Estimating Answer Sizes for XML Queries

Yuqing Wu, Jignesh M. Patel, and H.V. Jagadish

Univ. of Michigan, Ann Arbor, MI, USA*
{yuwu,jignesh,jag}@eecs.umich.edu

Abstract. Estimating the sizes of query results, and intermediate results, is crucial to many aspects of query processing. In particular, it is necessary for effective query optimization. Even at the user level, predictions of the total result size can be valuable in "next-step" decisions, such as query refinement. This paper proposes a technique to obtain query result size estimates effectively in an XML database.

Queries in XML frequently specify structural patterns, requiring specific relationships between selected elements. Whereas traditional techniques can estimate the number of nodes (XML elements) that will satisfy a node-specific predicate in the query pattern, such estimates cannot easily be combined to provide estimates for the entire query pattern, since element occurrences are expected to have high correlation.

We propose a solution based on a novel histogram encoding of element occurrence position. With such *position histograms*, we are able to obtain estimates of sizes for complex pattern queries, as well as for simpler intermediate patterns that may be evaluated in alternative query plans, by means of a *position histogram join* (pH-join) algorithm that we introduce. We extend our technique to exploit schema information regarding allowable structure (the *no-overlap* property) through the use of a *coverage histogram.*

We present an extensive experimental evaluation using several XML data sets, both real and synthetic, with a variety of queries. Our results demonstrate that accurate and robust estimates can be achieved, with limited space, and at a miniscule computational cost. These techniques have been implemented in the context of the TIMBER native XML database [22] at the University of Michigan.

1 Introduction

XML data [2] is becoming ubiquitous, and an XML document (or database) is naturally modeled as a (collection of) node-labeled tree(s). In such a tree, each node represents an XML *element*, and each tree edge represents an *element-subelement inclusion relationship.*

A natural way to query such hierarchically organized data is by using small node-labeled trees, referred to as *twigs*, that match portions of the hierarchical

* H.V. Jagadish and Yuqing Wu were supported in part by NSF under grant IIS-9986030 and DMI-0075447. Jignesh M. Patel was supported in part by a research gift donation from NCR Corporation.

C.S. Jensen et al. (Eds.): EDBT 2002, LNCS 2287, pp. 590–608, 2002.

data. Such queries form an integral component of query languages proposed for XML (for example, [4]), and for LDAP directories [11]. For example, the XQuery expression

```
FOR $f IN document("personnel.xml")//department/faculty
WHERE count($f/TA) > 0 AND count($f/RA) > 0
RETURN $f
```

matches all faculty members that has at least one TA and one RA, in the example data set shown in Fig. 1. This query can be represented as a node-labeled tree, with the element tags department and faculty as labels of non-leaf nodes in the tree, and the element tags TA and RA as labels of leaf nodes in the tree, as shown in Fig. 2.

A fundamental problem in this context is to accurately and quickly estimate the number of matches of a twig query pattern against the node-labeled data tree.

An obvious use is in the cost-based optimization of such queries: knowing selectivities of various sub-queries can help in identifying cheap query evaluation plans. For example, the query of Fig. 2 can be evaluated by identifying all faculties with RAs, and joining this set with the set of departments, then joining the result of this with the set of all the TAs. An alternative query plan is to join the faculties and RAs first, and then join the result set with TAs, then, departments. Depending on the cardinalities of the intermediate result set, one plan may be substantially better than another. Accurate estimates for the intermediate join result are essential if a query optimizer is to pick the optimal plan. Furthermore, if there are multiple join algorithms, the optimizer will require accurate estimates to enable it to choose the more efficient algorithm. Similar choices must be made whether the underlying implementation is a relational or a native XML database.

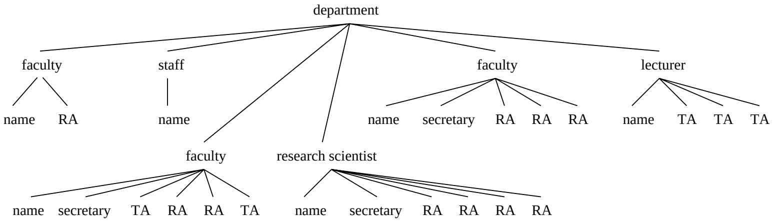

Fig. 1. Example XML document

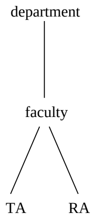

Fig. 2. Pattern Tree

Result size estimation has additional uses in an Internet context. For instance, there may be value in providing users with quick feedback about expected result sizes before evaluating the full query result. Even when the query involved is an on-line query where only partial results are requested, it is helpful to provide an estimate of the total number of results to the user along with the first subset of results, to help the user choose whether to request more results of the same query or to refine the query. Similarly, result size estimation can be very useful when space allocation or parallelism are involved.

Histograms are by far the most popular summary data structures used for estimating query result sizes in (relational) databases. When used in the XML context, they could indeed be used to estimate accurately the number of nodes satisfying a specified node predicate. One could build a histogram for the predicate associated with each node in a query pattern, and obtain individual estimates for the number of occurrences of each. However, structural relationship information is not captured in traditional histograms, and it is not obvious how to combine estimates for individual nodes into estimates for the whole query tree pattern.

The central contribution of this paper is the introduction of *position histograms* to capture this structural information. A position histogram is built over "base" predicates, such as "elementtag = faculty". The position histograms on two base predicates, P_1 and P_2, can be used to accurately estimate the selectivity of queries with the pattern $P_1//P_2$, which matches all "P_2" nodes that are descendants of all "P_1" nodes in the data tree. Some special features of predicates, such as no-overlap property, which dramatically affects the selectivity of the pattern matching, are also considered. Even though these histograms are two-dimensional, they behave like one-dimensional histograms for many purposes, including in their storage requirements.

We formally define our problem in Section 2, and summarize our overall solution approach in Section 3. We also establish various properties of this new summary data structure, and show how to use this to obtain query result sizes estimates efficiently in Section 3. Schemata often impose constraints on allowed structural relationships. In Section 4, we show how, at least in some key cases, such schema information can be exploited to obtain better estimates. We experimentally demonstrate the value of our proposal in Section 5, considering not just the quality of the estimate, but also issues such as computation time and storage requirement. And related work is discussed in Section 6. Conclusions and directions for future work are outlined in Section 7.

2 Problem Definition

We are given a large rooted node-labeled tree $T = (V_T, E_T)$, representing the database.

We are given a set of boolean predicates, $\mathcal{P}$: $\{v : v \in T\} \rightarrow \{0, 1\}$. For each predicate $\alpha \in \mathcal{P}$, for each node $v \in T$, we have either $\alpha(v)$ is true or $\alpha(v)$ is false. (See Sec 3.4 for a discussion of how to obtain this set $\mathcal{P}$ for a real database).

A query is a smaller, rooted, node-labeled tree $Q = (V_Q, E_Q)$. The goal is to determine the number of "matches" of Q in T. The labels at the nodes of Q are boolean compositions of predicates from $\mathcal{P}$.

A *match* of a pattern query Q in a T is a total mapping $h : \{u : u \in Q\} \rightarrow \{x : x \in T\}$ such that:

– For each node $u \in Q$, the predicate node label of u is satisfied by $h(u)$ in T.
– For each edge (u, v) in Q, $h(v)$ is a descendant of $h(u)$ in T.

Fig. 1 shows a very simple XML document. The personnel of a department can be faculty, staff, lecturer or research scientist. Each of them has a name as identification. They may or may not have a secretary. Each faculty may have both TAs and RAs. A lecturer can have more than one TAs, but no RA. A research scientist can have numerous RAs, but no TA. Consider a simple twig pattern with only two nodes, faculty and TA, with parent-child relationship among them. There are three faculty nodes and five TA nodes in the XML document. The schema says that a faculty can have any number of TAs. Without any further schema information, the best we can do in estimating the result size is to compute the product of the cardinality of these two nodes, which yields 15. Consider the fact that faculty nodes are not nested, one TA can only be the child of one faculty node, we can tell that the upper-bound of the result number is the cardinality of TA nodes, which is 5. But as we can see from the figure, the real result size is 2. The question we address in this paper is how to capture the structure information of the XML document to get a better estimation.

Our problem can be stated succinctly as follows:

> Define a summary data structure T' corresponding to a node-labeled data tree T, and a set of primitive predicates of interest $\mathcal{P}$, such that the size of T' is a small percentage of the size of T; and for any query Q, defined as a structural pattern of nodes satisfying combinations of predicates from $\mathcal{P}$, correctly estimate the total number of matches of Q in T, using only Q and the summary data structure T'.

3 Our Proposal

3.1 The Basic Idea

We associate a numeric start and end label with each node in the database, defining a corresponding interval between these labels. We require that a descendant node has an interval that is strictly included in its ancestors' intervals.

This numbering scheme is inspired by, and quite similar to, the node numbering based on document position frequently used in information retrieval and adopted for XML database use by Univ. of Wisconsin researchers in the course of the Niagara [18] project.

We obtain these labels as follows. First, we merge all documents in the database into a single mega-tree with a dummy element as the root, and each document as a child subtree. We number nodes in this tree to obtain the desired labels – the start label by a pre-order numbering and the end label of a node is assigned to be at least as large as its own start label and larger than the end label of any of its descendant.

Given a limited set $\mathcal{P}$ of predicates of interest, one should expect that there will be index structures that identify lists of nodes satisfying each predicate in $\mathcal{P}$. For many, even most, predicates, these lists can be very long. While queries may be answered through manipulating such lists, the effort involved is far too great for an answer size estimation task. The standard data structure for maintaining summary data in a database is a histogram. We compress each such list into a two-dimensional histogram summary data structure, as we describe next.

We take the pairs of start and end pair of values associated with the nodes that satisfy a predicate α, and construct a two-dimensional histogram $Hist_\alpha$ with them. Each grid cell in the histogram represents a range of start position values and a range of end position values. The histogram $Hist_\alpha$ maintains a count of the number of nodes satisfying α that have start and end positions within the specified ranges. We call such a data structure a *position histogram.*

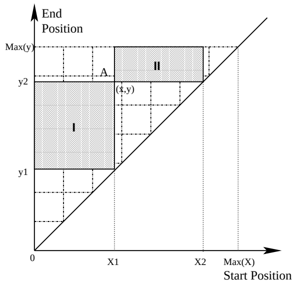

Fig. 3. Forbidden Regions in a Position Histogram due to one node

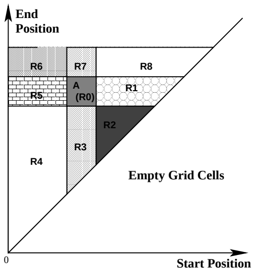

Fig. 4. Layout of Position Histogram

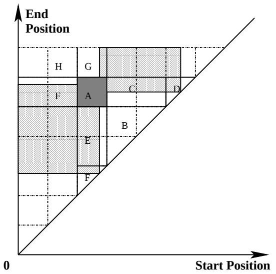

Fig. 5. Estimating Join Counts with Position Histogram

Position histograms, even though defined over a two-dimensional space, have considerable structure, as shown in Fig. 3.

Since the start position and end position of a node always satisfies the formula that start <= end, none of the nodes can fall into the area below the diagonal of the matrix. So, only the grid cells to the upper left of the diagonal can have count of more than zero.

Given a point A with coordinates (x,y), the regions marked I and II are guaranteed to be empty, since the start and end ranges of any two nodes can either have no overlap, or the range of one node fully contained within the range of the other node. This leads to the following Lemma:

Lemma 1 *In a position histogram for any predicate, a non-zero count in grid cell* (i, j) *implies a zero count in each grid cell* (k, l) *with (a)* $i < k < j$ *and* $j < l$*, or (b)* $i < l < j$ *and* $k < i$*.*

3.2 Primitive Estimation Algorithm

Each document node is mapped to a point in two dimensional space (in each position histogram corresponding to a predicate satisfied at the node) . Node u is an ancestor of node v iff the start position of u is less than the start position of

v and the end position of u is no less than the end position of v. In other words, u is to the left of and above every node v that it is an ancestor of, and vice versa.

Consider the grid cell labeled A in Fig. 4. There are nine regions in the plane to consider, marked A (R_0), R_1 through R_8 in the Figure. All points v in region R_2 are descendants of each point u in the grid cell A. All points v in region R_6 are ancestors of each point u in grid cell A. No point in region R_4 and R_8 is a descendant or ancestor of any point in the grid cell A. Points in region R_1 and R_3 *may* be descendants of points in grid cell A. Similarly, points in region R_5 and R_7 *may* be ancestors of points in grid cell A. To estimate how many, we exclude the forbidden region, and then assume a uniform distribution over the remainder of each grid cell. For this purpose, we overlap Fig. 3 with Fig. 4 to get Fig. 5.

Given predicates P_1 and P_2, both in $\mathcal{P}$, we show how to estimate the number of pairs of nodes u, v in the database such that u satisfies P_1, v satisfies P_2 and u is an ancestor of v, using two position histograms, one for predicate P_1 and one for predicate P_2.

When computing the estimate of a join, we can compute the estimate based on either the ancestor or the descendant. When computing an *ancestor-based* estimate, for each grid cell of the ancestor we estimate the number of descendants that join with the ancestor grid cell. Similarly, for the *descendant-based* estimate, for each grid cell of the descendant we estimate the number of ancestors that join with the grid cell.

The formulae for these two types of estimation are different, and are derived in the next two subsections. But first, we need the following definition:

Definition 1 *A grid cell in a position histogram is said to be* on-diagonal *if the intersection of the start-position interval (X-axis) and end-position interval (Y axis) is non-empty. Otherwise, the grid cell is said to be* off-diagonal.

Ancestor-Based Join Estimation. If A is off-diagonal, as shown in Fig. 5, all points in the grid cells in region B are descendants of all points in grid cell A. Using the position histogram for predicate P_2, we can simply add up the counts of all grid cells in this region. Now consider region E. Each point in grid cell A introduces two forbidden regions. No points in region E can fall in the forbidden regions of the right-most point in A (as shown in Fig. 5), so all points in region E must be descendants of all points in grid cell A. Similarly, for a given point in grid cell A, part of region F is forbidden; the points that fall in the right triangle of F are descendants of A, and the points in the left triangle are not. Integrating over the points in region F, we estimate that half the points in F, on average, are descendants of any specific point in grid cell A. Similar discussions apply to regions C and D. For the points in the same grid cell (grid cell A) in the histogram for predicate P_2, for each point in grid cell A of the histogram for the predicate P_1, only the points in the bottom-right region can be descendants. Assuming a uniform distribution and performing the necessary integrals in each dimension, we derive on average a quarter chance. Putting all these estimates together, the ancestor-based estimation for each off-diagonal grid cell can be expressed as the first formula in Fig. 6.

When grid cell A is on-diagonal, regions B, C, D, E, F don't exist. Since a diagonal grid cell is a triangle rather than a rectangle, the chance that a descendant point can join with an ancestor point is 1/12.

```
Primitive Estimation: ancestor-based
        For off-diagonal grid cell A:
            $Est_{P_{12}}[A] = Hist_{P_1}[A] \times \{\frac{1}{4} \times Hist_{P_2}[A] + Hist_{P_2}[B] + Hist_{P_2}[C] + Hist_{P_2}[E]$
                $+\frac{1}{2} \times (Hist_{P_2}[D] + Hist_{P_2}[F])\}$
        For on-diagonal grid cell A:
            $Est_{P_{12}}[A] = \frac{1}{12} \times Hist_{P_1}[A] \times Hist_{P_2}[A]$
Primitive Estimation: descendant-based
        For off-diagonal grid cell A:
            $Est_{P_{12}}[A] = Hist_{P_2}[A] \times \{Hist_{P_1}[F] + Hist_{P_1}[G] + Hist_{P_1}[H] + \frac{1}{4} \times Hist_{P_1}[A]\}$
        For on-diagonal grid cell A:
            $Est_{P_{12}}[A] = Hist_{P_2}[A] \times \{Hist_{P_1}[F] + Hist_{P_1}[G] + Hist_{P_1}[H] + \frac{1}{12} \times Hist_{P_1}[A]\}$

Notation:
        $Hist_P$: position histogram for predicate $P$
        $Est_{P_{12}}$: estimation histogram of a twig pattern, where the ancestor
                satisfies $P_1$ and the descendant satisfies $P_2$.
        $H[A]$ : summation of the grid cells in region $A$ in histogram $H$.
```

Fig. 6. Formulae for Primitive Join Estimation

Descendant-Based Join Estimation. Referring to Fig. 5, no matter whether A is on-diagonal or off-diagonal, all ancestors of a point in the grid cell A will be in the regions A, F, G, or H. Following argument similar to those in the ancestor-based estimation above, all points in region F, G and H are guaranteed to be ancestors of all points in grid cell A. For the points in the same grid cell (grid cell A), the chance is 1/4 for an off-diagonal grid cell, while it is 1/12 for an on-diagonal grid cell.

faculty		
	0	1
	2	

TA		
	0	3
	2	

Fig. 7. Position Histograms

Coverage for (0.0)		
	0.5	

Coverage for (1,1)		
		0.3

Fig. 8. Coverage Histogram for faculty

Let's have a look at the example XML document in Fig. 1 again, with the same query pattern we discussed in section 2. The 2×2 histograms of predicates "element tag = faculty" and "element tag = TA" are shown in Fig. 7. Using the primitive estimation algorithm introduced above, we estimate the result size to be 0.6, much closer to the real result size. Note that the position histograms we used here are 2×2. By refining the histogram to use more buckets, we can get a more accurate estimate.

3.3 Analysis

The primary concern with any estimation technique, of course, is how good the estimates are. We will perform an extensive evaluation in Section 5. However, there are two other metrics of concern: the storage required and the time to compute the estimate. We address these issues next.

Storage Requirement. There can only be $O(g)$ non-zero grid cells in a $g \times g$ grid, unlike the $O(g^2)$ one might expect in general. Therefore the storage requirements are quite modest. We establish this result as a theorem here, and verify experimentally in Section 5.

Theorem 1 *In a $g \times g$ grid, the number of position histogram grid cells with non-zero counts is $O(g)$*

Time Required. Based on the formulae for both ancestor-cased estimation and descendant-based estimation, the procedure to compute the expected size of result for a simple 2-node pattern is to loop through all grid cells for counts of nodes satisfying the outer predicate, and for each grid cell loop through the histogram for the inner predicate, adding up the regions as described in the preceding section, and multiplying by the count of the outer grid cell. The grand summation of all these is the desired result. We have a choice of which of the two nodes in the pattern is the inner loop, and the other is the outer.

The summation work in the inner loop is repeated several times in the simple nested loop algorithm outlined above. A small amount of storage for intermediate results can result in the much more efficient algorithm shown in Fig. 9.

Algorithm *pH-Join* is a three-pass algorithm. In the first pass, column partial summations (as on region E and columns in region B in Fig. 5) are obtained. In the second pass, row partial summations (as in region C) , as well as region partial summations (as in region B using column partial summations) are obtained. In the third pass, these partial summations are used, along with the matrix entries themselves, to obtain the necessary multiplicative coefficients derived from the inner matrix operand and these can be multiplied by the corresponding elements of the outer operand matrix and the summation taken.

Algorithm *pH-Join*, as stated, computes coefficients assuming that the inner operand is the descendant node in the pattern. Obvious minor changes are required if the inner operand is the ancestor node.

Observe also that all of Algorithm *pH-Join*, except for the final multiplication, deals with the histogram of only one predicate in the join operation. In consequence, it is possible to run the algorithm on each position histogram matrix in advance, pre-computing the multiplicative coefficients at each grid cell. The additional storage required is approximately equal to that of the original position histogram. So such pre-computation may provide a useful space-time tradeoff in some situations. In any event, the time required for the computation is simply $O(g)$ for a $g \times g$ grid.

```
Algorithm pH_Join (histA, histB)
// Inputs: Two histograms histA and histB,
// Output: Estimation of answer of A join with B .
for (i=0; i<grid_size; i++)
      for (j=i; j<grid_size; j++) {
           pSum[i][j].self = HistB[i][j];
           if (j == i) pSum[i][j].down = 0; // column summation
           else if (j == i+1) pSum[i][j].down = pSum[i][j-1].self;
           else pSum[i][j].down = pSum[i][j-1].self + pSum[i][j-1].down;
      }
for (j=grid_size-1; j>=0; j--)
      for (i=j; i>=0; i--) {
           if (i == j) {
                pSum[i][j].right = 0;
                pSum[i][j].descendant = 0;
           }
           else if (i == j-1) {
                pSum[i][j].right = pSum[i+1][j].self; // row summation
                pSum[i][j].descendant = pSum[i+1][j].down; // region summation
           }
           else{
                pSum[i][j].right = pSum[i+1][j].self + pSum[i+1][j].right;
                pSum[i][j].descendant = pSum[i+1][j].down + pSum[i+1][j].descendant;
           }
      }
for (i=0; i<grid_size; i++)
      for (j=i; j<grid_size; j++){
           if (i==j) rHist[i][j] = HistA[i][j] * pSum[i][j].self / 12;
           else rHist[i][j] = HistA[i][j] * (pSum[i][j].descendant
                     + pSum[i][j].self / 4 + pSum[i][j].down - pSum[i][i].self / 2
                     + pSum[i][j].right - pSum[j][j].self / 2 );
           total+=rHist[i][j]
      }
output(total);
```

Fig. 9. Algorithm *pH-Join* for Computing the Join Estimate

3.4 Predicate Set Selection

Often, the predicats applied at node may not belong to the set of basic predicates $\mathcal{P}$. In such a case, there may be no precomputed position histogram of start and end positions for nodes satisfying the specified predicate. However, if the specified predicate can be expressed as a boolean combination of basic predicates, we can estimate a position histogram assuming independence (between basic predicate components of the compound predicate) *within a grid cell.* To be able to manipulate counts, we need to convert these into the appropriate probabilities. For this purpose, we can compute a position histogram for the predicate "TRUE", including all elements in the database, and simply using their start and end positions to obtain the needed grid cell counts. For each grid cell, this count is the appropriate normalization constant to carry out the conversion.

Compound predicates can arise not only because the query expression has a compound predicate in it, but also because of the choices made in defining the set $\mathcal{P}$ of basic predicates. Predicates in XML queries fall into two general categories:

Element-Tag Predicates. These predicates are defined on the element tags. An example of such predicate is $elementtag = faculty$. Element tag predicates are likely to be common in XML queries, and are a good candidates for building position histograms on. Usually, there are not many element tags defined in an XML document, so it is easy to justify the storage requirement of one histogram for each such predicate and build a histogram on each one of these distinct element tags.

Element-Content Predicates. These predicates specify either an exact or partial match on the contents of element. For example, text nodes with a parent node year are numerical values (integer) within a small range. It is not unreasonable to build histogram for each of these values. In some cases, some part of the content has some general meaning, and tends to be queried extensively. It would be helpful to set a predicate that evaluates to true if the prefix (suffix) of the content of a text element matches to a certain value. We will see some examples of both in Section 5.

It is likely that such predicates far outnumber the element-tag predicates, and position histograms will only be built on element-content predicates that occur frequently. In any event, minimizing error in the estimation of these values is likely to be more important than errors in estimates of less frequent items. The value of this general concept has been amply demonstrated in the context of end-biased histograms [8].

4 Factoring in Schema Information

Up to this point, we assumed that the data was uniformly distributed within any grid cell, and this is indeed a reasonable thing to do if no other information is available. However, we may frequently have information from the schema that can substantially modify our estimate.

For instance, if we know that no node that satisfies predicate P_2 can be a descendant of a node that satisfies P_1, then the estimate for the number of results for a query that asks for P_1 satisfied at a node that is an ancestor of P_2 is simply zero – there is no need to compute histograms. Similarly, if we know that each element with tag author must have a parent element with tag book, then the number of pairs with book as ancestor and author as descendant is exactly equal to the number of author elements.

We recommend that such schema information be brought to bear when possible. Our work here concerns itself with the vast majority of the cases where schema information alone is insufficient.

4.1 No Overlap

We frequently know, for a given predicate, that two nodes satisfying the predicate cannot have any ancestor-descendant relationship. For instance, in Fig. 1, a faculty node cannot contain another faculty node. It follows that there can be no node that is a descendant of two distinct faculty nodes. (For instance, a particular TA node can appear under at most one faculty node). In such situations, the uniformity assumption within a histogram grid cell can lead to erroneous estimates. We present, in this section, an alternative estimation technique appropriate when the ancestor node predicate in a primitive two-node pattern has the no-overlap property. It turns out that there is no impact on the estimation of the descendant node in the pattern having a no-overlap property since multiple descendants could still pair with the same (set of nested) ancestor node(s).

Definition 2 *A predicate P is said to have the* no-overlap *property if for all elements x, y such that $P(x)$ and $P(y)$ are TRUE, we have: $endpos(x) < startpos(y)$ or $endpos(y) < startpos(x)$.*

4.2 Summary Data Structure for Predicates with No-Overlap

For a primitive pattern with a no-overlap ancestor node a, the number of occurrences is upper-bounded by the count of the descendant node d in the pattern. (Since each descendant node may join with at most one ancestor node). The question is how to estimate the fraction of the descendant nodes that participate in the join pattern. Within any one grid cell, the best one can do is to determine what fraction of the total nodes in the cell are descendants of a, and assume that the same fraction applies to d nodes. We call this fraction, the *coverage* of a in that particular cell. Thus, our technique for dealing with the no-overlap situation is to keep additional information, in the form of *coverage*. Formally, we define the *coverage histogram* for predicate P: $Cvg_P[i][j][m][n]$ to be the fraction of nodes in grid cell (i, j) that are descendants of some node that satisfies P and fall in grid cell (m, n).

At first glance, it may appear that the storage requirements here are exorbitant – rather than store counts for each grid cell, we are keeping information for *cell pairs*. However, for a given grid cell r in the position histogram, and consider its coverage in grid cell s, the coverage fraction is guaranteed to be one whenever cell s is both to the right of and below r. And the coverage fraction is obviously zero for cells that cannot include descendants of elements in r. As such, it is only the cells s along the "border" for which one is required explicitly to keep coverage information. In fact, one can establish the following theorem:

Theorem 2 *In a $g \times g$ grid, the number of coverage histogram cell pairs with partial (non-zero and non-one) coverage is $O(g)$. In other words, the coverage histogram requires only $O(g)$ storage.*

The proof relies on the fact that, due to the overlap property, if the grid cell of focus is populated (by a node satisfying the "ancestor" predicate), then there can be no node (satisfying this predicate) in any of the cells in the (black) region with coverage = 1.

This, together with the algorithms we established for position histograms in Section 3, estimation formulae for No-Overlap predicates can be derived, as shown in Fig. 10.

Let's go back to the example XML document again, and estimate the result size for the same query pattern. This time, the no-overlap estimation algorithm

Ancestor-based pattern count estimate:

$Est_{AB}[i][j] = Jn\_Fct_{A\_P_1}[i][j] \times \sum_{m=i..j,n=m..j}(Cvg_{A\_P_1}[m][n][i][j] \times Hist_{B\_P_2}[[m][n]$
$\times Jn\_Fct_{B\_P_2}[m][n])$

Descendant-based pattern count estimate:

$Est_{AB}[i][j] = Hist_{B\_P_2}[i][j] \times Jn\_Fct_{B\_P_2}[i][j] \times \sum_{m=0..i,n=j..max\_y}(Cvg_{A\_P_1}[i][j][m][n]$
$\times Jn\_Fct_{A\_P_1}[m][n])$

Join factor estimate:

$Jn\_Fct_{AB\_P_x}[i][j] = \frac{Est_{AB}[i][j]}{Hist_{AB\_P_x}[i][j]}$ if $Hist_{AB\_P_x}[i][j] > 0$, $= 0$ otherwise

Participation Estimation:

case1: the node (P_x) that the estimation is based on can overlap
$Hist_{AB\_P_x} = Est_{AB}$

case2: P_1 is no-overlap, estimation is ancestor-based
$N[i][j] = Hist_{A\_P_1}[i][j]$
$M[i][j] = \sum_{m=i..j,n=m..j} Hist_{B\_P_2}[[m][n]$
$Hist_{AB\_P_1}[i][j] = N[i][j] \times (1 - (\frac{N[i][j]-1}{N[i][j]})^{M[i][j]})$

case3: item P_2 is no-overlap, estimation is descendant-based
$Hist_{AB\_P_2}[i][j] = Hist_{B\_P_2}[i][j] \times \sum_{m=0..i,n=j..max\_y}(notzero(Hist_{A\_P_1[m][n]})$
$\times Cvg_{A\_P_1}[i][j][m][n])$
Here, function $notzero(x) = 1$ if $x \neq 0$, $= 0$ otherwise

Coverage Estimation:

case1: P_1 is no-overlap, the join is ancenstor-based
$Cvg_{AB\_P_1}[i][j][m][n] = Cvg_{A\_P_1}[i][[j][m][n] \times \frac{Hist_{AB\_P_1}[m][n]}{Hist_{A\_P_1}[m][n]}$

case2: P_2 is no-overlap, and the join is descendant-based
$Cvg_{AB\_P_2}[i][j][m][n] = Cvg_{B\_P_2}[i][[j][m][n] \times \frac{Hist_{AB\_P_2}[i][j]}{Hist_{B\_P_2}[i][j]} \times notzero(Hist_{A\_P_1}[m][n])$

* Notation:

Est_{AB}: estimation histogram of the pattern obtained by joining subpattern A,B
$Hist_{AB\_P_x}$: number of nodes that satisfy P_x and participate in the join of A and B.
$Jn\_Fct_{AB\_P_1}$: number of nodes satisfying P_2 that join with each distinct node, that satisfy P_1, in the join of subpattern A and B.
$Cvg_{AB\_P_1}$: the coverage histogram of the distinct nodes that satisfy P_1 and participate in the join of subpattern A and B.

Fig. 10. Estimation Formulae for No-Overlap Predicates

is used. The Coverage Histogram of predicate "element tag = faculty" is shown in Fig. 8. The estimate we get is 1.9, almost the same as the real result size.

5 Experimental Evaluation

We tested our estimation techniques extensively on a wide variety of both real and synthetic data sets. First, we report on the accuracy of the estimates obtained. Later, we present results on the storage size and the impact of storage size on the accuracy of the estimate.

5.1 The DBLP Data Set

We ran experiments on several well-known XML data sets, including the XMark Benchmark [15] and the Shakespeare play data set [20]. Results obtained in all cases were substantially similar. In the interests of space, we present results only for the DBLP data set [19] that is probably most familiar to readers of this paper. The DBLP data set is 9M bytes in size and has approximately 0.5M nodes.

For the DBLP data set, we picked a mix of element-tag and element-content predicates and built histograms on exact matching of all the element tags, the content value of years, and the prefix matching of the content of 'cite' (e.g conf, journal, etc.). A few of these predicates, along with the count of the nodes that match each predicate, and the overlap property of the predicate is summarized in Table 1. Note that the predicates 1990's and 1980's are compound predicates, obtained by adding up 10 corresponding primitive histograms for element-content predicate (e.g. 1990, 1991 ...). In all, there are 63 predicates; and the total size of all the corresponding histograms added up to about 6K bytes in all – roughly 0.7% of the data set size. (We used 10×10 histograms in all experiments, except where explicitly stated otherwise.)

Table 1. Characteristics of Some Predicates on the DBLP Data Set

Predicate Name	Predicate	Node Count	Overlap Property
article	element tag = "article"	7,366	no overlap
author	element tag = "author"	41,501	no overlap
book	element tag = "book"	408	no overlap
cdrom	element tag = "cdrom"	1,722	no overlap
cite	element tag = "cite"	33,097	no overlap
title	element tag = "title"	19,921	no overlap
url	element tag = "url"	19,542	no overlap
year	element tag = "year"	19,914	no overlap
conf	text start-with "conf"	13,609	N/A
journal	text start-with "journal"	7,834	N/A
1980's	compound	13,066	N/A
1990's	compound	3,963	N/A

Estimating Simple Query Answer Sizes. We tested the effectiveness of position histograms on a number of queries using a combinations of predicates from Table 1. In the interest of space, we only present results for a few representative queries in Table 2. The first row of this table considers a query pattern where an element with author tag appears below an element with article tag. Other rows consider similar other simple queries.

Without the position histograms, and without any schema information, a (very) naive estimate for the answer size is the product of the cardinalities of the node counts for the two predicates (*i.e.*, article and author). The naive estimate is far from the real result, since it does not consider the structural relationship between nodes. With the schema information and no position histogram, if the ancestor node has no-overlap property, the best (upper-bound) estimate of the

Table 2. Result Size Estimation for Simple Queries on DBLP Data Set

		Naive	**Desc**	**Overlap**		**No-Overlap**		**Real**
Ance	**Desc**	**Estimate**	**Num**	**Estimate**	**Est Time**	**Estimate**	**Est Time**	**Result**
article	author	305,696,366	41,501	2,415,480	0.000344	14,627	0.000263	14,644
article	cdrom	12,684,252	1,722	4,379	0.000290	112	0.000261	130
article	cite	243,792,502	33,097	671,722	0.000229	3,958	0.000261	5,114
book	cdrom	702,576	1,722	179	0.000142	4	0.000259	3

result size is the number of descendants involved in the join. When position and coverage histograms are available, overlap or no-overlap estimation algorithms can be used. When no schema information is available, using position histograms and the primitive *pH-Join* estimation algorithm brings the estimate closer to the real answer size. In some cases, the primitive estimation is better than the upper-bound estimation using only the schema information, while the no-overlap estimation using position histogram and coverage histogram gives almost exactly the right answer size.

Finally, the time spent on estimating the result size of a simple twig query pattern, in all cases, using both the overlap algorithm and the no-overlap algorithm, is only a few tenths of a millisecond, which is very small compared to most database operations.

5.2 Synthetic Data Set

Whereas our tests on real data give us confidence, real data sets like DBLP are limited in size and complexity. We wanted to understand how our techniques would do given a more complex situation, with deeply nested and repeating element tags. For this purpose we used the IBM XML generator[21] to create synthetic data using a realistic DTD involving managers, departments and employees, as shown below:

```
<!ELEMENT manager (name,(manager | department | employee)+)>
<!ELEMENT department (name, email?, employee+, department*)>
<!ELEMENT employee (name+,email?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT email (#PCDATA)>
```

The predicates that we consider for this DTD are summarized in Table 3.

Table 3. Characteristics of Predicates on the Synthetic Data Set

Predicate Name	**Predicate**	**Node Count**	**Overlap Property**
manager	element tag = “manager”	44	overlap
department	element tag = “department”	270	overlap
employee	element tag = “employee”	473	no overlap
email	element tag = “email”	173	no overlap
name	element tag = “name”	1,002	no overlap

On the synthetic data set, we ran all types of queries we presented above. Here, for lack of space, we present only the results of some representative simple queries in Table 4.

Table 4. Synthetic Data Set: Result Size Estimation for Simple Queries

		Naive	**Overlap**		**No-Overlap**		**Real**
Ancs	**Desc**	**Est**	**Estimate**	**Est Time**	**Estimate**	**Est Time**	**Result**
manager	department	11,880	656	0.000070	N/A	N/A	761
manager	employee	20,812	1,205	0.000054	N/A	N/A	1,395
manager	email	7,612	429	0.000052	N/A	N/A	491
department	employee	127,710	2,914	0.000050	N/A	N/A	1,663
department	email	46,710	1,082	0.000054	N/A	N/A	473
employee	name	473,946	8,070	0.000062	559	0.000082	688
employee	email	81,829	1,391	0.000054	96	0.000080	99

In this data set, some of the nodes have the no-overlap property, some don't. We obtain the estimate with the *pH-Join* algorithm for all the queries, and use no-overlap estimation algorithm whenever possible. From Table 4 we can see that whenever there is no-overlap property, the no-overlap estimation algorithm gives an estimate that is much closer to the real answer size than those obtained by using the primitive *pH-Join* algorithm. For joins where the ancestor node doesn't have the no-overlap property, the primitive *pH-Join* algorithm computes an estimate that is very close to the real answer size. In spite of the deep recursion, the time to compute estimates remains a small fraction of a millisecond.

5.3 Storage Requirements

In this section, we present experimental results for the storage requirements of both position histograms and coverage histograms (recall as per Theorem 2, we expect the storage requirement to be $O(n)$). We also consider the impact of storage space on the accuracy of the estimates.

Fig. 11 shows the effect of increasing grid size on the storage requirement and the accuracy of the estimate, for the department-email query on the synthetic data set. Since the predicate department does not have the no-overlap property, the department-email pair join does not require any coverage information, therefore, only position histograms are built on predicate department and predicate email. The storage requirement for the two predicates are all linear to the grid size, with a constant factor close to 2. The result estimate is not very good when the histogram is very small. However, the ratio of the estimate to the real answer size drops rapidly and is close to 1 for grid sizes larger than 10-20.

Article-cdrom join is an example of query with no-overlap property. Here, both predicates (article, cdrom) have the no-overlap property, and consequently, we store both a position histogram and a coverage histogram for each of them. The storage requirement of these two predicates, as well as the accuracy of the

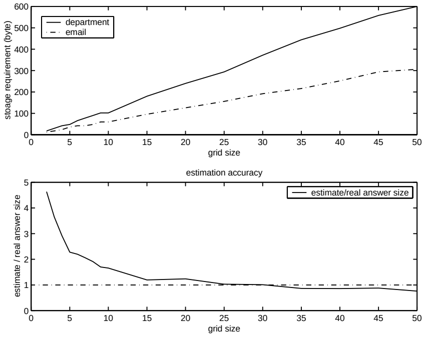

Fig. 11. Storage Requirement and Estimation Accuracy for Overlap Predicates (department-email)

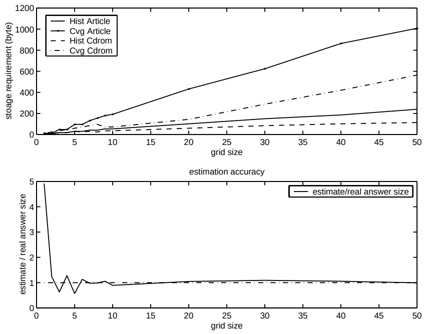

Fig. 12. Storage Requirement and Estimation Accuracy for No-Overlap Predicates (article-cdrom)

estimation is shown in Fig. 12. Note that the storage requirement for both the position histogram and the coverage histogram are linear to the grid size, which results in the total storage requirement grow linearly with a constant factor between 2 and 3. Another observation is that the estimate is not good when the grid size is very small, but it very quickly converges to the correct answer. Starting from the point where the grid size is larger than 5, the ratio of estimate to the real answer size is within 1 ± 0.05, and keeps in this range thereafter. The reason is that more information is caught by the coverage histogram than by only the position histogram.

6 Related Work

In [5], estimation techniques have been suggested for "twig" pattern queries in a hierarchical database, including XML. These techniques generalize the work on pruned suffix trees, presented in [16,10], and the notion of set hashing [3,6]. These techniques, while powerful where they apply, suffer from some limitations. For one thing, the techniques only apply to fully specified twig patterns, involving only parent-child links. However, one expects many, perhaps even the majority, of XML queries to involve patterns with ancestor-descendant links. For another thing, the computation techniques only provide the selectivity estimate for the entire query pattern. If estimates are required for sub-patterns, representing intermediate results in a potential query plan, these have to be computed separately. Finally, the entire technique relies on notions of pruning in suffix trees, and on maintaining small hashes for set similarity. These space-saving solutions obviously lose information, and much ingenuity is used to minimize this loss. In contrast, our techniques are insensitive to depth of tree, and require no pruning and do not admit the possibility of a non-local information loss. (However, our techniques are sensitive to the size of "symbol alphabet", and the techniques in the reference are probably more appropriate if a large number of basic node predicates are required).

McHugh and Widom [13] describe Lore's cost-based query optimizer, which maintains statistics about subpaths of length $\leq k$, and uses it to infer selectivity estimates of longer *path queries*. Estimating selectivity of path queries has also been the focus of a recent paper by Aboulnaga et al. [1], in which the authors propose two techniques for estimating the selectivity of path expressions. The first technique called *path trees* are similar to the pruned suffix trees of [5], but are more accurate for estimating the selectivity of certain path expressions. The second technique uses a Markov table to maintain statistics about all paths up to a certain length. The Markov table approach is similar to [13], but can be aggressively summarized, thereby reducing the amount of memory used to maintain statistics. The techniques presented in these two papers do not maintain correlations between paths, and consequently, these techniques do not allow them to accurately estimate the selectivity of tree query patterns, which are very natural in XML query languages.

Histograms of various types, including multi-dimensional histograms, have been used for query estimation in databases [14,12,7,8,9]. However, XML queries often involve an ancestor-descendant or parent-child relationships among nodes. Traditional one dimensional histograms are not enough to catch the position information of each single node, relationship among nodes, as well as other structure information of the XML data. Therefore, a novel histogram is introduced here which can capture the structure information native to XML data and estimate the result size effectively and accurately.

7 Conclusions and Future Work

As XML continues to grow in popularity, large repositories of XML documents are likely to emerge, and users are likely to pose complex queries on these data sets. Efficient evaluation of these complex queries will require accurate estimates. Queries in XML frequently specify structural patterns that specify specific relationships between the selected elements. Obtaining accurate estimates for these is not easy, by traditional means. In this paper we have proposed a novel histogram technique called *position histogram*, and estimation algorithms using the position histograms, that can be used for accurately estimating the answer size for arbitrarily complex pattern queries. While the histograms we develop are two-dimensional, they are sparse and only require storage that grows linearly (rather than quadratically) with grid size. The estimation algorithms are computationally efficient and require only a very small running time.

In many cases, schema information may be available, and frequently can be used to set an estimate to zero or (through uniqueness) equal to some other simpler estimate. We identify one specific schema restriction that occurs frequently in XML, namely the *no-overlap property*. We exploit this property in a modified estimation algorithm, which produces estimates that are more accurate than the estimates produced without factoring in this schema information. An open question is which other schema information can be considered together with the position histogram to further improve the accuracy.

Extensive experimental evaluation using both real and synthetic data sets demonstrates the effectiveness of the proposed techniques, for different type of queries, simple or complex, and on XML documents of different structure, shallow or deep and nested. The summary data structures and estimation techniques developed in this paper are an important piece of the query optimizer in the TIMBER[22] native XML database system under development at the University of Michigan.

Theoretical and experimental studies have also be done on how to exploit the estimation technique, using both position histograms and coverage histogram, to estimate the answer size for query patterns that are arbitrarily complex. Issues on estimation for queries with ordered semantics, with parent-child relationship, and estimation using histogram with non-uniform grid cells are also looked into. Please refer to [17] for detailed information of these techniques, as well as proofs of all the lemmas and theorems in this paper.

References

1. A. Aboulnaga, A.R. Alameldeen and J.F. Naughton. Estimating the Selectivity of XML Path Expressions for Internet Scale Applications. *VLDB*, 2001
2. T. Bray, J. Paoli, and C. M. Sperberg-McQueen. Extensible markup language (XML) 1.0. W3C Recommendation. Available at http://www.w3.org/TR/1998/REC-xml-19980210, Feb. 1998.
3. A. Broder. On the Resemblance and Containment of Documents. *IEEE SEQUENCES '97*, pages 21–29, 1998.
4. D. Chamberlin, J. Clark, D. Florescu, J. Robie, J. Siméon and M. Stefanescu XQuery 1.0: An XML Query Language. *W3C Working Draft*, http://www.w3.org/TR/xquery/, June 7, 2001.
5. Z. Chen, H. V. Jagadish, F. Korn, N. Koudas, S. Muthukrishnan, R.T. Ng, D. Srivastava. Counting Twig Matches in a Tree. *ICDE*, 2001.
6. Z. Chen, F. Korn, N. Koudas, and S. Muthukrishnan. Selectivity estimation for boolean queries. In *Proceedings of the ACM Symposium on Principles of Database Systems*, 2000.
7. Yannis E. Ioannidis. Universality of Serial Histograms. In *VLDB*, pages 256-267, 1993.
8. Y.E. Ioannidis, V. Poosala. Balancing Histogram Optimality and Practicality for Query Result Size Estimation. In *SIGMOD Conference*, pages 233-244, 1995.
9. H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K.C. Sevcik, T. Suel. Optimal Histograms with Quality Guarantees. *VLDB*, pages 275-286, 1998.
10. H. V. Jagadish, O. Kapitskaia, R. T. Ng, and D. Srivastava. One-dimensional and multi-dimensional substring selectivity estimation. In *VLDB Journal*, 9(3), pp.214–230, 2000.
11. H. V. Jagadish, L. V. S. Lakshmanan, T. Milo, D. Srivastava, and D. Vista. Querying network directories. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, Philadelphia, PA, June 1999.
12. R. J. Lipton and J. F. Naughton. Query size estimation by adaptive sampling. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, March 1990.

13. J. McHugh and J. Widom. Query optimization for XML. In *Proceedings of the International Conference on Very Large Databases*, pages 315–326, 1999.
14. M. Muralikrishna and D.J. DeWitt. Equi-Depth Histograms For Estimating Selectivity Factors For Multi-Dimensional Queries. In *SIGMOD Conference*, pages 28-36, 1988.
15. A.R. Schmidt, F. Waas, M.L. Kersten, D. Florescu, I. Manolescu, M.J. Carey and R. Busse. The XML Benchmark Project. Technical Report INS-R0103, CWI, Amsterdam, The Netherlands, April 2001.
16. M. Wang, J. S. Vitter, and B. Iyer. Selectivity estimation in the presence of alphanumeric correlations. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 169–180, 1997.
17. Yuqing Wu, Jignesh M. Patel, H.V.Jagadish. Histogram-based Result Size Estimation for XML Queries. University of Michigan Tech Report, 2002.
18. C. Zhang, J.F. Naughton, D.J. DeWitt, Q. Luo and G.M. Lohman. On Supporting Containment Queries in Relational Database Management Systems. *SIGMOD*, 2001
19. DBLP data set. Available at http://www.informatik.uni-trier.de/ley/db/index.html.
20. ibiblio Organization. XML dataset for Shakesapeare drama. Available at http://sunsite.unc.edu/pub/sun-info/xml/eg/shakespeare.1.10.xml.zip.
21. IBM. XML generator. Available at http://www.alphaworks.ibm.com/tech/xmlgenerator.
22. TIMBER Group. TIMBER Project at Univ. of Michigan. Available at http://www.eecs.umich.edu/db/timber/.