

# A Decomposition-Based Probabilistic Framework for Estimating the Selectivity of XML Twig Queries

Chao Wang, Srinivasan Parthasarathy, and Ruoming Jin

Department of Computer Science and Engineering, The Ohio State University  
Contact: {srini}@cse.ohio-state.edu

**Abstract.** In this paper we present a novel approach for estimating the selectivity of XML twig queries. Such a technique is useful for answering approximate queries as well as for determining an optimal query plan for complex queries based on said estimates. Our approach relies on a summary structure that contains the occurrence statistics of small twigs. We rely on a novel probabilistic approach for decomposing larger twig queries into smaller ones. We then show how it can be used to estimate the selectivity of the larger query in conjunction with the summary information. We present and evaluate different strategies for decomposition and compare this work against a state-of-the-art selectivity estimation approach on synthetic and real datasets. The experimental results show that our proposed approach is very effective in estimating the selectivity of XML twig queries.

## 1 Introduction

XML is gaining acceptance as a standard for data representation and exchange over the World Wide Web. However, for wide-spread deployment and use it is becoming increasingly clear that the design of an efficient high-level querying mechanism is necessary. Since XML documents may be represented as a rooted and labeled tree, this necessity has led to the development of tree-based (twig) querying mechanisms. Twig queries describe a complex traversal of the document graph and retrieve document elements through an intertwined (i.e., joint) evaluation of multiple path expressions.

Given the importance of twig queries as a basic selection mechanism in XML [1–3], efficient support for accurately estimating their selectivity is crucial for the optimization of complex queries. This is analogous to selectivity estimation in relational databases [4–7]. Accurate selectivity estimation is also desirable in interactive settings and for approximate queries. For instance, an end-user can interactively refine their query if they know it will return an overwhelmingly large result set. Similarly, the estimated value can be returned as an approximate answer to aggregate queries using the COUNT primitive.

The early work in this area has focused on determining the selectivity of path expressions (a special case of twig queries) [8–13]. The Lore system [8] adopts a Markov model-based approach for this purpose. The Markov table method [10] improves on the Lore system through the use of intelligent pruning and aggregation to reduce space requirements. Recently, Lim and Wang proposed XPathLearner [9], an on-line, tunable

Markov table method which has been shown to be effective for path expression selectivity. A key limitation of these methods is that they do not adapt well to twig queries because they do not account for path correlations.

More recently, researchers have focused on selectivity estimation for twig queries [14, 3, 15, 2, 1]. Examples include Correlated Sub-Trees [3], XSketches [15, 1] and TreeSketches [2]. Among these it has been shown that TreeSketches is the most accurate and efficient method [2]. TreeSketches [2], a successor of XSketches, clusters the similar fragments of XML data together to generate its synopsis. The granularity of the clustering depends on the memory budget.

To estimate the selectivity of XML twig queries, the above approaches, as well as the approach presented in this paper, define a summary data structure that houses important statistics about the data from which the selectivity may be estimated. Important issues at hand include: the quality of estimation from the given summary; the time to construct the summary; and finally, the time to estimate the selectivity of queries from the summary. To address these issues we present a new approach to selectivity estimation. The key contributions of our approach are highlighted below.

First, we present a framework under which the selectivity of a query (represented as a rooted tree) can be estimated from its subtrees. We present and evaluate different strategies for decomposing the query into subtrees. These subtrees can then be used to arrive at a selectivity estimate. We present a theoretical basis for this approach and furthermore show that it subsumes the Markov model-based XML path selectivity estimation as a special case.

Second, to summarize an XML dataset we leverage the use of frequent tree mining. A dynamically-determined subset<sup>1</sup> of all the discovered subtrees up to a certain size (number of nodes), coupled with associated occurrence statistics, forms the basis of our summary structure. More specifically, the dynamic subset we store is based on the notion of (*non*)-*derivable* patterns. We also rely on fast searching mechanisms to locate the subtrees of a given twig query within our summary structure.

Third, we conducted an extensive experimental study to examine the benefits of our approach and compare it against TreeSketches<sup>2</sup>. Empirical results show that our approach takes less time to construct the summary, and is usually much faster when computing the selectivity estimates. In our qualitative assessment we also find that our approach compares favorably with TreeSketches. We also offer a detailed explanation as to why the new approach (called *TreeLattice*) outperforms TreeSketches [2] under certain conditions.

The rest of the paper is organized as follows. We formally define our problem and give an overview of TreeLattice in Section 2. In Section 3, we detail our proposed summary structure and twig decomposition-based XML twig selectivity estimation framework. We present experimental results in Section 4 and related work in Section 5. Finally we discuss the future work and conclude in Section 6.

<sup>1</sup> Due to storage costs, the complete lattice (all frequent patterns) cannot be held in memory, thus we only store a portion of it, which is dynamic and data dependent.

<sup>2</sup> We are grateful to Neoklis Polyzotis for providing us with the TreeSketches executable and also for helping us tune the algorithm for a fair comparison.

## 2 Problem Definition and TreeLattice Overview

In the following section, we formally define the problem of estimating XML twig selectivity (Subsection 2.1). We follow with a discussion of the basic ideas and key challenges in our new approach, TreeLattice (Subsection 2.2).

### 2.1 Problem Definition

An XML document can be structurally modeled as a tree where each node is typically associated with a tag or a value. In practice, values are almost always associated with leaf nodes and tags with interior nodes. As with prior work by Polyzotis and Garofalakis [16], we do not model value elements.

A twig query  $T_Q$  is defined as a node-labeled tree  $T_Q(V_Q, E_Q)$ , where each node  $t_i \in V_Q$  is labeled with a path expression  $P_i$ . At an abstract level, each node  $t_i$  corresponds to a subset of elements, while the path  $P_i$  describes the structural relationship that must be satisfied between the elements in  $t_i$  and the elements in its parent node. In particular, we only consider the parent/child relationship between different elements. Research on the more general ancestor/descendant relationship is underway. We next present the definition of a twig match as given by Chen *et al.* [3].

**Definition 1.** A match of a twig query  $T_Q = (V_Q, E_Q)$  in a node-labeled data tree  $T = (V_T, E_T)$  is defined by a 1 – 1 mapping:  $f : V_Q \mapsto V_T$  such that if  $f(u) = v$  for  $u \in V_Q$  and  $v \in V_T$ , then (i)  $Label(u) = Label(v)$  and (ii) if  $(u, u') \in E_Q$ , then  $(f(u), f(u')) \in E_T$ .

The selectivity  $\sigma(T_Q)$  of twig query  $T_Q$  is defined as the number of matches of  $T_Q$  in the data tree. Our objective is to accurately estimate the selectivity of an XML twig query  $T_Q$  as efficiently as possible given constraints in space (summary storage) and time (summary construction and estimation time).

### 2.2 Basic Ideas and Key Challenges of TreeLattice

The first basic idea in TreeLattice comes from the observation that in many cases, the selectivity of a given twig query  $\sigma(T_Q)$  can be reasonably estimated from the selectivity information of its sub-twig queries. For example, suppose twig  $T_Q$  is the union of two sub-twigs  $T_1$  and  $T_2$ , which differ by only one edge and share a common part  $T$  (Figure 1a). We can expect  $\sigma(T_1)$ ,  $\sigma(T_2)$  and  $\sigma(T)$  to provide good clues for estimating  $\sigma(T_Q)$  in many real datasets. Furthermore, if the twig  $T_Q$  is the union of a set of sub-twigs, the selectivity of all these sub-twigs can be used to estimate  $\sigma(T_Q)$ . Therefore, the first problem we face is *can we develop a reasonable selectivity estimate for a given twig query  $T_Q$  by utilizing the selectivity of its sub-twigs?* This problem is answered in Subsection 3.1, where we construct such an estimator based on the *conditional independence assumption for growing a tree*. Note that in order to systematically estimate the selectivity of twig queries with this approach, we need to pre-compute a group of small twigs as the basis.

However, we can also expect that our assumption will likely be violated for some twig queries on a given XML dataset. To deal with this issue, we use another basic idea

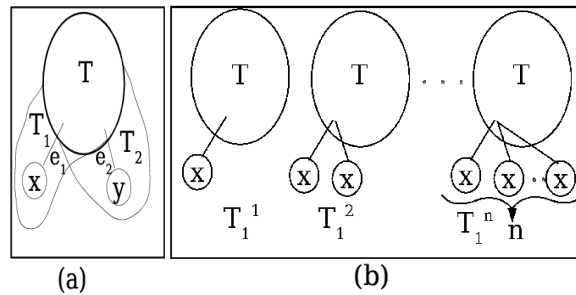
from the observation that the selectivity information of different twigs can be of differing importance in terms of capturing the underlying twig distribution. For example, if the selectivity of  $T_1, T_2$ , and  $T$  are available and  $T_Q$  can be *precisely* estimated from them, then the selectivity of  $T_Q$  should not be pre-computed. Here, we face another key challenge in TreeLattice: *how can we select a group of twigs as the basis for selectivity estimation in order to minimize the estimation error?* In particular, such selection needs to be performed under the budget of user-defined memory cost. Another problem closely related to this challenge is *how can we decompose a large twig query into basic twigs and perform estimations if different decompositions exist?* The solution to the latter actually helps us determine a solution for the former. In Subsection 3.2, we discuss the decomposition problem and in Subsection 3.3, we introduce our method to select basis for selectivity estimation.

Given the above discussion, we can see that our TreeLattice has three basic components: *Basis Building*, *Twig Decomposition*, *Augmenting Estimation*. The basis building is off-line and the other two components are computed at runtime while processing a query. When a new query arrives, we first decompose it into the small twigs in the basis and use the pre-computed selectivity of these basic twigs to infer the selectivity of the complex (larger) one.

### 3 An Estimation Framework based on Twig Decomposition

In this section, we will answer the three questions posed in the previous section. The twig decomposition-based selectivity estimation framework will be formulated during the course of this discussion.

#### 3.1 Augmenting Twigs



**Fig. 1.** (a) Augmented twigs  $T_1 \cup T_2$ ; (b) Growing  $T_1$  from  $T$

Suppose we have two basic twigs  $T_1$  and  $T_2$ , and they differ by only one edge (Figure 1(a)). If  $T$  is common to both, then we can express  $T_1$  as  $T \cup \{e_1\}$  and  $T_2$  as  $T \cup \{e_2\}$ , where  $e_1$  and  $e_2$  are two *distinct* edges. The edges are distinct in that they either attach to different nodes of  $T$ , or the two additional nodes  $x$  and  $y$  introduced by

these two edges are different. The two twigs can be augmented together to generate a larger twig, denoted as  $T_1 \cup T_2 = T \cup \{e_1\} \cup \{e_2\}$ . Assuming the counts of  $T_1$ ,  $T_2$  and their common part  $T$  are available and denoted as  $\sigma(T_1)$ ,  $\sigma(T_2)$  and  $\sigma(T)$ , respectively, we are interested in estimating the count of the augmented twig,  $T_1 \cup T_2$ , based on this information.

A complication arises when the occurrence of  $T$  is coupled with one or more instances of edge  $e_1$ , as shown in Figure 1(b). Let  $T_1^i$  denote the occurrence of  $T$  with  $i$  edges of type  $e_1$ . Then it is easy to see that the selectivity of  $T_1$  is given by the decomposition formula<sup>3</sup>:

$$\sigma(T_1) = \sigma(T_1^1) + 2 \times \sigma(T_1^2) + \dots + n \times \sigma(T_1^n)$$

and similarly the selectivity of  $T_2$  is given by:

$$\sigma(T_2) = \sigma(T_2^1) + 2 \times \sigma(T_2^2) + \dots + m \times \sigma(T_2^m)$$

In order to derive our formula for estimating the augmented twig  $T_1 \cup T_2$ , we assume that the event of growing  $T_1$  from  $T$  is conditionally independent from the event of growing  $T_2$  from  $T$  (called the *tree-growing independence assumption*). More formally we have:

$$Pr(T_1^i \cup T_2^j | T) = Pr(T_1^i | T) \times Pr(T_2^j | T)$$

where:

$$Pr(T_1^i | T) = \sigma(T_1^i) / \sigma(T)$$

and:

$$Pr(T_2^j | T) = \sigma(T_2^j) / \sigma(T)$$

**Theorem 1.** *Given two non-trivial rooted and labeled twigs  $T_1$  and  $T_2$ , which differ by only one edge, let  $T$  be the common part between  $T_1$  and  $T_2$ . Under the tree-growing independence assumption, the expected count of  $T_1 \cup T_2$  is given by  $\sigma(T_1) \times \sigma(T_2) / \sigma(T)$ .*

**Proof:** Given the tree-growing independence assumption, we can treat the count of  $T_1 \cup T_2$  as a random variable. The expected value of this random variable,  $E(\sigma(T_1 \cup T_2))$ , is as follows:

$$\begin{aligned} & \text{(From the decomposition formula)} \\ E(\sigma(T_1 \cup T_2)) &= \sum_{i=1}^n \sum_{j=1}^m E(\sigma(T_1^i \cup T_2^j)) \\ &= \sum_{i=1}^n \sum_{j=1}^m (i \times j \times Pr(T_1^i \cup T_2^j | T) \times \sigma(T)) \end{aligned}$$

(By the conditional independence assumption)

<sup>3</sup> The coefficients in front of each term represents the number of choices one has to grow from  $T$  to  $T_1$ .  $n$  is the maximal number of  $e_1$  edges under  $T$ . Similarly,  $m$  is the maximal number of  $e_2$  edges under  $T$ .

$$\begin{aligned}
&= \sum_{i=1}^n \sum_{j=1}^m i \times j \times Pr(T_1^i|T) \times Pr(T_2^j|T) \times \sigma(T) \\
&= \sigma(T) \times \sum_{i=1}^n i \times Pr(T_1^i|T) \times \left( \sum_{j=1}^m j \times Pr(T_2^j|T) \right) \\
&= \sigma(T) \times \sum_{i=1}^n i \times Pr(T_1^i|T) \times \left( \sum_{j=1}^m j \times \frac{\sigma(T_2^j)}{\sigma(T)} \right) \\
&= \sigma(T) \times \sum_{i=1}^n i \times Pr(T_1^i|T) \times \frac{1}{\sigma(T)} \times \left( \sum_{j=1}^m j \times \sigma(T_2^j) \right) \\
&\quad \text{(The decomposition of count of } \mathbf{T}_2, \sigma(\mathbf{T}_2)) \\
&= \sigma(T) \times \sum_{i=1}^n i \times Pr(T_1^i|T) \times \frac{\sigma(T_2)}{\sigma(T)} \\
&\quad \text{(The decomposition of count of } \mathbf{T}_1, \sigma(\mathbf{T}_1)) \\
&= \sigma(T) \times \frac{\sigma(T_1)}{\sigma(T)} \times \frac{\sigma(T_2)}{\sigma(T)} \\
&= \sigma(T_1) \times \sigma(T_2) / \sigma(T)
\end{aligned}$$

□

In our approach, we will use the expected count of  $T_1 \cup T_2$  as the estimate of the true count of  $T_1 \cup T_2$ , denoted as  $\hat{\sigma}(T_1 \cup T_2) = \sigma(T_1) \times \sigma(T_2) / \sigma(T)$ .

An important lemma that follows from this theorem is stated next and its proof can be found in the full version of this paper [17].

**Lemma 1.** *Given two subtrees  $T_1$  and  $T_2$  that share a common subtree  $T$ , where*

$$|T| = \min(|T_1|, |T_2|) - 1$$

*then  $\sigma(T_1 \cup T_2)$  can be estimated as follows:*

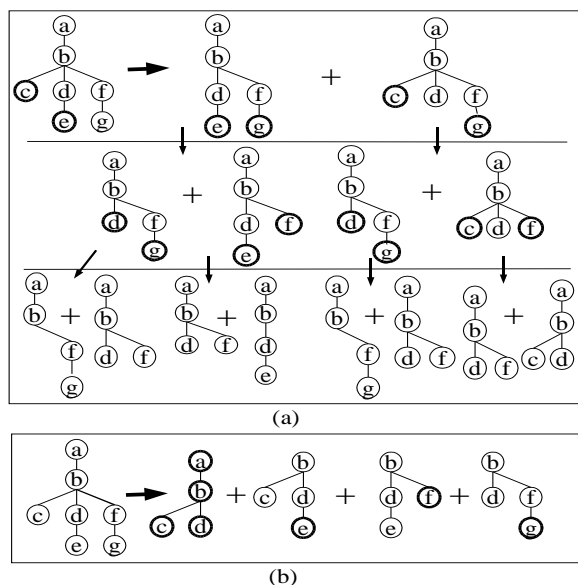
$$\sigma(T_1 \cup T_2) = \frac{\sigma(T_1) \times \sigma(T_2)}{\sigma(T)}$$

### 3.2 Twig Decomposition

In this section, we discuss how to decompose a large twig query into basic twigs and also how to estimate its selectivity.

**Recursive Decomposition Scheme** This decomposition is obtained directly from Lemma 1. Since each tree has at least two leaf nodes (if the root node has degree 1, it can also be

considered a leaf node for our purposes), we can always obtain two subtrees of the original tree by removing one leaf node or the other. These subtrees are labeled  $T_1$  and  $T_2$ , respectively. If the size of  $T$  is  $k$ , then the size of  $T_1$  and  $T_2$  will be  $(k - 1)$ . Suppose the common part between  $T_1$  and  $T_2$  is  $T_3$ , then we can apply the above formula to estimate the selectivity of  $T$ , given the selectivity of  $T_1$ ,  $T_2$  and  $T_3$ .



**Fig. 2.** (a) Recursive decomposition scheme; (b) Fixed-sized decomposition scheme

This decomposition scheme ensures that the overlap between  $T_1$  and  $T_2$  is maximal and thus ensures that the correlation of occurrence is well captured. If  $T_1$  and  $T_2$  are too large to fit in the lattice summary, then we execute the above decomposition process recursively, until we reach the brim of the lattice summary. We present an example of this recursive decomposition in Figure 2a. Here a twig of size 7 is decomposed into a set of sub-twigs of size 4. The bold nodes are chosen to be eliminated at each step in the recursion. Figure 3 presents the formal algorithm of the estimator.

**Voting Scheme Extension:** We note that a twig may have more than two leaves. In this case the choice of leaf nodes for decomposition may result in different estimates. Correspondingly, we can have multiple estimations at each recursive step. As an optimization, we record all the estimations at a given level and average them to obtain a resulting estimate to be used in the next step. Intuitively, we expect to avoid skewed estimates resulting from poor initial choices and that this optimization will prevent the propagation of errors during the course of the decomposition. Different voting schemes can be applied here. We will demonstrate the effect of this optimization in Section 4.

```

Algorithm: Estimate ( $T, L$ )
Input:  $T$ , an XML twig;
        $L$ , the lattice summary;
Output:  $\hat{\sigma}$ , selectivity estimation for  $T$ ;
1. if  $T$  is in  $L$ 
   return the associated count;
2. else
   pick a pair of  $T$ 's nodes( $v_1, v_2$ ) having degree of 1;
   remove  $v_1$  from  $T$  to get  $T_1$ ,
   remove  $v_2$  from  $T$  to get  $T_2$ ;
   evaluate  $T_3 = T_1 \cap T_2$ ;
    $\hat{\sigma} = \frac{Estimate(T_1, L) * Estimate(T_2, L)}{Estimate(T_3, L)}$ 

```

**Fig. 3.** Algorithm for recursive decomposition estimator

**Fast Fixed-sized Decomposition Scheme** Assuming we can keep the information of all subtrees no larger than  $k$  in the lattice summary, we can decompose a large query  $T$  in the following way: We use small fixed-sized subtrees to progressively cover  $T$ . First, we sort all nodes in the twig in pre-order fashion. Then we choose a  $k$ -subtree of  $T$  to cover the first  $k$  nodes. Let the covered portion of  $T$  be denoted as  $T_c$ . At each following step we cover a new node  $v$  using  $T_{new}$ , where all the nodes of  $T_{new}$  is a subset of  $T_c$  except  $v$ . Correspondingly, we update  $T_c$  as the union of the previous  $T_c$  and  $T_{new}$ . Thus,  $T_c$  will progressively grow until it covers all the nodes in  $T$ . Also, it holds that the part common between  $T_c$  and  $T_{new}$  is a  $(k - 1)$ -subtree. Clearly,  $T$  can be covered by exactly  $(size(T) - k + 1)$   $k$ -subtrees. The correlation between two subtree patterns is captured by their common part. In Figure 2b, we present an example of this decomposition. Newly covered nodes are highlighted at each step. Figure 4 presents the formal algorithm of the fixed-sized decomposition scheme.

The correctness of the above algorithm is formally stated as Lemma 2. Furthermore, Lemma 3 describes the corresponding selectivity estimator using such a decomposition scheme. Again, the detailed proofs of Lemma 2 and 3 can be found in the full version of this paper.

**Lemma 2.** *Given a rooted ordered labeled tree  $T$  of size  $n$ , it can be covered by  $n - k + 1$  of its subtrees of size  $k$  ( $n > k$ ), i.e.,  $T_1, T_2, \dots, T_i, \dots, T_{n-k+1}$ , such that  $T_i \cap (\bigcup_{j=1}^{i-1} T_j)$  is a  $(k - 1)$ -subtree.*

**Lemma 3.** *Assume we have a twig query  $T$  decomposed into  $k$ -subtrees, i.e.,  $T_1, T_2, \dots, T_i, \dots, T_{n-k+1}$ , and,  $C_{i-1} = T_i \cap (\bigcup_{j=1}^{i-1} T_j)$ ,  $2 < i \leq n - k + 1$ . Then the selectivity of  $T$  may be estimated as follows:*

$$\hat{\sigma}(T) = \frac{\prod_{i=1}^{n-k+1} \sigma(T_i)}{\prod_{j=1}^{n-k} \sigma(C_j)}$$

The advantage of this scheme is that it is very simple and the decomposition is very fast. In reality however, the lattice summary does not necessarily store all patterns up

```

Algorithm: FSD( $T, k$ )
Input:  $T$ , an XML twig of size  $n$ ;  $k$ , a fixed size;
Output:  $D$ , a set of  $k$ -subtrees satisfying the condition
in Theorem 2;
1. Order all nodes of  $T$  according to pre-order:
   i.e.,  $v_1, v_2, v_3, \dots, v_n$ ;
2. Choose the subtree  $t_1$  consisting of the first  $k$  nodes
   from the node list and label them as covered;
   //  $t_1$  must be a valid subtree;
   Initialize  $T_c$  by  $t_1$ , add  $t_1$  to  $D$ ;
3. for each remaining uncovered node  $v_i$ :
4.   pick a subtree  $t_i$  containing  $v_i$  as the
      rightmost node, all other nodes are from  $T_c$ ;
5.   add  $v_i$  to  $T_c$ , label  $v_i$  as covered
      and add  $t_i$  to  $D$ ;
6. return  $D$ ;

```

**Fig. 4.** Fixed-sized decomposition algorithm

to some size. Thus, the above decomposition can not be applied directly. To overcome this problem, we devise a hybrid version of this scheme and the recursive decomposition scheme with voting. The hybrid scheme works as follows: For a large twig, we first decompose it into fixed-sized sub-twigs and then use the recursive decomposition scheme with voting to estimate the selectivity of all of these sub-twigs. Finally, we use Lemma 3 to obtain the estimation for the original query. The advantage of this scheme is that it is much faster than the recursive decomposition scheme with voting. Additionally, it utilizes the summary information more effectively through voting, compared to the recursive decomposition scheme without voting. We call this hybrid version the *fast fixed-sized decomposition scheme* and refer it as *fast decomposition* in Section 4.

### 3.3 Building Basis Statistics

The summary records the occurrence statistics of basic twigs. There exists redundancy in the summary that can be pruned to reduce its size. With this in mind, we formally define the notion of a  $\delta$ -derivable pattern.

**Definition 2.** A twig pattern is  $\delta$ -derivable if and only if its true selectivity is within an error tolerance of  $\delta$  to its expected selectivity (according to *TreeLattice*).

By Definition 2, 0-derivable ( $\delta$ -derivable with  $\delta = 0$ ) patterns have the exact true selectivity as their expected selectivity. It is therefore safe to prune away the 0-derivable patterns from the lattice summary without sacrificing the quality of the estimations. This observation is formally stated as Lemma 4. As a result, we have more space to store more non-derivable patterns in the lattice summary.

**Lemma 4.** The estimation given by *TreeLattice* with a lattice summary  $L$  is exactly the same as that when 0-derivable patterns are removed from  $L$ .

**Proof:** The proof is trivial and is omitted.  $\square$

The above idea can be generalized by varying  $\delta$ , thereby controlling the trade-off between accuracy and memory utilization. We build the basis statistics in a bottom-up fashion. We collect the selectivity information of small twigs first, followed by the larger twigs. Essentially, we give more priority to smaller twigs, since they are more basic building blocks. Furthermore, at each level, we give priority to the more frequent twigs, as they are more important in capturing the overall twig distribution. Note that we only keep the information of non-derivable patterns in the lattice summary. Figure 5 presents the formal algorithm of building the basis statistics.

```

Algorithm: TreeLattice-Build ( $D, S, \delta$ )
Input: XML document  $D$ ; space budget  $S$ ; error tolerance  $\delta$ ;
Output: TreeLattice summary  $L$  of size  $\leq S$ ;
1. Obtain all 1-subtree and 2-subtree patterns in  $D$ 
   and their counts; Use them to initialize  $L$ ;
2.  $k = 3$ ;
3. While  $k < MAX\_LEVEL$ 
4.   Obtain all  $k$ -subtree patterns in  $D$  and their counts;
5.   Sort these patterns in decreasing order of their counts;
6.   For each  $k$ -subtree pattern  $p$ :
7.     Estimate  $\sigma(p)$  and compute the estimation
       error  $e$ ;
8.     if  $e > \delta$  then add  $p$  to  $L$ ;
9.     if  $size(L) \geq S$  exit;
7.    $k++$ ;
8. return  $L$ ;

```

**Fig. 5.** Algorithm TreeLattice-Build

## 4 Experiments

In this section, we examine the performance of our proposed approach for XML twig selectivity estimation on synthetic and real-life datasets. We compare our approach with TreeSketches, a state-of-the-art scheme [2].

### 4.1 Experimental Setup

All the experiments were conducted on a Pentium 4 2.66GHz machine with 1GB RAM running Linux 2.6.8. Below we detail the datasets, workloads and error metric considered in our evaluation.

**Datasets:** We use four publicly available datasets in our experiments: *Nasa*, a real-life dataset converted from legacy flat-file format into XML and made available to the public; *PSD* (Protein Sequence Database), a real-life dataset of integrated collection of functionally annotated protein sequences; *XMark*, a synthetic dataset that models transactions in an on-line auction site and *IMDB*, a real-life dataset from the Internet Movie Database Project. We would like to note that for the PSD dataset, both algorithms take a long time to process, so we present results on a sample. The main characteristics of the datasets are summarized in Table 1.

**Query Workloads:** In our experiments, we consider three different kinds of workloads: random, frequent-twig and negative-query. Regardless of workload, the first step is to enumerate all possible queries for a given dataset. This set of queries is further partitioned, where each partition corresponds to twig queries of a certain size. For the *random* workload, we sample a fixed amount from each partition under a uniform random distribution to yield a total of 1000 queries. This level-wise partitioning and sampling also enables us to evaluate the performance of our strategies, in particular their error propagation, in a controlled manner.

For the *frequent-twig* workload, we pick the most frequent 1000 twig queries as the workload. An alternative strategy would be to sample twigs as a function of the frequency of occurrence (a stratified sampling model). However, we observed little difference in the performance of these two frequency-based strategies and thus limit our discussion to the frequent-twig workload. Twig queries in the frequent-twig workload will have large selectivity rates, as expected.

We also generate and evaluate various *negative-query* workloads (workloads exclusively consisting of queries with zero selectivity). To generate these workloads, we followed the initial step of enumerating all possible queries. For each twig we then replaced node labels in accordance with their frequency of occurrence. More frequent labels are used for replacement more often so there is a greater chance for erroneous predictions (since sub-twigs are more likely to occur frequently). We then filter those queries whose selectivity is above 0. Once again we limit the workloads to be of size 1000. Experimental results show that TreeSketches is always accurate (100% of the time), and that TreeLattice is almost always as accurate (99% of the time), and returns the correct answer (zero). There is little difference between these two strategies for negative workloads, so we do not consider this workload further.

**Error Metric:** We quantify the accuracy of estimations using the average absolute relative error over all queries in the workload. The absolute relative error is defined as  $|\sigma - \hat{\sigma}| / \max(s, \sigma)$ , where the sanity bound  $s$  is used to avoid the artificially high percentages of low selectivity queries. Following common practice [2, 1], we set  $s$  to be the 10-percentile of true query counts. We use a lower bound of 10 if  $s$  should fall below that value.

## 4.2 Accuracy of Estimators

Here we examine the accuracy of the estimators on our workloads. For both TreeLattice and TreeSketches, we limit the summary size to 50KB. Figures 6a-d show the average selectivity estimation error on various frequent-twig workloads for all four datasets.

An obvious trend that stands out is that as the size of the twig query increases, the quality of the estimation decreases. This is not surprising, since the estimation errors grow for larger-sized queries for both strategies. Specifically for TreeLattice, the smaller sized queries are closer to the lattice boundary (exact information maintained in the summary) and thus subject to less estimation error. In contrast, for larger queries, depending on the number of decomposition and estimation steps, the error will accumulate, finally affecting the quality of the estimations. On the Nasa dataset, for example, the recursive decomposition estimator yields very accurate estimations on frequent-twig workloads of size 5 and 6, with error 0.0% and 4.2%, respectively. In contrast, on the

frequent-twig workloads of size 8 and 9, the error increases to 11.6% and 17.8%, respectively. The effect of error accumulation can be clearly seen from the results. The other two estimators have a similar trend when working on various workloads for all datasets.

We would like to note that the voting scheme refines the estimations effectively by mitigating the error propagation. The recursive decomposition estimator with voting usually yields the most accurate estimations. Additionally, one should note that the estimations returned by the fast decomposition estimator is very similar to that returned by recursive decomposition estimator with voting.

When comparing the two strategies, it can be observed that TreeLattice significantly outperforms TreeSketches for both the Nasa and PSD workload on all query sizes. On the XMark dataset, TreeLattice is near perfect and TreeSketches is marginally worse (note the Y-axis scale). On the IMDB workload, TreeSketches outperforms TreeLattice significantly on larger query sizes. On smaller query sizes, the difference is not as significant. Note that on Nasa, PSD and XMark, in most cases, even the weakest estimation strategy in TreeLattice, recursive decomposition estimator without voting, does better than TreeSketches.

Figures 7a-d show the average selectivity estimation error on the random workloads for all four datasets. The trends are very similar to the ones observed for the frequent-twig workloads. Two differences are that TreeLattice is closer in performance to TreeSketches on the IMDB dataset and on the XMark dataset TreeSketches performs poorly (the errors are well above 100% in some cases).

To examine a possible outlier effect, we plotted the cumulative distribution function of the errors. Figures 8a-d present the results for frequent-twig workloads. The results are consistent with Figure 6, showing that TreeLattice outperforms TreeSketches consistently on all datasets except IMDB. The results on random workloads are similar and are omitted in the interest of space. The complete results can be found in the full version of this paper [17].

In conclusion, these results demonstrate that TreeLattice is effective in summarizing the distribution of the underlying twigs. Furthermore, we show that TreeLattice is effective in processing both frequent and infrequent twig queries. When the query size is increased, the quality of the estimation is reduced (due to the error propagation). Specifically among the strategies evaluated, the recursive decomposition with voting estimator usually yields the best estimations. Finally, the fast decomposition estimator yields close estimations to the recursive decomposition estimator with voting.

### 4.3 Impact of Varying Summary Size

In this experiment, we measure the estimation error while varying the summary size. We use a frequent-twig workload containing frequent 8-twig queries. Figures 9a-d show the average selectivity estimation error when varying the summary size for Nasa, PSD, XMark and IMDB, respectively. As expected, we observe that an increase in the size of the summary yields more accurate estimations. As before, TreeLattice works extremely well for Nasa, PSD and XMark. An important point here is that the estimation error for these datasets is well below 10% when we use at least a 40KB summary. For the IMDB dataset, TreeSketches is better than TreeLattice.

#### 4.4 Implications on Estimation Time

In this experiment, we compare TreeLattice against TreeSketches in terms of selectivity-estimation time. Figures 10a-d present the response times of the different approaches on frequent-twig workloads for Nasa, IMDB, PSD and XMark, respectively. The results on random workloads are similar and are omitted in the interest of space, though the complete results can be found in the full version of the paper. As seen in the figures, in most cases, all TreeLattice estimators are much more efficient than TreeSketches. Specifically, TreeLattice runs extremely fast when processing relatively small twig queries. As we increase the query size of the workload, the recursive decomposition estimator with voting becomes much slower. The degradation of response time becomes more significant as we increase the size of the twig queries. This is not surprising, since the number of all possible decompositions increases exponentially with the number of recursion levels. The recursive decomposition without voting is fastest. However in terms of accuracy, this strategy is the weakest among the three. *The overall performance of the fast decomposition estimator is clearly the best since it is close to recursive decomposition estimator in terms of response time, and it is close to the recursive decomposition estimator with voting in terms of estimation quality.*

#### 4.5 Impact of $\delta$ -derivable Pruning

The pruning strategy we describe earlier allows us to replace  $\delta$ -derivable patterns with non-derivable patterns in the lattice summary. Here we examine the potential benefits of this strategy on the IMDB dataset. We use the frequent-twig workload for this experiment. Figure 11 presents the estimation quality at the different  $\delta$ -levels. All summary sizes are fixed at 50KB. As can be seen in the figure, when we increase  $\delta$ , the estimations become more accurate for large twig queries. This comes at a small sacrifice in the estimation accuracy for small twig queries. If the estimation error for the small twig queries is tolerable, we can continue to increase  $\delta$  for the benefit of improved estimations for large twig queries. If we consider a single large workload consisting of a uniform number of different-sized queries (4 to 9),  $\delta = 20\%$  gives the lowest average error (17.9%).

#### 4.6 Comparison of Summary Construction Times

In this experiment, we evaluate the cost of constructing the summary. In TreeSketches, this is a very expensive operation as it involves a bottom-up clustering of similar substructures in the XML data tree. In contrast, our approach relies on fast off-the-shelf efficient tree-mining algorithms to build the summary. Table 3 presents the time required by both approaches to construct a 50KB summary on each of the four datasets. The advantage of our approach over TreeSketches is quite telling—with an improvement of about one order of magnitude.

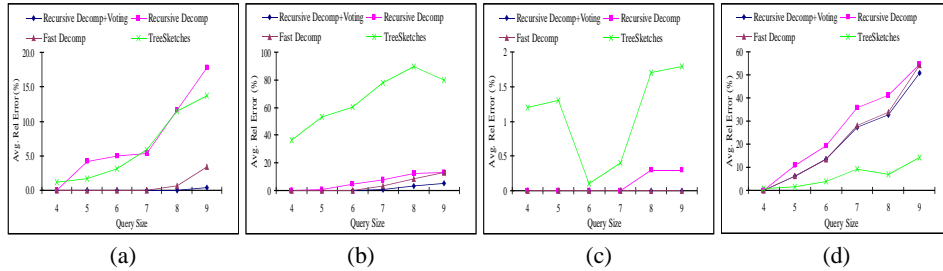
#### 4.7 Result Summary and Rationale

The experimental results have shown that TreeLattice is very effective and efficient in estimating selectivity of the XML twig queries. In most cases, TreeLattice outperforms TreeSketches in terms of both accuracy and response time. In addition, pruning  $\delta$ -derivable patterns can further refine the selectivity estimation for large queries. We also notice that TreeLattice is outperformed by TreeSketches on IMDB, though it still

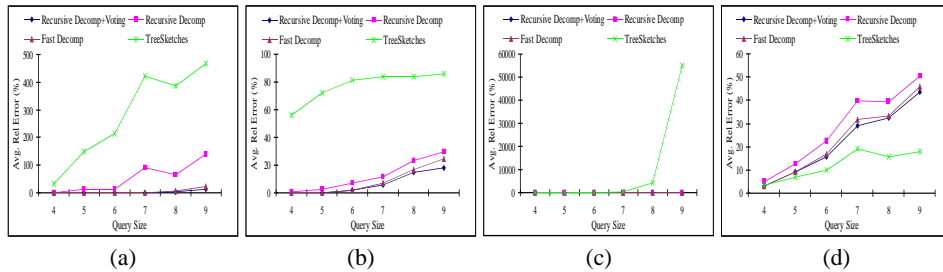
yields reasonable estimations. Here we attempt to explain the rationale behind these results.

We know that TreeLattice is based on the conditional independence assumption of twig growing. If real XML data satisfy this assumption well, then TreeLattice will perform well. On the other hand, if the assumption does not hold, it will not. From the experimental results, it would appear that Nasa, PSD and XMark satisfy the assumption well and IMDB does not. Our expectation is supported by Table 4, which lists the number of patterns satisfying the assumption at different lattice levels on the four datasets. From the table, we see for the Nasa, PSD and XMark datasets, the ratio of 0-derivable patterns to total patterns is quite high, meaning they satisfy the assumption well. In contrast, the ratio on IMDB is much lower, which is reflected in the results.

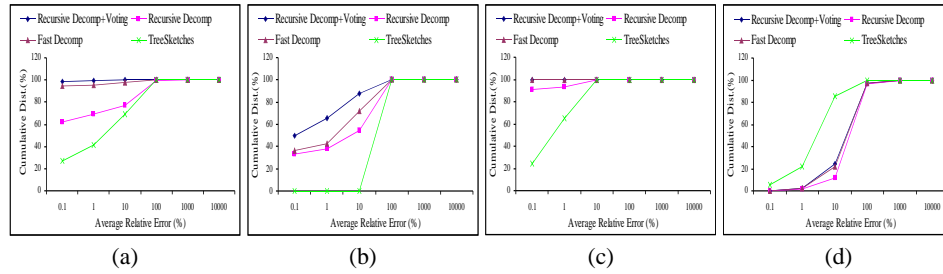
Now let us take a closer look at TreeSketches. The TreeSketches synopsis is constructed by a bottom-up clustering of the similar substructures in the XML data tree. In it, an edge  $(x, y)$  with weight  $\alpha$  represents that on *average*, each node in set  $x$  has  $\alpha$  children in set  $y$ . Assume we have  $n$  nodes in set  $x$ , and the nodes have  $\alpha_1, \dots, \alpha_n$ , children in set  $y$ , respectively. If there are many similar substructures in the XML data tree found by bottom-up clustering, then TreeSketches should work very well (e.g., IMDB). On a detailed examination of IMDB, we find this to be the case. However, if this does not hold, then one is forced to cluster substructures that are not very similar in order to compress the XML data tree. This results in a large variance of  $\alpha_i$  which leads to larger errors that propagate rapidly. We believe this explains the poor performance of TreeSketches on the other three datasets.



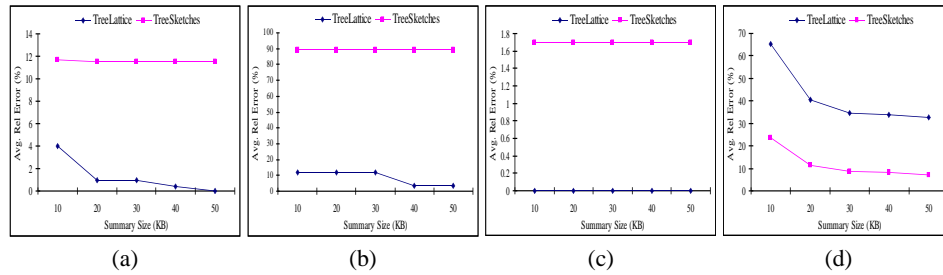
**Fig. 6.** Average estimation error on frequent-twig workload: (a)Nasa (b)PSD (c)XMark (d)IMDB



**Fig. 7.** Average estimation error on random workload: (a)Nasa (b)PSD (c)XMark (d)IMDB



**Fig. 8.** Average estimation error distribution on frequent-twig workload: (a)Nasa (b)PSD (c)XMark (d)IMDB



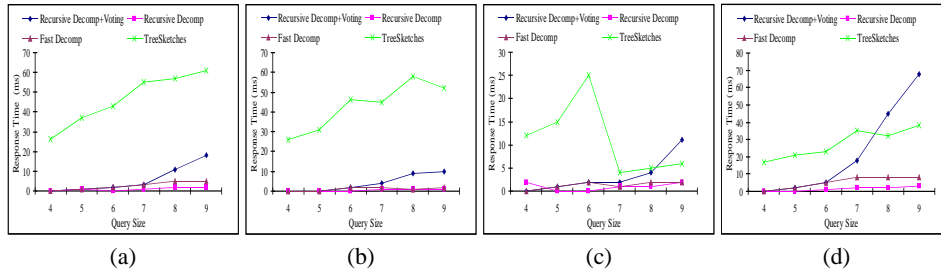
**Fig. 9.** Average estimation error when varying summary size: (a)Nasa (b)PSD (c)XMark (d)IMDB

Dataset	Elements	File Size(MB)
Nasa	476646	24
PSD	335193	12
XMark	167864	12
IMDB	155898	7

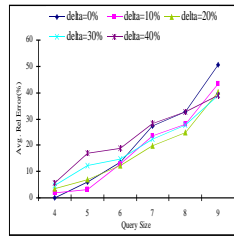
**Table 1.** Dataset characteristics

## 5 Related Work

Chen *et al.* [3] were among the first to study the problem of estimating twig counts. They propose the Correlated Sub-path Tree (CST) method for estimating the selectivity of XML twig queries. A CST is a suffix tree-based data structure used to store all the paths up to certain length. To estimate the selectivity of a given twig query, this approach needs to decompose a twig into a set of paths stored in the CST. Note that even though both the CST and our TreeLattice approach depend on decomposing a large twig into basic twigs, they are quite different in several respects. First, our approach utilizes the *subtrees* instead of paths as the summary of an XML document. Our results have shown that these subtrees capture the structure of an XML document very effectively. In contrast, in order to perform selectivity estimation, CST has to store additional information, called *set hashing signature*, in order to capture the correlation among paths. Our approach is essentially a generalization of the Markov model-based approach for



**Fig. 10.** Average response time on frequent-twig workload: (a)Nasa (b)PSD (c)XMark (d)IMDB



**Fig. 11.** Average selectivity estimation error when varying delta (IMDB)

Dataset	Nasa		PSD		XMark		IMDB	
	Frequent	Random	Frequent	Random	Frequent	Random	Frequent	Random
4	5377	4073	6601	3321	2774	1722	3519	784
5	8282	3742	11563	2827	5995	2058	40703	985
6	21334	3978	28160	2398	6347	3251	11815	1982
7	58920	4004	68877	2383	169993	6641	17193	2937
8	29558	2855	129892	2920	288944	10394	29962	3559
9	18814	2608	148993	2464	281808	5748	37963	5825

**Table 2.** Workload characteristics (average no. of binding tuples)

Dataset	TreeLattice	TreeSketches
Nasa	10	80
PSD	21	102
XMark	15	78
IMDB	1	8

**Table 3.** Summary construction time (in minutes)

Dataset	Nasa		PSD		XMark		IMDB	
	# total	# 0-derivable	# total	# 0-derivable	# total	# 0-derivable	# total	# 0-derivable
3	213	174	282	201	365	302	877	156
4	668	434	1284	1016	1283	1138	9839	3625
5	2296	1866	6728	5778	4378	3948	-	-
6	8274	7768	34976	31580	14492	13251	-	-
7	30492	29232	-	-	46628	43373	-	-

**Table 4.** Number of total and 0-derivable patterns on four datasets

XML path selectivity estimation. When dealing with XML path queries, TreeLattice yields the same selectivity estimation as the Markov model-based approaches, which have been shown to be more effective than the CST-based approach [10].

XSketches [16] exploits localized graph stability in a graph-synopsis model to approximate path and branching distribution in an XML data graph. Its successor integrates support for value constraints as well, by using a multidimensional synopsis to capture value correlations [15]. They augment the XSketches model with new distribution information [1] to estimate the selectivity of XML twig queries and show that XSketches performs better than CST, yielding estimates with significantly lower estimation error.

TreeSketches [2], a successor of XSketches, clusters similar fragments of XML data together to generate its synopsis. The granularity of the clustering depends on the memory budget. Also, it outperforms its predecessors in terms of both accuracy and construction time. We note that the scope of TreeSketches is much broader than that of TreeLattice, since they are able to handle more general twigs (containing `//` operator).

A particular case of the twig query is the XML path query. The wide use of XML path queries has motivated many researches on estimating their selectivity. The Lore system [8] is one of the earliest works in this direction. It stores statistics of all distinct paths up to length  $m$ , with  $m$  being a tunable parameter. Selectivity of paths longer than  $m$  are estimated assuming the Markov property. Abounaga *et al.* [10], extends the idea used by Lore system in their Markov table method. It consists of a set of pruning and aggregation techniques on the statistics used in the Lore system and therefore offers an improvement by reducing the space requirements. Abounaga *et al.* [10], also propose a tree-based method known as the *path tree*, for estimating the selectivity of XML paths without data values. A path tree is a summarized form of the XML data tree. Compared with the Markov table method, this approach is inferior in terms of estimation accuracy for real datasets [10].

XPathLearner [9], is an on-line, self-tuning, Markov table-based approach used to estimate the selectivity of XML paths. The statistics of the data are collected in an on-line fashion, thus it is workload-aware. By design, our approach is also incremental in nature and can maintain summaries on-line, though we do not evaluate this aspect here. Our method is a generalization of these Markov model-based approaches for more complex twig queries. Recently, Wang *et al.* [12] propose the use of Bloom Histograms to estimate XML path selectivity. It is the first approach that gives a theoretical bound on the estimation error. However, it does not handle twig queries.

## 6 Conclusions and Future Work

In this paper, we have described a new approach, TreeLattice, to estimate the selectivity of XML twig queries with branching predicates. TreeLattice is shown to be comparable or better than TreeSketches in terms of estimation accuracy. Moreover, our technique is significantly faster both in terms of summary construction and in terms of selectivity estimation. Furthermore, we have provided theoretical foundations for the estimation process and have shown that TreeLattice subsumes the successful Markov model-based XML path selectivity estimation approach as a special case.

In the future, we will study the following issues: First, we would like to extend TreeLattice to handle more complex twig queries with recursion predicates (`//` operator).

In this case, we are allowed to grow the twig in a more relaxed fashion. We conjecture that the conditional independence assumption of tree growing will still hold even for this case. Second, an error bound associated with the estimation would be very useful and we have made some initial progress towards this end. Third, we would like to adapt TreeLattice in a manner similar to XPathLearner, where information learned from an on-line workload can dynamically guide what is to be maintained in the summary structure.

## References

1. Polyzotis, N., Garofalakis, M., Ioannidis, Y.: Selectivity estimation for xml twigs. In: Proceedings of the International Conference on Data Engineering. (2004)
2. Polyzotis, N., Garofalakis, M., Ioannidis, Y.: Approximate xml query answers. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. (2004)
3. Chen, Z., H.V.Jagadish, *et al.*: Counting twig matches in a tree. In: Proceedings of the International Conference on Data Engineering. (2001)
4. Chen, Z., Korn, F., *et al.*: Selectivity estimation for boolean queries. In: Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. (2000)
5. Jagadish, H., Kapitskaia, O., *et al.*: Multi-dimensional substring selectivity estimation. In: Proceedings of the International Conference on Very Large Data Bases. (1999)
6. Jagadish, H., T.Ng, R., *et al.*: Substring selectivity estimation. In: Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. (1999)
7. P.Krishnan, Vitter, J.S., Iyer, B.: Estimating alphanumeric selectivity in the presence of wildcards. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. (1996)
8. McHugh, J., Widom, J.: Query optimization for xml. In: Proceedings of the International Conference on Very Large Data Bases. (1999)
9. Lim, L., Wang, M., *et al.*: Xpathlearner: An on-line self-tuning markov histogram for xml path selectivity estimation. In: Proceedings of the International Conference on Very Large Data Bases. (2002)
10. Aboulnaga, A., Alameldeen, A.R., Naughton, J.F.: Estimating the selectivity of xml path expressions for internet scale applications. In: Proceedings of the International Conference on Very Large Data Bases. (2001)
11. Wu, Y., Patel, J.M., Jagadish, H.: Estimating answer sizes for xml queries. In: Proceedings of the International Conference on Extending Database Technology (EDBT). (2002)
12. Wang, W., Jiang, H., Lu, H., Yu, J.X.: Bloom histogram: Path selectivity estimation for xml data with updates. In: Proceedings of the International Conference on Very Large Data Bases. (2004)
13. Jiang, W., Jiang, H., Lu, H., Yu, J.X.: Containment join size estimation: Models and methods. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. (2003)
14. Freire, J., Haritsa, J.R., *et al.*: Statix: Making xml count. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. (2002)
15. Polyzotis, N., Garofalakis, M.: Structure and value synopses for xml data graphs. In: Proceedings of the International Conference on Very Large Data Bases. (2002)
16. Polyzotis, N., Garofalakis, M.: Statistical synopses for graph-structured xml databases. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. (2002)
17. Wang, C., Parthasarathy, S., Jin, R.: A decomposition-based probabilistic framework for estimating the selectivity of xml twig queries. In: The Ohio State University, Technical Report. (2005)