Knowl Inf Syst (2008) 16:97–127
DOI 10.1007/s10115-007-0089-z

REGULAR PAPER

A relational model for XML structural joins and their size estimations

Cheng Luo · Zhewei Jiang · Wen-Chi Hou · Feng Yan · Qiang Zhu

Received: 6 June 2006 / Revised: 21 April 2007 / Accepted: 28 April 2007
Published online: 4 July 2007

Abstract XML structural joins, which evaluate the containment (ancestor-descendant) relationships between XML elements, are important operations of XML query processing. Estimating structural join size accurately and quickly is crucial to the success of XML query plan selection and the query optimization. XML structural joins are essentially complex θ-joins, which render well-known estimation techniques for relational equijoins, such as discrete cosine transform, wavelet transform, and sketch, not applicable. In this paper, we model structural joins from a relational point of view and convert the complex θ-joins to equijoins so that those well-known estimation techniques become applicable to structural join size estimation. Theoretical analyses and extensive experiments have been performed on these estimation methods. It is shown that discrete cosine transform requires the least memory and yields the best estimates among the three techniques. Compared with state-of-the-art method IM-DA-Est, discrete cosine transform is much faster, requires less memory, and yields comparable estimates.

Keywords Semi-structured databases · XML databases · Query optimization · Selectivity estimation

1 Introduction

Boosted by the popularity of the Internet and online applications, Extensible Markup Language (XML) has recently become the de facto standard for presenting, storing, and exchanging data on the Internet. Different from the relational paradigm, XML data are

C. Luo (✉) · Z. Jiang · W.-C. Hou · F. Yan
Department of Computer Science, Southern Illinois University Carbondale, Carbondale, IL 62901, USA
e-mail: cluo@cs.siu.edu

Q. Zhu
Department of Computer and Information Science, University of Michigan-Dearborn, Dearborn, MI 48128, USA

semi-structured and usually modeled as trees. Therefore, queries over XML data are usually specified as pattern trees [22] or path expressions [6,9].

There have been considerable efforts devoted to XML query optimization with the objective to select an efficient query execution plan. Usually, the plan selection is based on the cost estimates of alternative plans, in which selectivity estimation plays a major role. Existing approaches focus on estimating selectivity of query patterns at different structural levels. For instance, some estimate the selectivities of the whole path expressions or pattern trees [1,7,12,23], while others study the structural relationships between pairs of node sets. Those methods focusing on path expressions or pattern trees generally rely on some pre-stored statistics to capture the structures of XML documents. For example, path trees and Markov tables [1] were proposed to aid in estimating the selectivities of simple XML path expressions. Freire et al. [12] adopted XML Schema types to gather statistics and used histograms to store the statistics. Polyzotis et al. [23] introduced a graph-synopsis model to provide statistical summaries for large XML data graphs. The model exploits localized graph stability to efficiently partition XML data nodes.

Instead of estimating the selectivities of path expressions or pattern trees, some methods opt to estimate the selectivities of the structural constraints between pairs of nodes in the query patterns. Structural joins that study the structural relationships between pairs of XML nodes have been recognized as vital operations in this regard. Due to the importance of structural join operations, a variety of methods have been proposed. While most of them concentrate on efficient execution of structural join operations[2,18,20,27], few [25,26] address the issue of structural join size estimation, which is nevertheless crucial to the query optimization.

Compared with estimating the selectivities of path expressions or query patterns, it is usually faster, easier, more flexible, and more precise to estimate structural join sizes between pairs of nodes. The estimates of structural joins can serve to locate the performance bottleneck of candidate query plans. Moreover, if necessary, the selectivity of path expressions or query patterns can be derived by multiple structural join size estimates. Based on these arguments, our research focuses on estimating structural join sizes.

Wu et al. [26] proposed a grid model for XML structural join size estimation. The grid model represents the entire XML dataset as a two-dimensional feature space and partitions this space into predefined grid cells. Each grid cell is associated with a count that indicates the number of nodes that fall in it. Wang et al. [25] proposed an interval model and a position model. The interval model represents each ancestor node as an interval and each descendant node as a point. The position model represents the structural information in two tables, covering table and start table. The covering table models the structural information of ancestor nodes while the start table stores the structural information of descendant nodes.

XML structural joins are essentially complex θ joins, which render well-known estimation techniques for relational equijoins, such as discrete cosine transform, wavelet transform, and sketch, not directly applicable to their size estimations [25]. In this paper, we propose an innovative relational model for XML structural joins. Our model captures the structural information of XML data by relations. It converts structural joins to simple equijoins and thus makes those well-known estimation techniques applicable to structural join size estimation. Unlike Wang's IM-DA-Est method [25], which requires extensive search on external index structures, these three methods, namely discrete cosine transform, wavelet transform, and sketch, require only simple computations and little memory space for structural join size estimation. Theoretical analyses and extensive experiments have been performed. The experimental results show that, the estimation method using discrete cosine transform requires the least memory space and generates the best estimates among the three methods. Compared with state-of-the-art method IM-DA-Est [25], discrete cosine transform is much faster, requires less memory, and yields comparable estimates.

The rest of the paper is organized as follows. Section 2 briefly reviews related research in XML structural join size estimation. Section 3 introduces the three approximation techniques, namely discrete cosine transform, wavelet transform and sketch. Section 4 discusses the relational modeling of the structural joins and their estimations utilizing the three approximation techniques. Section 5 compares the three estimation methods and the IM-DA-Est method. Detailed theoretical analyses and experimental results are presented. Finally, Sect. 6 concludes this paper.

2 Preliminaries

To facilitate structural join operations, Wu et al. [26] proposed a region coding scheme, which is similar to the one adopted in the Niagara [27] project. Specifically, the coding scheme assigns a pair of values, *start* and *end*, called the region codes, to each node in the XML data tree. The region codes specify the nodes' locations and coverage. A structural join between an ancestor node a and a descendant node d is essentially to evaluate the logical expression of $a.start \leq d.start$ && $d.end \leq a.end$.

Existing models for structural join include grid, interval and position models [25,26]. The grid model [26] is a two-dimensional model. It represents the entire XML dataset as a two-dimensional feature space and partitions this space into predefined grid cells. Each grid cell represents a range of *start* region codes and a range of *end* region codes. It keeps count of XML nodes that fall in it. The structural join size is estimated by examining the spatial relationships between the grid cells based on the assumption that XML nodes are uniformly distributed in the two-dimensional space. However, such an assumption could lead to poor estimation accuracy especially when the ancestor nodes are not self-nested. The Coverage histogram [26] is thus proposed to remedy this problem by estimating the fraction of coverage. However, the estimation accuracy is still severely impaired by the additional assumption that the local coverage statistics could substitute for the global coverage statistics.

Wang et al. [25] proposed the interval model and the position model. The interval model represents each ancestor node as an interval and each descendant node as a point. For each node with the region codes $(start, end)$, it is represented as an interval of $[start, end]$ when it acts as an ancestor and it is represented as a point with value $start$ when it acts an a descendant. The position model stores the structural information of XML data in two tables, covering table and start table. Both tables have two attributes. The covering table has attributes position and coverage. The coverage attribute indicates the number of ancestor nodes that cover each respective position. The start table has attributes position and start. The start attribute indicates the number of descendant nodes that start at each respective position.

Wang et al. [25] have also proposed two sampling-based methods, IM-DA-Est and PM-Est. Both methods estimate structural join size by first computing the structural join sizes over samples and then scaling up the results proportionally. IM-DA-Est adopts the interval model and samples only the descendant set while PM-Est uses the position model and samples both the ancestor and the descendant sets. IM-DA-Est was shown to yield better estimates than all other methods including those based on the grid model [25]. However, since excessive data accesses are required, the sampling approaches are generally very slow. To expedite structure join size estimation, external index structures, such as XR tree and T^+ tree, are used in IM-DA-Est and PM-Est. However, there is still much room for improvement on the estimation speed.

Fourier transform is a versatile linear transform that decomposes a waveform or function as a sum or integral of sinusoidal functions multiplied by some coefficients (amplitudes). As a variant of the Fourier transform, the discrete cosine transform is known to have excellent energy compaction properties, where most of the signal information tends to be concentrated in a few low-frequency components of the transform [5]. Therefore, the original waveform or function can be approximated, without much information loss, by its first few terms.

Wavelet transform is another special form of mathematical transforms. It decomposes the original signal by applying highpass and lowpass filters repeatedly until a predefined decomposition level is reached. The Haar transform is conceptually the simplest wavelet transform. For the Haar transform, it is proved [13] that the largest coefficients in absolute value carry the most important information of the original signal. Thus the original signal can be compressed using a few coefficients that have large absolute values.

Recently, a randomizing technique, called sketch [3,4], has been proposed. It uses a set of independent randomized linear-projection variables, termed atomic sketches, to estimate the join size. Each atomic sketch has the expected value of the true join size and its variance is relatively small.

Discrete cosine transform, wavelet transform, and sketch have been successfully applied in selectivity estimation. However, it is not clear how these techniques can be applied to structural join size estimation [25]. In this paper, we propose a relational modeling of structural joins to accomplish this task.

3 Technical background

In this section, we briefly discuss the discrete cosine transform, wavelet transform, and the sketch method. While wavelet and sketch have been applied to join size estimation [4,11,19,21], to the best of our knowledge, discrete cosine transform has only been applied to range query size estimation [16]. Therefore, we shall explain in more detail how to apply discrete cosine transform to join size estimation. We attempt to use these techniques to approximate frequency functions of attributes for structural join size estimation.

A relational equi-join is generally performed between attributes with discrete domains. Since a categorical domain can be easily mapped to a numerical domain, we from now on assume all attributes are discrete numerical.

3.1 Discrete cosine transform

The discrete cosine transform is a special form of mathematical transforms that can be used to approximate a signal with only a few coefficients.

3.1.1 Normalization

Let X be an attribute of a relation. We attempt to describe the frequencies of attribute values by a mathematical frequency function. To simplify the notations and implementation of the discrete cosine transform, we opt to normalize the domain of X to a predetermined domain [0, 1]. Let $maxX$ and $minX$ be the maximum and minimum value of X, respectively. Then, each value $x \in X$ is normalized as follows:

$$x^z = \frac{x - minX}{maxX - minX} \tag{1}$$

where x^z denotes the normalized value of x. For example, suppose {0, 1, 2, 3, 4} is the domain of function f. Then the normalized domain is {0, 1/4, 1/2, 3/4, 1}.

3.1.2 Discrete cosine transform

Let N be the total number of tuples in the relation, and n the size of domain X, i.e. $|X| = n$. Let v_i, $1 \leq i \leq n$, be the ith value of attribute X and v_i^z be its normalized value. Let $count_{v_i^z}$ be the number of times that v_i appears in the relation.

The frequency function of X defined on the normalized domain [0, 1], denoted as f, is defined as

$$f(x) = count_x \tag{2}$$

The frequency function has the following property: $\sum_{x=0}^{1} f(x) = N$.

By the theory of discrete cosine transform, $f(x)$ can be represented as

$$f(x) = \sum_{k=0}^{\infty} \alpha_k \phi_k(x)$$

where $\phi_k(x) = N$ for $k = 0$, and $\phi_k(x) = \sqrt{2}\cos k\pi x$ for $k > 0$. α_k is computed as:

$$\alpha_k \approx \sum_{j=1}^{N} \phi_k(t_j^z) = \sum_{i=1}^{n} count_{v_i^z} \phi_k(v_i^z) \tag{3}$$

where t_j^z is the normalized X value of the tuple t_j, $1 \leq j \leq N$.

The Cosine transform is known to have excellent energy compaction properties, where most of the signal information tends to be concentrated in a few low-frequency components of the transform [5]. Therefore, in practice, $f(x)$ is often approximated, without much information loss, by its first few terms as

$$f(x) \approx \sum_{k=0}^{m} \alpha_k \phi_k(x)$$

where m is a small number.

Example Consider a function $f(x)$ defined on the normalized domain [0, 1]. The values of $f(x)$ are: $f(0) = 1$, $f(0.32) = 2$, $f(0.56) = 1$, $f(1) = 1$. The Cosine transform of $f(x)$ is derived as follows. Given that $\phi_k(x)$ is $\sqrt{2}\cos k\pi x$ for $k \geq 1$, α_k is calculated as:

$$\begin{aligned} \alpha_0 &= 5 \\ \alpha_1 &= \sum_{all\ x} f(x)\phi_1(x) \approx 1.2504 \\ \alpha_2 &= \sum_{all\ x} f(x)\phi_2(x) \approx 0.3092 \\ &\cdots\cdots\cdots \end{aligned}$$

Finally, function $f(x)$ is approximated as $f(x) \approx 5 + 1.2504\sqrt{2}\cos \pi x + 0.3092\sqrt{2}\cos 2\pi x + \cdots$.

3.2 Wavelet transform

Wavelet Transform [8] is another special mathematical transform that is able to compress a signal using a few coefficients.

Our application focuses on the Haar wavelet transform, which is conceptually the simplest and has been used in various database applications [19,21]. Haar wavelet transform is usually computed by recursive averaging and differencing, as illustrated below by an example.

Example Consider a series of numbers, listed as {12, 9, 3, 7, 1, 8, 4, 6}. Note that the size of the series is deliberately selected to be a power of 2, which simplifies the computation and analysis. The computation of its Haar transform H_f takes 3 stages.

Stage 1:

$$\begin{aligned} H_{f_1} &= \frac{\{12+9, 3+7, 1+8, 4+6\}}{\sqrt{2}} \\ H_{f_2} &= \frac{\{12-9, 3-7, 1-8, 4-6\}}{\sqrt{2}} \\ H_f &= H_{f_1} \cup H_{f_2} \\ &= \{21, 10, 9, 10, 3, -4, -7, -2\}/\sqrt{2} \end{aligned}$$

Stage 2:

$$\begin{aligned} H_f &= \frac{\{\frac{21+10}{\sqrt{2}}, \frac{9+10}{\sqrt{2}}, \frac{21-10}{\sqrt{2}}, \frac{9-10}{\sqrt{2}}, 3, -4, -7, -2\}}{\sqrt{2}} \\ &= \{\frac{31}{\sqrt{2}}, \frac{19}{\sqrt{2}}, \frac{11}{\sqrt{2}}, \frac{-1}{\sqrt{2}}, 3, -4, -7, -2\}/\sqrt{2} \end{aligned}$$

Stage 3:

$$\begin{aligned} H_f &= \frac{\{\frac{31+19}{(\sqrt{2})^2}, \frac{31-19}{(\sqrt{2})^2}, \frac{11}{\sqrt{2}}, \frac{-1}{\sqrt{2}}, 3, -4, -7, -2\}}{\sqrt{2}} \\ &= \{\frac{50}{(\sqrt{2})^2}, \frac{12}{(\sqrt{2})^2}, \frac{11}{\sqrt{2}}, \frac{-1}{\sqrt{2}}, 3, -4, -7, -2\}/\sqrt{2} \\ &\simeq \{17.68, 4.24, 5.5, -0.5, 2.12, -2.83, -4.95, -1.41\} \end{aligned}$$

To summarize, let the size of the signal be *len*. Then, its Haar transform takes $\log_2 len$ stages to compute. In stage i, only the first $len/2^{i-1}$ elements change their values. The new values are resulted from pair-wise averaging and then differencing of the first $len/2^i$ elements in the previous stage $i-1$. The computation of Haar wavelet is $O(len)$, which is much faster than the computation of Cosine transform whose complexity is $O(m \cdot len)$.

The entries in the Wavelet transform are referred to as Wavelet coefficients. It is proved [13] that for Haar transform, the largest coefficients in absolute value carry the most important information and thus should be selected to represent the original signal. Assume that we are to represent the original series in the above example by three coefficients. Then, they are {17.68, 5.5, −4.95}.

3.3 Sketch

The sketch technique utilizes a set of randomized linear-projection variables to estimate the join size [11]. Consider a join between relations R_1 and R_2 on attribute X and let f_1 and

f_2 be the frequency functions of R_1 and R_2 on X, respectively. Assume $|X| = n$. Then, a randomized linear-projection variable X for this join is computed as follows:

- Generate n four-wise independent variables $\{\xi_i\}$, where each $\xi_i \in \{-1, 1\}$ and Prob[$\xi_i = 1$]=Prob[$\xi_i = -1$] $= 1/2$ (i.e., $E[\xi_i] = 0$).
- $J = (\sum_{i=1}^{n} f_1(v_i) \cdot \xi_i) \times (\sum_{i=1}^{n} f_2(v_i) \cdot \xi_i)$, where v_i is the ith value in the domain of X.

As shown above, each randomized linear-projection variable J is a product of two numbers, namely $\sum_{i=1}^{n} f_1(v_i) \cdot \xi_i$ and $\sum_{i=1}^{n} f_2(v_i) \cdot \xi_i$. These two numbers are termed atomic sketches, whose values only depend upon one relation.

To improve the precision of join size estimation, usually $s_1 \cdot s_2$ J's are used. These randomized linear-projection variables are organized in s_1 groups, of which each has s_2 members. The values of s_1 and s_2 are determined by the desired estimation precision and the desired estimation confidence, respectively.

Example Consider a join on attribute X between relations R_1 and R_2. Suppose the frequency function of R_1 on attribute X is: $f_1(0) = 2$, $f_1(1) = 3$, $f_1(2) = 1$, $f_1(3) = 7$ and the frequency function of R_2 on attribute X is: $f_2(0) = 1$, $f_2(1) = 3$, $f_2(2) = 4$, $f_2(3) = 0$.

To calculate the randomized liner-projection variables, we first generate their corresponding four-wise independent random number series. Assume we intend to use 3 randomized liner-projection variables and the 3 four-wise independent random number series generated are: $fw_1 = \{-1, -1, 1, 1\}$, $fw_2 = \{-1, 1, -1, -1\}$, and $fw_3 = \{1, 1, 1, -1\}$. Then the 3 J's would be calculated as: $J_1 = 3 \times 0 = 0$, $J_2 = (-7) \times (-2) = 14$, and $J_3 = (-1) \times 8 = -8$.

4 Estimating XML structural join size

In this section, we develop a model to capture the structural information of XML data in a relational manner so that an XML structural join, which is essentially a complex θ join, can be modeled as an equi-join. This model makes well known selectivity estimation techniques for relational equi-joins, such as discrete cosine transform, wavelet transform, and sketch, applicable to structural join size estimation.

4.1 Assumptions and definitions

An XML dataset usually consists of multiple XML documents. To ensure a coherent region coding scheme, all documents are first integrated into one single XML document, which is often accomplished by creating a pseudo root tag above all the existing root tags of the XML documents. Hereafter, for simplicity, we shall assume an XML dataset consists of only one document.

An XML document is generally represented by an XML data tree. We assume an XML data tree encompasses all the information of the original XML dataset, with elements, attributes, and text data of the dataset represented by nodes and their relationships by edges in the tree. Consequently, queries on an XML dataset can be specified against its XML data tree.

We use capital letters for node types or node sets, and small letters for individual data nodes. For instance, A represents the type of node or a set of nodes whose tag name is A, while a represents a data node of type A.

The region coding scheme [26] assigns a pair of integer values ($start$, end) to each node in the XML data tree. The root node has the region code (0, n), where n is the smallest

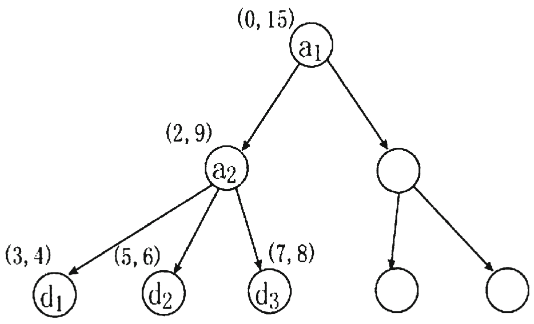


Fig. 1 An XML Data Tree

integer number for the root node to cover all its descendants. The region [0, n] is also termed the coding domain *Dom*. The region coding scheme [26] is able to capture the structural information of XML data elegantly. A node d is a descendant of a node a iff $a.start \leq d.start$ && $d.end \leq a.end$. A structural join between two sets A and D, where A acts as the ancestor set and D the descendant set, is to find all pairs of (a, d) such that $a \in A$, $d \in D$ and a is an ancestor (or parent) of d. We assume the node sets involved in a structural join are distinct.

4.2 A relational model for structural join

Wang et al. [25] use two models to capture the structural information. In their Interval Model (IM), each node is represented by an interval when it acts as an ancestor in the query, and a point when acts as a descendant in the query. In their Position Model (PM), two tables, a covering table and a start table, are used to capture the structural information. In the covering table, each position is associated with the number of nodes covering it, and in the start table, each position is associated with the number of nodes starting at it. In this research, we model the structural information of XML data as relations so that a structural join of XML data can be treated as an equi-join of relations. Our model can be viewed as a semantic extension of Wang 's position model [25].

For each node set T, we model it by two relations, the Coverage and Start-position relations, for its different roles in structural joins. Both relations have two attributes, Sequence number and Position. The Sequence number attribute serves no purpose other than avoiding generating duplicate tuples in the relation. The Position attribute has the coding domain *Dom* as its domain. For each T node with a region code $(start, end)$, it is represented by a set of end–$start$ + 1 tuples whose Position attribute values are $start, start + 1, \ldots, end$ in the Coverage relation for T, and as a single tuple whose Position value is $start$ in the Start-position relation for T. The Coverage relation is used when T acts as an ancestor set in a structural join, while the Start-position relation is used when T acts as a descendant set in a structural join. It should be noted that the Coverage and Start-position relations are introduced only to serve the purpose of modeling the structural join. They are not materialized in any form.

Figure 1 shows an example of an XML data tree, in which two nodes, namely a_1 and a_2, are of type A, and three other nodes, namely d_1, d_2 and d_3, are of type D. Nodes that are of interest also have their region codes labeled close by. For example, node d_2 has region codes (5, 6).

Tables 1, 2, 3 and 4 show how A and D nodes are represented in our relational model. Each node with a region code $(start, end)$ is represented by end–$start$ + 1 tuples in the

Table 1 Coverage relation for *A*

Sequence no.	Position
1	0
2	1
3	2
4	3
5	4
6	5
7	6
8	7
9	8
10	9
11	10
12	11
13	12
14	13
15	14
16	15
17	2
18	3
19	4
20	5
21	6
22	7
23	8
24	9

Table 2 Start-position relation for *A*

Sequence no.	Position
1	0
2	2

Table 3 Coverage relation for *D*

Sequence no.	Position
1	3
2	4
3	5
4	6
5	7
6	8

Coverage relation and a single tuple in the Start-position relation. In the coverage relation, the Position attribute values of these tuples start from $start$ to end. In the Start-position relation, the tuple's Position attribute value is $start$.

Table 4 Start-position relation for *D*

Sequence no.	Position
1	3
2	5
3	7

The structural join between an ancestor set A and a descendant set D is to find all pairs of (a, d) such that $a \in A$, $d \in D$ and a contains d. It can be easily observed that this operation is equivalent to find all pairs of tuples from A's Coverage relation and D's Start-position relation, respectively, such that the two tuples have the same Position attribute value.

Lemma 1 *The structural join between an ancestor set A and a descendant set D is the equi-join between A's Coverage relation and D's Start-position relation on the Position attribute.*

The above reasoning can be easily extended to multi-structural joins that are essentially simple linear path queries. For example, consider a query $//A_1//A_2//D$, where only the deepest node set D acts as the descendant while the other node sets, A_1 and A_2, act as the ancestors. Theorem 1 generalizes this reasoning to multi-structural join size computation.

Theorem 1 *The structural join among distinct ancestor sets $A_1, A_2, \ldots, A_n$ and a descendant set D, is the equi-joins among the Coverage relations of $A_1, A_2, \ldots, A_n$ and D's Start-position relation on the Position attribute.*

In what follows, we shall discuss the computation of structural join size using frequency functions.

4.3 Join size computation

To compute the equi-join size, we need to know the number of occurrences of each position in the relations, that is, the frequency distribution of the Position attribute. Therefore, for each node set, we define two frequency functions, the Coverage and Start-position functions, to describe the frequency distributions of the Position attributes in the Coverage and Start-position relations, respectively.

Definition 1 *The coverage function of a node set T at position $pos \in \mathrm{Dom}$, denoted as $\mathcal{C}_T(pos)$, is the number of T nodes whose region codes cover pos or the number of tuples whose Position attributes values are pos in the Coverage relation of T.*

Definition 2 *The start-position function of a node set T at position $pos \in \mathrm{Dom}$, denoted as $\mathcal{S}_T(pos)$, is the number of T nodes whose start region codes equal pos or the number of tuples whose Position attribute values are pos in the Start-position relation of T.*

Figures 2, 3, 4 and 5 show the corresponding frequency functions for the two node sets A and D in the example XML data tree. For instance, $\mathcal{C}_A(6) = 2$ because both A nodes, namely $a1$ and $a2$, cover position 6.

The coverage function is intended for use when the node set acts as the ancestor in a structural join and the start-position function is used when the node set acts as the descendant. The value of $\mathcal{C}_T(pos)$ can be greater than 1 since T nodes covering pos might be nested;

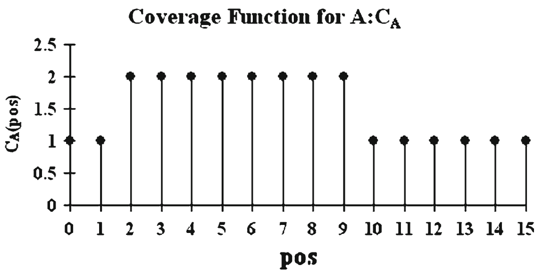


Fig. 2 Coverage function for *A*

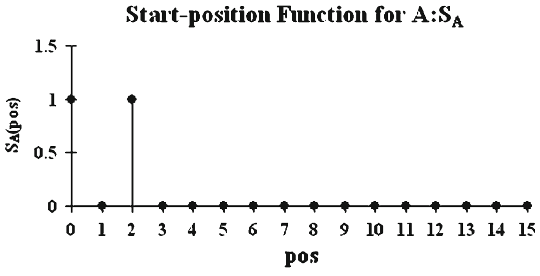


Fig. 3 Start-position function for *A*

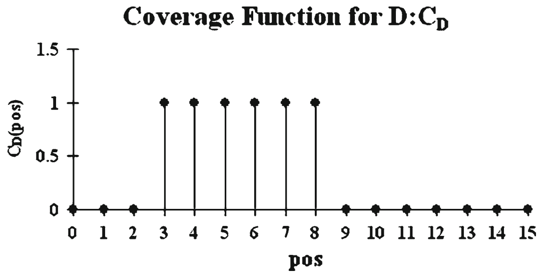


Fig. 4 Coverage function for *D*

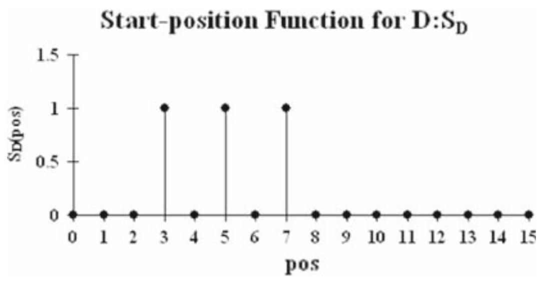


Fig. 5 Start-position function for *D*

the value of $\mathcal{S}_T(pos)$ can only be either 0 or 1 because no nodes can have two identical start region codes.

Thus, the structural join size between an ancestor set A and a descendant set D is the inner product of $\mathcal{C}_A(pos)$ and $\mathcal{S}_D(pos)$, denoted as $\langle \mathcal{C}_A, \mathcal{S}_D \rangle$, over the coding domain, namely:

$$\sum_{pos \in Dom} \mathcal{C}_A(pos) \times \mathcal{S}_D(pos) \tag{4}$$

Moreover, the structural join size among any distinct ancestor sets A_1, $A_2, \ldots,$ A_n and a descendant set D, is the inner product of $\mathcal{C}_{A_1}(pos), \mathcal{C}_{A_2}(pos), \ldots, \mathcal{C}_{A_n}(pos)$ and $\mathcal{S}_D(pos)$, denoted as $\langle \mathcal{C}_{A_1}, \mathcal{C}_{A_2}, \ldots, \mathcal{C}_{A_n}, \mathcal{S}_D \rangle$, over the coding domain, namely: $\sum_{pos \in Dom} \mathcal{C}_{A_1}(pos) \times \cdots \times \mathcal{C}_{A_n}(pos) \times \mathcal{S}_D(pos)$.

4.4 Structural join size estimation

The storage of the frequency functions can consume a lot of space. Therefore, we opt to use three techniques, namely the discrete cosine transform, wavelet transform, and sketch, that need only small amounts of storage space to approximate the frequency functions. In addition, unlike Wang's approaches [25], which could require a considerable number of disk accesses, these join size estimation methods need only simple computations. In this section, we will mainly focus on the discrete cosine transform as the other two methods, namely the wavelet transform and sketch, have been discussed thoroughly in the literature [3,4,8,13].

4.4.1 Estimation via the discrete cosine transform

Recall that before applying the discrete cosine transform to a function, a normalization of the function's domain is performed first. In our case, the coding domain *Dom*, namely [0, *n*], is mapped to the region [0, 1] and each position *pos* in *Dom* is normalized to pos^z by Eq. (1) accordingly. For example, $0^z = 0$ and $n^z = 1$.

We denote the coverage function $\mathcal{C}_A$ on the new domain [0, 1] as $\mathcal{C}'_A$, such that $\mathcal{C}_A(pos) = \mathcal{C}'_A(pos^z)$. Likewise, the start-position function $\mathcal{S}_D$ is redefined on [0, 1] as $\mathcal{S}'_D$ and $\mathcal{S}_D(pos) = \mathcal{S}'_D(pos^z)$.

The two new frequency functions can now be expressed in discrete cosine series as: $\mathcal{C}'_A(pos^z) = \sum_{k=0}^{\infty} a_k \phi_k(pos^z)$ and $\mathcal{S}'_D(pos^z) = \sum_{k=0}^{\infty} b_k \phi_k(pos^z)$ respectively, where a_k and b_k are computed following Eq. 3.

Now the structural join size computation (Eq. (4)) can be rewritten as:

$$\begin{aligned}
&\sum_{pos \in Dom} \mathcal{C}_A(pos) \times \mathcal{S}_D(pos) \\
&= \sum_{pos=0}^{n} \mathcal{C}_A(pos) \times \mathcal{S}_D(pos) \\
&= \sum_{i=0}^{n} count A_i \cdot count D_i
\end{aligned} \tag{5}$$

where $count A_i$ is the number of tuples whose Position values are i in the coverage relation for A and $count D_i$ is the number of tuples whose Position values are i in the start-position relation for D.

On the other hand, by the definition of frequency function, we obtain

$$\langle \mathcal{C}'_A, \mathcal{S}'_D \rangle = \sum_{i=0}^{n} count A_i \cdot count D_i \tag{6}$$

By Parsevel's identity [14], the following equation holds:

$$\langle \mathcal{C}'_A, \mathcal{S}'_D \rangle = \sum_{k=0}^{\infty} a_k \times b_k \tag{7}$$

From Eqs. (5), (6) and (7), the structural join size is computed as $\sum_{k=0}^{\infty} a_k \times b_k$.

The join size can be approximated by using only the first m coefficients of each series as $\sum_{k=0}^{m-1} a_k \times b_k$.

4.4.2 Estimation via wavelet transform

The Haar transforms of the frequency functions can be computed as discussed in Sect. 3. The structural join size between an ancestor node set A and a descendant node set D is equal to the inner product between A's coverage function and D's start-position function, which is also equal to the inner product of their respective Haar transform [19].

The structural join size between an ancestor node set A and a descendant node set D is estimated as follows. Firstly, the m largest coefficients in absolute value of A's and D's Haar transforms are selected respectively. Secondly, those non-selected coefficients are assumed to be 0s. Finally, the estimation is carried out by computing the inner product of the revised coefficients.

4.4.3 Estimation via sketch

Consider a structural join between an ancestor node set A and a descendant node set D. We attempt to use $s_1 \cdot s_2$ randomized linear-projection variables to approximate the structural join size between A and D, where s_1 determines the estimation precision and s_2 determines the estimation confidence. These randomized linear-projection variables are organized in s_1 groups, of which each has s_2 members. The structural join size estimate is the median of the averages of the groups [4].

5 Experimental results

We have implemented four estimation methods, the discrete cosine transform, the Haar wavelet transform, sketch, and IM-DA-Est [25]. IM-DA-Est is included in the comparison because it was shown [25] to outperform other approaches, such as the PH Histogram and Coverage Histogram [26], PL Histogram and PM-Est [25]. In this section, we report the experimental results. All experiments are conducted on a PC with a Pentium 4, 1.6 GHz processor and 256 MB RAM.

The datasets we used include a synthetic XML benchmark dataset XMARK [24] and a real XML database DBLP [10]. For each dataset, we select pairs of node sets and then perform a structural join on each pair. Tables 5 and 6 show the set of queries performed on each dataset.

To investigate how dataset size would affect the estimation accuracy, we used two versions of each dataset. For XMARk, one version corresponds to the scale factor 1 (roughly 100 MB) and the other scale factor 5 (roughly 500 MB). These two XMARK datasets are called X1 and X5 for short. For DBLP, we used an old version of about 120 MB and a current version of about 370 MB. For short, these two versions are called Do and Dc, respectively. Tables 7 and 8 show the selected node sets and their detailed statistics in each dataset.

5.1 Storage space

The discrete cosine transform and sketch derive structural join size estimates directly from the cosine coefficients and atomic sketches, respectively. Only the cosine coefficients or atomic sketches need to be stored for these methods.

Table 5 Queries on XMARK

Query	Ancestor	Descendant
Q1	Closed_auction	Annotation
Q2	Closed_auction	Quantity
Q3	Closed_auction	Seller
Q4	Item	Name
Q5	Item	Quantity
Q6	Person	Name
Q7	Person	Emailaddress
Q8	Open_auction	Itemref
Q9	Open_auction	Quantity
Q10	Open_auction	Seller
Q11	Open_auction	Text
Q12	Open_auction	Type

Table 6 Queries on DBLP

Query	Ancestor	Descendant
Q1	Inproceedings	Author
Q2	Inproceedings	Title
Q3	Inproceedings	Cite
Q4	Article	Title
Q5	Article	Cite
Q6	Incollection	Year

The Haar wavelet transform stores not only the coefficients but also their indexes so that the coefficients can be correctly matched. Therefore, it needs twice as much space as the above two methods for the same number of coefficients or atomic sketches.

As for the IM-DA-Est method, it stores a start-position table for each descendent node set, from which samples are drawn. The table size is in the order of the number of descendent nodes in question. In addition, it also requires space for an external index structure, such as an XR-tree or T-tree [25], to expedite the examination of the structural relationships between sample descendant nodes and ancestor nodes. An XR-tree (or a T-tree) is built for each ancestor node set and requires a space in the order of the size of the ancestor nodes in question.

Let us use an example to illustrate the memory usage of these methods. In the experiments, we use three space budget settings, namely 200, 400, 800, and 1,600 bytes memory for discrete cosine, wavelet, and sketch methods. As for IM-DA-Est, excluding the external index structure, even the smallest table, created for the descendant node set *annotation* in X1, has $21,750$ entries, which amounts to $4 \times 21,750 = 87,000$ bytes of memory consumption. Therefore, the memory consumption of IM-DA-Est is much greater, for instance, about 100 times greater than the three proposed methods in our experiments. In short, the three proposed methods require much less memory space than IM-DA-Est.

Table 7 Statistics for XMARK

Tag name	Node count in X1	Node count in X5
Annotation	21,750	108,750
Closed_auction	9,750	48,750
Emailaddress	25,500	127,500
Item	21,750	108,750
Itemref	21,750	108,750
Name	48,250	241,250
Open_auction	12,000	60,000
Person	25,500	127,500
Quantity	43,500	217,500
Seller	21,750	108,750
Text	105,114	527,147
Type	21,750	108,750

Table 8 Statistics for DBLP

Tag name	Node count in Do	Node count in Dc
Article	213,949	322,755
Author	1,331,301	2,105,709
Cite	172,406	172,401
Incollection	1,535	2,530
Inproceedings	357,848	531,130
Title	581,570	876,759
Year	580,420	866,382

5.2 Estimation error

The relative estimation error is used to determine the accuracy of the estimation. It is defined as $\frac{|x-\hat{x}|}{x} \times 100\%$, where x is the actual join size and $\hat{x}$ is the estimate.

The estimation accuracy depends on the number of samples drawn in IM-DA-Est, and the number of coefficients or atomic sketches used in the other three methods. Unfortunately, it is difficult to compare the estimation accuracy based on the same amount of memory consumption because of the outstanding difference in memory usage between IM-DA-Est and the proposed methods. Recall that the proposed methods store coefficients and atomic sketches, while IM-DA-Est stores tables and external index structures. IM-DA-Est, excluding the external index structures, can consume several orders of magnitude more memory than the other three methods. Therefore, the memory (consumption) constraints used in the experiments, such as 200, 400, 800, and 1,600 bytes apply, strictly speaking, only to the proposed methods. However, to observe the effects of sample size on estimation accuracy, these constraints are also used to specify the size of samples for IM-DA-Est. Readers are advised to be aware of this difference when interpreting the results.

The estimation accuracy is measured based on various memory constraints: 200, 400, 800, and 1,600 bytes. For example, a 200-byte memory constraint would mean 50 sample nodes (4 bytes each), or 50 cosine coefficients, or 50 atomic sketches, or 25 wavelet coefficients, recalling that each wavelet coefficient is stored with its index. Note that the memory constraints indicate only the sample size of IM-DA-Est, but they are all the memory space required by the three proposed methods.

We first compare the estimation accuracy between sketch, discrete cosine transform and wavelet. Then the best of the three is compared with IM-DA-Est.

5.2.1 Comparisons between sketch, discrete cosine transform and wavelet

In general, sketch performs much worse than the other two methods. Therefore, we separate its results from the other two so that the comparisons between the cosine and wavelet methods are clearer. Figure 6 shows the performance of the sketch method on the X1 dataset. The results are the averages of 50 runs. The performance of the sketch method on the other datasets, namely X5, Do, and Dn, is very similar and is thus omitted. Figure 6 has four sub-figures that correspond to the four memory space constraints respectively. The x axis denotes the queries executed and the y axis shows the relative estimation errors.

As shown in Fig. 6, the sketch method performs very poorly in structural join size estimation. Even with the largest 1,600 byte space budget, the average relative estimation errors for all the queries are still around 40%. In addition, the variance of the relative estimation error is also quite large, as demonstrated by the huge performance variation of query 3 under the 800 and 1,600 byte constraints.

Theoretically, join size estimation via sketch has its best case when all the tuples in the join relations have the same and sole join attribute value, while it has its worst case when the attribute values are uniformly distributed. The frequency functions of the datasets in the experiments are more uniformly distributed than being concentrated on a few specific points, which explains the poor performance of the sketch method.

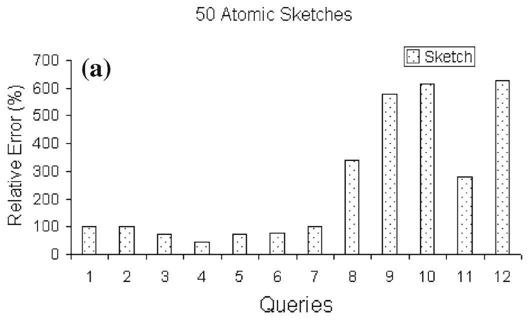
Figures 7 and 8 show the performance of discrete cosine transform and wavelet on the X1 and X5 datasets while Figs. 9 and 10 show their performance on the Do and Dn datasets. Results from both methods are placed side by side for easy comparison.

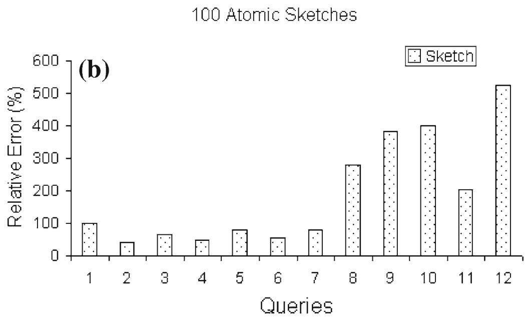
Recall that the Haar wavelet transform consumes twice as much space as the discrete cosine transform for storing the same number of coefficients. Therefore, under the same memory space constraint, the Haar wavelet transform method can use only half the number of coefficients that the discrete cosine transform method uses. The titles of the subfigures indicate the total number of discrete cosine coefficients used for estimation.

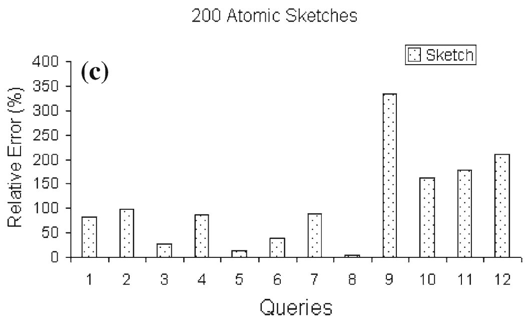
The performance of both the discrete cosine transform and wavelet methods depends on how well the frequency functions are approximated or compressed. In general, the smoother the frequencies, the better the approximation.

As shown in Figs. 7, 8, 9, and 10, overall discrete cosine transform performs better than wavelet on all the four XML datasets. Indeed, the estimation accuracy of the discrete cosine transform method is generally 2–3 times better than that of the wavelet method. Even with the same number of coefficients for both methods, discrete cosine transform is still generally better than wavelet, which can be observed by comparing their performance in neighboring figures. These figures also show that it is the distribution characteristics of the data, instead of the data size, that primarily determines the estimation accuracy of discrete cosine transform. For instance, X5 is about five times as large as X1, but since both datasets have similar data distribution characteristics, the performance of discrete cosine transform on them is very similar.

As discrete cosine transform is shown to have the best estimation accuracy, in the following, it is compared with the IM-DA-Est method. Note that the comparison is based not on the same memory consumption but the same numbers of coefficients and samples.







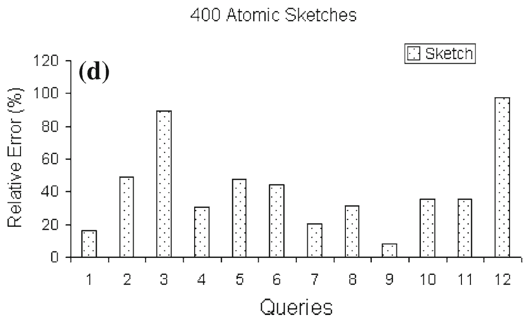


Fig. 6 Sketch on XMARK. **a** Space limit:200 bytes, **b** Space limit:400 bytes, **c** Space limit:800 bytes, **d** Space limit:1,600 bytes,

Fig. 7 Performance on X1. **a** Space limit:200 bytes, **b** Space limit:400 bytes, **c** Space limit:800 bytes, **d** Space limit:1,600 bytes

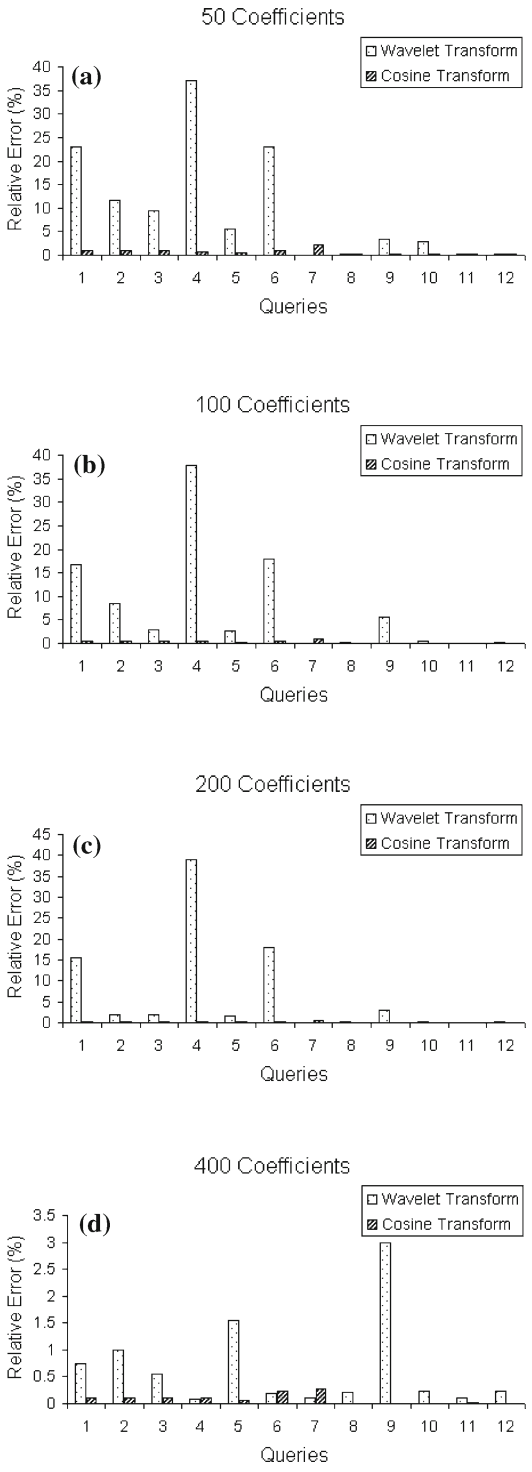


Fig. 8 Performance on X5. **a** Space limit:200 bytes, **b** Space limit:400 bytes, **c** Space limit:800 bytes, **d** Space limit:1,600 bytes

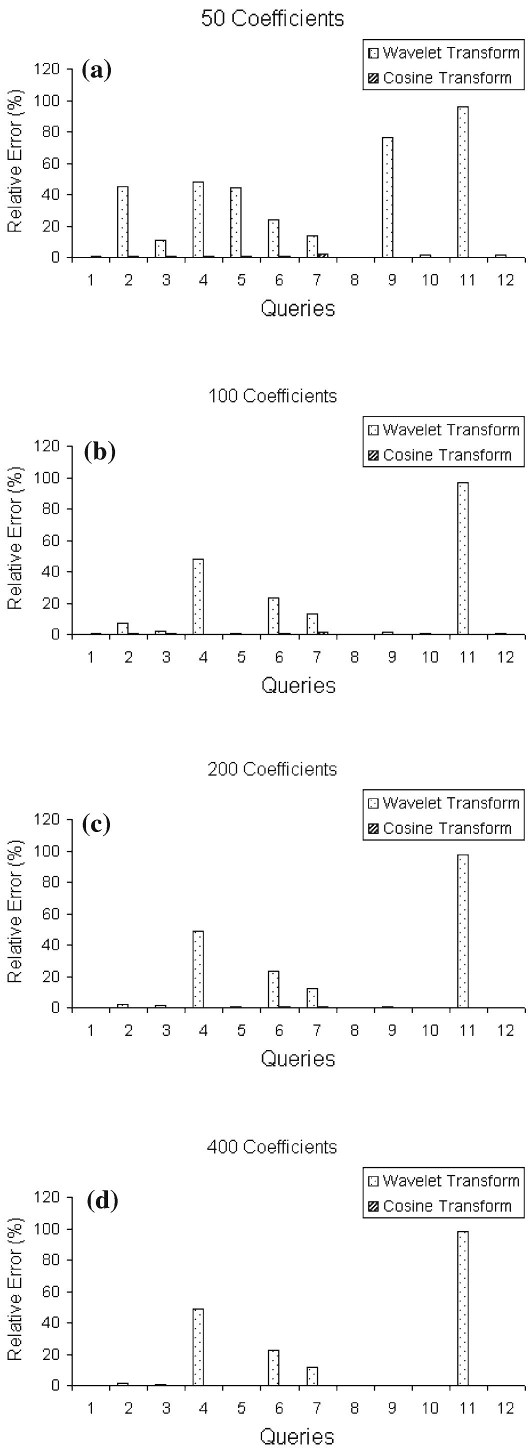


Fig. 9 Performance on Do. **a** Space limit:200 bytes, **b** Space limit:400 bytes, **c** Space limit:800 bytes, **d** Space limit:1,600 bytes

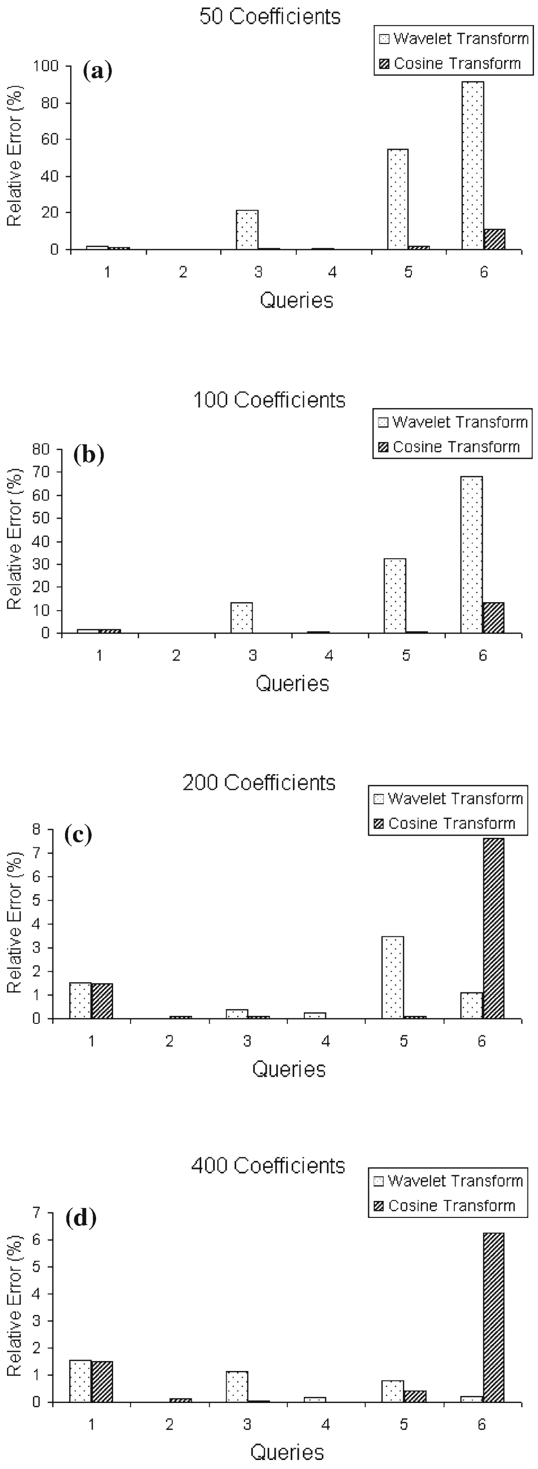
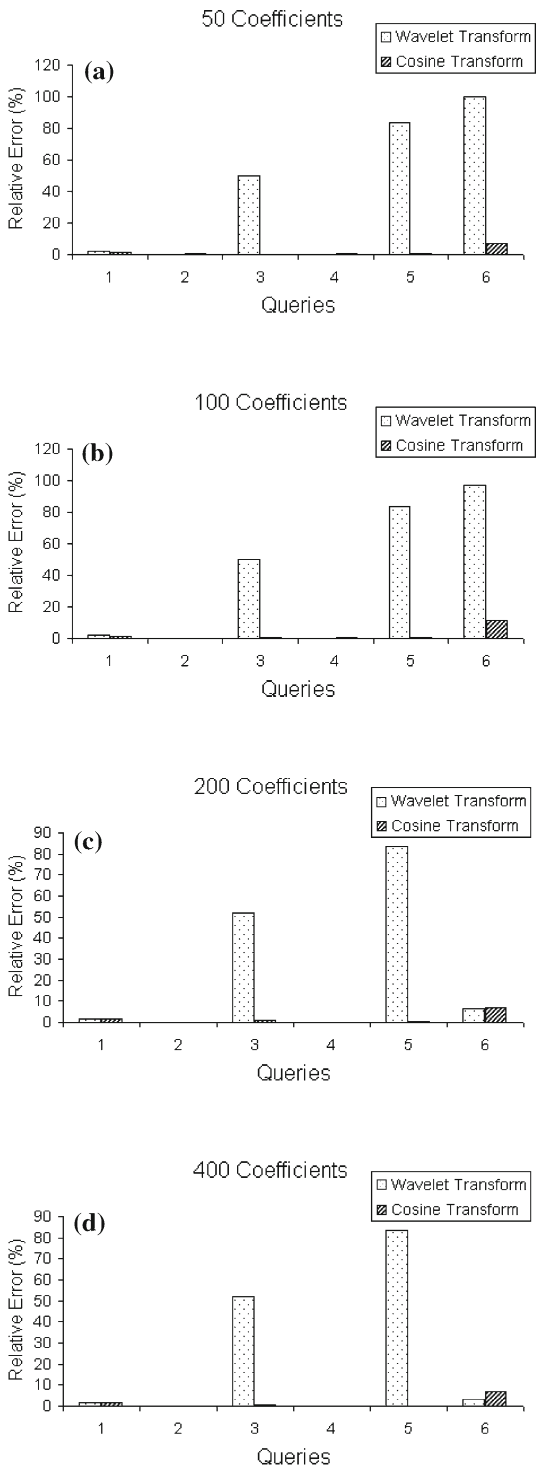


Fig. 10 Performance on Dn. **a** Space limit:200 bytes, **b** Space limit:400 bytes, **c** Space limit:800 bytes, **d** Space limit:1,600 bytes



5.2.2 Comparisons between discrete cosine transform and IM-DA-Est

The IM-DA-Est method works very well when the structural relationships between the ancestor and descendant nodes are regular, that is, when the number of ancestor nodes that cover a descendant node, namely the size of a subjoin, does not vary substantially. In this situation, a small number of sample subjoins would be sufficient to yield a good estimate for the join. On the other hand, if the relationships are irregular, poor estimate might be generated.

The performance of discrete cosine transform depends on how well the frequency functions are approximated. In general, the smoother the frequency functions, the better the approximation. Specifically, for the coverage function, discrete cosine transform approximates better when the nodes on average have larger ranges of coverage. For the start-position function, the more 1's, the denser the distribution, and the better the approximation.

Figures 11 and 12 show the performance of IM-DA-Est and discrete cosine transform on the X1 and X5 datasets. Both methods perform quite well. The good performance is due to the regularity exhibited in the ancestor-descendant relationships for IM-MD-Est and the smoothness of the coverage functions for discrete cosine transform. Indeed, there exist perfect regularities between the ancestors and descendants in query 7, that is, the number of ancestor nodes covering a descendant node is a constant. IM-DA-Est yields error-free estimation in this situation. Discrete cosine transform also yields very accurate estimate, though not error free.

Figures 13 and 14 show the performance on the Do and Dn datasets. Discrete cosine transform yields below or near 10% errors for all queries and outperforms IM-DA-Est for most of the queries.

5.3 Estimator Construction Time

Since IM-DA-Est applies random sampling directly to descendant sets, there is virtually no set-up requirement.

As discussed earlier in Sect. 3, given the region coding domain size $|Dom|$ and the number of coefficients m in an estimator, cosine transform takes $O(|Dom| \cdot m)$ to calculate the first m coefficients. Haar wavelet transform takes $O(|Dom|)$ to calculate all the $|Dom|$ coefficients and $\Theta(|Dom| \cdot m)$ to identify the m most significant ones. However, in practice, since elements are sparsely distributed in the region coding domain, we can do much better. Assume the element under consideration has n_e entries in the XML document, cosine transform only takes $\Theta(n_e \cdot m)$, where $n_e \ll |Dom|$, to calculate the first m coefficients. Besides, the sparse distribution of the element under study implies that the overwhelming majority of its wavelet coefficients are 0's. We thus only need to identify the m most significant coefficients among the non-zero coefficients. Assume the number of non-zero wavelet coefficients is n_{nz}, it takes $O(n_{nz} \cdot m)$, where $n_{nz} \ll |Dom|$, to identify the m most significant ones. Finally, the time complexity to calculate m atomic sketches is $\Theta(|Dom| \cdot m)$. As there is no optimization possible, sketch requires the longest construction time.

Tables 9, 10, and 11 show the construction time for the wavelet coefficients, cosine coefficients, and atomic sketches, respectively. The construction time is categorized by the number of coefficients and dataset. It is the average time to construct the coefficients/atomic sketches for each element. In other words, the time covers the construction of the coefficients/atomic sketches required to approximate both the coverage and start-position functions of an element.

Table 9 shows that wavelet transform takes the least amount of construction time and there is little time difference in constructing estimators with different numbers of coefficients for the same dataset, which is because all wavelet coefficients have to be computed

Fig. 11 Performance on X1. **a** Space limit:200 bytes, **b** Space limit:400 bytes, **c** Space limit:800 bytes, **d** Space limit:1,600 bytes

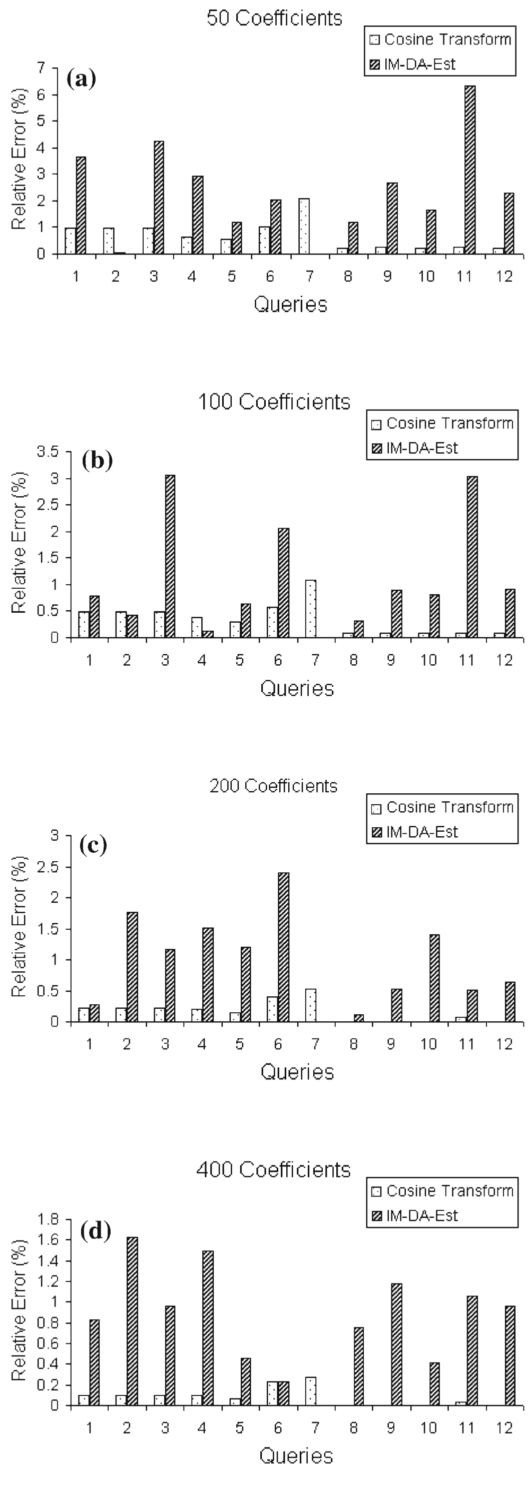
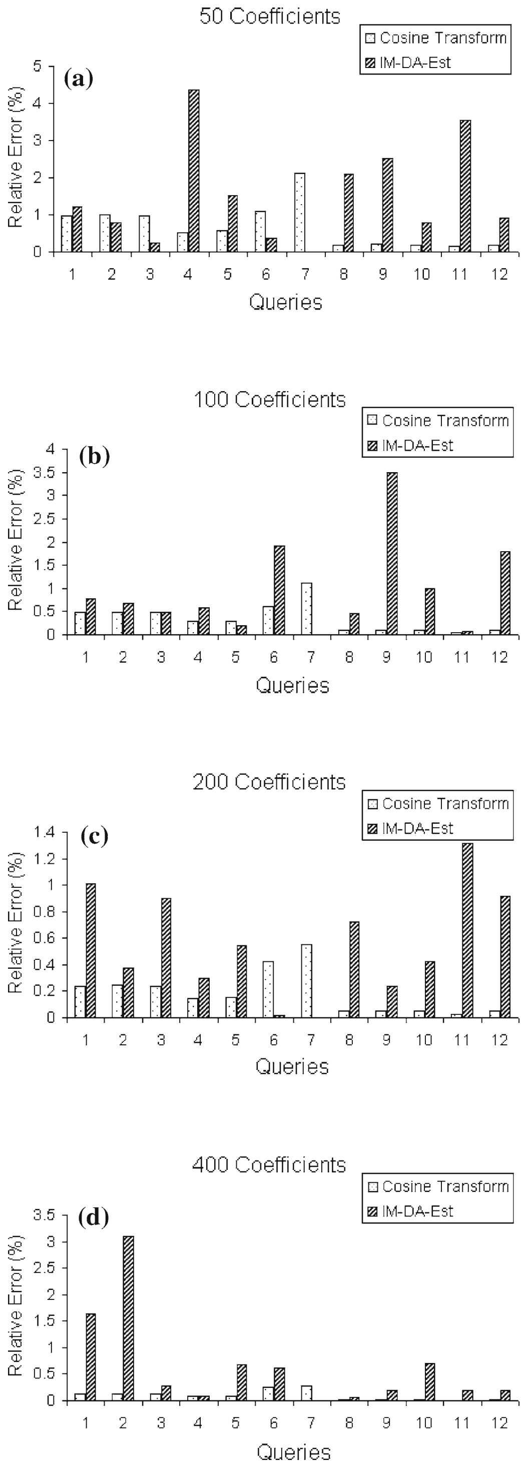


Fig. 12 Performance on X5. **a** Space limit:200 bytes, **b** Space limit:400 bytes, **c** Space limit:800 bytes, **d** Space limit:1,600 bytes



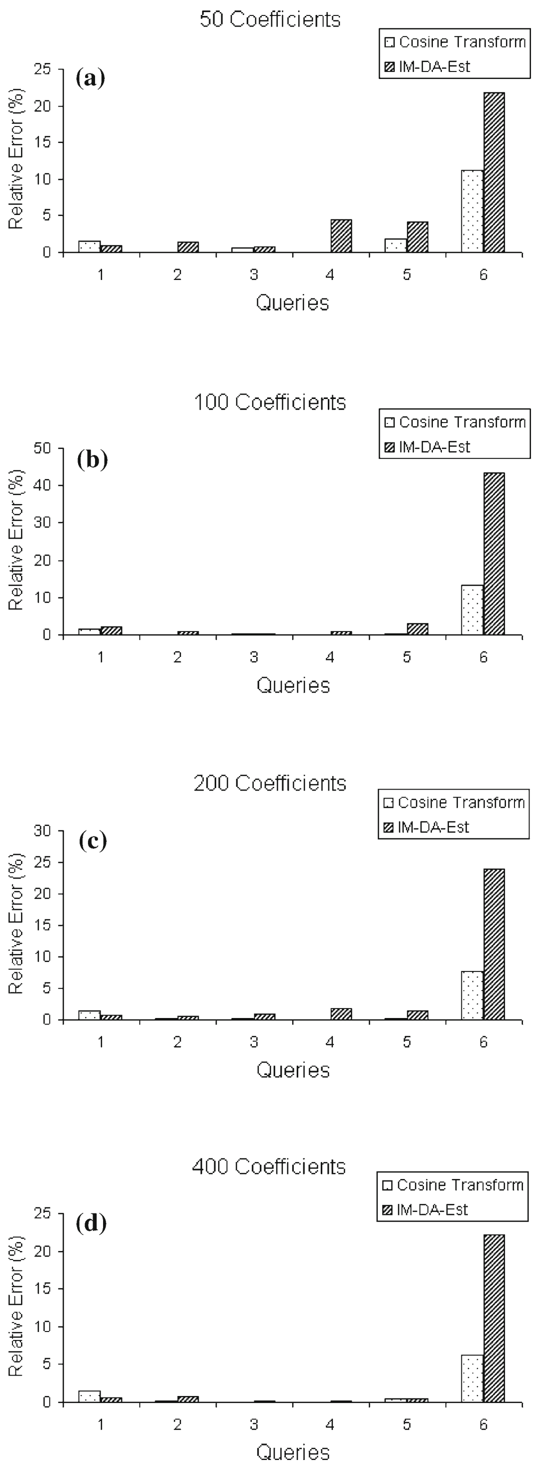
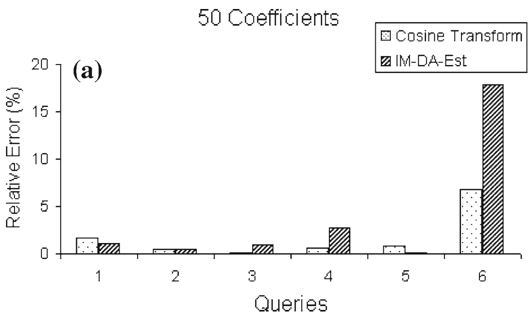
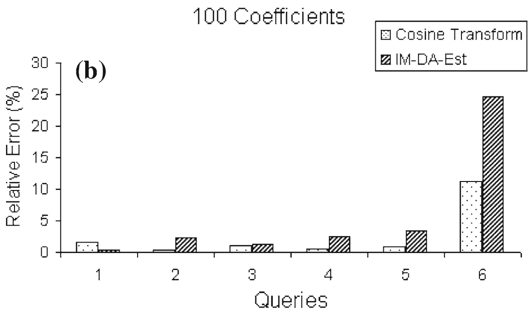
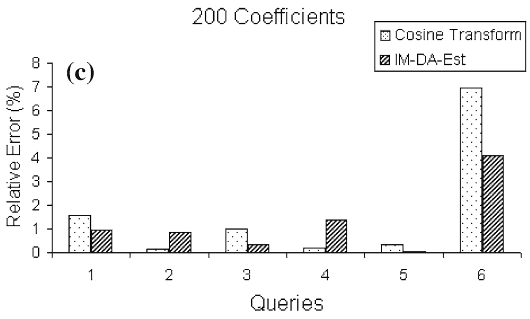


Fig. 13 Performance on Do. **a** Space limit:200 bytes, **b** Space limit:400 bytes, **c** Space limit:800 bytes, **d** Space limit:1,600 bytes

Fig. 14 Performance on Dn. **a** Space limit:200 bytes, **b** Space limit:400 bytes, **c** Space limit:800 bytes, **d** Space limit:1,600 bytes







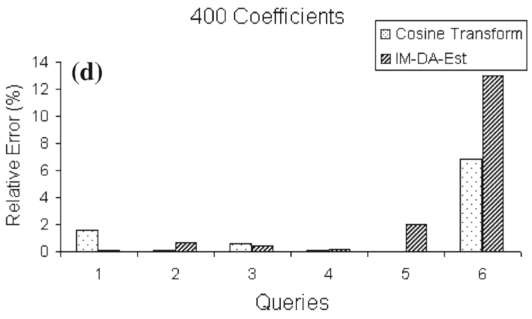


Table 9 Construction time for wavelet coefficients (S)

Coefficients	X1	X5	Do	Dc
200	0.36	1.482	0.386	1.29
400	0.365	1.486	0.39	1.294
800	0.372	1.501	0.393	1.298
1,600	0.377	1.504	0.398	1.303

Table 10 Construction time for cosine coefficients (S)

Coefficients	X1	X5	Do	Dc
200	0.337	1.675	1.674	4.961
400	0.644	3.23	3.261	9.66
800	1.265	6.338	6.414	18.458
1,600	2.505	12.546	12.759	37.743

Table 11 Construction time for atomic sketches (S)

Coefficients	X1	X5	Do	Dc
200	54	231.2	95.47	225.8
400	103.8	452.4	177.94	438.6
800	212.45	912.8	356.88	873.2
1,600	424.7	1817.6	736.76	1760.4

no matter how many of them would be used. Table 10 shows that cosine transform generally takes much longer construction time than wavelet transform. The construction time for cosine coefficients primarily relies on the number of element entries. For instance, although dataset Dc is smaller than X5, it takes longer construction time as its elements on average have larger number of entries. Table 11 shows that sketch takes the longest construction time. The construction time for atomic sketches relies heavily on the region coding domain size $|Dom|$.

It should be pointed out that the construction costs for these estimators are incurred only once. All the three proposed methods can dynamically update their coefficients in the face of data additions and deletions. Dynamic update for wavelet transform is discussed in [19]. For sketch, addition or deletion of an entry is handled by adding/subtracting its mapped value from affected atomic sketches. In the case of cosine transform, its coefficients are computed as the averages of the basis functions on the values. One can add/subtract the portion contributed by the newly added/deleted entries easily [17].

5.4 Estimation time

All the three proposed methods perform simple mathematical calculations for structural join size estimation. For discrete cosine transform and wavelet, the estimation is performed by summing up the products of pairs of coefficients. For sketch, the estimate is derived by selecting the median of the group averages of products of atomic sketch pairs.

Consider using 400 pairs of coefficients or atomic sketches (with $s1 = s2 = 20$) as an example. To derive a join size estimate, it takes less than 80, 80 and 82 μs for discrete cosine transform, wavelet transform, and sketch, respectively. All these three methods can estimate XML structural join size quickly and have a complexity of $O(m)$, where m is the number of coefficients for discrete cosine transform and wavelet, or the number of atomic sketches for sketch.

As for the IM-DA-Est method, it needs to probe an external index structure for each sample to find out the number of covering ancestor nodes, which may require several disk accesses[25]. Indeed, it may take, for example, $O(log_F N + R)$ disk accesses on an XR-tree in the worst case [15], where N is the number of indexed nodes, F is the fanout of the XR-Tree and R is the output size. The modern hard disk access time is about 4–14 ms. Therefore, even if there is only one disk access for each estimation, IM-DA-Est is still significantly slower than the other three methods. Our experiments show that the estimation time of IM-DA-Est varies with the size of the ancestor node set. In general, it takes tens to hundreds of milliseconds to derive an estimate from 100 sample nodes using an XR-tree. For instance, the open_auction ancestor node set in X1 requires about 60 ms to derive the estimate using 100 samples. As part of the query optimizer, the structural join size estimation is expected to be performed quickly and frequently. The three proposed methods certainly demonstrate their superiority in this regard.

In summary, the three proposed methods consume much less memory space and are much faster than IM-DA-Est. As for the estimation accuracy, discrete cosine transform is slightly better than or, at least comparable to, IM-DA-Est.

6 Conclusions

Well-known estimation techniques for relational equijoins, such as discrete cosine transform, wavelet transform, and sketch, have been applied successfully to selectivity estimation for relational data. These methods perform simple calculations and require little memory space. They provide a quick and economical estimation approach. However, no existing models make these techniques applicable to XML structural join size estimation.

In this paper,we propose an innovative relational model for XML structural join size estimation. Our model captures the structural information of XML data by relations. It converts structural joins, which are essentially complex θ-joins, to simple equijoins and makes those well-known estimation techniques applicable to structural join size estimation.

Theoretical analyses and extensive experiments have been performed. The experimental results show that, the three proposed methods consume much less memory space and are much faster than IM-DA-Est [25]. As for the estimation accuracy, discrete cosine transform is slightly better than or, at least comparable to, IM-DA-Est.

References

1. Aboulnaga A, Alameldeen A, Naughton J (2001) Estimating the selectivity of XML path expressions for internet scale applications. In: Proceedings of 27th international conference on very large data bases, pp 591–600
2. Al-Khalifa S, Jagadish V, Koudas N, Patel M, Srivastava D, Wu Y (2002) Structural joins: a primitive for efficient XML query pattern matching. ICDE, pp 141–152
3. Alon N, Matias Y, Szegedy M (1996) The space complexity of approximating the frequency moments. In: Proceedings of the 28th annual ACM symposium on theory of computing, pp 20–29

4. Alon N, Gibons P, Matias Y, Szegedy M (1999) Tracking join and self-join sizes in limited storage. In: Proceedings of the 18th ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems, pp 10–20
5. Briggs L, Henson E (1995) DFT: an owner's manual for the discrete Fourier transform. Philadelphia Society for Industrial and Applied Mathematics
6. Chamberlin D, Florescu D, Robie J, Simeon J, Stefanescu M (2004) XQuery 1.0: an XML query language. W3C Working Draft http://www.w3.org/TR/xquery/
7. Chen Z, Jagadish V, Korn F, Koudas N, Muthukrishnan S, Ng T, Srivastava D (2001) Counting twig matches in a tree. In: Proceedings of the 17th International Conference on Data Engineering, pp 595–604
8. Chui C (1992) An introduction to wavelets. Academic, New York
9. Clark J, DeRose S (1999) XML path language (XPath). W3C Working Draft http://www.w3.org/TR/xpath
10. Dobra A, Garofalakis M, Gchrkc J, Rastogi R (2002) Processing complex aggregate queries over data stream. ACM-SIGMOD, Madison, pp 61–72
11. Freire J, Haritsa R, Ramanath M, Roy P, Siméon J (2002) Statix: making XML count. In: Proceedings of the 2002 ACM SIGMOD international conference on management of data, pp 181–191
12. Gilbert A, Kotidis Y, Muthukrishnan S, Strauss M (2001) Surfing wavelets on streams: one-pass summaries for approximate aggregate queries. In: Proceedings of the 27th international conferrence on VLDB, pp 79–88
13. Issacson E, Keller B (1994) Analysis of numerical methods theorem 3. Dover Publications, New York 238
14. Jiang H, Lu H, Wang W, Ooi B (2003) XR-Tree: indexing XML data for efficient structural join. In: Proceedings of ICDE, India, pp 253–264
15. Jiang Z, Luo C, Hou W-C, Yan F, Zhu Q, Wang C-F (2007) Join size estimation over data streams using cosine series. Int J Inf Technol 12(9): 27–45
16. Lee J, Kim, Chung C (1999) Multi-dimensional selectivity estimation using compressed histogram information. In: Proceedings ACM SIGMOD conference, pp 205–214
17. Ley M (2002) The dblp computer science bibliography: Evolution, research issues, perspectives. In: SPIRE 2002, Lisbon, Portugal, September 11–12, 2002. Springer, Heidelberg, pp 1–10
18. Li Q, Moon B (2001) Indexing and querying XML data for regular path expressions. VLDB, pp 361–370
19. Matias Y, Vitter J, Wang M (1998) Wavelet-based histograms for selectivity estimation. SIGMOD
20. McHugh J, Widom J (1999) Optimizing branching path expressions. VLDB, pp 315–326
21. Nievergelt Y (1999) Wavelets made easy. Birkhauser, Basel
22. Paparizos S, Al-Khalifa S, Chapman A, Jagadish V, Lakshmanan S, Nierman A, Patel M, Srivastava D, Wiwatwattana N, Wu Y, Yu C (2002) TIMBER: a native system for querying XML. VLDB J 11(4): 274–291
23. Polyzotis N, Garofalakis N (2002) Statistical synopses for graph-structured XML databases. In: Proceedings of the 2002 ACM SIGMOD international conference on management of data, pp 358–369
24. Schmidt A, Waas F, Kersten M, Florescu D, Manolescu L, Carey J, Busse R (2001) The XML benchmark project. Technical report CWI
25. Wang W, Jiang H, Lu H, Yu X (2003) Containment join size estimation: models and methods. In: Proceedings of the 2003 ACM SIGMOD international conference on management of data, pp 145–156
26. Wu Y, Patel M, Jagadish V (2002) Estimating answer sizes for xml queries. In: 8th International conference on extending database technology, pp 590–608
27. Zhang C, Naughton F, DeWitt J, Luo Q, Lohman M (2001) On supporting containment queries in relational database management systems. SIGMOD

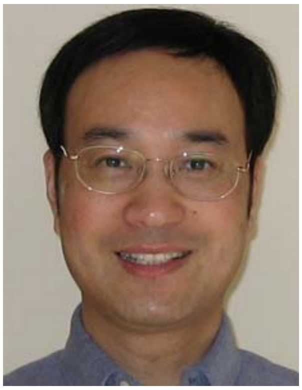
Authors Biography

Cheng Luo received a B.E. degree from Southwest Jiaotong University, Chengdu, China, in 1998 and two M.S. degrees from Southern Illinois University, Carbondale, USA, in 2001 and 2004, respectively. He is currently a Ph.D student in the Computer Science Department at Southern Illinois University, Carbondale, IL, USA. His interests are in databases and data mining.

Zhewei Jiang received her B.S. and M.S. degrees from Southwest Jiaotong University, Chengdu, China, in 1998 and 2001, respectively. She is currently a Ph.D student in the Computer Science Department at Southern Illinois University, Carbondale, IL, USA. Her interests are in databases and data mining.

Wen-Chi Hou received his M.S. and Ph.D. degrees in computer science and engineering from Case Western Reserve University, Cleveland Ohio, in 1985 and 1989, respectively. He is presently an associated professor of computer science at Southern Illinois University at Carbondale. His interests include statistical databases, mobile databases, XML databases, and data streams.

Feng Yan received his M.S. in computer science and Ph.D. in Mathematics from Southern Illinois University at Carbondale in 1999. He is currently a senior quantitative analyst at FPL Energy. His interests are in statistical databases, data mining, optimization and stochastic calculus and its applications.



Qiang Zhu received his Ph.D. degree in computer science at the University of Waterloo, Canada, in 1995. He is currently an associate professor of computer and information science at The University of Michigan, Dearborn, MI, USA. He is also an IBM CAS Faculty Fellow at the IBM Toronto Laboratory. His research interests include database query optimization, data streams, multidimensional indexing, self-managing database systems, data mining, and Web data management.