Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/080452

Cost-based Query Optimization for XPath

Dong Li[1,*], *Wenhao Chen*[1], *Xiaochong Liang*[1], *Jida Guan*[1], *Yang Xu*[1] *and Xiuyu Lu*[2]

[1] School of Software Engineering, South China University of Technology, Guangzhou 510006, P. R. China
[2] School of Computer Science and Technology, South China University of Technology, Guangzhou 510006, P. R. China

Received: 21 Aug. 2013, Revised: 23 Nov. 2013, Accepted: 24 Nov. 2013
Published online: 1 Jul. 2014

Abstract: In this article, we address the issues which are related to the cost-based XML query optimization for XPath. Specially, we focus on the issue of how to determine the execution order for a given XPath expression according to the cost models. The main impact factor that dominates the execution order is the size of the intermediate results during the evaluation of queries. The hierarchy encoding scheme and the value-encoding histogram are introduced to support the size estimation of the path expressions. Two cost models are proposed to describe the costs of different types of join operations in the path expressions. A heuristic-based dynamic programming approach is proposed to determine the optimal execution tree. The primary experimental results demonstrate the validity of our approaches.

Keywords: Query Optimization, XPath, Hierarchy Encoding Scheme, Histogram, Dynamic Programming

1 Introduction

XML (Extensible Markup Language) [1] has become the de facto standard for information or data representation and exchange on the Internet. More and more data is expressed in XML, which is stored in XML-enabled databases, e.g. Oracle, DB2, SQL Server, or in native XML databases [2]. The optimization of XML queries, a key issue that every XML DBMS is facing, is an extreme challenge in the context of XML databases. The main reason is that the semi-structure property of XML data allowing an irregular or changing organization makes the data management be of high complexity. This nature of semi-structure requires more efficient XML query processing and optimization.

1.1 Motivation

The irregularity of the XML data makes the statistical data collection as a big challenge for the XML optimization. In XML documents, the node with the same tag in different parts of the same document may have a different meaning or semantics. For example, considering the sample XML document (named Xmark.xml) tree in Fig.1 (conforming to the XMark [11]), the node *name* can be of different meanings under the node *person* or under the node *item*. In addition, nodes with the same names may have different numbers of children, e.g. the node *people* can have one or more *person*.The XML schemes, including DTD and XML schema, determine what kinds of sub-elements or attributes can happen under certain elements and their cardinalities of sub-elements and so on.

The order of XML data is another significant characteristic difference from the relational data, which limits the flexibility of the transformation rules. To cope with the order characteristics, path expressions become the core part of XML query languages, such as XPath and XQuery. The evaluation of the path expressions can be carried out from the left to the right or from the right to the left or in a hybrid way of them. For example, considering the path expression *"/site/people/person/name"* in Fig.1, to get the query results we can travel from the node site to the node name or from the node name to the node site, or we can evaluate the path using the order of *"/site/(people/person)/name"*. By applying this kind of associative laws, we can get many different alternative execution (evaluation) plans. We believe that it can reduce the total evaluation cost by appropriately applying the associative laws to regulate the execution order of the path expressions.

To more accurately estimate the cost of queries, more statistical information is needed. Because of the

* Corresponding author e-mail: cslidong@scut.edu.cn

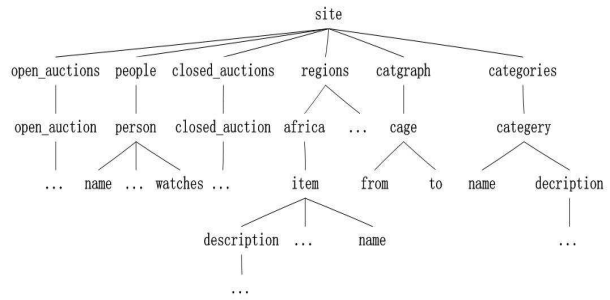


Fig. 1: Xmark.xml

complexity of the XML data model, the traditional cost estimation methods are not practical, thus we should find out new methods to deal with the cost estimation of XML queries in which the path expression selectivity is the core issue. Selectivity analysis on complex path expressions is to estimate the size of the query results. Minimizing the temporary results can reduce the overall cost of the query. If the different sizes of different objects are ignored (e.g. using objects' IDs for join operation), the estimation of the cost can be reduced to measure the operation times of the joining (or matching) the ascendant elements with the descendant elements. Correspondingly, finding the optimal execution plan is to find which execution plan has the minimal join times. Inspired by the other previous work, we use the dynamic programming method to select the best execution plan for XML queries.

1.2 Scope of the Article

In this article, we address the issues related to the cost-based XML query optimization.

Cost-based XML query optimization is to utilize the cost model and statistics to evaluate the costs of different execution plans, and to select the least costly one to execute. The main impact factor that dominates the join order is the size of the intermediate results during the evaluation of queries. To this end, we adopted a hierarchy encoding scheme along with a method of value-encoding histogram to estimate the size of the results of the expressions including the predicates, e.g., A[B>2], furthermore, the design of the structural join order selection algorithm, which is the kernel of an XML query optimizer , is another focus of this article.

The techniques used in most existing native system (such as Lore, TIMBER, Natix and ToX) for evaluating path expressions can roughly be classified into two categories [26]: the structural joins and the holistic algorithms. Our approach belongs to the structural joins family.

For the experimental study of our approaches, we adopted the in-memory version of the XSQS [12] as the host of our test-bed, the XMark [11] as the test database, and conducted a series of experiments. The major contributions of our work are summarized below:

- Based on the hierarchy encoding scheme and its corresponding hierarchy encoding counting table, we introduce a value-encoding histogram to support the size estimation of the path expressions including the predicates.
- Different cost models are proposed to describe the execution costs of the path expressions according to the join types.
- We apply a dynamic programming based on the heuristic rules to determine the join order with the above techniques and the cost models. We demonstrate the validity of our approach via the experimental evaluation.

1.3 Organization of the Article

The remainder of this article is organized as follows. Section 2 surveys the related work in this area. Section 3 presents preliminaries about the hierarchy encoding scheme. Section 4 describes the value-encoding histogram. Section 5 discusses the cost models and the heuristic-based dynamic programming method for cost estimation. Section 6 provides the performance study of query optimization and Section 7 concludes the article.

2 RELATED WORK

XML query optimization which plays an important role in XML database systems [2] can be implemented at the logical level or at the physical level, or at both levels. At the logical level, the XML query optimization is to utilize the XML Schema, and to apply the rules of normalization or simplification to the inner query expressions so as to simplify the expressions that can improve the query performance. Lots of researches focusing on logical optimization have been done [3,4,5,6,7,9,10]. Our work focuses on the cost-based query optimization at the physical level and deals with how to determine the join order in query trees according to the cost estimation.

Although numerous articles on XML query processing have been published, only a few have addressed the cost-based optimization of XML queries. Most of those adapt or extend the relational optimization techniques. The major native XML DBMSs employ the cost-based optimization such as Lore, Niagara, TIMBER, Natix and ToX. Some XML-enable DBMSs, such as DB2 XML, use the cost-based optimization approach. Lore [15] is the first DBMS with the cost-based optimization originally designed for the semi-structured data and later migrated to the XML-based data model. The Lore optimizer is cost-based and does not perform logical-level optimization. In [15], the cost model does not take into account the data clustering but does make an assumption

that each I/O operation gets only one object. So estimating the cost I/O can be converted to estimate the size of the intermediate query results. Instead of estimating the size of the intermediate results, we directly take the number of executing join operations as the main factor of the cost models. Another aspect of modeling cost for XML query optimization is described in [8] which proposes a CPU cost model with a statistical learning technique to predict the overall cost. The occurring of the phenomenon of the high fraction of CPU cost in XML query processing is also the main reason why in this article we focus our work on optimizing the execution order of the path expressions by evaluating the costs of the join operations. In [26], DB2XML adapts the holistic algorithms approach to support XML query optimization in a relational-XML hybid way.

In [17], Wu et al proposes five algorithms based on dynamic programming for the structural join order optimization for XML tree pattern matching. This work models the costs of physical operations to optimize the XML queries. What differs from [17] is that we utilize much more order-related constraints (e.g., the associated law is proposed to constrain the set of the impossible join orders, thus narrowing down the search space for the enumeration algorithm) for optimization and we use different cost models for the join operations, different dynamic programming method for the join order selection and we take advantage of the histogram and the hierarchy encoding technique to estimate the size of XML query results. The idea of using the dynamic programming method for the query optimization is not new [28] and some novel dynamic programming methods [29,30] are proposed in the relational context. We adapt a new heuristic?Cbased dynamic programming method in the XML context, which is different from these researches.

The main factor dominating the order of the operations is the size of the intermediate query results. The selectivity analysis on complex path expressions is a technique for estimating the size of the result set. The selectivity estimation of Lore is based on the DataGuide path index together with stored statistics to provide the structure knowledge about the XML data for optimization. This idea is later extended by pruned count-suffix tree (PST) based on correlated subpath trees in [19], and Markov model based on XSketch in [18]. In [13], a method based on position histogram is proposed to collect the distribution information about the ancestor-descendant. This method does not consider the selectivity estimation of the path with a predicate condition. In [14], a PL histogram of one dimension is built and used to evaluate the selectivity. This method only considers the node with predicates and skips the other nodes when evaluating the selectivity. Other work on XML cardinality estimation includes StatiX in [2], Bloom histograms in [20] and CXHist in [21]. These cardinality estimation techniques differ in whether they are online or offline algorithms, whether they handle subtree queries or linear path queries only, whether they handle leaf values, or whether the leaf values are assumed to be strings or numbers. Our approach explores many other aspects of applying the structure knowledge of XML data by the hierarchy encoding scheme and the related statistics management method for query optimization elaborated in this article. This is where our approach is distinguished from these previous researches.

Finally, with regard to the encoding schema for XML data, Dewey encoding [22] and its variations (e.g. [23], [24]) are worth mentioning. In [22], each node label is a combination of its parent label and an integer number indicating its local order among its siblings. Ordpaths [24] modifies the original Deway encoding method so that it becomes insert-friendly. Some other typical work on labeling schema includes Extended Dewey Encoding [23] which exploits some schema information for labeling XML tree. Our research is different from the previous work in that we adopt the suffix encoding of the binary sequence called the hierarchy encoding scheme for labeling the XML tree. This method is easy to judge the AD and the PC relationship between nodes.

To the best of our knowledge, our work uses a new hierarchy encoding scheme with the corresponding value-encoding histogram, new cost models for the join operations and a new heuristic-based dynamic programming method to achieve the cost-based XML query optimization.

3 PRELIMINARIES

Hierarchy encoding scheme is a kind of width-first encoding scheme whose definition is as follows (more details can be seen in [16]):

- If $N_{i,level}$ is the root node (level=0) of an XML document tree, its hierarchy encoding ID is 0, i.e , Hid($N_{i,level}$)=Hid(Root)=0.
- If N_i,level is an element on level one(level=1) in an XML document tree, its hierarchy encoding ID is a binary sequence according to its order among all its siblings. That is, if $N_{i,level}$ is the i^{th} (starting from zero) distinct element on level one in the XML document tree, then $(N_{i,level})'$s hierarchy encoding ID is a binary sequence $S_{i,1}$ whose i^{th} bit ,from the right side to the left side, of the binary sequence is set one, while all other bits are set zeros. So we get Hid ($N_{i,level}$) =Hid ($N_{i,1}$) = $S_{i,1}$.
- If $N_{i,level}$ is an element with the level greater than one(level>1) in an XML document tree, its hierarchy encoding ID is a binary sequence made up of two parts, $S_{i,level}$ and $S_{j,level-1}$ (i.e., Hid($N_{i,level}$)= $S_{i,level} \cdot S_{j,level-1}$) , where $S_{j,level-1}$ is the encoding ID of $(N_{i,level})'$s parent and $S_{i,level}$ is the binary sequence representation for i which is the i^{th} child among all the siblings in the same level.

As an example, a part of the hierarchy encoding scheme for Xmark.xml is shown in Fig.2.

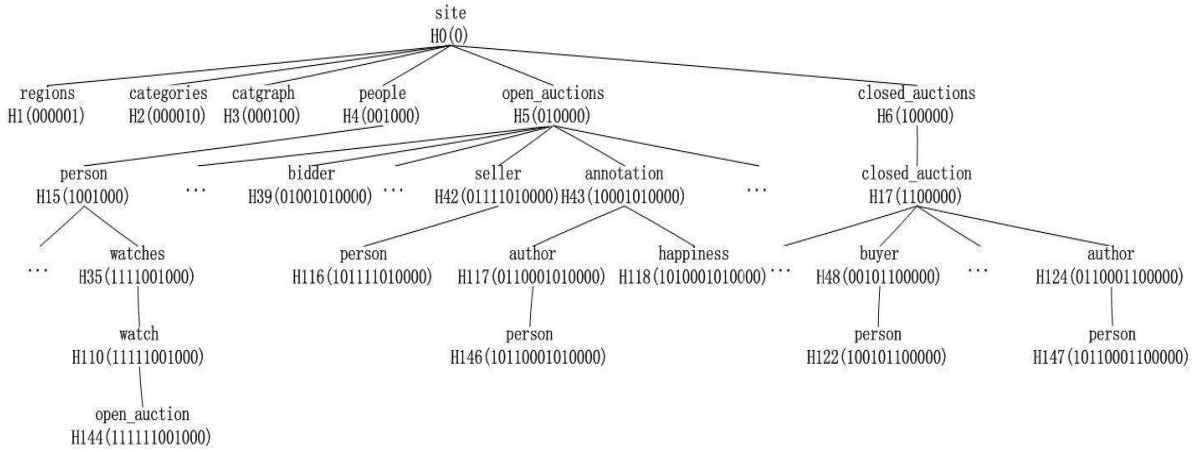


Fig. 2: Hierarchy encoding for the sample document

In order to judge whether two elements are of AD or PC relationship, we extend the encoding information to a five-ary structure. The new extension can also be applied to estimating the selectivity of the path expressions in XML queries. The five-ary tuple is depicted as <Hid, nodeCount, childNameNum, textType, isElement>. Wherein, the Hid denotes the hierarchy encoding ID. The nodeCount denotes the number of nodes with the same Hid. The childNameNum denotes the number of distinct tag names among all its children. The textType denotes the type of text which can be further classified into three subtypes, the numeric text, the string text, and the element text (i.e., element without subelements or text content). The implementation of our approach is shown in Fig.3, integer 0 stands for the NULL, integer 1 stands for the type of numeric text, integer 2 stands for the type of sting text, and integer 3 stands for the element text. The isElement denotes a node type which is either an element or an attribute.

We apply the hash table technique to organize these records called Encoding Counting Table (ECT) as shown in Fig.3 (because of the space limit of the article, we just keep a part of the whole data set here).

Through the hierarchy encoding scheme, it is convenient to judge the relationship between the XML nodes. Given two nodes X, Y with their corresponding tuples of the forms of <Hid, nodeCount, childNameNum, textType, isElement>, we can judge whether X and Y are in AD relationship by formula (1). For clarity, let Hid(X) denote $X.Hid$, childNameNum(X) denote $X.childNameNume$.

$$Hid(Y)\&(2^{encodinglength(Hid(X))}-1)=Hid(X) \qquad (1)$$

If the result of (1) is true, then X is the ancestor of Y. For example, a node people with $H4=001000$, a node person with $H15=1001000$. $H15\&(2^6-1)=001000=H4$, so $H4$ is the ancestor of $H15$. If we further like to judge whether the nodes, X and Y ,are in PC relationship, we can figure it out by formula (2)

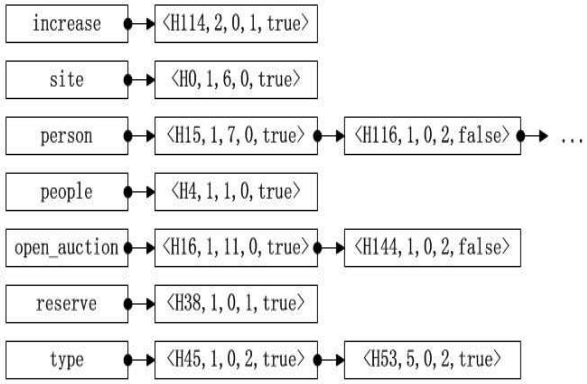


Fig. 3: The encoding counting table of hash table organization for the document shown in Fig.2

$$\begin{aligned} length(Child(X)) &= length(Hid(X)) \\ &+Ceiling(log_2(childNameNum(X)+1)) \end{aligned} \qquad (2)$$

Because the equation $length(H4)+Ceiling(log_2(1+1))=length(H15)$, is satisfied, the element person is the child of the element people, that is, they are in PC relationship.

4 VALUE-ENCODING HISTOGRAM

Through the ECT, we can estimate the result size of expression of A/B or A//B, but not the (predicate) filter operations. The filter operations, such as nodeName1[nodeName2θvalue], are very common in XPath queries. To estimate the result size of the filter operations, an efficient way is to build histograms. In our

method, a value-encoding histogram is to capture the distribution of the numeric values of the elements or the attributes.

The constructed Value-Encoding histogram can be described by several matrixes. Given a document with N distinct numeric elements, a Primary Matrix A is a 2×N matrix in which each column corresponds to a hierarchy encoding ID, the first row stores the minimal values for all the numeric elements and the second row stores the maximal value intervals of each numeric element. The Primary Matrix A is formatted as the following.

$$A = \begin{bmatrix} minV_1 & minV_2 & minV_3 & \cdots & minV_n \\ interval_1 & interval_2 & interval_3 & \cdots & interval_n \end{bmatrix}$$

Assuming that the buckets' number of the histogram is M, the Secondary Matrix B is a (M+1)×2 matrix with the first column set ones and the second column is a vector of 0 to M. Through the Primary Matrix A and the Secondary Matrix B, a Value Distribution Matrix C can be computed by B multiplying A (or BA for simplicity). These two matrixes are as follows.

$$B = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ \vdots & \vdots \\ 1 & m \end{bmatrix}, C = BA =$$

$$\begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ \vdots & \vdots \\ 1 & m \end{bmatrix} \begin{bmatrix} minV_1 & minV_2 & minV_3 & \cdots & minV_n \\ interval_1 & interval_2 & interval_3 & \cdots & interval_n \end{bmatrix}$$

$$= \begin{bmatrix} C_{11} & C_{12} & \cdots & C_{1n} \\ C_{21} & C_{22} & \cdots & C_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ C_{m1} & C_{m2} & \cdots & C_{mn} \end{bmatrix}$$

Based on matrix A, B and C, the Count Distribution Matrix D (initially ,a zero matrix) for an XML document can be constructed by applying the rule "when C(i, j) ≤ value < C(i+1, j) is held, D is to be updated with D(i, j) = D(i, j) + value". The value in this rule means the numeric value of an element in the XML document. Each column in D corresponds to one hierarchy encoding ID. D can be formally expressed as the below.

$$D = \begin{bmatrix} D_{11} & D_{12} & \cdots & D_{1n} \\ D_{21} & D_{22} & \cdots & D_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ D_{m1} & D_{m2} & \cdots & D_{mn} \end{bmatrix} = \begin{bmatrix} d_1 & d_2 & \cdots & d_n \end{bmatrix}$$

For evaluating the result size f of the predicate [nodeNameθvalue], If C(i, j) ≤ value < C(i+1, j) is held, an 1×M matrix e should be constructed according to the type of θ. The corresponding rules are shown in Table 1. The result size f is obtained by e * d_j , i.e., $\sigma(f) = e * d_j$.

For example, the matrixes, for the factor =0.535 of the XMark Database(see Section 6.1) and M =10, are as follows.

$$A = \begin{bmatrix} 1 & 3 & 1 & 1 & 1 & 0.24 & 1.5 \\ 0.33 & 4 & 0.22 & 1 & 0.33 & 705.85 & 15 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ \vdots & \vdots \\ 1 & 10 \end{bmatrix}$$

$$C = BA = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \\ 1 & 5 \\ 1 & 6 \\ 1 & 7 \\ 1 & 8 \\ 1 & 9 \\ 1 & 10 \end{bmatrix} \begin{bmatrix} 1 & 3 & 1 & 1 & 1 & 0.24 & 1.5 \\ 0.33 & 4 & 0.22 & 1 & 0.33 & 705.85 & 15 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 3 & 1 & 1 & 1 & 0.24 & 1.5 \\ 1.33 & 7 & 1.22 & 2 & 1.33 & 706.09 & 16.5 \\ 1.66 & 11 & 1.44 & 3 & 1.66 & 1411.94 & 31.5 \\ 1.99 & 15 & 1.66 & 4 & 1.99 & 2117.79 & 46.5 \\ 2.32 & 19 & 1.88 & 5 & 2.32 & 2823.64 & 61.5 \\ 2.65 & 23 & 2.1 & 6 & 2.65 & 3529.49 & 76.5 \\ 2.98 & 27 & 2.32 & 7 & 2.98 & 4235.34 & 91.5 \\ 3.31 & 31 & 2.54 & 8 & 3.31 & 4941.19 & 106.5 \\ 3.64 & 35 & 2.76 & 9 & 3.64 & 5674.04 & 121.5 \\ 3.97 & 39 & 2.98 & 10 & 3.97 & 6352.89 & 136.5 \\ 4.3 & 43 & 3.2 & 11 & 4.3 & 7058.74 & 151.5 \end{bmatrix}$$

For a given predicate [hapiness<6] denoted as f and the hierarchy encoding ID of this happiness is H148 which corresponds to the 4^{th} column in the D matrix. According to the rule in Table 1, the $\sigma(f)$ can be computed as the following.

$$\sigma(f) = ed_4 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 519 \\ 543 \\ 516 \\ 532 \\ 559 \\ 536 \\ 477 \\ 521 \\ 497 \\ 516 \end{bmatrix} = 2699$$

Table 1: Matrix e constructing rules for different θ

Operator θ	Matrix e
$=$	The ith coloum in e is set one, others are set zeros.
$<$	The colums in e in front of i-1 are set ones, others are set zeros.
$\leq$	The colums in e in front of i are set ones, others are set zeros.
$>$	The colums in e in front of i are set zeros, others are set ones.
$\geq$	The colums in e in front of i-1 are set zeros, others are set ones.
$\neq$	The ith coloum in e is set zero, others are set ones.

For string predicates (e.g. author='Smith'), the selectivity estimation is supported by the EMO algorithm or the EKVI algorithm which is the extended versions of the MO and the EKVI respectively. Both of these new algorithms are based on the RPST. The statistics information of strings is stored in the RPST which is a kind of modified PST. More details are given in [27].

5 COST-BASED QUERY OPTIMIZATION

An XML document tree can be modeled as a rooted, node-labeled tree $T = (V_T, E_T)$. V_T denotes the set of nodes of the tree, and E_T the set of edges of the tree. An XPath query (pattern) tree is a rooted, node-labeled tree $XQ = (V_Q, E_Q)$. Nodes of XQ are the Boolean compositions of the predicates. Edges of XQ are also labelled (PC or AD) to specify (the immediate or arbitrary) the relationship between nodes.

A match of the XQ in T is a total mapping M:$\{q : q \in XQ\} \rightarrow \{n : n \in T\}$ such that:

- For each node $q \in XQ$, the predicate node label of q is satisfied by M(q) in T ;
- For each edge $(p,q) \in XQ, M(p)$ is the parent or ancestor of M(q) in T.

Given an XQ, there are many alternative query (execution) plans that should be considered. The query optimizer can evaluate these plans to choose the optimal one (or at least sub-optimal one) based on their estimated costs.

An XPath evaluation plan is modeled as a rooted, labeled tree $XEP = (V_P, E_P)$, whose nodes are cataloged into a set of Operand Node (e.g., element, attribute or value node) or a set of Operator Node(e.g., "/", "//", "[ ]" (or Filter), ">" ,"=" ,and so on.), thus $V_P = V_{operand} \bigcup V_{operator}$ in this article.

5.1 Cost Model

In general, a path expression P can be expressed as $\omega N_1[P_1]\omega N_2[P_2]\omega \ldots \omega N_m[P_m]$, wherein, ω is an absolute path "/" or a relative path "//", N_k (k=1...m) can be an element operand or an * , and P_k is a set of predicates which can include path expressions. For example, a path expression can be /A//*/B[C/D]/ E[@F]/G. It is obvious that different join orders can produce different costs. It is worth mentioning that we only consider the abbreviated syntax of XPath, the transformation from the Axis XPath expression to the abbreviated one is out of this scope.

In this article, eight join operations and eight predicate filter operations summarized in Table 2 are considered. In order to accurately estimate the size of the result set, the selectivity of the predicates should be taken into account. How to calculate the selectivity is shown in Table 2. It should be mentioned that in this article the selectivity of a path expression without predicates can be accurately obtained through the ECT. Together with the ECT, the selectivity of a path expression can be obtained by applying the calculation method of the selectivity of the predicates.

Some explanations about the notations in Table 2 are described briefly. EN, EN1 and EN2 are the tag names, Path, LeftPath and RightPath denote the path expressions, PP denotes a path with predicates. Count is a function for computing the size of the results of a path expression through hierarchy encoding information, selec denotes the selectivity, such as Path.selec for the selectivity of the Path.Path(i).level denotes the level of the i^{th} element in the result set of the path expression Path.θ denotes one operator of =, $<$, $>$. $EN2_{cv}$ (see No.10 in Table 2) denotes the result set satisfying that EN2 and EN1 are of PC relationship, and at the same time that the EN2θValue is held when EN1[EN2θValue] is being evaluated. For example, given a query "$/site/open\_auctions/open\_auction[@id =$ '$open\_auction5$'$][bidder//date]/annotation$", the sub-path $open\_auction[@id =$ '$open\_auction5$'$]$ conforms to the type of EN1[EN2θValue](see No.10 in Table 2), the $[bidder//date]$ conforms to the type of EN1//EN2 (see No.7 in Table 2), and the operation that joins the $[bidder//date]$ and the $open\_auction[@id =$ '$open\_auction5$'$]$ conforms to the type of PP[/Path] (see No.14 in Table 2),etc.

The operations in Table 2 are based on five atomic operations which are called the Element-Path Parent-Child Matching primitive(short for EPPC), the Path-Path Parent-Child Matching primitive(short for PPPC),the Element-Path Ancestor-Descendent Matching primitive (short for EPAD),the Path-Path Ancestor-Descendent Matching primitive(short for PPAD) and the Element-Or-Attribute Fetching primitive(short for EOA). The unit costs of these atomic operations are denoted as F1, F2, F3, F4 and F5 respectively in Table 2.

The join operations are divided into the associative joins and the commutative joins in this article. The associative joins such as A/B/C can be executed in an associative way, that is, the execution plan of A/B/C can be (A/B)/C or A/(B/C); while the commutative joins such as A[B][C] (both [B] and [C] are not position predicates, or else the A[B][C] is not commutative one)

Table 2: Estimation Formulae for JoinCost

No.	Join Type	Cost Formula	Estimation Formula for Selectivity
1	EN/Path	$F1*Count(Path)*Path.selec$	Path.selec
2	LeftPath/RightPath	$F2*Count(LeftPath)*Count(RightPath)*LeftPath.selec*RightPath.selec$	$LeftPath.selec*RightPath.selec$
3	EN//Path	$F3*Path.selec*\sum_{i=1}^{n}Path(i).level$	$Path.selec$
4	LeftPath//RightPath	$F4*LeftPath.selec*RightPath.selec*\sum_{i=1}^{n}(RightPath(i).level)-LeftPath(i).level$	$LeftPath.selec*RightPath.selec$
5	EN1/EN2	$F5*totalNodes+F1*Count(EN2)*Path.selec$	1
6	Path / EN	$F5*totalNodes+F2*Count(Path)*Count(EN)*Path.selec$	$Path.selec$
7	EN1// EN2	$F5*totalNodes+F3*\sum_{i=1}^{n}EN2(i).level$	1
8	Path // EN	$F5*tatalNodes+F4*Path.selec*\sum_{i=1}^{n}(EN(i).level-Path(i).level)$	$Path.selec$
9	EN1[EN2]	$F5*totalNodes+F1*Count(EN2_C)$	$Count(EN2_C)/Count(EN1)$
10	EN1[EN2 θ Value]	$F5*2*totalNodes+F2*Count(EN1)*Count(EN2_{CV})$	$Count(EN2_{CV})/Count(EN1)$
11	PP[EN]	$F5*totalNodes+F2*Count(PP)*Count(EN)*PP.selec$	Count(EN)/Count(PP)
12	PP[EN θ Value]	$F5*totalNodes+F2*Count(PP)*Count(EN_{CV})*PP.selec$	$Count(EN_{CV})/Count(PP)$
13	EN[/Path]	$F5*totalNodes+F1*Count(Path)*Path.selec$	Count(Path)/Count(EN)
14	PP[/Path]	$F5*totalNodes+F1*Count(Path)*Path.selec$	Count(Path)/Count(PP)
15	EN[//Path]	$F5*totalNodes+F3*\sum_{i=1}^{n}Path(i).level*Path.selec$	Count(Path)/Count(EN)
16	PP[//Path]	$F5*totalNodes+F3*\sum_{i=1}^{n}Path(i).level*Path.selec$	Count(Path)/Count(PP)

cannot be executed in an associative way but in a commutative way, that is, the execution plan of A[B][C] can be of A[C][B] but not of A([B][C]).In this article, the expression with only the associative join(s) is called the association-enabled expression (AE), the expression with only the commutative join(s) is called the commutation-enabled expression (CE), and the expression with associative joins and commutative joins is called the hybrid expression (HE). The cost of HE comprise those of AE and CE.

The cost model of evaluating AE type P with N (location) steps is defined as formula (3)

$$Cost(i,j)=\begin{cases}0 & \text{i=j}\\ Cost(i,split)+Cost(split+1,j) & \\ +joinCost((i,split),(split+1,j)) & \text{(i+1<j<N)}\end{cases} \quad (3)$$

The cost model of evaluating CE type P with N (location) steps is defined as formula (4).

$$Cost(i,j)=\begin{cases}joinCost(i,j) & \text{i+1=j}\\ Cost(i,j-1) & \\ +joinCost((i,j-1),j) & \text{(i+1<j$\leq$N)}\end{cases} \quad (4)$$

The joinCost in formula (3) and (4) is the cost function for computing the join operations shown in Table 2.

5.2 Dynamic Programming for Optimizing the Execution Plan

5.2.1 Framework for dynamic programming method

The procedure of using the Dynamic Programming is made up of four main steps: initializing, cost estimating for alternative execution plans, recording the costs and split points in DP Information Record(short for DPIR), and constructing the optimal execution plan. The task of initialization that focuses mainly on building the DPIR is completed by Algorithm 1.

Algorithm 1: InitializingDPIR
*Input: path /*the path is a XEQ tree*/*
*Output: ArrayList< S_path[ ][ ]> S_paths /*of the format of DPIR*/*
Procedure:
{ */* Parsing the XPath expression path and storing the operands and operators into*
S_operand[] and S_operator[].S_operand[], S_operator[], S_path[][] are arrays/*
S_operand← path.Voperands;
S_operator←Path.Voperators; /
do {
if(i==j) S_path[i][j]←(0,i);
else {
S_path[i][j]←(0, path_split(S_operand [i],
S_operand [j]));
}
} *while((0 ⩽ i ⩽ j ⩽ S_operand.size())/*end of initializing the DPIR*/*
S_operands←S_operand[];
S_operators←S_operator[];
S_paths←S_path[][];
}

The task of the path_split(S_operand [i],S_operand [j]) is to determine the split point for a path S_operand starting from location i to location j. For a path expression with commutative join , such as A[B][C]/D (i=0 and j=3), there is no split point for A[B][C] in our approach because it makes no sense for A[B]([C]/D), in other words, A[B][C] should be taken as a whole part first although the order of [B] and [C] can be exchanged.

Taking "/site // open_auction[initial>'100'][//bidder] // description[//text/bold]/*/listitem[*/listitem//bold]" (i.e., the query 7 in Table 6) as an example, the initial and final states of the DPIRs are shown as Table 3 and Table 4 respectively. Each cell of the tables is a kind of the triple of <splitting point, minCost, resultSet> (each real result of the resultSet is just denoted as * in these tables for simplicity). In these tables, the values of the splitting point are integers from 0 to 8. The integer 0 indicates that the splitting position is just between the expression "site" and "open_auction", the integer 1 indicates that the splitting position is between the "open_auction" and "initial>100" , etc. Corresponding to the final state of DPIR, the execution order is the form of ((/site) // ((open_auction [initial>'100'][//bidder]) // (description[//text/bold])))/((*)/(listitem[*/listitem//bold])).

By applying the associative law and/or the commutative law, a path expression P can have alternative execution plans with different smaller pieces of path expressions in different orders. One way of evaluating the P can be realized by step joins. The main procedure of evaluating the cost of P is the EvalCost shown in Algorithms 2.

Algorithm 2: EvalCost
*Input: path /*path expression*/*
Output: minimal cost of path
Procudure:
{
*/*calculating the cost according to the type of the input path MAX_VALUE is a predefined big number*/*
if (Path.cost == MAX_VALUE)
{

Table 3: Initialization State of DPIR

	site	open_ auction	initial >100	// bidder	descrip tion	//text /bold	*	listitem	* /listitem //bold
site	0,max,*	0,max,*	0,max,*	0,max,*	0,max,*	0,max,*	0,max,*	0,max,*	0,max,*
open_ auction		1,max,*	1,max,*	1,max,*	1,max,*	1,max,*	1,max,*	1,max,*	1,max,*
initial >100			2,max,*	2,max,*	2,max,*	2,max,*	2,max,*	2,max,*	2,max,*
//bidder				3,max,*	3,max,*	3,max,*	3,max,*	3,max,*	3,max,*
descrip tion					4,max,*	4,max,*	4,max,*	4,max,*	4,max,*
//text /bold						5,max,*	5,max,*	5,max,*	5,max,*
*							6,max,*	6,max,*	6,max,*
listitem								7,max,*	7,max,*
* /listitem //bold									8,max,*

Table 4: Final State of DPIR

	site	open_ auction	initial >100	// bidder	descrip tion	//text /bold	*	listitem	* /listitem //bold
site	0,0.00,*	0,max,*	0,max,*	0,3.85,*	0,max,*	0,8.15,*	0,30.92,*	0,max,*	5,20.46,*
open_ auction		1,max,*	1,max,*	2,3.83,*	3,max,*	3,8.14,*	5,30.90,*	3,max,*	5,20.45,*
initial >100			2,max,*	2,max,*	2,max,*	2,max,*	2,max,*	2,max,*	2,max,*
// bidder				3,max,*	3,max,*	3,max,*	3,max,*	3,max,*	3,max,*
descrip tion					4,max,*	4,4.31,*	5,172.71,*	5,max,*	5,20.63,*
//text /bold						5,max,*	5,max,*	5,max,*	5,max,*
*							6,0.00,*	6,max,*	6,11.73,*
listitem								7,max,*	7,11.53,*
* /listitem //bold									8,max,*

if (Singlestep)
{ Path.cost←stepCost(Path); }
else if (Short-path-with-filters)
{ Path.cost← predicatePathCost(Path); }
else (longpath)
{ Path.cost←pathCost(Path); }
}
*return m_cost;/*return the minimum cost and update DPIR*/*
}

The Single-step is a kind of node (path) expressions without predicates such as A or B. The Short-path-with-filters is a kind of path expressions which contain an element (or node) operation followed by predicates, such as open_auction[@id], bidder[increase≤'19.50'],open_auction[@id='open_auction0'], bidder[1],bidder[increase] or the combination of these types of predicates. The Long-path is the combination of the Single-step and the Short-path-with-filters. Given split points, the cost can be evaluated through formula (3) and/or formula (4) together with the value-encoding histogram. The discussions on details of functions, such as stepCost(Path), predicatePathCost(Path) and pathCost(Path), are beyond the scope of this article.

After the costs are estimated and the updates of DPIR are completed, an optimized plan can be constructed to be the final execution plan by using Algorithm 3.

Algorithm 3: BuildOptimizedAST
Input: S_operands[ ],S_operators[ ], S_paths[ ][ ];
Output: Rtn / optimized execution plan */*
Procedure:
{
if (path_start ==path_end)
Rtn ← S_operands.get(loc)[path_start];
else
{
/ using recursion method to construct the final execution plan*/*
Rtn← s_operators.get(loc)[split];
Lhs←s_paths.get(loc)[path_start][split].buidOptimizedAST( );
Rhs←s_paths.get(loc)[split+1][path_end].buidOptimizedAST( );
LeftLink(Rtn)← Lhs;
RightLink(Rtn)←Rhs;
}
return Rtn;
}

In this article, InitializingDPIR, EvalCost and BuildOptimizedAST are the basic components of the framework for the dynamic programming method. We present two heuristic rules for this framework in the next section.

5.2.2 Heuristic rules for dynamic programming method

In relational databases, the project and the selection with predicates are evaluated first to reduce the intermediate results to fasten the query execution. In XML databases, we believe that some similar measurements will optimize the query execution. From our observations, the predicates and the wildcard * should be considered seriously. Two rules for query optimizer using the bottom-up fashion are the following.

Rule 1: if there are some wildcards (*) in a path expression, the lower * is evaluated first, while for a join between * and a sub-path, the sub-path is evaluated first.

Rule 2: If there are no predicates in the path expression, the candidate execution plan can be selected from the set of the right-deep query trees.

These two heuristic rules are implemented in the function pathCost(Path) in the algorithm EvalCost. The approach to deal with the path expression with predicates has been discussed in section 5.1.

The heuristic rules can degrade the time complexity of the search algorithm. For rule 2, given an XPath query A1/A2/A3.../An as the example, the number of all the possible query plan trees can be figured out by the formula (5)

$$f(n) = \begin{cases} 1 & \text{n=1} \\ \sum_{i=1}^{n-1} f(i) f(n-i) & \text{n>1} \end{cases} = \begin{cases} 1 & \text{n=1} \\ \frac{(2n-2)!}{n((n-1)!)^2} & \text{n>1} \end{cases} \quad (5)$$

Using the basic dynamic programming method, the time complexity of the search algorithm is O(n^3). When the heuristic rule 2 is applied to the basic dynamic programming method, the size of the search space can be computed by the formula (6).

$$F(n) = \begin{cases} 1 & \text{n=1,2} \\ F(1)F(n-1)+F(n-1)F(1) & \text{n>2} \end{cases} = \begin{cases} 1 & \text{n=1,2} \\ 2^{n-2} & \text{n>2} \end{cases} \quad (6)$$

In this case, the time complexity of the search algorithm based on the basic dynamic programming is O(n^2).

6 EXPERIMENTS

In this section, the primary experimental results are provided here to verify the validity of our approaches. The performance comparisons between the original approach and our optimized approach focus on the execution time, the scaling effect of the optimization approach and the speedup ratios of the optimized queries over the original ones. The experimental results are the average cost of the executions carried out five times in the same tests.

The original approach constructs the query expression tree from the left to the right according to the XPath expression and executed in a bottom-up fashion. The optimized approach corresponds to our heuristic-based dynamic programming method.

Table 5: Some Selected Query Statements for Testing

Group	QNum	Path Expression
A	1	/*/regions//*/description//parlist/*/text/emph
	2	/site/closed_auctions/closed_auction/*/description /*/listitem/parlist/*/text/emph
B	3	/site/regions//item[1]/description//*/listitem/text/emph
	4	/site/regions//item[quantity=1] /description[//parlist/listitem/text]//emph
	5	/site/closed_auctions/closed_auction[price>'40.00'] /annotation[/author/@person]/description/* /listitem/parlist/listitem/text/emph
C	6	/site/regions//item[@featured][quantity][payment='Creditcard'] /mailbox[mail]/mail/text[bold]/bold
	7	/site//open_auction[initial>'100'][//bidder] //description[//text/bold]/*/listitem[*/listitem//bold]
	8	/site/closed_auctions/closed_auction[price>'10'] [//listitem/text/*/emph][price<'15'][quantity=1] /annotation/description//text/emph
D	9	/site/categories/category[@id = 'category0']//listitem[//text[bold]]/parlist
E	10	/site/regions//item/description//parlist/listitem/text/emph
	11	/site/closed_auctions/closed_auction/annotation/description /parlist/listitem/parlist/listitem/text/emph

Table 6: XML Databases for Testing

Database	Size(MB)	factor	#of Node
1	60	0.535	1734772
2	90	0.795	2582727
3	120	1.065	3450402
4	150	1.35	4376400
5	180	1.60	5179247

6.1 Experiment Environment

The test bed is the in-memory version of the XSQS[12] which is a native XML DBMS prototype. All experiments were run on a machine with a 2.8 MHz Pentium processor, 3GB memory, and a 150G hard disk. The underlying OS is Windows 7/64bit.

The test data or database is generated by XMark[11] shown in Table 5. The factors are 0.535, 0.795, 1.065, 1.345 and 1.595 of the size of 60M, 90M, 120M, 150M and 180M, respectively.

The selected running examples which are cataloged into five groups are shown in Table 6. Each group has some similar characteristics. Group A includes queries with the wildcards (*) but with no predicates, group B includes queries with single predicate, group C includes queries with the conjunctive predicates, group D includes a query with an embedded predicate and group E includes queries without the predicates and the wildcards (*).

6.2 Experimental Results

6.2.1 Performance Results

Fig.4 depicts the performance comparisons between the original execution plans and their optimized counterparts (the optimization cost is included) on database1 as an example (size of database1 is 60MB, see Table 3). By the way, the costs of generating the optimized execution plans of the queries are less than 30ms except that of Q8 which is about 60ms-80ms in our tests. The experiments on the other databases illustrate similar effects.

6.2.2 Scaling Effect

To know how our optimization approach responds to the growth of the size of databases, we carried out tests by using, one by one, the same queries over the five databases (shown in Table 3). Because the execution time of Q4, Q6 and Q7 is one magnitude higher than other queries, the experimental results of these eleven queries are depicted in (a) and (b) of Fig.5 for clarity. The similar consideration is taken for describing the experimental results of the speedup ratio in section 6.2.3.

6.2.3 Speedup Ratio of the Optimization

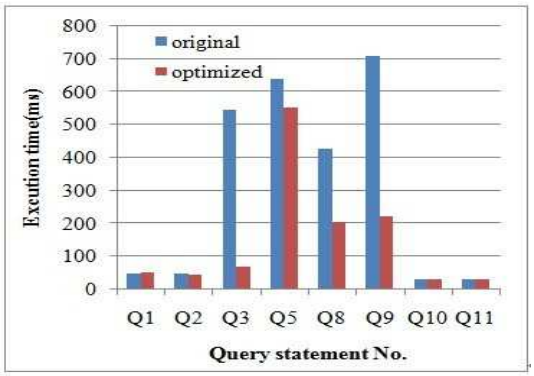
To know to what extension the optimization achieved, we did the tests on speedup ratios by using these eleven queries over these five databases. Depicted in (a) and (b) of Fig.6 are the speedup ratios which are the costs of the original execution plan divided by the costs of the optimized queries.

6.2.4 Remarks on Experimental Results

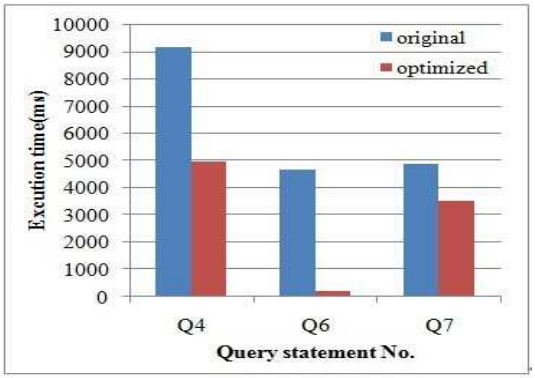
It can be seen in Fig.4 that the execution time of the optimized queries of Q3-Q9 over database1 (of size 60MB) is much less than that of the original ones. While the execution time of the optimized plans of Q1, Q2, Q10 and Q11 is slightly more than that of the original ones. The reasons are that the optimized execution trees are the same as those of the original ones and the execution time of each optimized one should include the cost of generating the optimized execution tree. The same results are obtained for other databases. All of the Q1, Q2, Q10 and Q11 are the path expressions without the predicates, their optimal query trees are all the right-deep trees in our tests. Our heuristic rules make sense.

Fig.5 shows that these optimized queries almost have linear scaling effects with the growth of the size of database, taking into account the presence of system errors.

As Fig.6 shown, among these eleven queries, Q3, Q6, Q8 and Q9 have approximate linear behavior. Q6 containing a string predicate has the maximal speedup ratio. The rest of the queries have nearly constant (or in a narrow band) speedup ratios. It can

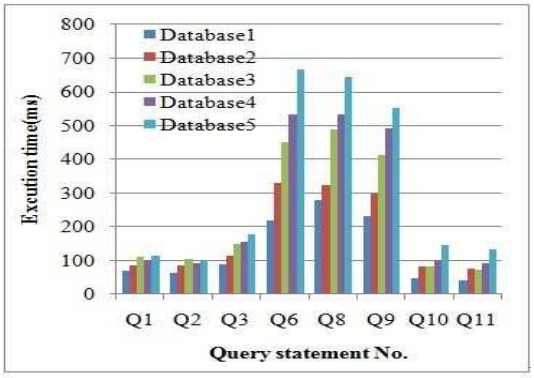


(a)

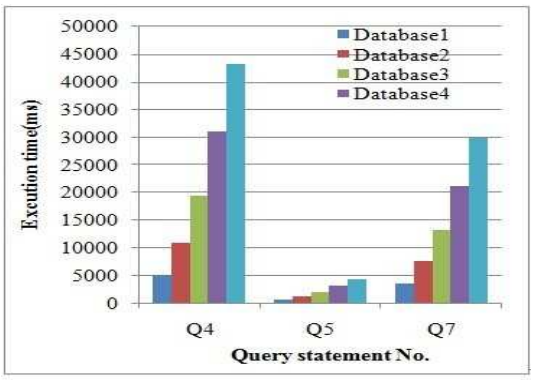


(b)

Fig. 4: Performance Results of Q1-11 on Database 1

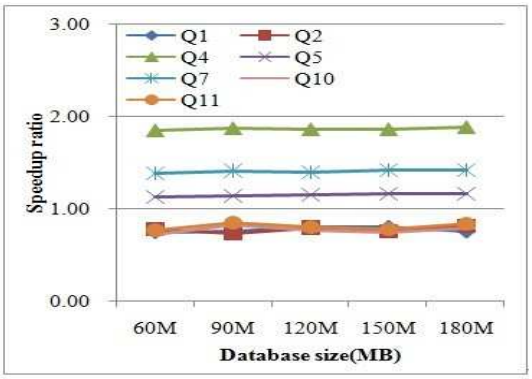


(a)

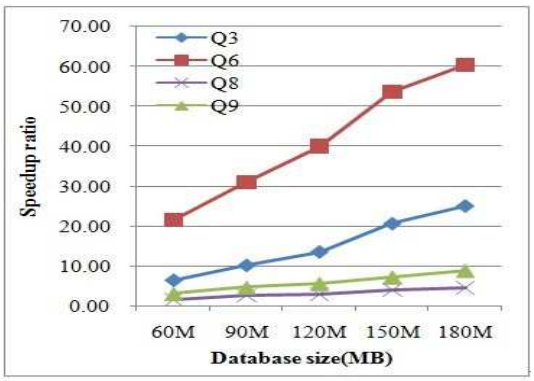


(b)

Fig. 5: Scaling Effect of Optimized Q1-11



(a)



(b)

Fig. 6: Speedup Ratios of the Optimized Queries for Q1-Q11 over the Original Counterparts

be concluded that by using our approach a good speedup ratio can be achieved for XPath.

7 CONCLUSIONS AND FUTURE WORK

In this article, a value-encoding histogram is proposed to collect statistics of the XML database and the cost models are presented to describe the costs of different types of join operations. Together with the hierarchy encoding technique, a heuristic-based dynamic programming method is used to optimize the join order. The experiments demonstrate that the optimized query plans have linear scaling effects, greater performance improvements and significant speedup effects for evaluating XPath queries. Our approaches can always select the optimal execution plan. To parallelize our algorithms and to extend them for XQuery is our next step.

8 ACKNOWLEDGMENTS

The work is partially supported by National Natural Science Foundation of China (Grant No. 71090403), Education Ministry of Education of P.R.C (Grand No. x2rjB7110020), Bureau of Science and Information Technology of Guangzhou (Grant No.2011J4100061 or No.x2rjB2111420).The authors are very grateful to Professor Ting Lin for many valuable discussions. We also thank the anonymous reviewers for their helpful suggestions that improve the quality of this paper.

References

[1] Extensible Markup Language, DOI http://xml.coverpages.org/xml.html.
[2] Fiebig, T., Helmer, S., Kanne, C.-C., Moerkotte, G., Neumann, J., Schiele, R., Westmann, T. Anatomy of a native XML base management system. The VLDB Journal, **11**, 292-314 (2002).
[3] P. T. Wood. Minimizing simple XPath expressions. In Proc. 4th Int. Workshop on the Web and Databases(Santa Barbara, California, May, 13-18 (2001).
[4] Amer-Yahia, S., Cho, S., Lakshmanan, L., Srivastava, D. Minimization of tree pattern queries. In Proc. of ACM Conf. on Management of Data (SIGMOD), 497-508 (2001).
[5] Flesca, S., Furfaro, F., Masciari, E. On the minimization of Xpath queries. In Proc. of VLDB, 153-164 (2003).
[6] Kwong, A.,Gertz, M. Schemabased optimization of XPath expressions.Technical report, Dept. of Computer Science, University Of California, (2001).

[7] M. Fernández and D. Suciu. Optimizing regular path expressions using graph Schemas. In Proc. 14th International Conference on Data Engineering, Orlando, Florida, USA, February (1998).
[8] N. Zhang, P.J. Hass, V. Josifovski, G. M. Lohman, and C. Zhang. Statistical learning techniques for costing XML queries. In Proc. 31st Int. Conf. on Very Large Data Bases, 289-300 (2005).
[9] S. Groppe and S. Böttcher. Schema-based Query Optimization for XQuery Queries, In Prodeedings of the Advances in Databases and Information Systems, Tallinn, Estonia, (2005).
[10] D. Che, K. Aberer and özsu T. Query optimization in XML structured-document Databases. VLDB Journal, **15**, 263-289 (2006) .
[11] A. Schmidt, F. Waas, M. Kersten, D. Florescu, I. Manolescu, M. Carey, and R. Busse. XMark: A Benchmark for XML Data Management. In Proceedings of International Conference on Very Large Data Bases, 974-985 (2002).
[12] Dong Li, Xiuyu Lu, Xifeng Huang, Wenhao Chen. Structural Join in the 'XSQS' Native XML Database. Accepted by Journal of Software, (2012).
[13] Wu YQ, Patel JM, Jagadish HV. Estimating answer sizes for XML queries. In roceedings of the 8th International Conference on Extending Database Technology: Advances in Database Technology, March, 590-608 (2002).
[14] W. Wang, H. Jiang, H. Lu, and J. X. Yu. Containment join size estimation: Models and methods. In SIGMOD, 145-156 (2003).
[15] McHugh, J., Widom, J. Query optimization for XML. In Proceedings of the 25th International Conference on Very Large Databases, Edinburgh, Scotland, September, 315-326 (1999).
[16] Li Dong, Gu Ning. Hierarchy Encoding Based XML Query Estimation, In International Forum on Information Technology and Applications, May, **2**, 451-456 (2009).
[17] Y. Wu, J. M. Patel, and H. V. Jagadish. Structural join order selection for XML query optimization. In Proc. 19th IEEE Int'l Conf. Data Eng.(ICDE), March, 443-454 (2003).
[18] N. Polyzotis and M. N. Garofalakis. Statistical Synopses for Graph-Structured XML Databases. In Proceedings of the ACM SIGMOD Conference on the Management of Data, 358-369 (2002).
[19] Z. Chen, H. V. Jagadish, F. Korn, N. Koudas,S. Muthukrishnan, R. T. Ng, and D. Srivastava. Counting Twig Matches in a Tree. In Proceedings of the 17th International Conference on Data Engineering(ICDE), 595-604 (2001).
[20] W. Wang, H. Jiang, H. Lu, and J. X. Yu. Bloom Histogram: Path Selectivity Estimation for XML Data with Updates. In Proceedings of the 30th International Conference on Very Large Data Bases(VLDB), 240-251 (2004).
[21] L. Lim, M. Wang, and J. S. Vitter. CXHist: An On-line Classification-Based Histogram for XML String Selectivity Estimation. In Proceedings of the 31st International Conference on Very Large Data Bases(VLDB), 1187-1198 (2005).
[22] S. Tatarinov, K. S. Viglas., J. Beyer, E. Shanmugasundaram, J. Shekita, C. Zhang. Storing and querying ordered XML using a relational database system. In SIGMOD, 204-215 (2002).
[23] J. Lu, T. Wang Ling, C. Chan, T. Chen. From Region Encoding To Extended Dewey: On Efficient Processing of XML Twig Pattern Matching. In VLDB, 193-204 (2005).
[24] P. O'Neil, E. O'Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury. ORDPATHs: Insert-friendly XML node labels. In SIGMOD, 903-908 (2004).
[25] N. Bruno, D. Srivastava, N. Koudas. Holistic twig joins: optimal XML pattern matching. In SIGMOD, 310-321 (2002).
[26] Balmin A, Eliaz T, Hornibrook J, et al. Cost-based optimization in DB2 XML. IBM Systems Journal, **45**, 299-319 (2006).
[27] Dong Li, Qixu Zhang, Xiaochong Liang, et al. Selectivity estimation for string predicates based on modified pruned count-suffix tree, submitted to Chinese Journal of Electronics.
[28] Selinger P G, Astrahan M M, Chamberlin D D, et al. Access path selection in a relational database management system,Proceedings of the 1979 ACM SIGMOD international conference on Management of data. ACM, 23-34 (1979).
[29] Moerkotte G, Neumann T. Analysis of two existing and one new dynamic programming algorithm for the generation of optimal bushy join trees without cross products,Proceedings of the 32nd international conference on Very large data bases. VLDB Endowment, 930-941 (2006).
[30] Moerkotte G, Neumann T. Dynamic programming strikes back, Proceedings of the 2008 ACM SIGMOD international conference on Management of data. ACM, 539-552 (2008).

Dong Li received his BS from Harbin Institute of Technology in 1992 and his Master and PhD from Huazhong University of Science and Technology in 1995 and 2001, respectively. He had been to UCSB as a visiting scholar from 2008 to 2009. He is a professor at South China University of Technology in China. His research interests include XML databases, mobile database and service computing.

Wenhao Chen received his BS from South China University of Technology in 2011. He is currently a Master student in South China University of Technology in China, and his research interest is XML database.

Xiaochong Liang received her BS from South China University of Technology in 2010. She is currently a Master student in South China University of Technology in China, and her research interest is XML database.

Xiuyu Lu received her BS from Sun Yat-sen University in 2010. She is currently a Master student in South China University of Technology in China, and her research interest is XML database.

Jida Guan received his BS from Guangdong University of Business Studies in 2012. He is currently a Master student in South China University of Technology in China, and his research interest is XML database.

Yang Xu received his Master from Huazhong University of Science and Technology in 2001 and his PhD from Wuhan University in 2007, respectively. He is a reseacher and lecture at South China University of Technology in China. His research interests include business process management, service computing and semantic Web.