XQuery dialect shown in Table 1. The compiler's target language is a table algebra (Table 2) that has been designed to ease query analysis—the focus in this work—as well as efficient execution on modern SQL-style database engines [10, 12].

literal values	document order (e_1 `<<` e_2)
sequences (e_1 `,` e_2)	node identity (e_1 `is` e_2)
variables (`$`v)	arithmetics (`+`, `-`, ...)
`let $`v `:=` e_1 `return` e_2	comparisons (`eq`, `lt`, ...)
`for $`v `in` e_1 `return` e_2	Boolean op.s (`and`, `or`, ...)
`if (`e_1`) then` e_2 `else` e_3	`fn:doc`(e)
`typeswitch` clauses	`fn:root`(e)
`element {`e_1`} {`e_2`}`	`fn:data`(e)
`text {`e`}`	`fs:distinct-doc-order`(e)
e_1 `order by` e_2`,` ...`,` e_n	`fn:count`(e), `fn:sum`(e), ...
XPath (e/ax::nt)	`fn:empty`(e)
user-defined functions	`fn:position()`, `fn:last()`

Table 1: Supported XQuery subset (excerpt).

$\pi_{\ldots,\mathsf{b:a},\ldots}$	column projection, renaming (a into b)
σ_{a}	selection (select rows with $\mathsf{a} = \mathsf{true}$)
$\bowtie_{\mathsf{a=b}}, \times$	equi-join, Cartesian product
$\dot\cup, \setminus$	disjoint union (append), difference
δ	duplicate row elimination
$\varrho_{\mathsf{a}:\langle \mathsf{b}_1,\ldots,\mathsf{b}_n\rangle \Vert \mathsf{c}}$	row numbering (grouped by c)
$\circledast_{\mathsf{a}:(\mathsf{b}_1,\mathsf{b}_2)}$	arithmetic/comparison operator $*$
$\lrcorner_{\mathsf{a}:ax::nt(\mathsf{b})}$	XPath step operator ($\mathsf{a} = \mathsf{b}/ax::nt$)
⚛$_{\mathsf{a:b}}$	XQuery atomization ($\mathsf{a} =$ `fn:data(b)`)
$\mathtt{doc}_{\mathsf{a:b}}$	XML document access ($\mathsf{a} =$ `fn:doc(b)`)
ε, τ	element/text node construction
$agg_{\mathsf{a}\Vert\mathsf{b}}$	aggregation, grouped by b

Table 2: Table algebra used for cardinality estimation ($agg \in \{$`count`, `sum`, `max`, ...$\}$).

We work with rather restricted operator variants that consume and emit tables (not sets) of rows. Projection π extracts and renames columns and does not incur duplicate elimination— the latter is explicit in terms of operator δ. Selection σ_{a} selects rows whose Boolean column a carries value true. Such a Boolean column a is typically established through prior application of comparison operators like $\circledless_{\mathsf{a}:(\mathsf{b}_1,\mathsf{b}_2)}$—the instance of $\circledast$ with $* = <$—which map a comparison of columns b_1 and b_2 over the entire input table. The loop lifting compilation scheme guarantees that all occurring $\bowtie$ are indeed equi-joins and that the rows of the input tables of operator $\dot\cup$ are disjoint. The primitive $\varrho_{\mathsf{a}:\langle \mathsf{b}_1,\ldots,\mathsf{b}_n\rangle \Vert \mathsf{c}}$ groups its input table by column c and then attaches new column a holding unique row numbers in $\mathsf{b}_1, \ldots, \mathsf{b}_n$ order (this mimics the SQL:1999 clause `RANK() OVER (PARTITION BY c ORDER BY` $\mathsf{b}_1,\ldots,\mathsf{b}_n$`) AS a`). The compilation scheme uses ϱ mainly to embed row order information in the table data itself—this enables the use of compilation targets whose native data model is unordered (*e.g.*, relational database systems).

The non-textbook operators, $\lrcorner$, ⚛, doc, ε, τ reflect specific aspects of the XQuery semantics and are discussed in Section 4 when they are in context.

In the loop-lifting compiler, the XQuery `for` clause is the core language construct: *any* expression e is considered to be

$$\frac{}{const\,(\pi_{\mathsf{a}:v}(q)) \supseteq \{\mathsf{a}^{=v}\}}\,(\textsc{Const-1}) \qquad \frac{\mathsf{a}^{=v} \in const\,(q)}{const\,(\pi_{\dots,\mathsf{b}:\mathsf{a},\dots}(q)) \supseteq \{\mathsf{b}^{=v}\}}\,(\textsc{Const-2}) \qquad \frac{}{const\,(\sigma_{\mathsf{a}}(q)) \supseteq \{\mathsf{a}^{=\mathsf{true}}\}}\,(\textsc{Const-3})$$

Figure 2: Examples of $const\,(\cdot)$ (columns holding a constant value) inference rules.

in the scope of its innermost enclosing `for` iteration.[1] For each such e, the compiler emits *two* algebraic plan pieces which jointly compute a tabular encoding of e's result:

(1) loop_e, a unary table with single column iter that holds value i iff e is evaluated in the ith iteration of its enclosing `for` loop, and

(2) q_e, a ternary table iter|pos|item in which a row $[i, p, v]$ indicates that, in iteration i, e evaluates to an item sequence in which v (an atomic value or XML node identifier) occurs at position p.

To illustrate, consider the subexpression ⑤ in the following slightly contrived XQuery Query Q_2 which evaluates to `(<gust>80 mph</gust>,<gust/>,<gust>95 mph</gust>)`:

```
for $gust in (80,5,95) return                              (Q2)
 element gust { if ($gust > 70)                    ⑤        ⑥
                then (string($gust),"mph") else ()  }
                                                           ⑦
```

iter
1
3

According to compilation invariant (1), the algebraic plan for Q_2 will contain a sub-plan that computes the unary two-row table $\mathsf{loop}_{⑤}$ shown on the left: subexpression ⑤ is evaluated for the first and third binding of variable `$gust`.

iter	pos	item
1	1	"80"
1	2	"mph"
3	1	"95"
3	2	"mph"

As per invariant (2), the compiler further emits a sub-plan that evaluates to the sequence-encoding table $q_{⑤}$ on the right: expression ⑤ evaluates to two `xs:string` items each in iterations 1 and 3 of the enclosing `for` loop. From the cardinality of table $q_{⑤}$ we infer that, overall, expression ⑤ contributes four items.

Exactly this direct correspondence of the cardinalities of tables loop_e and q_e and the number of items returned by XQuery expression e is what we exploit in this work. If we annotate the original input XQuery expression Q_2 with the table cardinalities we have observed, we get

```
for $gust in (80,5,95) return
 element gust { if ($gust > 70)                    4        0
                then (string($gust),"mph") else ()  }
                                          4 (1.33/iteration)
```

The empty sequence `()` maps to an empty iter|pos|item table [12] while the `if-then-else` essentially maps into a disjount union $\uplus$, so we can derive its cardinality as $4 + 0 = 4$. With the cardinality three of $\mathsf{loop}_{⑦}$ at hand, we can even report that the entire `if-then-else` clause will contribute an average of $4/3 \approx 1.33$ items per iteration of the `for` loop.

Figure 3 depicts the complete plan DAG for Query Q_1 of the Introduction. Note that one source of plan sharing in this DAG is due to the fact that all subexpressions in the scope of the same `for` loop may share their loop tables. The desired forecast for the cardinalities of subexpressions ① to ④ in Q_1 may be made based on the table cardinalities of $q_{①}$ to $q_{④}$. Note that this forecast does *not* depend on the actual presence of a relational execution engine. Any (non-relational) XQuery processor can exploit this agreement of row count and number of XQuery items. Since loop lifting defines a fully *compositional* compilation scheme [11], it is especially simple to keep track of the correspondence of XQuery subexpression e and its associated subplan q_e.

[1] Around a top-level expression e we assume the void loop `for $_ in (0) return` e where `$_` does not occur free in e.

2.1 Analyzing Relational XQuery Plans

We analyze the relational plan DAGs in a peephole-style fashion. Based on a set of inference rules, we perform a single pass over the plan to annotate a set of properties to each operator. These annotations are local in the sense that they only depend on the operator's immediate plan vicinity.

Figure 2 shows a subset of the inference rules for one such property, $const\,(\cdot)$—we will add more in the course of this work. An entry $\mathsf{a}^{=v}$ in the set-valued property $const\,(q)$ (denoted $const\,(q) \supseteq \{\mathsf{a}^{=v}\}$) indicates that, in the result of sub-plan q, column a holds value v in *all* rows. We use Rules Const-1 through Const-3 to infer such columns that hold a constant value. The projection operator π is one means to introduce columns of this kind (reflected by Rule Const-1), but other operators may lead to statically-known constants, too (as shown, *e.g.*, in Rule Const-3). Rule Const-2 propagates constant column information after column renaming. Further inference rules for $const\,(\cdot)$ are explained in [10].

In the remainder of this text, we will also use the property $cols(\cdot)$ to access the column schema of its table argument, such that $cols(q_e) = \{\mathsf{iter}, \mathsf{pos}, \mathsf{item}\}$ according to compilation invariant (2), for example.

3. CARDINALITY INFERENCE

The ultimate goal of this work is to infer an additional property $|q|$ for each sub-plan q, the estimated number of rows after sub-plan evaluation. For a large class of algebra operators, we can directly turn their definition into an inference rule for $|\cdot|$. Most of the unary operators, *e.g.*, preserve the cardinality of their input, as captured by Rule Card-1:

$$\frac{\square \in \{\pi_{\dots}, \varrho_{\mathsf{a}:\langle \mathsf{b}_1,\dots,\mathsf{b}_n\rangle \| \mathsf{c}}, \circledast_{\mathsf{a}:(\mathsf{b}_1,\dots,\mathsf{b}_n)}, ⚛_{\mathsf{a}:\mathsf{b}}\}}{|\square(q)| = |q|}\,(\textsc{Card-1})$$

The binary operators $\uplus$ and $\times$ are other examples where the operator definition straightforwardly translates into a cardinality inference rule (recall that $\uplus$ preserves duplicates):

$$\frac{}{|q_1 \uplus q_2| = |q_1| + |q_2|}\,(\textsc{Card-2}) \qquad \frac{}{|q_1 \times q_2| = |q_1| \cdot |q_2|}\,(\textsc{Card-3})$$

In relational XQuery evaluation plans, operator $\uplus$ is used, *e.g.*, to combine the subexpressions of the `then` and `else` branches of an `if` conditional (sub-plan $q_{②}$ in Figure 3) or to implement sequence construction (remaining $\uplus$ operators in Figure 3).

The traditional approach to estimate the cardinality of the selection σ_{a} and equi-join operators $\bowtie_{\mathsf{a}=\mathsf{b}}$ operators in relational databases is the one taken in System R [20]. Rules Card-4 and Card-5 implement the heuristic "10 %" rule of System R:

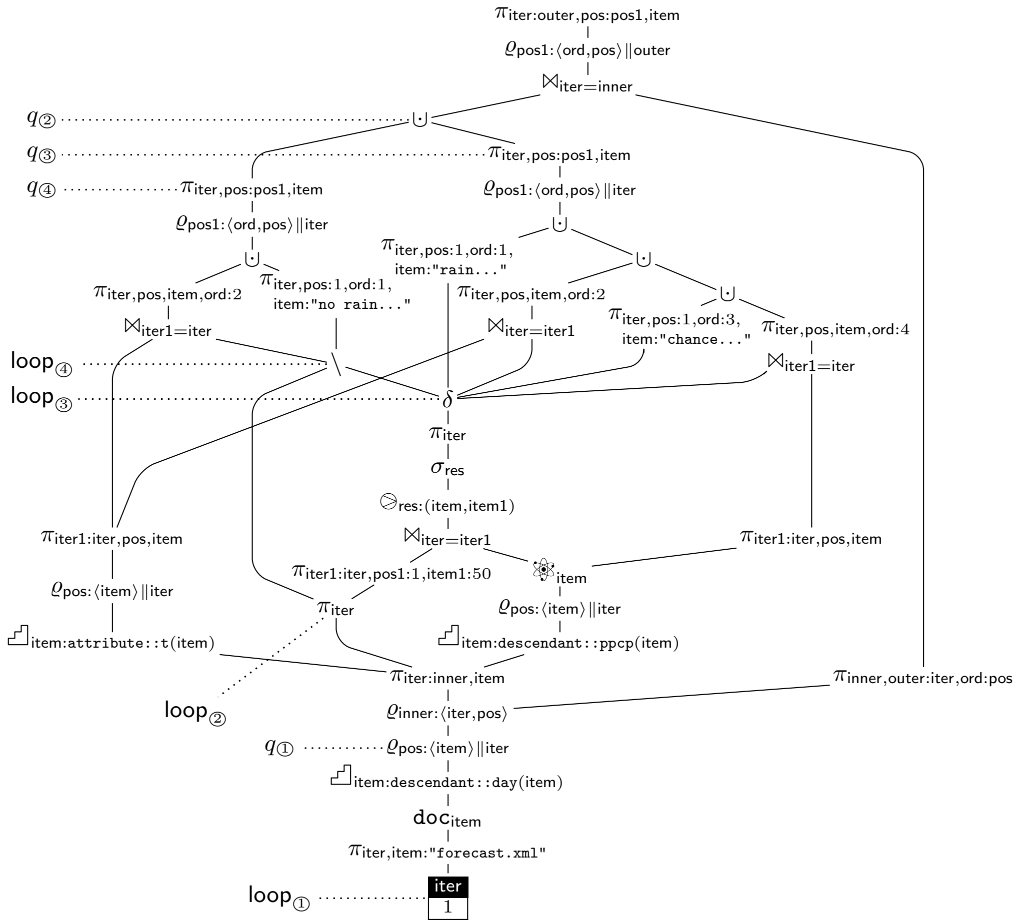


Figure 3: Complete plan DAG for XQuery expression Q_1 (see text for annotations).

$$\frac{}{|\sigma_{\mathsf{a}}(q)| = |q| \cdot {}^{1}\!/_{10}}\,(\textsc{Card-4}) \qquad \frac{}{|q_1 \underset{\mathsf{a=b}}{\bowtie} q_2| = |q_1| \cdot |q_2| \cdot {}^{1}\!/_{10}}\,(\textsc{Card-5})$$

Both rules implement rather crude estimates, since the System R optimizer assumes that persistent indexes are typically present to provide statistics for the argument relations q_1 and q_2. Unfortunately, this assumption is met only rarely in plans generated from XQuery. Nested XQuery expressions more often lead to selections and joins over *computed* subexpressions, instead of input from persistent storage. A peephole-style implementation of *data flow analysis* in our estimator remedies this situation, as we discuss next. Our actual plan analyzer uses Rules Card-4 and Card-5 as a last resort only if no useful information can be inferred about the input relations computed by q_1 and q_2. Only rarely did we see them being applied to real-world queries by our prototype.

3.1 Abstract Domain Identifiers

The selectivities involved in the evaluation of the σ or $\bowtie$ operators could be estimated more accurately with knowledge about the *active domain* of any column c in the operand relations, *i.e.*, the actual values taken by c at runtime. Obviously, the actual value space of any column is not yet known during static query analysis. Instead, we introduce *abstract domain identifiers*, denoted by Greek letters α, β, γ in this text, to represent the runtime domains. We infer just as much information about value domains as necessary to compute reliable cardinality estimates. A similar device has been used in [10] to aid the algebraic optimization of XQuery joins.

Fresh value domains are usually introduced by operators that establish new table columns, such as, *e.g.*, the row-numbering operator $\varrho_{\mathsf{a}:\langle \mathsf{b}_1,\ldots,\mathsf{b}_n\rangle}$. We write $\mathsf{a}^\alpha \in \mathit{dom}\,(q)$ to indicate that in the result relation computed by q the active domain of column a is α (for a fresh identifier α, not used before):

$$\frac{}{\mathit{dom}\left(\varrho_{\mathsf{a}:\langle \mathsf{b}_1,\ldots,\mathsf{b}_n\rangle}(q)\right) \supseteq \mathit{dom}\,(q) \cup \{\mathsf{a}^\alpha\}}\ .$$

In our example plan (Figure 3), other instances of operators that introduce fresh domains are projection operators that set up constant columns ($\pi_{\ldots,\mathsf{a}:v,\ldots}$), the comparison operator $\circleddash_{\text{res:(item,item1)}}$, or the XPath-related operators $\mathtt{doc}$, ⏋, and ⚛, whose semantics we discuss in Section 4.

3.1.1 Domain Sizes

For each newly established value domain, our plan analyzer also tries to infer additional information that is valuable for our aim, the inference of table cardinalities. Towards this end, we estimate the *size* of each domain, written as $\|\alpha\|$. $\|\alpha\|$ denotes the number of distinct values in the value domain α.

Domain sizes and table cardinalities often interact. Operator $\varrho_{\mathsf{a}:\langle \mathsf{b}_1,\ldots,\mathsf{b}_n\rangle}(q)$, *e.g.*, establishes a new key column a over the input relation q. The size of a's value domain, hence, coincides with the cardinality of q, as reflected in Rule Dom-1 of our inference rule set (this is a refinement of the above inference):

$$\frac{}{\mathit{dom}\left(\varrho_{\mathsf{a}:\langle \mathsf{b}_1,\ldots,\mathsf{b}_n\rangle}(q)\right) \supseteq \mathit{dom}\,(q) \cup \left\{\mathsf{a}^{\alpha} \wedge \|\alpha\| =^{!} |q|\right\}}\,(\textsc{Dom-1})$$

The notation $\|\alpha\| =^{!} |q|$ indicates that we are inferring the domain size of α to be the cardinality of q here (rather than deriving it from a domain size inferred earlier).

Conversely, the size of a domain determines the cardinality of aggregates:

$$\frac{\mathsf{b}^{\beta} \in \mathit{dom}\,(q)}{\left|\mathit{agg}_{\mathsf{a}\|\mathsf{b}}(q)\right| = \|\beta\|}\,(\textsc{Card-6})$$

or the output of the duplicate elimination operator δ for single-column inputs:

$$\frac{\mathit{cols}(q) = \{\mathsf{a}\} \qquad \mathsf{a}^{\alpha} \in \mathit{dom}\,(q)}{|\delta(q)| = \|\alpha\|}\,(\textsc{Card-7})\ .$$

Further examples of domain usage and inference are shown in Figure 4. The size of a constant-column domain is trivially 1 (Rule Dom-2). The output domain of operators with a Boolean result is the two-item set $\{\mathsf{true},\mathsf{false}\}$ (Rule Dom-3). Rules Dom-4 to Dom-6 propagate domain information bottom-up through the inference process.

Rule Dom-7 shows the domain inference for the row-numbering operator $\varrho_{\mathsf{a}:\langle \mathsf{b}_1,\ldots,\mathsf{b}_n\rangle\|\mathsf{c}}$ in the presence of a grouping column c. The operator creates $\|\gamma\|$ groups, where γ is the domain associated with c. Assuming equi-sized groups, the average group size is $|q|/\|\gamma\|$. Since ϱ produces numbers between 1 and the group size, this is also the domain size we estimate for the new column a. Rule Dom-8 is the dual to the aforementioned cardinality inference rule for $\mathit{agg}_{\mathsf{a}\|\mathsf{b}}$.

In our example plan (Figure 3), we use domain sizes, *e.g.*, to infer the cardinalities of the $\mathsf{loop}_{③}$ and $\mathsf{loop}_{④}$ relations, which correspond to the number of times the `then` and `else` branches are taken in the original query Q_1, respectively.

3.1.2 Domain Inclusion

So far we have only considered algebra operators that strictly propagate all values (*i.e.*, the full value domain) from one column to the operator output. Operators such as selection (σ_{a}), equi join ($\bowtie_{\mathsf{a}=\mathsf{b}}$), or difference ($\setminus$), by contrast, typically compute a restriction of their input domains.

The domain inference for the selection operator σ_{a} (see Rule Dom-9, Figure 4) uses the expression

$$\|\gamma_2\| =^{!} \|\gamma_1\| \cdot \left(1 - (1 - {}^{1}\!/_{10})^{|q|/\|\gamma_1\|}\right) \qquad (1)$$

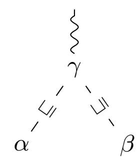
to compute the domain sizes for all output domains. The factor $^{1}\!/_{10}$ is the System R 10 % heuristic for the general selection operator. Details about the remaining terms in Expression 1 are beyond our current discussion. Interested readers may find them in Appendix B.

Restricting domains also leads to an *inclusion relationship* between the input and output domains. Domain α is a *subdomain* of β ($\alpha \sqsubseteq \beta$) if all values in α are also a member of β. The values in $\sigma_{\mathsf{a}}(q)$, *e.g.*, are a subset of those in q, hence the inference of $\gamma_2 \sqsubseteq \gamma_1$ in Rule Dom-9. Domain inclusion is transitive ($\alpha \sqsubseteq \beta \wedge \beta \sqsubseteq \gamma \Rightarrow \alpha \sqsubseteq \gamma$) and reflexive ($\alpha \sqsubseteq \alpha$).

System R-style domain inference for joins is covered by Rule Dom-10. However, our plan analyzer can typically avoid the application of this rule and rather derive more fine-grained domain information based on domain inclusion knowledge. Rule Dom-11 presupposes a subdomain relationship $\alpha \sqsubseteq \beta$ between the value domains α and β associated with the join attributes a and b of the input relations q_1 and q_2 (respectively). A common instance is the foreign key dependence when column a of q_1 references column b in q_2.

Under the premise of $\alpha \sqsubseteq \beta$, all rows from q_1 are retained in the join result, hence $\mathit{dom}\,(q_1) \subseteq \mathit{dom}\,(q_1 \bowtie_{\mathsf{a}=\mathsf{b}} q_2)$. Rows from the right-hand-side relation, by contrast, are filtered depending on the containment of their b values in domain α. α has been derived earlier as a restriction of β with a selectivity of $\|\alpha\|/\|\beta\|$. Rule Dom-11 uses this factor to compute the sizes of those domains that originally come from q_2. Columns a and b are identical in the join result by definition, hence $\mathsf{b}^{\alpha} \in \mathit{dom}\,(q_1 \bowtie_{\mathsf{a}=\mathsf{b}} q_2)$.

In Figure 3, the input to the ⊝ operator is a join operation of this kind. The domain associated with the iter column of the right join input is a subdomain of its left-hand-side counterpart here.



More generally, domain inclusion defines a tree-shaped hierarchy of domains. In Rule Dom-12, we consider join operators $\bowtie_{\mathsf{a}=\mathsf{b}}$ where α and β, the domains of a and b, have a subdomain relationship to a common superdomain γ (as illustrated here on the right). Domain α in join operand q_1 contains $\|\alpha\|/\|\gamma\|$ of the values of γ. Based on the assumption that α and β have been derived from γ independently,[2] Rule Dom-12 uses this factor and the domain restriction formula from Appendix B to derive the domain sizes associated with all columns coming from input relation q_2 (and, vice versa, $\|\beta\|/\|\gamma\|$ for column values from q_1).

The join operator on top of subexpression $\mathsf{loop}_{④}$ in Figure 3 is an instance of this pattern. The right input of this join contains the iter values of those iterations over `doc("forecast.xml")/descendant::day` that did not satisfy `$ppcp > 50` (*i.e.*, the iterations that belong to the `else` branch). The left-hand side contains information only for iterations for which a `@t` attribute could be found. The domain established by $\varrho_{\mathsf{inner}:\langle \mathsf{iter},\mathsf{pos}\rangle}$ (in the bottom part of the plan) is a common superdomain of both join attributes.

Note that Rule Dom-11 actually is a special instance of Rule Dom-12.

Rule Dom-13 introduces a System R-like 10 % factor for domains that result from the relational difference operator $\setminus$. In Pathfinder-generated plans, operator $\setminus$ is predominantly used to work over single-column tables.[3] Virtually

[2]The last premise in Rule Dom-12 ensures that γ is the smallest common subdomain of α and β.

[3]Pathfinder uses $\setminus$ operators over single columns, *e.g.*, to compute the loop relation of an XQuery `else` branch (as shown in Figure 3) or to handle empty sequences (which are encoded as the absence of their iter value in the loop-lifted sequence encoding). Multi-column differences are, in fact, only needed to evaluate the XQuery `except` operator.

$$\frac{}{\mathit{dom}\,(\pi_{\dots,\mathsf{a}:v,\dots}(q)) \supseteq \left\{\mathsf{a}^{\alpha} \wedge \|\alpha\| \stackrel{!}{=} 1\right\}}\,(\textsc{Dom-2}) \qquad \frac{\square \in \{\circledless, \circledgtr, \circledvee, \dots\}}{\mathit{dom}\,\left(\square_{\mathsf{a}:(\mathsf{b}_1,\mathsf{b}_2)}(q)\right) \supseteq \mathit{dom}\,(q) \cup \left\{\mathsf{a}^{\alpha} \wedge \|\alpha\| \stackrel{!}{=} 2\right\}}\,(\textsc{Dom-3})$$

$$\frac{\mathsf{a}^{\alpha} \in \mathit{dom}\,(q)}{\mathit{dom}\,\left(\pi_{\dots,\mathsf{b}:\mathsf{a},\dots}(q)\right) \supseteq \{\mathsf{b}^{\alpha}\}}\,(\textsc{Dom-4}) \qquad \frac{}{\mathit{dom}\,(\delta(q)) = \mathit{dom}\,(q)}\,(\textsc{Dom-5}) \qquad \frac{}{\mathit{dom}\,(q_1 \times q_2) = \mathit{dom}\,(q_1) \cup \mathit{dom}\,(q_2)}\,(\textsc{Dom-6})$$

$$\frac{\mathsf{c}^{\gamma} \in \mathit{dom}\,(q)}{\mathit{dom}\,\left(\varrho_{\mathsf{a}:\langle \mathsf{b}_1,\dots,\mathsf{b}_n\rangle \| \mathsf{c}}(q)\right) \supseteq \mathit{dom}\,(q) \cup \left\{\mathsf{a}^{\alpha} \wedge \|\alpha\| \stackrel{!}{=} |q|/\|\gamma\|\right\}}\,(\textsc{Dom-7}) \qquad \frac{\mathit{cols}(q) = \{\mathsf{b}\} \qquad \mathsf{b}^{\beta} \in \mathit{dom}\,(q)}{\mathit{dom}\,\left(\mathit{agg}_{\mathsf{a}\|\mathsf{b}}(q)\right) \supseteq \mathit{dom}\,(q) \cup \left\{\mathsf{a}^{\alpha} \wedge \|\alpha\| \stackrel{!}{=} \|\beta\|\right\}}\,(\textsc{Dom-8})$$

$$\frac{}{\mathit{dom}\,(\sigma_{\mathsf{a}}(q)) \supseteq \left\{\mathsf{a}^{\alpha} \wedge \|\alpha\| \stackrel{!}{=} 1\right\} \cup \left\{\mathsf{c}^{\gamma_2} \mid \mathsf{c}^{\gamma_1} \in \mathit{dom}\,(q) \wedge \gamma_2 \stackrel{!}{\sqsubseteq} \gamma_1 \wedge \|\gamma_2\| \stackrel{!}{=} \|\gamma_1\| \cdot \left(1 - (1 - 1/10)^{|q|/\|\gamma_1\|}\right)\right\}}\,(\textsc{Dom-9})$$

$$\frac{}{\mathit{dom}\,(q_1 \bowtie_{\mathsf{a}=\mathsf{b}} q_2) \supseteq \left\{\mathsf{c}^{\gamma_2} \mid \mathsf{c}^{\gamma_1} \in \mathit{dom}\,(q_1) \cup \mathit{dom}\,(q_2) \wedge \gamma_2 \stackrel{!}{\sqsubseteq} \gamma_1 \wedge \|\gamma_2\| \stackrel{!}{=} \|\gamma_1\| \cdot \left(1 - (1 - 1/10)^{|q_2|/\|\gamma_1\|}\right)\right\}}\,(\textsc{Dom-10})$$

$$\frac{\mathsf{a}^{\alpha} \in \mathit{dom}\,(q_1) \qquad \mathsf{b}^{\beta} \in \mathit{dom}\,(q_2) \qquad \alpha \sqsubseteq \beta}{\mathit{dom}\,(q_1 \bowtie_{\mathsf{a}=\mathsf{b}} q_2) \supseteq \mathit{dom}\,(q_1) \cup \{\mathsf{b}^{\alpha}\} \cup \left\{\mathsf{c}^{\gamma_2} \mid \mathsf{c}^{\gamma_1} \in \mathit{dom}\,(q_2) \wedge \gamma_2 \stackrel{!}{\sqsubseteq} \gamma_1 \wedge \|\gamma_2\| \stackrel{!}{=} \|\gamma_1\| \cdot \left(1 - \left(1 - \frac{\|\alpha\|}{\|\beta\|}\right)^{|q_2|/\|\gamma_1\|}\right)\right\}}\,(\textsc{Dom-11})$$

$$\frac{\mathsf{a}^{\alpha} \in \mathit{dom}\,(q_1) \qquad \mathsf{b}^{\beta} \in \mathit{dom}\,(q_2) \qquad \alpha \sqsubseteq \gamma \qquad \beta \sqsubseteq \gamma \qquad \nexists \gamma' : (\alpha \sqsubseteq \gamma' \wedge \beta \sqsubseteq \gamma' \wedge \sqsubseteq \gamma \wedge \gamma \neq \gamma')}{\begin{array}{l}\mathit{dom}\,(q_1 \bowtie_{\mathsf{a}=\mathsf{b}} q_2) \supseteq \left\{\mathsf{c}^{\gamma_2} \mid \mathsf{c}^{\gamma_1} \in \mathit{dom}\,(q_1) \wedge \gamma_2 \stackrel{!}{\sqsubseteq} \gamma_1 \wedge \|\gamma_2\| \stackrel{!}{=} \|\gamma_1\| \cdot \left(1 - \left(1 - \frac{\|\beta\|}{\|\gamma\|}\right)^{|q_1|/\|\gamma_1\|}\right)\right\} \\ \qquad \cup \left\{\mathsf{c}^{\gamma_2} \mid \mathsf{c}^{\gamma_1} \in \mathit{dom}\,(q_2) \wedge \gamma_2 \stackrel{!}{\sqsubseteq} \gamma_1 \wedge \|\gamma_2\| \stackrel{!}{=} \|\gamma_1\| \cdot \left(1 - \left(1 - \frac{\|\alpha\|}{\|\gamma\|}\right)^{|q_2|/\|\gamma_1\|}\right)\right\}\end{array}}\,(\textsc{Dom-12})$$

$$\frac{}{\mathit{dom}\,(q_1 \setminus q_2) \supseteq \left\{\mathsf{c}^{\beta} \mid \mathsf{c}^{\alpha} \in \mathit{dom}\,(q_1) \wedge \beta \stackrel{!}{\sqsubseteq} \alpha \wedge \|\beta\| \stackrel{!}{=} \|\alpha\| \cdot \left(1 - (1 - 1/10)^{|q_2|/\|\gamma_1\|}\right)\right\}}\,(\textsc{Dom-13})$$

$$\frac{\begin{array}{c}\mathit{cols}(q_1) = \mathit{cols}(q_2) = \{\mathsf{a}\} \qquad \mathsf{a}^{\alpha} \in \mathit{dom}\,(q_1) \\ \mathsf{a}^{\beta} \in \mathit{dom}\,(q_2) \qquad \beta \sqsubseteq \alpha\end{array}}{\mathit{dom}\,(q_1 \setminus q_2) \supseteq \left\{\mathsf{a}^{\gamma} \wedge \gamma \stackrel{!}{\sqsubseteq} \alpha \wedge \|\gamma\| \stackrel{!}{=} \|\alpha\| - \|\beta\|\right\}}\,(\textsc{Dom-14}) \qquad \frac{\begin{array}{c}\mathit{cols}(q_1) = \mathit{cols}(q_2) = \{\mathsf{a}\} \qquad \mathsf{a}^{\alpha} \in \mathit{dom}\,(q_1) \\ \mathsf{a}^{\beta} \in \mathit{dom}\,(q_2) \qquad \alpha \sqsubseteq \beta\end{array}}{\mathit{dom}\,(q_1 \setminus q_2) \supseteq \left\{\mathsf{a}^{\gamma} \wedge \gamma \stackrel{!}{\sqsubseteq} \alpha \wedge \|\gamma\| \stackrel{!}{=} 0\right\}}\,(\textsc{Dom-15})$$

Figure 4: Domain inference. Abstract domain identifiers $\alpha, \dots, \gamma$ represent static approximations of runtime value domains. Inference rules also infer estimated domain sizes $\|\cdot\|$ and subdomain relationships $\cdot \sqsubseteq \cdot$.

all cases thus benefit from more specific domain inference rules such as the ones shown in Rule Dom-14 and Dom-15. The former covers the situation that we also see in Figure 3: The value domain of the single-column relation $\mathsf{loop}_{③}$ is a subset of the values in $\mathsf{loop}_{②}$. Hence, subtraction of the two input domain sizes $\|\alpha\|$ and $\|\beta\|$ yields the domain size of the output domain γ, $\|\gamma\| \stackrel{!}{=} \|\alpha\| - \|\beta\|$ (see Rule Dom-14). Rule Dom-15 is the complimentary rule that decides $\|\gamma\| \stackrel{!}{=} 0$ based on $\alpha \sqsubseteq \beta$.

3.2 Table Cardinalities

The collected domain information can now be used to compute meaningful cardinalities for subexpressions in the relational plan. Figure 5 lists the missing inference rules that correspond to the plan situations discussed earlier.

Rules Card-8 and Card-9 correspond to Rules Dom-11 and Dom-12 in Figure 4, respectively. Domain sizes are used here to estimate the selectivity of the join predicate $\mathsf{a} = \mathsf{b}$. In Rule Card-8, the result contains $\|\alpha\|$ distinct values in the join attribute. Each of these finds $|q_1|/\|\alpha\|$ and $|q_2|/\|\beta\|$ rows from the left- and right-hand-side of the join, respectively, such that the cardinality can be estimated to $\frac{|q_1| \cdot |q_2|}{\|\beta\|}$. The domain size of the join attribute in the result becomes $\frac{\|\alpha\| \cdot \|\beta\|}{\|\gamma\|}$ in the generalized rule Card-9.

The latter two rules, Card-10 and Card-11, implement cardinality estimation for the general case of the relational difference (Rule Card-10) and for the special case we discussed in the previous section (Rule Card-11).

Still, our rules do not yet cover the estimation of sub-plans that depend on XPath navigation or improve the estimation accuracy for the selection operator σ_{a}. Both tasks require access to statistical information about the underlying (XML) data.

4. INTERFACING WITH XPATH

Access to XML documents is made explicit in our relational algebra (Table 2) in terms of the ⌟, doc, and ⚛ operators. These operators may be backed by system-dependent implementations for the respective XQuery functionality. In a purely relational XQuery setup, all three operators are typically expanded into relational "micro-plans" that operate over a relational XML encoding.

Given node identifiers γ_i stored in a column b, the step operator ⌟$_{\mathsf{a}:ax::nt(\mathsf{b})}$ evaluates the location step $ax::nt$ for each node in b and populates a new column a with the node identifiers of the result nodes. Figure 6 illustrates this for the step `child::*` and a five-node XML instance.

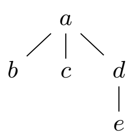
The XML document access operator $\mathtt{doc}_{\mathsf{a}:\mathsf{b}}$ uses the URIs in column b to look up the document nodes of XML instances. Their node identifiers are populated into the new column a.

Operator ⚛$_{\mathsf{a}:\mathsf{b}}$ implements XQuery atomization [4], *i.e.*, the extraction of simple-typed data from XML node content. The new column a holds the values obtained from atomizing the XML nodes referenced by the identifiers in column b. In Figure 3, we used ⚛ to extract the simple-typed (chance of

$$\frac{\mathsf{a}^{\alpha} \in \mathit{dom}\,(q_1) \qquad \mathsf{b}^{\beta} \in \mathit{dom}\,(q_2) \qquad \alpha \sqsubseteq \beta}{|q_1 \underset{\mathsf{a}=\mathsf{b}}{\bowtie} q_2| = \dfrac{|q_1| \cdot |q_2|}{\|\beta\|}}\,(\textsc{Card-8}) \qquad \frac{\begin{array}{c}\mathsf{a}^{\alpha} \in \mathit{dom}\,(q_1) \qquad \mathsf{b}^{\beta} \in \mathit{dom}\,(q_2) \qquad \alpha \sqsubseteq \gamma \qquad \beta \sqsubseteq \gamma \\ \nexists \gamma' : (\alpha \sqsubseteq \gamma' \wedge \beta \sqsubseteq \gamma' \wedge \sqsubseteq \gamma \wedge \gamma \neq \gamma')\end{array}}{|q_1 \underset{\mathsf{a}=\mathsf{b}}{\bowtie} q_2| = \dfrac{|q_1| \cdot |q_2|}{\|\gamma\|}}\,(\textsc{Card-9})$$

$$\frac{}{|q_1 \setminus q_2| = |q_1| \cdot {}^{1}/_{10}}\,(\textsc{Card-10}) \qquad \frac{\begin{array}{c}\mathsf{a}^{\alpha} \in \mathit{dom}\,(q_1) \qquad \mathsf{a}^{\beta} \in \mathit{dom}\,(q_2) \\ \beta \sqsubseteq \alpha \qquad \mathit{cols}(q_1) = \mathit{cols}(q_2) = \{\mathsf{a}\}\end{array}}{|q_1 \setminus q_2| = |q_1| - |q_2|}\,(\textsc{Card-11}) \qquad \frac{\begin{array}{c}\mathsf{a}^{\alpha} \in \mathit{dom}\,(q_1) \qquad \mathsf{a}^{\beta} \in \mathit{dom}\,(q_2) \\ \alpha \sqsubseteq \beta \qquad \mathit{cols}(q_1) = \mathit{cols}(q_2) = \{\mathsf{a}\}\end{array}}{|q_1 \setminus q_2| = 0}\,(\textsc{Card-12})$$

Figure 5: Cardinality estimation rules that correspond to the domain inference shown in Figure 4.



(a) XML tree.

b	c
γ_a	1
γ_b	2
γ_d	3

$\xrightarrow{\lrcorner_{\mathsf{a:child::*(b)}}}$

a	b	c
γ_b	γ_a	1
γ_c	γ_a	1
γ_d	γ_a	1
γ_e	γ_d	3

(b) Operator $\lrcorner_{\mathsf{a:child::*(b)}}$.

Figure 6: Semantics of step operator $\lrcorner_{\mathsf{a}:ax::nt(\mathsf{b})}$.

precipitation) data from the `ppcp` elements returned by the XPath navigation $\lrcorner_{\mathsf{item:descendant::ppcp(item)}}$.

4.1 XPath Navigation

With all operations on XML data made explicit using distinguished algebra operators, our cardinality estimation procedure remains mostly independent of its XPath estimation subsystem and can play well together with existing XPath estimation techniques proposed in the literature. We have successfully built implementations based on straight line tree grammars [8] and data guides [9].

To facilitate the interaction with the XPath estimation subsystem, we adapt the idea of *projection paths* [15]. Our query analyzer infers a trace of all XPath navigation steps that have been followed to compute the result of a plan subexpression. Traces (projection paths) are then used to guide the interaction with the XPath estimation subsystem. In [15], similar information was used to pre-filter XML instances at document loading time (and, consequently, reduce the main-memory requirements of the Galax XQuery processor).

4.1.1 Projection Paths

Projection paths are inferred as the $\mathit{path}\,(q)$ property of a plan operator q. An entry $\mathsf{a}^{\Rightarrow p}$ in $\mathit{path}\,(q)$ indicates that all node identifiers in column a in the output of q have been reached by an XPath navigation along the path p, possibly constrained by additional predicates. The node references in a, therefore, are a subset of those returned by the XPath expression p.

Unlike [15], we allow projection paths composed of arbitrary XPath axes and node tests. Our current plan analyzer does not generate predicated projection paths (*i.e.*, paths of the form p_1[p_2]), since those are normalized into explicit `for` iterations prior to query compilation [7]. Our setup could easily be modified to generate such paths if an XPath estimation subsystem provides specialized support for predicates, however. Further, we do not label projection paths with any additional flags (such as the `#` in [15]).

Informally, a new projection path is instantiated for every call to the XQuery built-in function `fn:doc`$(\cdot)$ (operator $\mathtt{doc}_{\mathsf{a:b}}$ in the algebraic plan). As the analyzer processes the plan bottom-up, each occurrence of an XPath navigation step ax::nt (operator $\lrcorner_{\mathsf{a}:ax::nt(\mathsf{b})}$) is recorded as an appendix to the existing projection path information. The remaining plan operators only propagate projection paths bottom-up.

The process is covered by Inference Rules Path-1 to Path-7 in Figure 7. Rule Path-1 establishes a new projection path for the result column of a of operator $\mathtt{doc}_{\mathsf{a:b}}$. On occurrence of a step operator $\lrcorner_{\mathsf{a}:ax::nt(\mathsf{b})}$, this path is extended by the step ax::nt and annotated to the output column b of the operator (Rule Path-2).

Otherwise, projection path information is propagated bottom-up. Rules Path-3 to Path-7 thereby ensure proper treatment of attribute renaming and projection (Rule Path-3) and of the semantics of the set operators $\setminus$ and $\uplus$ (Rules Path-5 and Path-7, respectively).

In Figure 3, the output column item of operator $\mathtt{doc}_{\mathsf{item}}$ (bottom of the plan) is annotated with the projection path `fn:doc("forecast.xml")`. Operators $\lrcorner_{\mathsf{item}:ax::nt(\mathsf{item})}$ in the upstream DAG then update the projection path information recorded for column item to read `fn:doc("forecast.xml")/attribute::t` and `fn:doc("forecast.xml")/descendant::ppcp` in the left and right branches of the query plan, respectively.

4.1.2 Cardinality Estimation for XPath Steps

XPath navigation also affects table cardinalities, as can be seen in Figure 6. Three rows are contributed to the operator output by the first input row (since a has three children in Figure 6(a)). The second row disappears during step evaluation (b/`child::*` is empty), while the last row produces one output row.

The effect of operator $\lrcorner_{\mathsf{a}:ax::nt(\mathsf{b})}$ on the table cardinality is determined by the *fanout* of the node identifiers in column b with respect to the location step ax::nt. In Figure 6, the average fanout of the input nodes (nodes a, b, and d) is

$$f_{\text{avg}} = \frac{3+0+1}{3} = {}^{4}/_{3} \ .$$

Multiplication with the input cardinality yields the row count of the result:

$$\left|\lrcorner_{\mathsf{a:child::*(b)}}(q)\right| = |q| \cdot f_{\text{avg}} = 3 \cdot {}^{4}/_{3} = 4 \ .$$

We can estimate the factor f_{avg} involved in determining the cardinality of $\lrcorner_{\mathsf{a}:ax::nt}(q)$ based on the projection path p that has been inferred for the context column b in q (*i.e.*, $\mathsf{b}^{\Rightarrow p} \in \mathit{path}\,(q)$). We base the estimate for f_{avg} on statistical information about the XML document:

$$f_{\text{avg}} \approx \mathsf{Pr}_{ax::nt}\,(p) := \frac{\mathtt{fn:count}\,(p/ax::nt)}{\mathtt{fn:count}\,(p)} \ .$$

$$\frac{\mathsf{b}^{=v} \in \mathit{const}\,(q)}{\mathit{path}\,(\mathtt{doc}_{\mathsf{a:b}}(q)) \supseteq \mathsf{a}^{\Rightarrow \mathtt{fn:doc}(v)} \cup \mathit{path}\,(q)}\,(\textsc{Path-1}) \qquad \frac{\mathsf{b}^{\Rightarrow p} \in \mathit{path}\,(q)}{\mathit{path}\,\left(\lrcorner_{\mathsf{a}:ax::nt(\mathsf{b})}(q)\right) \supseteq \mathsf{a}^{\Rightarrow p/ax::nt} \cup \mathit{path}\,(q)}\,(\textsc{Path-2})$$

$$\frac{\mathsf{b}^{\Rightarrow p} \in \mathit{path}\,(q)}{\mathit{path}\,\left(\pi_{\dots,\mathsf{a:b},\dots}(q)\right) \supseteq \{\mathsf{a}^{\Rightarrow p}\}}\,(\textsc{Path-3}) \quad \frac{\Box \in \left\{\sigma_{\mathsf{a}}, \delta, \varrho_{\mathsf{a}:\langle \mathsf{b}_1,\dots,\mathsf{b}_n\rangle \| \mathsf{c}}, \circledast_{\mathsf{a}:(\mathsf{b}_1,\mathsf{b}_2)}, \text{⚛}_{\mathsf{a:b}}, \mathtt{count}_{\mathsf{a}\|\mathsf{b}}\right\}}{\mathit{path}\,(\Box(q)) \supseteq \mathit{path}\,(q)}\,(\textsc{Path-4}) \quad \frac{}{\mathit{path}\,(q_1 \setminus q_2) \supseteq \mathit{path}\,(q_1)}\,(\textsc{Path-5})$$

$$\frac{\Box \in \{\times, \bowtie_{\mathsf{a}=\mathsf{b}}\}}{\mathit{path}\,(q_1 \,\Box\, q_2) \supseteq \mathit{path}\,(q_1) \cup \mathit{path}\,(q_2)}\,(\textsc{Path-6}) \qquad \frac{}{\mathit{path}\,(q_1 \uplus q_2) \supseteq \left\{\mathsf{a}^{\Rightarrow p_1 | p_2} \mid \mathsf{a}^{\Rightarrow p_1} \in \mathit{path}\,(q_1) \wedge \mathsf{a}^{\Rightarrow p_2} \in \mathit{path}\,(q_2)\right\}}\,(\textsc{Path-7})$$

$$\frac{}{|\mathtt{doc}_{\mathsf{a:b}}(q)| = |q|}\,(\textsc{Card-13}) \qquad \frac{\mathsf{b}^{\Rightarrow p} \in \mathit{path}\,(q)}{\left|\lrcorner_{\mathsf{a}:ax::nt(\mathsf{b})}(q)\right| = |q| \cdot \mathsf{Pr}_{ax::nt}\,(p)}\,(\textsc{Card-14})$$

$$\frac{\mathsf{b}^{\beta} \in \mathit{dom}\,(q) \qquad \mathsf{b}^{\Rightarrow p} \in \mathit{path}\,(q)}{\begin{array}{c}\mathit{dom}\,\left(\lrcorner_{\mathsf{a}:ax::nt(\mathsf{b})}(q)\right) \supseteq \left\{\mathsf{a}^{\alpha} \wedge \|\alpha\| \overset{!}{=} \|\beta\| \cdot \mathsf{Pr}_{ax::nt}\,(p)\right\} \\ \cup \left\{\mathsf{c}^{\gamma_2} \mid \mathsf{c}^{\gamma_1} \in \mathit{dom}\,(q) \wedge \gamma_2 \overset{!}{\sqsubseteq} \gamma_1 \wedge \|\gamma_2\| \overset{!}{=} \|\gamma_1\| \cdot \left(1 - (1 - \mathsf{Pr}_{[ax::nt]}\,(p))^{|q|/\|\gamma_1\|}\right)\right\}\end{array}}\,(\textsc{Dom-16})$$

Figure 7: XPath-related property inference. The inference of $\mathit{path}\,(\cdot)$ resembles the tracking of projection paths in [15]. Access to XML document statistics is encapsulated into functions $\mathsf{Pr}_{\dots}(p)$.

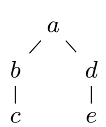
The cardinality of $\lrcorner_{\mathsf{a}:ax::nt(\mathsf{b})}(q)$ can then be approximated as shown in Rule Card-14 (Figure 7).

The approximation assumes that the nodes referenced in input column b are a random sample of those reachable by p. In particular, they are assumed to be picked from p independently of their fanout with respect to $ax::nt$. This assumption is met by most real-world queries that we could get hold of in experimental studies.

The *fanout function* $\mathsf{Pr}_{ax::nt}\,(p)$ is part of our interface to the XPath estimation subsystem. Every XPath estimator that provides $\mathsf{Pr}_{ax::nt}\,(p)$ and the two functions that we define in a moment can seamlessly be plugged into the XQuery estimator described here. In Section 4.3, we illustrate a naïve implementation based on Goldman and Widom's data guides [9].

4.1.3 Domains and XPath Location Steps

With regards to value domains, operator $\lrcorner_{\mathsf{a}:ax::nt(\mathsf{b})}(q)$ acts like a *filter* on all column values coming from the input relation q. Only rows for which at least one node can be found along the step $ax::nt$ will appear in the operator result. Since, *e.g.*, in Figure 6, node b has no children reachable by `child::*`, the row $[\gamma_b, 2]$ does not contribute to the operator output.



The selectivity of the filter is independent of the average fanout $\mathsf{Pr}_{ax::nt}\,(p)$ that we used before. Evaluated over the XML instance shown on the left, *e.g.*, the operation shown in Figure 6(b) would still yield four result rows ($f_{\text{avg}} = 4/3$), but all input rows would now "survive" operation $\lrcorner_{\mathsf{a}:\mathtt{child::*}(\mathsf{b})}$.

To judge the impact of $\lrcorner$ on domain sizes, we thus introduce the *selectivity function* $\mathsf{Pr}_{[ax::nt]}\,(p)$ as a second interface to request statistical information from the XPath estimation subsystem:

$$\mathsf{Pr}_{[ax::nt]}\,(p) := \frac{\mathtt{fn:count}\,(p[ax::nt])}{\mathtt{fn:count}\,(p)} \quad ,$$

where the location step $ax::nt$ is now used inside a predicate to the location path p. The selectivity function satisfies $0 \le \mathsf{Pr}_{[ax::nt]}\,(p) \le 1$ by definition.

Only two of the three input nodes in Figure 6 have children reachable via `child::*`, such that the selectivity of the step is $2/3$. Using the Domain Inference Rule Dom-16 in Figure 7 (right-hand input to $\cup$), we thus determine the domain size of columns b and c in the expression result as $3 \cdot 2/3 = 2$. Evaluated over the modified XML instance above instead, the selectivity function would now yield 1 (and, hence, a domain size of 3 for result columns b and c).

4.1.4 Node Construction

Apart from its navigation sub-language XPath, XQuery has been equipped with functionality also to construct new tree fragments at query runtime. Operators ε and τ in Table 2 make this functionality explicit in our algebra and mimic XQuery element and text node construction, respectively. Both operators expand into "micro plans" which essentially compute an aggregate over an input relation that holds the content node sequence (see [12] for details).

From the perspective of cardinality estimation, such functionality is straightforward to handle. We have already seen in Rules Dom-8 and Card-6 how domain information and cardinalities can be inferred for aggregation functions, respectively. The forecasted cardinality, typically the size of the domain associated with column iter, is consistent with the semantics of node construction in XQuery. Consider Query Q_2 again: each evaluation of `element gust { e }` yielded *exactly one* new element node, regardless of the cardinality of e.

More valuable than the projected size of the node construction *result* may be information about the cardinality of the constructor's *input*. Since we infer $|e|$ for any subexpression e, such information is readily available to, *e.g.*, allocate enough memory to hold the content of the new tree fragment below element `gust`.

4.2 Value-Based Predicates

The reliance on functions $\mathsf{Pr}_{\dots}(\cdot)$ to access statistical information about XPath navigation enables data-dependent cardinality estimation only for the structural aspects of XML document access. To judge the selectivity of the predicate `$ppcp > 50` in Query Q_1, we also need to have information about the distribution of *values* in `forecast.xml`.

4.2.1 Typed Value Histograms

Our estimator assumes the availability of such information in terms of *typed value histograms*, which can be set up by

$$\frac{\mathsf{b}^{\Rightarrow p} \in \mathit{path}(q) \qquad H = \mathsf{Hist}(p)}{\mathit{hist}(\text{⚛}_{\mathsf{a:b}}(q)) \supseteq \left\{\mathsf{a}^{\text{📊}\,H}\right\}} \text{(Hist-1)} \qquad \frac{\left\{\mathsf{b}_1^{\text{📊}\,H_1}, \ldots, \mathsf{b}_n^{\text{📊}\,H_n}\right\} \in \mathit{hist}(q)}{\mathit{hist}\left(\circledast_{\mathsf{a:(b_1,\ldots,b_n)}}(q)\right) \supseteq \left\{\mathsf{a}^{\text{📊}\,\boxast(H_1,\ldots,H_n)}\right\}} \text{(Hist-2)} \qquad \frac{\mathsf{a}^{\text{📊}\,H} \in \mathit{hist}(q)}{|\sigma_{\mathsf{a}}(q)| = |q| \cdot H[\mathsf{true}]} \text{(Card-15)}$$

$$\frac{\mathsf{a}^{\text{📊}\,H} \in \mathit{hist}(q)}{\mathit{dom}(\sigma_{\mathsf{a}}(q)) \supseteq \left\{\mathsf{a}^{\alpha} \wedge \|\alpha\| \stackrel{!}{=} 1\right\} \cup \left\{\mathsf{c}^{\gamma_2} \mid \mathsf{c}^{\gamma_1} \in \mathit{dom}(q) \wedge \gamma_2 \sqsubseteq^! \gamma_1 \wedge \|\gamma_2\| \stackrel{!}{=} \|\gamma_1\| \cdot \left(1 - (1 - H[\mathsf{true}])^{|q|/\|\gamma_1\|}\right)\right\}} \text{(Dom-17)}$$

Figure 8: Inference and use of typed value histograms (annotation $\mathit{hist}(\cdot)$).

the database administrator for frequently queried values in the XML document catalog. A histogram created with, for example,

```
create typed value histogram H1
  on '/descendant::day/descendant::ppcp'
  validate as xs:integer ,
```

can be used to judge the selectivity of the predicate in Query Q_1. Typed value histograms of this kind are readily provided by, *e.g.*, the XML data indices in IBM DB2 9 [13].

We leave histogram maintenance up to the XPath subsystem and remain fully agnostic with respect to its concrete implementation here. The only assumption we make is that the histogram for path p, if available, is accessible from the XPath estimation subsystem via the interface function $\mathsf{Hist}(p)$.

4.2.2 Trading Paths for Histograms

In XQuery, access to typed value information requires *atomization* of the respective XML tree nodes. Calls to the XQuery built-in function `fn:data(·)` make this process explicit in the query after normalization to XQuery Core [7]. In relational XQuery evaluation plans, the atomization operator $\text{⚛}_{\mathsf{a:b}}$ marks this situation where the query engine trades nodes for values.

It is the same spot where our plan analyzer trades projection path annotations for typed value histograms. In Rule Hist-1 (Figure 8), it uses the projection path inferred for the input column b to request the typed value histogram H from the XPath subsystem. The histogram is then recorded as $\mathsf{a}^{\text{📊}\,H}$ in the $\mathit{hist}(\cdot)$ annotation of the result expression.

Operations on values (arithmetic computations and value comparisons) are represented explicitly using the $\circledast$ operators in our algebra ($e.g.$, $\oplus_{\mathsf{a:(b_1,b_2)}}$, $\ominus_{\mathsf{a:(b_1,b_2)}}$, $\oslash_{\mathsf{a:(b_1,b_2)}}$, …).[4] To reflect these operations in algebraic histogram annotations, a new histogram is computed for the result column a based on histograms available for the input columns b_i. In Rule Hist-2, we used $\boxast(H_1,\ldots,H_n)$ to express arithmetics on histograms. A possible implementation for $\boxast$ is the histogram discretization technique by Berleant [3]. Histogram information is propagated bottom-up for the remaining algebra operators (not shown formally).

With histogram information available, the cardinality inference for the selection operator σ_{a} now becomes an educated guess. As shown in Rule Card-15, the two-bucket ($\mathsf{true}/\mathsf{false}$) histogram annotated to column a readily describes the selectivity of σ_{a}. In Rule Dom-17, we also use it to infer domain sizes associated with the output of σ_{a}.

In Figure 3, a typed value histogram is fetched from the XPath estimator to annotate column item of the atomization

[4] Selection σ_{a} and $\bowtie_{\mathsf{a=b}}$ can be implemented as first-order operators this way.

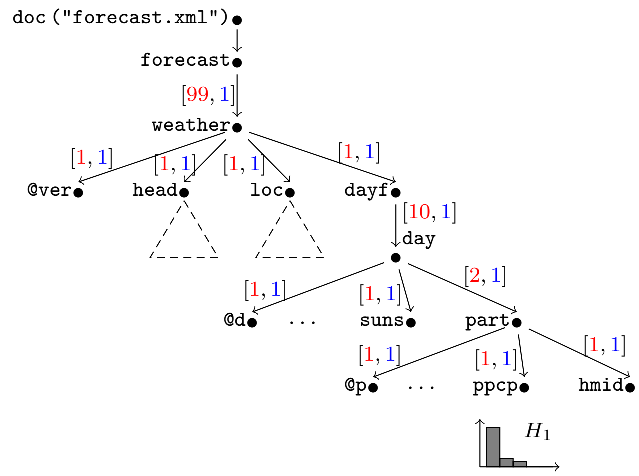


Figure 9: Data guide for `weather.com` data. Edge annotations $[f, s]$ indicate fanout f and selectivity s.

operator ⚛_{item}. After propagation through the join $\bowtie_{\mathsf{iter=iter1}}$ and histogram arithmetics in $\oslash_{\mathsf{res:(item,item1)}}$ this histogram is then used to judge the output cardinality of σ_{res}.

4.3 An Implementation for Pr: Data Guides

The two interface functions to access fanout, $\mathsf{Pr}_{ax::nt}(p)$, and selectivity, $\mathsf{Pr}_{[ax::nt]}(p)$, in Section 4.1 suggest the use of data guides [9] to maintain statistical information about the underlying document structure. In a nutshell, a data guide is built by reducing element nodes with identical root-to-leaf paths to a single instance in the guide. The outcome is a "skeleton tree" holding all distinct paths (we assume a *strong* data guide in the sense of [9]) that may be annotated with, *e.g.*, statistical information.

4.3.1 Fanouts and Selectivities

Figure 9 shows the data guide that corresponds to a small collection of weather data for cities in New Zealand that we retrieved from `weather.com` at the time of this writing.[5] To implement the two XPath interface functions, each edge in the data guide is labeled with a pair of values $[f, s]$ ($f > 0$ and $0 < s \leq 1$), which correspond to the average fanout and selectivity (respectively) along the corresponding axis in the full document. The document contains, *e.g.*, ten `day` elements below each `dayf`, each `day` contains two `parts` (day and night). The structure of weather data is more deterministic than the forecast it describes: each edge is guaranteed to be present for corresponding parent nodes in the document, hence $s \equiv 1$ in our example.

Based on these annotations, $\mathsf{Pr}_{ax::nt}(p)$ and $\mathsf{Pr}_{[ax::nt]}(p)$

[5] See `http://www.weather.com/services/xmloap.html` for instructions on how to use the `weather.com` web service.

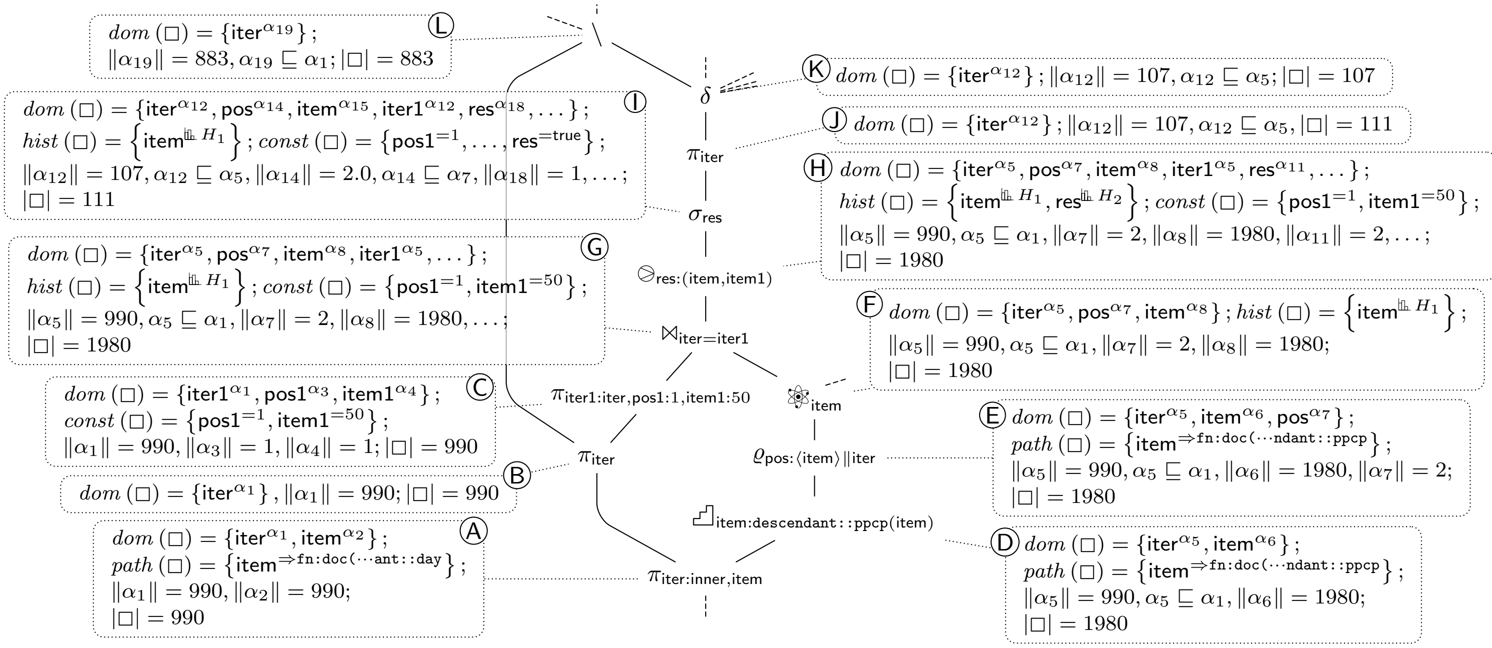


Figure 10: Full annotations for a sub-plan of Figure 3 (annotations explained in Section 5.1).

are straightforward to implement. For named `child` steps, both pieces of information can directly be read from the annotations. Otherwise, fanout information can be computed by adding horizontally and multiplying vertically along guide edges that qualify for the given step and node test. The aggregation of selectivity values is beyond the scope of this paper and uses similar observations as those that we sketch in Appendix B. Note that data guides do not capture the order between siblings and hence cannot be used to implement $\mathsf{Pr}_{ax::nt}(p)$ for order-sensitive axes *ax* such as, *e.g.*, the `following` or `preceding-sibling` axes. To support estimation of order-sensitive axes, other synopses can be used, such as [14] or the straight line tree grammars of [8].

4.3.2 Typed Value Histograms

Figure 9 also illustrates how our prototype implements typed value histograms as annotations to nodes in the data guide. Histogram H_1, created in Section 4.2.1, is annotated to the `ppcp` data guide node. (If, in $\mathsf{Hist}(p)$, more than one histogram can be found for path p, our implementation uses discretization [3] to compute the effective typed value histogram.)

5. FORECASTING IN PRACTICE

With all bits and pieces together, we are now ready to infer plan annotations and cardinalities for the query plan in Figure 3. For space reasons, we only report inferred plan properties for its most interesting sub-plan, illustrated in Figure 10. Towards the end of this section, we also report on empirical results for a realistic set of XQuery expressions, taken from the W3C XQuery Use Cases Note [6].

5.1 Zooming in on the Running Example

The input to the sub-plan in Figure 10 (operator Ⓐ) is essentially determined by the XPath subexpression result `doc("forecast.xml")/descendant::day`, whose cardinality we estimated to $99 \cdot 1 \cdot 10 = 990$, following the fanout annotations in the data guide. This information is propagated bottom-up along operators Ⓑ and Ⓒ.

Annotations in the right branch of the plan depend on the projection path information available for column item. Using $\mathsf{Pr}_{\texttt{descendant::ppcp}}(\texttt{fn:doc(}\cdots\texttt{)/descendant::day}) = 2$ (fanout annotation in Figure 9), we can infer the cardinality of operator Ⓓ to 1980 (Rule CARD-14), as well as the domain size of α_6, $\|\alpha_6\| = 1980$ (left part of Rule DOM-16). By contrast, since the selectivity $\mathsf{Pr}_{[\texttt{descendant::ppcp}]}(\cdots)$ of the step `descendant::ppcp` is 1, the domain size annotated to column iter remains $\|\alpha_5\| = 990$ (right part of Rule DOM-16). The factor $|\Box|/\|\alpha_5\| = 2$ stems from the two `ppcp` elements encountered in each iteration over `doc(···)/descendant::day`. This factor also leads to the domain size $\|\alpha_7\| = 2$ inferred for operator Ⓔ (using Rule DOM-7).

To annotate operator Ⓕ, we fetch the histogram H_1 from the data guide using the projection path `fn:doc(···)/···/descendant::ppcp` (Rule HIST-1). The histogram will be accessed later in the plan to judge the selectivity of the predicate `$ppcp > 50`.

Operator Ⓖ is an instance of the situation in Rules DOM-11 and CARD-8. Since each row in the right branch is guaranteed to find a (single) join partner in the left branch, we expect 1980 rows to flow upwards the execution plan.

To judge the effect of operators Ⓗ and Ⓘ, we access the histogram information for input column item. Together with the *const* $(\cdot)$ information available for column item1 (which we interpret as the second input histogram in Rule HIST-2), we compute a two-bucket histogram for column res that reflects the Boolean outcome of the comparison operator Ⓗ. Operator Ⓘ finally performs the selection and we determine its output cardinality based on Rule CARD-15.

After propagation of plan annotations through operator Ⓙ, the cardinality estimate for Ⓚ depends on the domain size annotated to column iter. Using Rule CARD-7, we infer 107 as its predicted row count. The cardinality of operator Ⓛ then is 883, according to Inference Rule CARD-11.

Recall that operators Ⓚ and Ⓛ correspond to the $\mathsf{loop}_{③}$ and $\mathsf{loop}_{④}$ relations, *i.e.*, the number of evaluations of the `then` and `else` branches of the original query, respectively. A query run over the real data reveals 158 and 832 evaluations of the two branches—the weather in New Zealand is slightly worse than expected. While this may look like a ominous forecast for VLDB, the deviation is actually caused by a skewed data distribution around a rain probability of 50 % (which meteorologists seem to avoid whenever possible). Substituted back into Q_1, we obtain the final forecast:

```
                                                  990
for $d in doc ("forecast.xml")/descendant::day
let $day := $d/@t
let $ppcp := data ($d/descendant::ppcp)
return                                           2301
  if ($ppcp > 50)                                 535
    then ("rain likely on", $day,
          "chance of precipitation:", $ppcp)
    else ("no rain on", $day)
                                  1766
```

Our overall cardinality forecast for Query Q_1 is 2301, which is $\approx 6\,\%$ off the actual result cardinality of 2454.

5.2 Forecasting the Real World

To judge the quality of computed estimates for realistic XQuery workloads, we analyzed a number of queries from the W3C XQuery Working Group Use Cases Note [6] using a prototype implementation of the full cardinality inference rule set. In addition to pure estimation accuracy, we also demonstrate how our prototype is able to gracefully recover from intermediate estimation errors, a behavior crucial to the construction of a robust XQuery optimizer.

We ran our experiments on extended sample data, as given in [6] and abstract from actual synopses by using histograms with one bucket per value for numeric node content (thus essentially keeping all of the data). We used a data guide-style XPath estimation subsystem equivalent to the one described in Section 4.3.

Figure 11 visualizes the deviation of our forecast from the tuple count observed over actual data. The deviation of each query subexpression is depicted by a circle ∘. A placement at coordinate 1 indicates that we predicted the subexpression cardinality correctly. If our estimate was 1/10 the actual value, the circle is placed at position 0.1. Respectively, the circle is printed at position 10 in case of a ten-fold over-estimate. If multiple subexpressions fall onto the same coordinate, we scale the circle proportionally to the number of such subexpressions. Finally, the estimate of the overall query is indicated by a filled circle •.

We report on a subset of the use case queries here and refer to the appendix for a more detailed experimental study.

As the figure shows, our system produces under-estimates for roughly 40 % of the subexpressions and over-estimates for only about 10 % of all subexpressions. Approximately 50 % of all estimates are exact. As detailed in Appendix A, the predominant cause for mis-estimations are string-based predicates (for which we did not set up any data synopses), value-based joins, and positional predicates.

Most valuable for the use of our inference system in practice is its ability to recover from intermediate mis-estimates gracefully. With the exception of query *XMP Q8*, our prototype implementation predicted the cardinality of the overall expression correctly, even though some of the subexpressions were forecast with lower accuracy. Our estimator takes advantage of the XQuery language semantics here. Language constructs with a grouping semantics (*e.g.*, node construction, the computation of effective Boolean values, or the existential semantics of comparisons) are pervasive in XQuery and allow the cardinality analysis to reasonably proceed even if the forecast went astray for a specific subplan.

For a deeper experimental assessment refer to Appendix A.

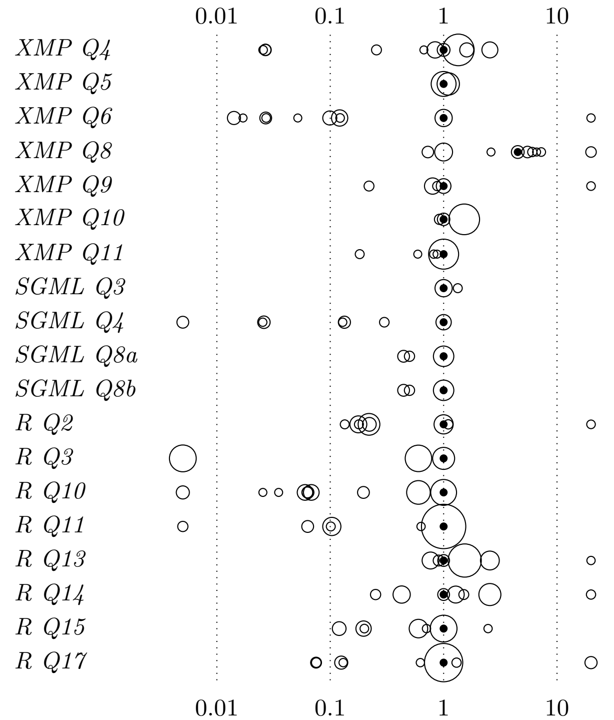


Figure 11: Estimation accuracy for selected W3C Use Cases (see appendix for further data).

6. MORE RELATED WORK

Work on cardinality estimation for semi-structured data has emerged even long before the advent of XML as a syntactical format and XPath/XQuery as a means to query XML data. Goldman and Widom have proposed data guides as a tool to hold statistical information in the *Lore* database manager [9]. We build on their work to demonstrate a simple implementation for the XPath aspect of our relational approach to XQuery estimation. Later work on XPath estimation (*e.g.*, [1, 8, 14, 16]) was mainly concerned with improved accuracy and the reduction of space. Since we strictly kept the estimation of XPath subexpressions separate in our work, all of them could serve as a drop-in replacement for the data guides in Section 4.3.

The separation of path estimation into fanout and selectivity has also been observed by Balmin *et al.* [2]. The pureXML query optimizer built into DB2 9 maintains statistics about XML data by means of the same two parameters. Their work also considers "lowest common ancestor" situations in a fashion similar to our domain analysis for tree-shaped domain relationships (Section 3.1.2), remains tied to certain syntactic patterns in the surface language, however.

Other estimation techniques that target XQuery (as opposed to just XPath) are surprisingly rare. Sartiani [18] looked at a restricted form of XQuery `for` clauses and their cardinalities. It remains unclear, however, whether his approach could be pushed to full XQuery support at all.

In Section 4.1.1, we picked up an idea of Marian and Siméon [15]. In the same way that the Galax XQuery processor analyzes navigation into XML documents, we use traces of XPath step navigation—projection paths—to associate statistical information about value distributions in simple-typed XML nodes to relational plans.

7. WRAP-UP

We have described a framework that fills the gap between the feature richness of the XQuery language and existing work on cardinality estimation for the XPath sub-language. Based on a relational representation of the input queries, we can re-use existing machinery from the relational domain to derive cardinality estimates for XQuery subexpressions in arbitrary compositions. Our strategy plays well together with estimation techniques for XPath proposed in earlier work (*e.g.*, [1, 8, 14, 16]), which can be plugged into our setup seamlessly.

To account for the characteristics of relational query plans that originate from XQuery, our work lays a focus on a peephole-style implementation of data flow analysis based on *abstract domain identifiers*. Abstract domain identifiers approximate the value space taken by individual table columns at runtime. Reasoning over inclusion relationship between domains and their sizes provides just the information that we need to derive cardinality estimates in a dependable manner.

Our setup remains agnostic with respect to the details of XPath location path estimation. A simple data guide-style implementation of this component proved sufficient to compute meaningful XQuery estimates in an experimental assessment.

The estimation procedure is defined in terms of a set of inference rules. As such, it provides a flexible basis for the addition of refined or domain-specific estimation rules. Currently we are looking into first-class support for positional predicates. With appropriate support in the statistics collection (*e.g.*, histograms as a replacement for the average fanout annotation in our data guide), an inference rule that matches the pattern

$$\varrho_{\mathsf{pos}:\langle\mathsf{b}\rangle\|\mathsf{c}}\left(\lrcorner_{\mathsf{a}:ax::nt(\mathsf{b})}\,(q)\right)$$

could annotate the output column `pos` with child distribution information. A ϱ operator of this kind is generated by the compiler for XPath location steps to set up a positional numbering according to the XML document order. A selection on `pos` later implements a positional predicate, whose effect we could judge with the annotated histogram.

Our experiments indicate that a closer look into value-based joins might be valuable for further accuracy and/or performance improvements. A possible approach could be the inspection of XML Schema information, or `ID/IDREF(S)` constraints in DTDs.

Acknowledgements

The experimental part of this work would have been impossible without the tremendous help we got from Jan Rittinger. Thanks, Jan!

8. REFERENCES

[1] A. Aboulnaga, A.R. Alameldeen, and J.F. Naughton. Estimating the Selectivity of XML Path Expressions for Internet Scale Applications. In *Proc. VLDB*, 2001.

[2] A. Balmin, T. Eliaz, J. Hornibrook, L. Lim, G.M. Lohman, D. Simmen, M. Wang, and C. Zhang. Cost-based Optimization in DB2 XML. *IBM Systems Journal*, 45(2), 2006.

[3] D. Berleant. Automatically Verified Reasoning with Both Intervals and Probability Density Functions. *Interval Computations*, No. 2, 1993.

[4] S. Boag, D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, and J. Siméon. XQuery 1.0: An XML Query Language. W3C Recommendation, 2007.

[5] P. Boncz, T. Grust, M. van Keulen, S. Manegold, J. Rittinger, and J. Teubner. MonetDB/XQuery: A Fast XQuery Processor Powered by a Relational Engine. In *Proc. SIGMOD*, 2006.

[6] Don Chamberlin, Peter Fankhauser, Daniela Florescu, Massimo Marchiori, and Jonathan Robie. XML Query Use Cases. W3C Working Group Note, 2007.

[7] D. Draper, P. Fankhauser, M.F. Fernández, A. Malhotra, K. Rose, M. Rys, J. Siméon, and P. Wadler. XQuery 1.0 and XPath 2.0 Formal Semantics. W3C Recommendation, 2007.

[8] D.K. Fisher and S. Maneth. Structural Selectivity Estimation for XML Documents. In *Proc. ICDE*, 2007.

[9] R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In *Proc. VLDB*, 1997.

[10] T. Grust. Purely Relational FLWORs. In *Proc. XIME-P*, 2005.

[11] T. Grust, S. Sakr, and J. Teubner. XQuery on SQL Hosts. In *Proc. VLDB*, 2004.

[12] T. Grust and J. Teubner. Relational Algebra: Mother Tongue—XQuery: Fluent. In *Proc. of the 1st Twente Data Management Workshop (TDM)*, 2004.

[13] International Business Machines Corp. (IBM). *DB2 Version 9 XML Guide*, 2006.

[14] H. Li, M.L. Lee, W. Hsu, and G. Cong. An Estimation System for XPath Expressions. In *Proc. ICDE*, 2006.

[15] A. Marian and J. Siméon. Projecting XML Documents. In *Proc. VLDB*, 2003.

[16] N. Polyzotis and M. Garofalakis. XSKETCH Synopses for XML Data Graphs. *ACM TODS*, 31(3), 2006.

[17] N. Reddy and J.R. Haritsa. Analyzing Plan Diagrams of Database Query Optimizers. In *Proc. VLDB*, 2005.

[18] C. Sartiani. A General Framework for Estimating XML Query Cardinality. In *Proc. DBPL*, 2003.

[19] A. R. Schmidt, F. Waas, M. L. Kersten, M. J. Carey, I. Manolescu, and R. Busse. XMark: A Benchmark for XML Data Management. In *Proc. VLDB*, 2002.

[20] P.G. Selinger, M.M. Astrahan, D. Chamberlin, R.A. Lorie, and T.G. Price. Access Path Selection in a Relational Database Management System. In *Proc. SIGMOD*, 1979.

APPENDIX

A. MORE REAL-WORLD FORECASTING

We have presented an estimation mechanism that provides dependable, subexpression-level cardinality forecasts for a significant and useful subset of the XQuery language. To support this claim, we conducted a series of experiments that compare the estimated cardinalities with actual results gathered over real data.

A prototype implementation of the inference rule set described in this work reads algebraic plan output generated by the Pathfinder XQuery compiler[6]. The prototype predicts the cardinality of each algebraic subexpression, then relates the projected table sizes to the actual cardinality as observed during a query evaluation over real data.

We used queries and data from two sources that provide realistic queries and data:

(1) From the W3C Working Group Use Cases Note [6], Use Case "XMP" exemplifies an application over bibliographic data. Use Case "R" assumes an XQuery front-end to a relational data of an online auction site. Use Case "SGML" is a traditional use case for markup languages.

(2) XMark [19] is a popular benchmark set in the XML database domain. It simulates an online auction site where data is held in an XML format.

The W3C Use Cases Note [6] describes the "XMP", "R", and "SGML" use cases over very small XML instances only. Although we added additional data to these instances to obtain reasonable data volumes, some of our measurements still suffered from the small data sizes. Even slight mis-estimations can lead to high error percentages over small numbers. For the XMark benchmark set, we generated XML instances using the `xmlgen` tool and made sure that their sizes were statistically sound.

The quality of value-based predicates obviously depends on the availability of histogram information. We generated histograms for all referenced elements and attributes whose content was numeric only. For all string-based comparisons, we assumed a System R-style 10 % selectivity.

We visualize the output of all our experiments in the same style as for Figure 11 in the experimental section of the main paper. To improve readability, we "truncated" all measurements at a 0.01 times under-estimate and a 10 times over-estimate (0.1 and 100 for the experiments on XMark data). Data points beyond these limits are collectively printed left and right of the respective cut-off lines. Typically, such data points are caused by subexpressions with an extremely low cardinality (a small but positive estimate becomes an ∞-fold over-estimate for an empty intermediate result).

A.1 W3C Use Case "XMP"

Use Case "XMP" may be seen as the most generic scenario among the W3C examples. It illustrates requirements from the database and document communities.

With the exception of Query *Q12*, our prototype had no problems processing all XMP example queries. Our current code does not support the XQuery built-in function `fn:deep-equal(·)`, which made it fail on the last use case query. For Queries *Q1*–*Q11*, the accuracy of our estimator is documented in Figure 12.

[6] `http://www.pathfinder-xquery.org/`

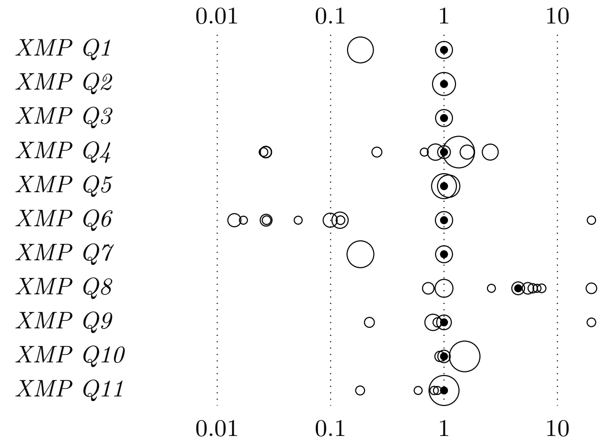


Figure 12: Estimation accuracy for the W3C Use Case "XMP" (experiences and exemplars) [6].

The most apparent outliers with significant under-estimations are the two Queries *Q4* and *Q6*. Those two queries depend on value-based joins and positional predicates, respectively. As we briefly sketched in Section 7, explicit synopses for the number of children of XML tree nodes could improve the accuracy of the latter class of expressions. Query *Q8* contains two substring matching predicates (`fn:contains(·)` and `fn:ends-with(·)`), which cause our prototype to over-estimate the query by a factor of ≈ 7.

A.2 W3C Use Case "R"

The W3C Use Case "R" mimics an application that queries XML data as it would typically come from a relational-to-XML translation scheme. The queries describe selections based on string (*Q1*–*Q3*, *Q5*, and *Q7*) and number (*Q3*, *Q6*, *Q9*, *Q15*) comparisons, value-based joins (*Q2*–*Q7*, *Q9*–*Q18*), and aggregation (*Q2*, *Q5*–*Q15*). In absence of any explicit foreign key information, we believe that particularly the multi-way joins (*Q5*, *Q10*–*Q13*, *Q17*) in Use Case "R" pose a challenge to any XQuery estimator.

Figure 13 documents the estimation error we observed for Use Case "R". Our query analyzer was able to handle all but two of the 18 use case queries. Our prototype does not support the XQuery built-in functions `fn:month-from-date(·)` and `fn:year-from-date(·)`, which prevented us from running Query *Q9*. Query *Q5* hit an uncaught exception in our code.

In Figure 13, the estimation error of a number of queries hit the x-coordinate limits we set ourselves for display purposes. The extreme deviations are caused by mis-estimated substring matching predicates (*Q1*, *Q7*), value-based joins (*Q3*, *Q10*–*Q12*), and date comparisons (for which we did not set up histograms; *Q8*). Yet, our prototype was able to "correct" the situation in most cases and showed higher accuracies as the estimation progressed bottom-up the plan DAG.

A.3 Use Case "SGML"

With Use Case "SGML", the W3C Working Group rephrased a traditional SGML use case using XQuery. It is characterized by a typical "document" DTD, where certain tags (*e.g.*, `title`) may appear at many different nesting lev-

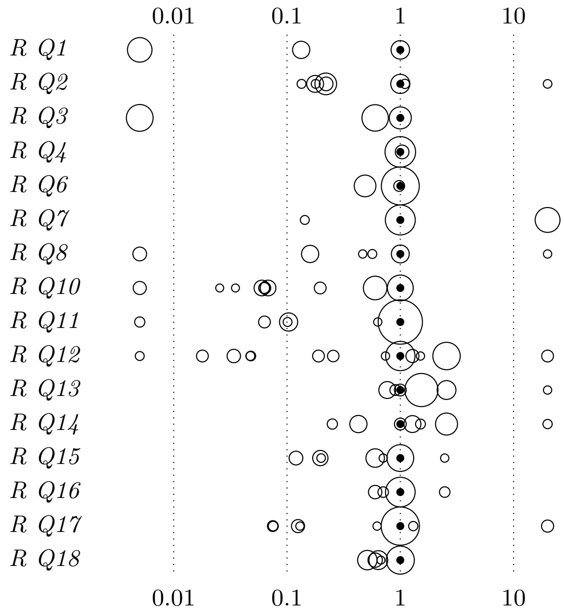


Figure 13: Estimation accuracy for the W3C Use Case "R" (access to relational data) [6].

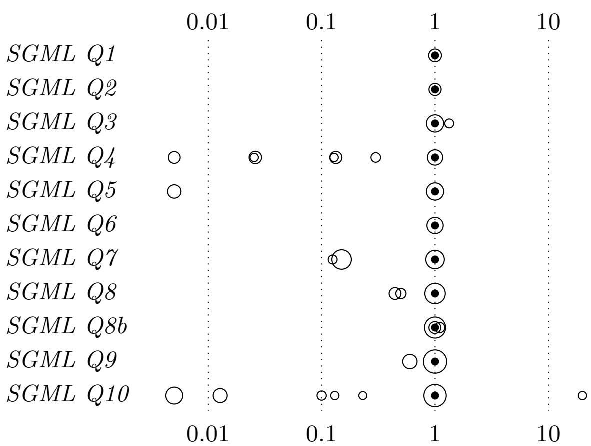


Figure 14: Estimation accuracy for the W3C Use Case "SGML" [6].

els (*e.g.*, within `titles`, `sections`, `topics`, etc.). Our prototype had no problems handling all 10 queries in this use case.

Some of the queries in Use Case "SGML" hit our estimator at its weakest spot. Without support for positional predicates, our prototype had problems predicting the result sizes in Queries *Q4*, *Q7*, and *Q10* accurately (shown in Figure 14). Explicit child distribution statistics (see Section 7) would help these queries. Otherwise, our prototype exploited ts data guide to come up with highly accurate queries in most cases.

A.4 XMark

Soon after its presentation in 2002, the XMark benchmark set [19], has quickly become the predominant way to judge the performance of XQuery-based database systems. Rather than looking at performance, our focus here is on an accurate estimation for each subexpression in the twenty benchmark queries. The scenario assumed in the benchmark is an online auction site.

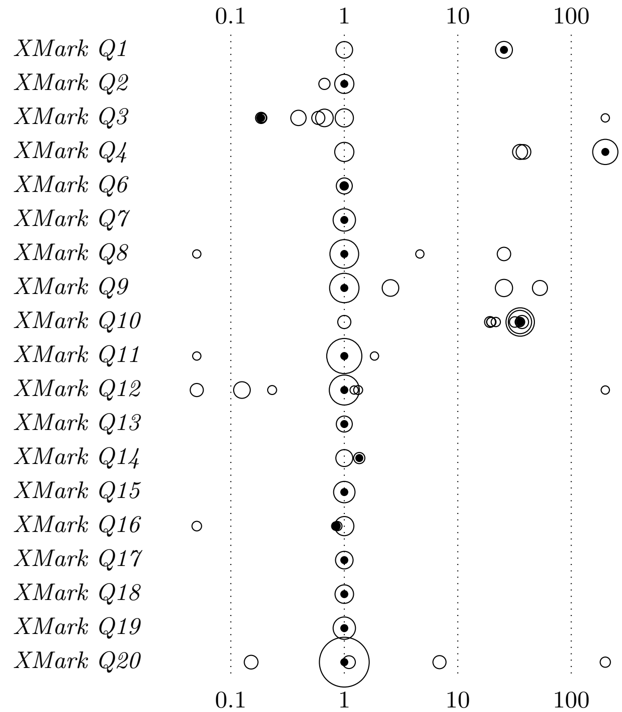


Figure 15: Estimation accuracy for the 20 XMark benchmark queries (simulating an online auction site) [19].

Our prototype successfully computed cardinality estimates for all twenty benchmark queries. The accuracy we observed is documented in Figure 15.

XMark Query *Q1* is a simple example of a lookup by a key value:

```
let $auction := doc("auction.xml") return
for $b in $auction/···/person[@id="person0"]
return $b/name/text() .
```

Our implementation does not (yet), however, interpret any DTD information to distinguish this query from an arbitrary string comparison. Absence of data statistics for strings in our estimator and the application of the System R-style 10 % rule leads to the high error margin in Figure 15. Query *Q4* contains similar lookups for two persons, one of which does not even exist in the generated XML instance. This is an instance of an "∞-fold over-estimate," as mentioned earlier.

Query *Q3* is dominated by a comparison that depends on document order—an aspect currently unsupported in our setup. Queries *Q8*–*Q12* represent value-based joins. They are known to be challenging to XML database optimizers. In Figure 15, we see that they also challenge our cardinality estimator. Again, however, our prototype recovered gracefully from the mis-estimation it made on intermediate results and produced an exact estimate for the join queries *Q8*, *Q9*, *Q11*, and *Q12*.

B. PROBABILITIES AND DOMAIN SIZES

Algebraic operators that discard rows from their input (most prominently the σ_{a} and $\bowtie_{\mathsf{a=b}}$ operators) affect the domain sizes associated with columns of the input table. Quantifying this effect requires a brief look into probability theory.

···	c	···
⋮	c_1	⋮
	⋮	
	c_1	
⋮	⋮	⋮
⋮	c_k	⋮
	⋮	
	c_k	

Suppose we apply a filter σ_{a} of selectivity s to table R (see right). Assuming an independence between σ_{a} and a column c, we can estimate the size of c's domain in the output, $\|\gamma_{\text{out}}\|$, based on s, $|R|$, and $k = \|\gamma_{\text{in}}\|$ (the domain size associated with column c in the input relation R).

On average, each value $a_i \in \gamma_{\text{in}}$ occurs $|R|/\|\gamma_{\text{in}}\|$ times in relation R. The chance that *all* of these occurrences are filtered out during $\sigma_{\mathsf{a}}(R)$ (which means that $a_i \notin \gamma_{\text{out}}$) is

$$P_{\notin} = (1-s)^{|R|/\|\gamma_{\text{in}}\|} \tag{2}$$

(since the chance of losing a single occurrence is $(1-s)$).

The chance that at least one instance of a_i is retained after $\sigma_{\mathsf{a}}(R)$ (*i.e.*, $a_i \in \gamma_{\text{out}}$) is $(1 - P_{\notin})$, hence,

$$\|\gamma_{\text{out}}\| = \|\gamma_{\text{in}}\| \cdot \left(1 - (1-s)^{|R|/\|\gamma_{\text{in}}\|}\right) \quad . \tag{3}$$