PERGAMON

Information Systems 28 (2003) 33–59

www.elsevier.com/locate/infosys

Using histograms to estimate answer sizes for XML queries ☆

Yuqing Wu*,[1], Jignesh M. Patel[2], H.V. Jagadish

RTCL/SSRL, EECS, University of Michigan, 1301 Beal Avenue, Ann Arbor, MI 48109-2122, USA

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 minuscule computational cost. These techniques have been implemented in the context of the TIMBER native XML database (available at http://www.eecs.umich.edu/db/timber) at the University of Michigan.

Keywords: XML; Estimation; Histogram; Schema

1. Introduction

XML data [1] 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

☆ Recommended by XXX.

*Corresponding author. Tel.: +1-734-647-1254; fax: +1-734-763-8094.

E-mail address: yuwu@eecs.umich.edu (Y. Wu)

[1] Supported in part by NSF under Grant IIS-9986030, DMI-0075447 and IIS-0208852.

[2] Supported in part by NSF under Grant IIS-0208852, and by a gift donation from IBM.

0306-4379/02/$ - see front matter © 2002 Elsevier Science Ltd. All rights reserved.
PII: S0306-4379(02)00048-0

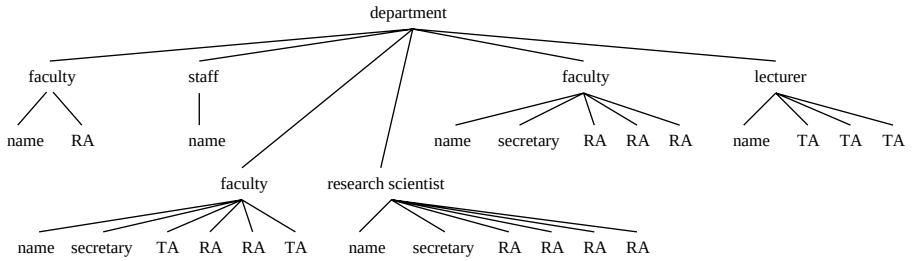


Fig. 1. Example XML document.

the hierarchical data. Such queries form an integral component of query languages proposed for XML (for example, [2]), and for LDAP directories [3].

Example 1.1. 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 selectivity of various subqueries can help in identifying cheap query evaluation plans.

Example 1.2. 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

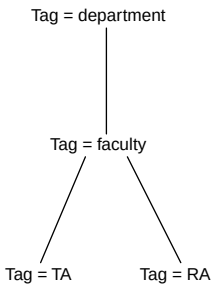


Fig. 2. Example pattern tree.

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.

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. Variants of the position histogram technique, leveled histogram for parent–child matching and estimation with non-uniform histogram, will be exploited in Sections 6 and 7. 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 8. Conclusions and directions for future work are outlined in Section 9.

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, $P: \{v : v \in V_T\} \to \{0, 1\}$. For each predicate $\alpha \in P$, for each node $v \in T$, we have either $\alpha(v)$ is true or $\alpha(v)$ is false. (See Section 3.6 for a discussion of how to obtain this set 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 P.

A *match* of a pattern query Q in a T is a total mapping $h: \{u : u \in Q\} \to \{x : x \in T\}$ 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 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 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 University of Wisconsin researchers in the course of the Niagara [4] 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.

Example 3.1. Fig. 3 shows the same example XML date as shown in Fig. 1. There are two numbers in the parenthesis following each element tag. The first one is the start label and second one is the end label. We can see that for all nodes, the end label is larger or equal to the start label. For leaf nodes, the two labels have the same value.

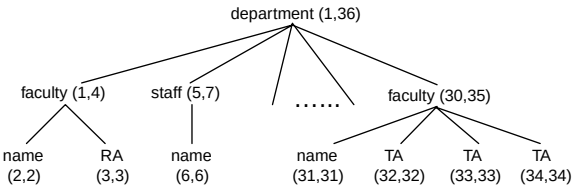


Fig. 3. Example XML data with numbering schema.

Given a limited set P of predicates of interest, one should expect that there will be index structures that identify lists of nodes satisfying each predicate in 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*.

Position histograms, even though defined over a two-dimensional space, have considerable structure, as shown in Fig. 4.

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

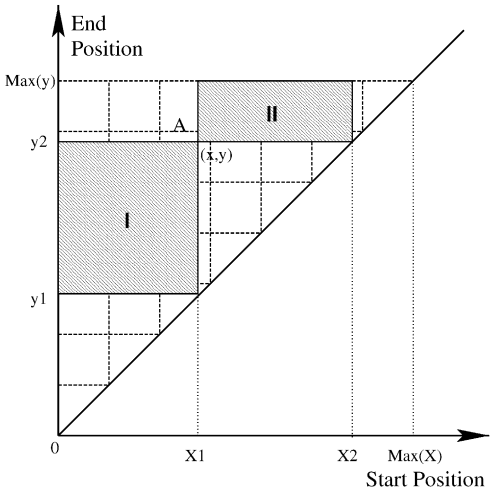


Fig. 4. Forbidden regions in a position histogram due to one node.

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 3.1. *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* $k<i$.

Proof. Assume grid cell (i,j) is non-zero. Then, there is at least one point (x,y) in this grid cell, such that $x[i-1]<x<x[i]$, and $y[j-1]<y<y[j]$, where $x[i-1]$ and $x[i]$ are the boundary of the grid cell (i,j) on x-axis.

The Diagonal line is $y=x$, so, in Fig. 4, we have $x_1=y_1=x$, $x_2=y_2=y$.

Assume that a grid cell (k,l) that satisfies condition (a) is non-zero. There is at least one point (x',y') in this grid cell such that $x_1<x[k-1]<x'<x[k]<x_2$, and $y_2<y[l-1]<y'<y[l]$. With condition (a), we have $x=x_1<x[k-1]<x'<x[k]<x_2=y<x[j-1]<y' \Rightarrow x<x'<y<y'$.

In a well-formed XML document, the intervals defined by the start position and end position of nodes cannot interleave with each other. So, an XML document with nodes on both points (x,y) and (x',y') is not well-formed. □

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. 5. There are nine regions in the plane to consider, marked A (R_0), R_1 through R_8 in 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. 4 with Fig. 5 to get Fig. 6, assuming that the forbidden ranges are based on node (x,y), which is the bottom-right node in grid cell A.

Given predicates P_1 and P_2, both in P, we show how to estimate the number of pairs of nodes u,v

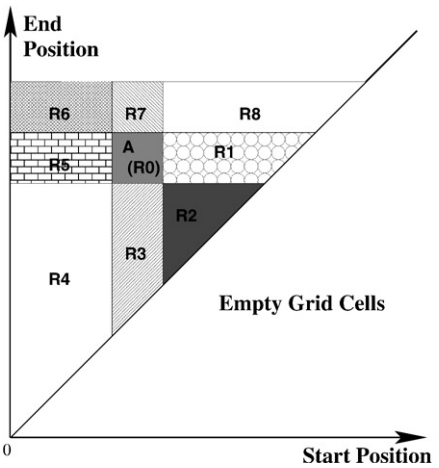


Fig. 5. Layout of position histogram.

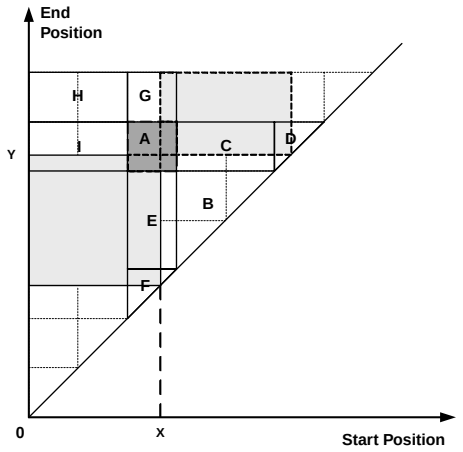


Fig. 6. Estimating join counts with position histogram.

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 3.1. 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*.

3.2.1. Ancestor-based join estimation

If A is off-diagonal, as shown in Fig. 6, 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. 6), 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 (see the following proof). Putting all these estimates together, the ancestor-based estimation for each off-diagonal grid cell can be expressed as the first formula in Fig. 7.

When grid cell A is on-diagonal, regions B, C, D, E, F do not 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 (see the following proof).

3.2.2. Descendant-based join estimation

Referring to Fig. 6, no matter whether A is on-diagonal or off-diagonal, all ancestors of a point in the grid cell A will be in regions A, G, H or I. Following argument similar to those in the ancestor-based estimation above, all points in region G, H and I 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.

Proof. Assume that P_1 is the predicate associated with the ancestor node, and P_2 is the predicate associated with the descendant node. H_1 and H_2 are the position histograms built on P_1 and P_2, respectively.

Primitive Estimation: ancestor-based

For off-diagonal grid cell A:

$$Est_{P_{12}}[A] = Hist_{P_1}[A] \times \{\tfrac{1}{4} \times Hist_{P_2}[A] + Hist_{P_2}[B] + Hist_{P_2}[C] + Hist_{P_2}[E] + \tfrac{1}{2} \times (Hist_{P_2}[D] + Hist_{P_2}[F])\}$$

For on-diagonal grid cell A:

$$Est_{P_{12}}[A] = \tfrac{1}{12} \times Hist_{P_1}[A] \times Hist_{P_2}[A]$$

Primitive Estimation: descendant-based

For off-diagonal grid cell A:

$$Est_{P_{12}}[A] = Hist_{P_2}[A] \times \{Hist_{P_1}[G] + Hist_{P_1}[H] + Hist_{P_1}[I] + \tfrac{1}{4} \times Hist_{P_1}[A]\}$$

For on-diagonal grid cell A:

$$Est_{P_{12}}[A] = Hist_{P_2}[A] \times \{Hist_{P_1}[G] + Hist_{P_1}[H] + Hist_{P_1}[I] + \tfrac{1}{12} \times Hist_{P_1}[A]\}$$

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. 7. Formulae for primitive join estimation.

Taking a grid cell A in H_1, and the grid cell at the same position in H_2, let us denote the node count of the grid cell in H_1 is A_1, and that in H_2 is A_2. We will estimate how many node pairs (*Est*) are to be generated from the ancestor–descendant join, with the nodes fallen in the grid cell A in both the ancestor and the descendant.

Assume that the boundary of grid cell A is (x_1, x_2, y_1, y_2).

- when A is not on-diagonal
  $Est = \int_{x_1}^{x_2} \int_{y_1}^{y_2} (x_2 - x)(y_2 - y)$
  $= 1/4 \times (x_2 - x_1)(y_2 - y_1) \times (x_2 - x_1)(y_2 - y_1)$
- when A is on-diagonal
  $Est = \int_{x_1}^{x_2} \int_{y_1}^{y_2} 1/2 \times (y - x)^2$
  $= 1/12 \times 1/2 \times (x_2 - x_1)(y_2 - y_1) \times 1/2 \times (x_2 - x_1)(y_2 - y_1)$ □

Example 3.2. Let us have a look at the example XML document in Fig. 1 again, with the query pattern we discussed in Section 2. The 2×2 histograms of predicates "element tag = faculty" and "element tag = TA" are shown in Fig. 8. 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.

faculty

0	1
2	

TA

0	3
2	

Fig. 8. Example position histograms.

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.

3.3.1. 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.

Lemma 3.2. *In a $g \times g$ position histogram, there are at most g 2-off-diagonal non-zero grid cells.*

Proof. Given a $g \times g$ histogram, H_g, we call the grid cell (i,j) "2-off-diagonal" if $j - i \geqslant 2$. They are the grid cells that are not on-diagonal and not 1 from diagonal. Let us denote the number of non-zero grid cells amongst all 2-off-diagonal cells in histogram H_g as $N_{2off}(H_g)$.

We prove it by induction.

- *Base case*:
  When $n = 1$ or 2, there are no 2-off-diagonal cells.
  $N_{2off}(H_1) = N_{2off}(H_2) = 0$.
  When $n = 3$, there is one 2-off-diagonal cell and it can be non-zero.
  $N_{2off}(H_3) = 1 < 3 = n$.
- *Inductive case*:
  Assume that the lemma holds for all grid sizes up to $g \times g$. For $(n+1) \times (n+1)$ histogram,
  - If there are no non-zero 2-off-diagonal cells, $N_{2off}(H_{n+1}) = 0 < n + 1$
  - If there are m non-zero 2-off-diagonal cells $(i_1,j_1)\ldots(i_m,j_m)$. Pick the grid cell (i_a,j_a) such that $i_a + j_a = minimum(i_k + j_k)(k = 1...m)$.

    As shown in Fig. 6, the whole histogram can be divided into six ranges. For the two forbidden regions, all grid cells that fall into these ranges have zero count. Due to the way (i_a,j_a) is chosen, all the grid cells in the rectangle to the up-left of the two forbidden regions have zero count. The three triangles are of size i_a, $j_a - i_a + 1$, and $n + 1 - j_a + 1$, respectively. Each of them can be regarded as a smaller histogram itself. The non-zero grid cell (i_a,j_a) is included in the second small histogram. So,

$$\begin{aligned} &N_{2off}(H_{n+1}) \\ &\quad = N_{2off}(H_{i_a}) + N_{2off}(H_{j_a - i_a + 1}) \\ &\qquad + N_{2off}(H_{(n+1)-j_a+1}) \\ &\quad \leqslant (i_a - 1) + (j_a - i_a + 1 - 1) \\ &\qquad + ((n+1) - j_a + 1 - 1) \\ &\quad = i_a - 1 + j_a - i_a + (n+1) - j_a \\ &\quad = n < n + 1 \quad \square \end{aligned}$$

Theorem 3.1. *In a $g \times g$ grid, the number of position histogram grid cells with non-zero counts is $O(g)$.*

Proof. Use Lemma 3.2, in an $g \times g$ position histogram, the number of 2-off-diagonal non-zero grid cells is at most g.

For a $g \times g$ position histogram, in the worst case when all the grid cells that are not 2-off-diagonal are non-zero, the total number of non-zero grid cells is at most $3g$. $\square$

3.3.2. Time required

Based on the formulae for both ancestor-based 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. 6) 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

```
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].desc = 0;
          }
          else if (i == j-1)
          {
               pSum[i][j].right = pSum[i+1][j].self; // row summation
               pSum[i][j].desct = 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].desc = pSum[i+1][j].down + pSum[i+1][j].desc;
          }
     }

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].desc
                    + 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.

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.

3.4. Complex patterns

Thus far we have dealt with a simple query pattern involving a single ancestor–descendant pair.

In general, of course, one can have query patterns that are much more complex. The technique presented above can be adapted for this purpose using the intermediate estimates, in the form of histograms. The basic idea is to construct the complex query pattern as a sequence of nodes added to a simple pattern. In other words, one can choose any one edge in the complex query pattern, estimate how many times that simple query pattern occurs, and then use the result to estimate how many times a slightly less simple query pattern, with two edges and three nodes occurs, and so on, adding one node at a time.

The only catch in being able to do this correctly is that the final summation to compute the total estimate should not be carried out. Instead, one should obtain a grid cell by grid cell estimate for the simpler query pattern, in effect constructing a histogram for the occurrence of this pattern, and then use this histogram in the next step. The question that arises is what is the start and end position for a pattern. These are concepts defined for a node, or an element in the XML document. The answer is that the positions we choose are exactly the positions at which a selected *distinguished* node in the pattern, which is the node used in the following estimation, occurs. It is to this distinguished node that the next node added to the pattern, in the sequential expansion of the pattern, must join.

There are many different ways in which a complex pattern can be decomposed, leading to different estimation formulae. However, careful algebraic manipulations can be used to establish the following reassuring theorem:

Theorem 3.2. *The count estimate for any pattern of predicates is independent of the join order for algorithm pH-join.*

Proof. Let us denote a tree pattern as (V, E), where V is the set of nodes in the pattern, and E is the set of edges. Each element in E is represented by a pair of nodes in V. $Hist_v$ is the position histogram of node v (in V). Let $Est(V, E)$ be the count estimation of the pattern (V, E). Prove that for any pattern (V, E), $Est(V, E)$ obtained in any order can be written in one formula.

Randomly choose one grid cell (i_k, j_k) from $Hist_{v_k}$, the position histogram of node v_k in V. Let $Ch_V = \{(i_1, j_1) \cdots (i_n, j_n)\}$ denote the set of grid cells chosen in this manner, one cell for each node from node set V. The estimation of the selected grid cells joined with respect to the pattern (V, E) is denoted by $E(Ch_V)$. $Est(V, E) = \sum E(Ch_V)$ for all combination of $i_1, j_1 \ldots i_n, j_n$. We are going to prove that for the selected grid cells Ch_V,

$$E(Ch_V) = \prod_{p \in 1 \ldots n} Hist_p[i_p][j_p] \times \prod_{q \in 1 \ldots n-1} Factor(e_q, i_{e_{q_a}}, j_{e_{q_a}}, i_{e_{q_d}}, j_{e_{q_d}}),$$

regardless of the order in which the estimation is computed and the node on which the result histogram is based.

Here, $Factor(e_q, i_{e_{q_a}}, j_{e_{q_a}}, i_{e_{q_d}}, j_{e_{q_d}})$ is defined as following:

- $e_q \in E$, is an edge in the pattern, with ancestor node e_{q_a} and descendant node e_{q_d}.

- $(i_{e_{qa}}, j_{e_{qa}}) \in Ch_V$, it is a grid cell randomly chosen from $Hist_{e_{qa}}$; and $(i_{e_{q_d}}, j_{e_{q_d}}) \in Ch_V$, it is a grid cell randomly chosen from $Hist_{e_{q_d}}$.
- The value of $Factor(e, i_a, j_a, i_d, j_d)$ is defined as:
  - value= 0, when $i_a < i_d$ or $j_a < j_d$
  - value = 1/4, when $i_a = i_d$, $j_a = j_d$, and $i_a \neq j_a$
  - value = 1/12, when $i_a = j_a = i_d = j_d$
  - value = 1, otherwise.

We prove by induction.

- *Base case*: For a two-node twig join pattern, the result of ancestor-based estimation is the same as descendant-based estimation.
  (V, E) is a two-node twig pattern, where the $V = \{v_1, v_2\}$, $E = \{e\}$, $e = (v_1, v_2)$. Two grid cells are randomly chosen from $Hist_{v_1}$ and $Hist_{v_2}$, they are $Hist_{v_1}[i_1, j_1]$ and $Hist_{v_2}[i_2, j_2]$. So, $Ch_V = \{(i_1, j_1), (i_2, j_2)\}$. Based on the formulas for pH-Join, we have: for ancestor-based estimation, $E\_A(Ch_V) = Hist_{v_1}[i_1][j_1] \times Hist_{v_2}[i_2][j_2] \times Factor(e, i_1, j_1, i_2, j_2)$ for descendant-based estimation, $E\_D(Ch_V) = Hist_{v_2}[i_2][j_2] \times Hist_{v_1}[i_1][j_1] \times Factor(e, i_1, j_1, i_2, j_2)$ so, $E\_A(Ch_V) = E\_D(Ch_V) = E(Ch_V)$.
- *Inductive case*: Assume that for any tree pattern (V, E) with t $(<n)$ nodes, Ch_V is the set of randomly selected grid cells, then,

$$E(Ch_V) = \prod_{p \in 1 \ldots t} Hist_p[i_p][j_p] \times \prod_{q \in 1 \ldots t-1} Factor(e_q, i_{e_{qa}}, j_{e_{qa}}, i_{e_{q_d}}, j_{e_{q_d}})$$

regardless of the join order and the node on which the result histogram is based.

For a pattern (V, E) with n nodes, Ch_V is the set of selected grid cells. Given a join plan, e is the edge on which the last join operation is based on. e divides the pattern into two subpatterns (V_1, E_1) and (V_2, E_2), both of them have less than n nodes, and $V_1 \cup V_2 = V$, $E_1 \cup E_2 \cup e = E$. Assume that the two nodes at the end of edge e are $v_a (\in V_1)$, $v_b (\in V_2)$.

With the assumption, the estimation of the subpattern (V_1, E_1), based on v_a, regardless of the join order, is

$$E(Ch_{V_1}) = \prod_{(i,j) \in Ch_{V_1}} Hist_p[i][j] \times \prod_{e \in E_1} Factor(e, i_{e_a}, j_{e_a}, i_{e_d}, j_{e_d}).$$

Similarly, the estimation of the subpattern (V_2, E_2), based on v_b, regardless of the join order, is

$$E(Ch_{V_2}) = \prod_{(i,j) \in Ch_{V_2}} Hist_p[i][j] \times \prod_{e \in E_2} Factor(e, i_{e_a}, j_{e_a}, i_{e_d}, j_{e_d}).$$

Join the intermediate result above on edge e, the final result of this set of join plan is

$$\begin{aligned} E' &= E(Ch_{V_1}) \times E(Ch_{V_2}) \times Factor(e, i_{e_a}, j_{e_a}, i_{e_d}, j_{e_d}) \\ &= \prod_{p \in 1 \ldots t} Hist_p[i_p][j_p] \times \prod_{q \in 1..t-1} Factor(e_q, i_{e_{qa}}, j_{e_{qa}}, i_{e_{q_d}}, j_{e_{q_d}}) \\ &= E(Ch_V) \end{aligned}$$

as we defined at the very beginning. □

3.5. Compound predicates

Often, the predicates applied at a node may not belong to the set of basic predicates 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*. Note that the basic predicates do *not* have to be independent over the entire data set—their respective position histograms will capture any correlation. Our independence assumption here is within a single grid cell, and is justifiable

in exactly the same way as a uniformity assumption within a single histogram bucket is justified without requiring that the entire data set be uniform globally.

To be able to manipulate counts, we need to convert these into the appropriate probabilities. What we require is the probability of a node satisfying some basic predicate, given that it is in a specific histogram grid cell (start and end position). 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. Now, given a specific grid cell, we can "normalize" the count associated with any basic predicate, dividing by the normalization constant, to obtain the probability of the basic predicate being satisfied by a point in that grid cell. For a compound predicate, we can manipulate probabilities assuming independence (taking the product for intersection, the sum minus the product for union, and one minus the original for negation). The final probability for the compound predicate can be "denormalized" by multiplying it with the normalization constant.

For example, consider two histograms for the predicates P_1 and P_2 corresponding to the predicates "author contains 'Jane'" and "author contains 'Doe'", respectively. To compute the position histogram for a predicate "author contains 'Jane' or author contains 'Doe' ", we use the position histograms on P_1 and P_2, and the position histogram on the predicate "TRUE", as shown in Fig. 10.

3.6. Predicate set selection

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 P of basic predicates. Predicates in XML queries fall into two general categories:

3.6.1. 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 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.

3.6.2. 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 a 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 [5].

Also, in cases where the predicate is over an attribute that takes a large number of values,

Formula for Computing Histogram of Compound Predicate:

$$Hist_{P_3}[i][j] = Hist_{P_1 \cup P_2}[i][j] = Hist_{TRUE}[i][j] \times \left\{ \frac{Hist_{P_1}[i][j]}{Hist_{TRUE}[i][j]} + \frac{Hist_{P_2}[i][j]}{Hist_{TRUE}[i][j]} - \left[\frac{Hist_{P_1}[i][j]}{Hist_{TRUE}[i][j]} \times \frac{Hist_{P_2}[i][j]}{Hist_{TRUE}[i][j]} \right] \right\}$$

Fig. 10. Formula for computing histogram of compound predicate.

standard histogram bucketizing techniques can be used to capture the number of elements that posses attributes in appropriate regions of values, and these can be used to estimate the number that satisfy any chosen range. In our case, each such histogram bucket becomes a "basic predicate". Given any predicate over values of this attribute, standard histogram estimation techniques can be used to evaluate the necessary "compound predicate". As an example, consider a bibliography database where each book element has an attribute for the number of pages in the book. We may bucketize this attribute value into disjoint adjacent ranges, such as 0–100, 101–200, 201–300, and so on. Now, given a query predicate that asks for books that have between 180 and 350 pages, we know how to estimate this number. For the position histograms introduced in this paper, we build one for each attribute value range bucket above. For any position histogram grid cell in question, the appropriate value histogram numbers can be combined to produce an appropriate estimate for the specific compound predicate position histogram.

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. 11, 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 4.1. 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

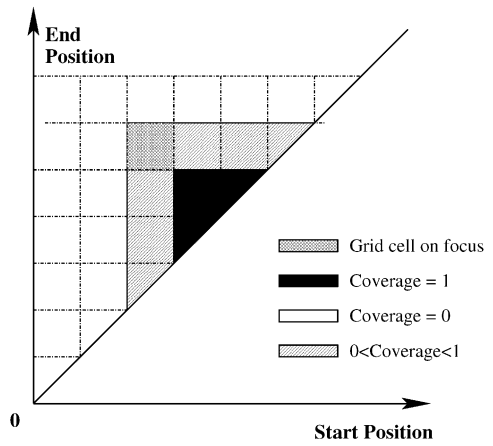


Fig. 11. Coverage histogram for no-overlap predicate.

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:

Lemma 4.1. *In a position histogram for a no-overlap predicate, every off-diagonal grid cell has count at most one.*

Proof. Let $[s_{\min}, s_{\max})$ and $[e_{\min}, e_{\max})$ be the start and end position ranges, respectively. For an off-diagonal grid cell, we must have $e_{\min} \geqslant s_{\max}$. Suppose there are two nodes u and v in this cell, with position (s_u, e_u) and (s_v, e_v), respectively. Due to the no-overlap property, we must have $s_v > e_u$. But we cannot then have both $s_v < s_{\max}$ and $e_u \geqslant e_{\min}$. Hence proved by contradiction. □

Theorem 4.1. *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.*

Proof. For a non-zero grid cell (i,j), we say that it covers the grid cells (p,q) if $p \geqslant i, q \leqslant j$. Among all the grid cells covered by (i,j), let us denote the number of on-diagonal grid cells covered by (i,j) as $D(i,j)$.

$$D(i,j) = |\{(p,q) | (p,q) \text{ is covered by } (i,j)\}| = j - i + 1.$$

Consider a non-zero grid cell (i,j) in the position histogram. In the coverage histogram,

- $Cvg[p][q][i][j]$ count zero, when $p < i$ or $q > j$.
- $Cvg[p][q][i][j]$ count one, when $p > i$ and $q < j$.
- Partial coverage can happen only in the remaining grid cells (that is, $p = i, q \leqslant j$ or $p \geqslant i, q = j$), the number of these cells is $2(j-i)+1 \leqslant 2 \times D(i,j)$.

With the no-overlap property, the on-diagonal grid cells covered by two non-zero grid cell cannot over lap, except on the boundary. That is, for any two non-zero grid cells (i_1,j_1), (i_2,j_2),

$|\{(p,q) | (p,q) \text{ is covered by } (i_1,j_1) \wedge (p,q) \text{ is covered by } (i_2,j_2)\}| \leqslant 1$.

From this, we have: for any two non-zero grid cells (i_1,j_1), (i_2,j_2),

$i_1 \leqslant i_2 \Rightarrow j_1 \leqslant i_2$.

In a $g \times g$ position histogram,

$$\sum_{nonzerogridcell(i,j)} D(i,j) \leqslant 4n.$$

$$P(Cvg) = \sum_{\substack{(i,j) \\ Hist[i][j] \neq 0}} 2 \times D(i,j) \leqslant 8n = O(g). \qquad \square$$

4.3. Estimation algorithm for no-overlap predicates

Consider descendant-based estimation (with the descendant node as the outer loop) first. For each grid cell s in the grid with a non-zero count of nodes satisfying P_2, as determined from the position histogram, we must determine the fraction of these nodes that have an ancestor that satisfies P_1. (Since P_1 has the no-overlap property, there can be no more than one such ancestor.) This is easily obtained as the sum of the coverage histograms for cell pairs (r,s) and nodes in r

satisfying P_1, for all the grid cell r's to the up left of grid cell s. The assumption here is that for any node in grid cell r, the fraction of P_2 satisfying nodes in s that are descendants of a P_1 satisfying node in r is the same as the fraction of all nodes in s that are descendants of a P_1 satisfying node in r. This gives us a position histogram of joined pairs for the two predicates, with counts of joined pairs attributed to the grid cell in which the descendant node occurs.

Turn now to ancestor-based estimation (with the ancestor node as the outer loop), where the counts of joined pairs in the result histogram are to be attributed to the grid cell in which the ancestor occurs. For a grid cell r, this is obtained by summing over each s the predicate P_1 coverage histogram for cell pairs (r,s) multiplied by the count in grid cell s of the predicate P_2 position histogram, for all the grid cell s's to the right and below grid cell r. Once again, this gives us a position histogram of joined pairs for the two predicates, but this time with counts of joined pairs attributed to the grid cell in which the ancestor node occurs.

To be able to evaluate estimates for complex patterns, by building up the patterns one edge at a time, we need to create as intermediate results, not just estimates of position (count) histograms but also of coverage histograms. In addition, we now have a further complication having to do with multiplicity of the join node versus the count of the (intermediate) pattern as a whole. These two are not necessarily the same, and their ratio is called the *Join Factor*.

Consider a simple three-node twig join query, with node A, B and C shown in Fig. 14(a) and node A has no-overlap property. Assume that we compute the A–B pair estimate first. After we get the estimate for the count of A–B pairs, we need to estimate the answer size of the join between the result of the A–B pair and C. Here, what we need is the number of distinct A's participating in A–B join ($Hist_{AB\_A}$, called *participation histogram*), the number of B's (on average) that join with each distinct A node in A–B pair($Jn\_Fct_{AB\_A}$), and the coverage information of the distinct A's participating in A–B join (Cvg_{AB_A}), all at grid cell level. An ancestor-based position histogram estimation, described two paras above, would merely compute the product of $Hist_{AB\_A}$ and $Jn\_Fct_{AB\_A}$. We can divide this product by $Hist_{AB\_A}$, estimated using a binomial distribution, to get the needed join factor value, to be used in the subsequent (A–C) join. These ideas, modulo a few messy details, lead to the estimation formulae presented in Fig. 12.

In the formulae in Fig. 12, what we are dealing with is not just a three-node twig as stated above, but a more general one where two subpatterns named A and B are joined, with ancestor–descendent relationship between nodes satisfying P_1 in subpattern A and nodes satisfying P_2 in subpattern B. What we have are the participation histogram, join factor and coverage histogram for the node that satisfy P_1 in the subpattern A, as well as those histograms for nodes that satisfy P_2 in the subpattern B. All of them come from previous estimation computation. What we are estimating here is the estimation histogram of AB (subpattern A join B), the participation histogram of P_x (x in 1,2), and the join factor and coverage histogram of P_x in AB.

Example 4.1. Let us 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 is used. The Coverage Histogram of predicate "element tag = faculty" is shown in Fig. 13. 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

Ancestor-based pattern count estimate:

$$Est_{AB}[i][j] = Jn\_Fct_{A\_P_1}[i][j] \times \sum_{\substack{m=i..j \\ n=m..j}} (Cvg_{A\_P_1}[m][n][i][j] \times Hist_{B\_P_2}[[m][n] \times Jn\_Fct_{B\ P_2}[m][n])$$

Descendant-based pattern count estimate:

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

Join factor estimate:

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

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

$$N[i][j] = Hist_{A\_P_1}[i][j]$$

$$M[i][j] = \sum_{\substack{m=i..j \\ n=m..j}} Hist_{B\_P_2}[[m][n]$$

$$Hist_{AB\_P_1}[i][j] = N[i][j] \times (1 - (\frac{N[i][j]-1}{N[i][j]})^{M[i][j]})$$

case3: item P_2 is no-overlap, estimation is descendant-based

$$Hist_{AB\ P_2}[i][j] = Hist_{B\ P_2}[i][j] \times \sum_{\substack{m=0..i \\ n=j..max\_y}} (notzero(Hist_{A\_P_1[m][n]}) \times Cvg_{A\_P_1}[i][j][m][n])$$

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. 12. Estimation formulae for no-overlap predicates.

[6] and the Shakespeare play data set [7]. Results obtained in all cases were substantially similar. In the interests of space, we present results only for the DBLP data set [8] that is probably most familiar to readers of this paper. The DBLP data set is 9 M bytes in size and has approximately 0.5 M 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.)

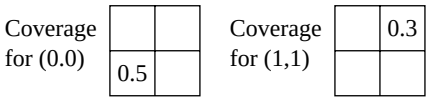


Fig. 13. Example coverage histogram for faculty.

5.1.1. 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.

Table 1
Characteristics of some predicates on the DBLP data set

Pred name	Predicate	Node count	Overlap property
article	element tag = "article"	7366	No overlap
author	element tag = "author"	41,501	No overlap
book	element tag = "book"	408	No overlap
cdrom	element tag = "cdrom"	1722	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"	7834	N/A
1980's	compound	13,066	N/A
1990's	compound	3963	N/A

Table 2
Result size estimation for simple queries on DBLP data set

Ance	Desc	Naive estimate	Desc num	Overlap		No-overlap		Real result
				Estimate	Est time	Estimate	Est time	
article	author	305,696,366	41,501	2,415,480	0.000344	14,627	0.000263	14,644
article	cdrom	12,684,252	1722	4379	0.000290	112	0.000261	130
article	cite	243,792,502	33,097	671,722	0.000229	3958	0.000261	5114
book	cdrom	702,576	1722	179	0.000142	4	0.000259	3

With the schema information and no position histogram, if the ancestor node has no-overlap property, the best (upper-bound) estimate of the 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.

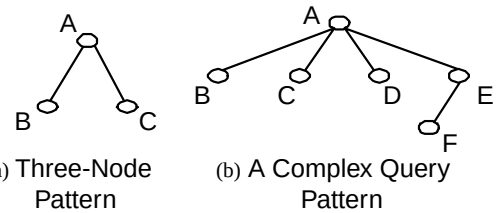
Finally, the time spent on estimating the result size of a simple twig query pattern, in all cases, using both the overlap algorithm and the no-overlap algorithm, is only a few tenths of a millisecond, which is very small compared to most database operations.

5.2. Answer size estimation for complex queries

For complex queries the final answer size estimation is computed by building the complex pattern one edge at a time and computing the estimates as we keep adding new edges. A set of supporting information (histogram for the joined nodes, join factor, etc.) is estimated and carried on from one simple join to another. In order to understand how the structure and the size of the pattern may affect the outcome of the estimation, we designed a set of queries with three-node-twig pattern, and a more complex pattern (with six nodes). The results of these cases are presented below.

5.2.1. Three-node twig queries

A three-node twig query pattern, as shown in Fig. 14(a), is an archetype for more complex patterns. We are required to compute at least one ancestor-based estimate, and use this in conjunction with the other descendant to obtain the final estimate. When that ancestor node has the no-overlap property, we not only estimate the number of answers of the subquery, but also estimate the required supporting information. A number of three node twig queries were run against the DBLP data set. The results for a few representative queries are shown in Table 3. The node names A, B, C refer to positions in Fig. 14(a). Results are presented not just for the entire pattern, but also for the intermediate results obtained if the AB edge or the AC edge is evaluated first. As expected, the final estimates obtained are virtually identical irrespective of how they were obtained: the (AB)C estimate first computes AB and then joins C to the result, whereas the (AC)B estimate first computes AC. The time to obtain these estimates is once again very small (approximately 1 ms each).



(a) Three-Node Pattern (b) A Complex Query Pattern

Fig. 14. Example query pattern.

Table 3
Result size estimation for three node twig queries on the DBLP data set

Node A	Node B	Node C	AB		AC		(AB)C est	(AC)B est	ABC real
			Estimate	Real	Estimate	Real			
article	author	cite	14,627	14,644	3960	5114	11,619	11,619	14,259
article	title	author	7323	7366	14,627	14,644	15,321	15,341	14,644
book	title	year	610	408	607	408	470	471	408
book	url	cdrom	493	33	5	3	3	3	3
proc	title	url	19	32	11	28	11	11	28

Table 4
Result size estimation for complex query on DBLP data set

Query	A	B	C	D	E	F	Estimate	Real size	Est time
Query1	article	title	author	year	cite	conf	2421	5826	0.003232
Query2	book	title	author	url	year	1990's	52	44	0.002540

5.2.2. Complex queries

Intuitively, when the number of nodes in the query pattern increases, the accuracy of the answer size estimation is expected to decrease since errors propagate multiplicatively. We ran a number of complex queries on the DBLP data set. Surprisingly, we found that even with very complex queries, the estimation error is not bad (within a factor of two). The reason is that each simple join brings in some additional selectivity factors. If the data nodes for a specific predicate are clustered in a small region of the histogram space (e.g. element book appears only in the first 1/10 of the DBLP data set), then the histogram produced after joining with this predicate will have many empty grid cells. Any predicate in the complex pattern that has this property brings about not just a numeric reduction but also a "focussing" on the region of interest. In the queries that we present, the article predicate in Query1 and the book predicate in Query2 exhibit this behavior.

In the interest of space, the results of only two complex queries are shown in Table 4. The structure of the query pattern is shown in Fig. 14(b). Note that the computation time for the entire complex estimate is still only about 3 ms.

5.3. 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 [9] to create synthetic data using a realistic DTD involving managers, departments and employees, as shown in Fig. 15.

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

Fig. 15. DTD used in generating synthetic data set.

The predicates that we consider for this DTD are summarized in Table 5.

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 6.

In this data set, some of the nodes have the no-overlap property, some do not. We obtain the estimate with the *pH-Join* algorithm for all the queries, and use no-overlap estimation algorithm whenever possible. From Table 6, 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 does not 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.

5.4. Storage requirements

In this section, we present experimental results for the storage requirements of both position histograms and coverage histograms (recall as per Theorem 4.1, we expect the storage requirement to

Table 5
Characteristics of predicates on the synthetic data set

Pred 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"	1002	No overlap

Table 6
Synthetic data set: result size estimation for simple queries

Ancestor	Descendant	Naive estimate	Overlap		No-overlap		Real result
			Estimate	Est time	Estimate	Est time	
manager	dept	11,880	656	0.000070	N/A	N/A	761
manager	employee	20,812	1205	0.000054	N/A	N/A	1395
manager	email	7612	429	0.000052	N/A	N/A	491
dept	employee	127,710	2914	0.000050	N/A	N/A	1663
dept	email	46,710	1082	0.000054	N/A	N/A	473
employee	name	473,946	8070	0.000062	559	0.000082	688
employee	email	81,829	1391	0.000054	96	0.000080	99

be $O(n)$). We also consider the impact of storage space on the accuracy of the estimates.

Fig. 16 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.

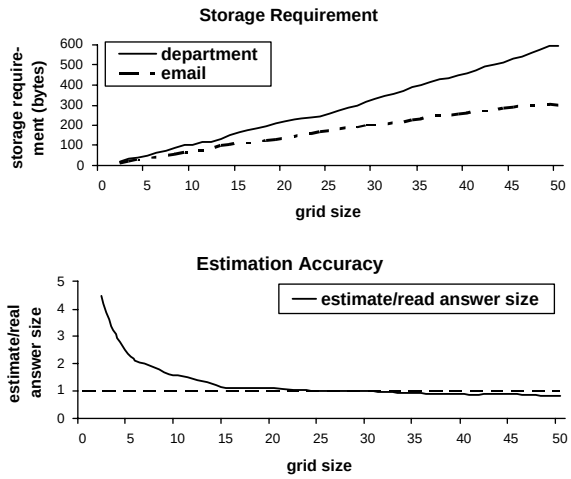


Fig. 16. Storage requirement and estimation accuracy for overlap predicates (department-email).

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 estimation is shown in Fig. 17. Note that the storage requirement for

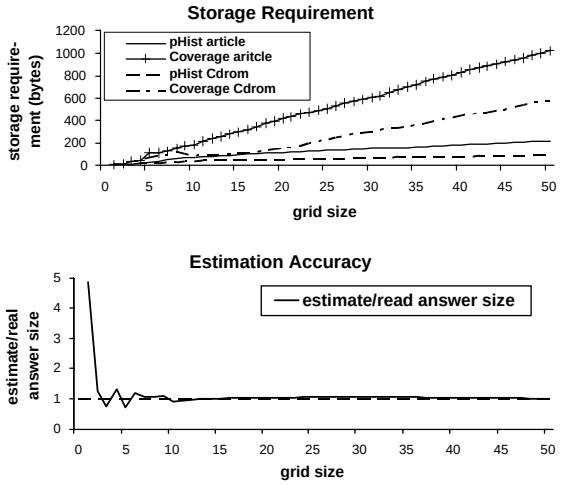


Fig. 17. Storage requirement and estimation accuracy for no-overlap predicates (article-cdrom).

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. Parent–child queries

With the node numbering scheme used so far, ancestor–descendant relationships can be determined, but one cannot be sure if any given pair of nodes have a direct parent–child relationship or have an intervening other node. To be able to answer this question, we introduce a level number with each node. The root is level 0, each of its children is level 1, and so on. Every child of any parent node is at a level exactly one greater than the parent. Thus, we associate a triple, rather than a pair, of values with each node.

For each predicate selected, a separate two-dimensional histogram is created for each level. The number of parent–child relationships is determined by computing the number of ancestor–descendant relationships between a node at level k and a node at level $k+1$, for all possible values of k. Since there is a different histogram for each value of k, this is easy to be accomplished, using exactly the techniques discussed above.

The storage required now could increase by a factor equal to the number of levels in the XML-tree. However, in practice, many predicates are satisfied only (or mostly) at a few levels in the tree, so that the storage blow-up is much smaller. Furthermore, since the number of entries in each level is fewer, one can use coarser grid cells to obtain the same accuracy. As a consequence, the storage required to handle parent–child queries, while larger than for ancestor–descendant queries with the same estimate error, is not larger by a huge factor, as the experiments below demonstrate.

Finally, a couple of observations regarding accuracy. Irrespective of the schema, it is not possible for two nodes at the same level ever to overlap. As such, the non-overlap criteria apply to all predicates in the parent–child case, and must be used to get good estimates. Also, for ancestor–descendant queries, since level numbers are available, they can be used to prune out many candidate nodes. Making an estimate by summation across levels, each paired only with higher (or lower, as the case may be) levels of the other predicate, one can frequently obtain ancestor–descendant estimates superior to those obtained with a single level-unaware structure.

For the level histogram, experiments were run on the synthetic data set only, since the DBLP data set is never more than 5-level deep. The result of the answer size estimation is shown in Table 7, and the storage requirement for the histograms is shown in Table 8. When a parent–child query is asked, without the level histogram, one can only compute the ancestor–descendant estimation. The result of this estimation is shown under the column "A-D Estimation". As shown in this table, with the level histogram we can compute a parent–child estimation more accurately. The extra storage

Table 7
Synthetic data set: result size estimation for simple queries. Histogram size 5×5

Node A	Node B	A-D estimate	P-C estimate	P-C real size
manager	department	1355	15	24
manager	employee	2509	20	31
department	employee	3858	199	442
department	email	1452	73	74
department	name	8137	348	270

Table 8
Storage requirement for leveled histogram and level-wise coverage

Predicate	pHist	LvHist	Cvg
manager	9	30	43
department	8	38	72
employee	6	37	79
name	6	46	87
email	5	25	87

cost, however, is linear (with a factor smaller than the number of levels in the database) to the grid size and level.

7. Estimation using non-uniform histograms

From the experiments on different data set, we observe that nodes satisfy a certain predicate tends to be clustered in a small portion of the XML document. If uniform histograms are built for all predicates selected, (that is, the grid cell boundary settings are the same for all the histogram), the result is that most of the histograms have only a few non-zero-count grid cells (or even one such grid cell).

In relational databases, there is considerable literature on the construction of good histogram in a one-dimensional case. There is even some literature on the construction of good histograms in the multi-dimensional case. While the specifics of these techniques are unimportant for our purpose here, being able to build a non-uniform histogram and estimate answer size using non-uniform histograms (histograms with unequal grid cells) would be helpful to obtain estimates that are closer to the real answer sizes.

7.1. Summary data structure

In a uniform histogram, the scale is defined by the number of grid cells boundaries on both X- and Y-axis. It is much more complex for a non-uniform histogram.

Definition 7.1. The scale of a non-uniform histogram *Hist* is (N_x, Vec_x, N_y, Vec_y), where N_x is the number of grid cell boundaries on the start position (X-axis), and Vec_x is the list of boundary value. N_y and Vec_y represent the same scaling value for end position (Y-axis).

A histogram is a uniform histogram when $N_x = N_y$, and $Vec_x[i] = Vec_y[i]$, for all $0 \leqslant i \leqslant N_x$.

When the two histograms involved in estimating the answer size for a pair join are both uniform histograms, but their scales are not exactly the same, the technique for estimation using non-uniform histogram is still in need.

Example 7.1. The histogram shown in Fig. 18(a) is a non-uniform histogram. The numbers of boundaries on start position and end position are not the same. And the intervals between the boundaries are not uniform. Note that with non-uniform boundaries, the shapes of the off-diagonal grid cells are no longer square, but rectangular, and the shapes of the on-diagonal grid cells are no longer triangular.

7.2. Transformation technique

Given two histograms with different scales, the estimation technique we stated above cannot be applied directly, since the grid cells from different histogram no longer exactly overlap. To be able to estimate the answer size of the pair join, we need to transform both histograms into histograms with same scale. Then, the estimation techniques

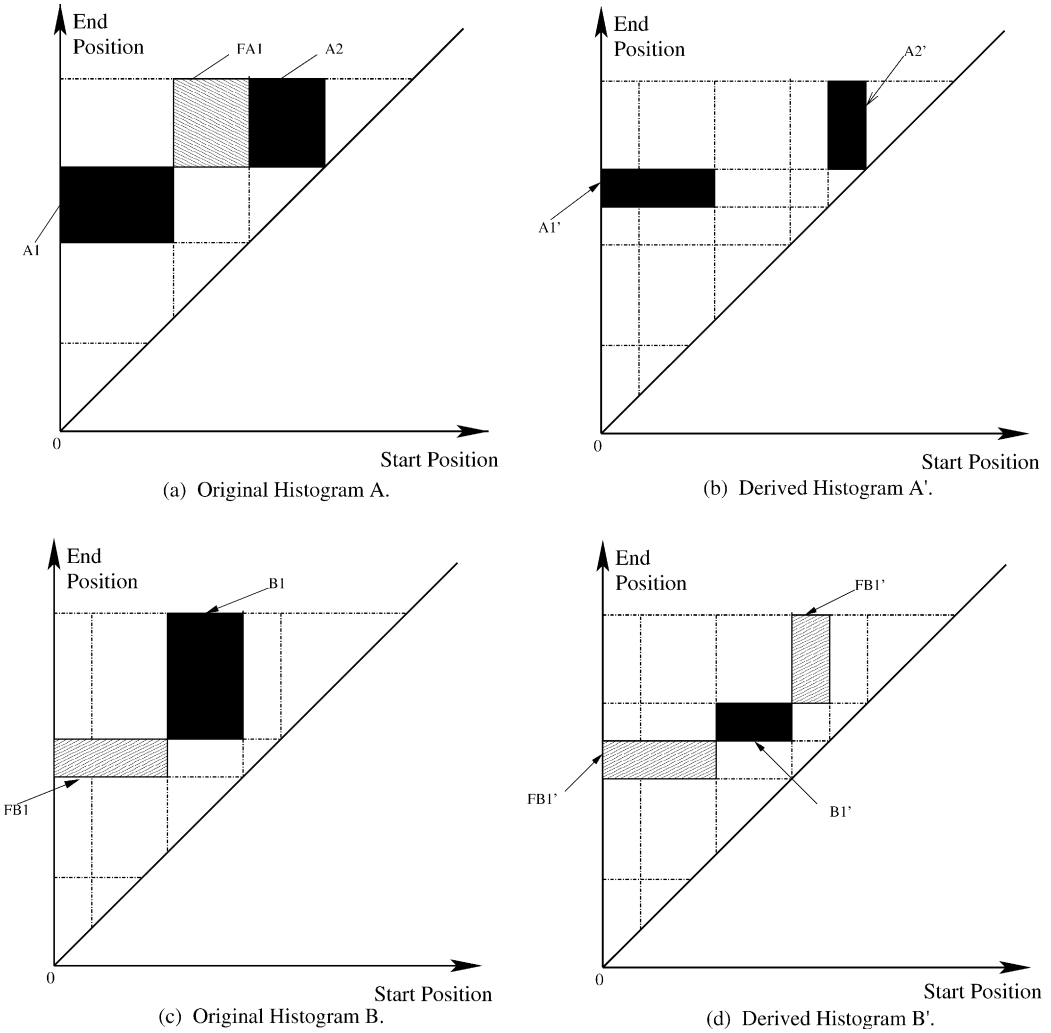


Fig. 18. Transforming non-uniform histograms to uniform histograms: (a) Original histogram A; (b) derived histogram A′; (c) original histogram B; and (d) derived histogram B′.

introduced in the sections above can be used to obtain the answer size estimate.

Definition 7.2. The *intersection scale* (*sc*) of a set of histogram setHist is a vector (N_x, Vec_x, N_y, Vec_y), which satisfies: for each $Hist \in setHist$, $Hist.Vec_x \subset Vec_x \wedge Hist.Vec_y \subset sc.Vec_y$.

Definition 7.3. For a given predicate P, the position histogram associated with the predicate, which is stored as the summary information of the data in the database, is called *original histogram* ($oHist(P)$). A histogram which is equivalent to the original histogram, but in some intersection scale, is called a *derived histogram* ($dHist_{sc}(P)$).

Given a set of original histograms, intersection scale can be found easily by merging their respective scales. Now, the problem is how to find the derived histogram of a given histogram, such that the derived histogram is at the intersection

scale, and is an approximation of the original histogram.

Lemma 3.1 states that appearance of nodes in one grid cell leads to zero count of some other grid cells. Here, we generalize the lemma to a non-uniform scenario.

Theorem 7.1. *In a position histogram for any predicate, with scale* (N_x, Vec_x, N_y, Vec_y)*, a non-zero count in grid cell* (x_1, y_1, x_2, y_2) *implies a zero count in each grid cell* (x'_1, y'_1, x'_2, y'_2) *with* (a) $x'_2 < x_1$, $y'_1 > x_1$ *and* $y'_2 < y_1$, *or* (b) $x'_1 > x_2$, $x'_2 < y_2$ *and* $y'_1 > y_2$.

This theorem can be further generalized to a set of histogram built on the same XML database.

Theorem 7.2. *Given a set of position histograms built on an XML database, a non-zero count in grid cell* (x_1, y_1, x_2, y_2) *implies a zero count in each grid cell* (x'_1, y'_1, x'_2, y'_2) *with* (a) $x'_2 < x_1$, $y'_1 > x_1$ *and* $y'_2 < y_1$, *or* (b) $x'_1 > x_2$, $x'_2 < y_2$ *and* $y'_1 > y_2$, *in all the histograms in the set.*

Given a set of original histogram, The derived histograms of the intersection scale could be found recursively, following the pseudo-code as shown in Fig. 19.

This algorithm takes two histogram with different scale definition, find the minimum-common scale and transform them into derived histogram of that scale. It can be generalized to handle more histograms, just by going through the same process for each histogram in the loop.

The following is a very simple example, which illustrates how this transformation is done using the algorithm in Fig. 19.

Example 7.2. Given Histograms A and B, in different scale definition, as shown in Fig. 18(a) and (c). Assume that there are two grid cells in Histogram A having non-zero count, A_1 and A_2. The grid cell with non-zero count in Histogram B is B_1.

First, the minimum-common scale is computed by combining the scale definition of Histogram A and B. The derived histogram should be in this new scale, as shown in Fig. 18(b) and (d). We map the non-zero grid cells in both original histograms into new scale.

In Histogram A', non-zero grid cell A_1 introduces one forbidden-region $F - A_1$. We map $F - A_1$ to Histogram B' and shrink the non-zero block to B'_1.

In Histogram B', the new non-zero block B'_1 introduce two forbidden regions. Each of these two forbidden regions intersects with one of the non-zero blocks in Histogram A'. Cutting off the zero-count grid cells, the two non-zero blocks in Histogram A' also shrink.

8. Related work

In [10], 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 [11,12], and the notion of set hashing [13,14]. 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 subpatterns, 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 [15] describe Lore's cost-based query optimizer, which maintains statistics

```
Algorithm HistogramTransform(HistA, HistB, HistA', HistB')
// Inputs:  Two original histograms HistA and HistB,
// Output:  Two derived histogram HistA' and HistB'

Combine the scale of HistA and HistB, define intersection scale sc.
Initialize HistA' and HistB' with scale sc.

For each non-zero grid cell in HistA (HistB)
     Mark the set of grid cells in HistA' (HistB') that are covered by the
          grid cell in HistA. (we call such a set a block here.)

while (1)
{
     Bool no_change = true;
     for each none-zero block blk_A in HistA'
     {
          find its forbidden regions F_blk_A;
          for each non-zero block blk_B in HistB'
          {
               find the intersection of blk_B with F_blk_A;
               if the intersection is not empty
                    mark the grid cells in the intersection zero.
               if the range of blk_B can be shrinked
                    shrink the range of blk_B;
                    set no_change = false;
          }
     }
     for each none-zero block blk_B in HistB'
          go through similar process as above.
     if no_change
          break from the loop;
}

for each none-zero block blk_A in HistA'
{
     if blk_A contains more than one grid cell
          divide the node count associated with blk_A to those grid cells
               according to their area.
     otherwise, set the node count of the grid cell to be the node account
               associated with blk_A
}

return HistA', HistB';
```

Fig. 19. Algorithm for transforming two non-uniform histograms to histograms with same scale.

about subpaths of length $\leqslant k$, 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. [16], 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 [10], 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 [15], 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 [5,17–20]. 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.

9. 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 [21] 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 and queries with value-based join operations are also looked into.

References

[1] T. Bray, J. Paoli, C.M. Sperberg-McQueen, Extensible markup language (XML) 1.0. W3C Recommendation, February 1998, Available at http://www.w3.org/TR/1998/REC-xml-19980210.

[2] D. Chamberlin, J. Clark, D. Florescu, J. Robie, J. Simon, M. Stefanescu, XQuery 1.0: an XML query language, W3C Working Draft, June 7, 2001, http://www.w3.org/TR/xquery/.
[3] H.V. Jagadish, L.V.S. Lakshmanan, T. Milo, D. Srivastava, D. Vista, Querying network directories, in: Proceedings of the ACM SIGMOD Conference on Management of Data, Philadelphia, PA, June 1999.
[4] C. Zhang, J.F. Naughton, D.J. DeWitt, Q. Luo, G.M. Lohman, On supporting containment queries in relational database management systems, SIGMOD, 2001.
[5] Y.E. Ioannidis, V. Poosala, Balancing histogram optimality and practicality for query result size estimation, in: SIGMOD Conference, 1995, pp. 233–244.
[6] A.R. Schmidt, F. Waas, M.L. Kersten, D. Florescu, I. Manolescu, M.J. Carey, R. Busse, The XML Benchmark Project, Technical Report INS-R0103, CWI, Amsterdam, The Netherlands, April 2001.
[7] ibiblio Organization, XML dataset for Shakespeare drama, Available at http://sunsite.unc.edu/pub/sun-info/xml/eg/shakespeare.1.10.xml.zip.
[8] DBLP data set, Available at http://www.informatik.uni-trier.de/ley/db/index.html.
[9] IBM, XML generator, Available at http://www.alphaworks.ibm.com/tech/xmlgenerator.
[10] Z. Chen, H.V. Jagadish, F. Korn, N. Koudas, S. Muthukrishnan, R.T. Ng, D. Srivastava, Counting twig matches in a tree, ICDE, 2001.
[11] M. Wang, J.S. Vitter, B. Iyer, Selectivity estimation in the presence of alphanumeric correlations, in: Proceedings of the IEEE International Conference on Data Engineering, 1997, pp. 169–180.
[12] H.V. Jagadish, O. Kapitskaia, R.T. Ng, D. Srivastava, One-dimensional and multi-dimensional substring selectivity estimation, VLDB J. 9 (3) (2000) 214–230.
[13] A. Broder, On the resemblance and containment of documents, IEEE SEQUENCES '97, 1998, pp. 21–29.
[14] Z. Chen, F. Korn, N. Koudas, S. Muthukrishnan, Selectivity estimation for boolean queries, in: Proceedings of the ACM Symposium on Principles of Database Systems, 2000.
[15] J. McHugh, J. Widom, Query optimization for XML, in: Proceedings of the International Conference on Very Large Databases, 1999, pp. 315–326.
[16] A. Aboulnaga, A.R. Alameldeen, J.F. Naughton, Estimating the selectivity of XML path expressions for internet scale applications, VLDB, 2001.
[17] M. Muralikrishna, D.J. DeWitt, Equi-depth histograms for estimating selectivity factors for multi-dimensional queries, in: SIGMOD Conference, 1988, pp. 28–36.
[18] R.J. Lipton, 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.
[19] Y.E. Ioannidis, Universality of serial histograms. in: VLDB, 1993, pp. 256–267.
[20] H.V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K.C. Sevcik, T. Suel, Optimal histograms with quality guarantees, VLDB, 1998, pp. 275–286.
[21] TIMBER Group, TIMBER Project at University of Michigan, Available at http://www.eecs. umich.edu/db/timber/.