A Sampling Approach for XML Query Selectivity Estimation

Cheng Luo
Department of Mathematics and Computer Science
Coppin State University
2500 West North Avenue,
Baltimore, MD, 21216, U.S.A.
cluo@coppin.edu

Zhewei Jiang
Computer Science Department
Southern Illinois University Carbondale
Carbondale, IL 62901, U.S.A.
zjiang@cs.siu.edu

Wen-Chi Hou
Computer Science Department
Southern Illinois University Carbondale
Carbondale, IL 62901, U.S.A.
hou@cs.siu.edu

Feng Yu
Computer Science Department
Southern Illinois University Carbondale
Carbondale, IL 62901, U.S.A.
fyu@cs.siu.edu

Qiang Zhu
Department of Computer and Information Science
University of Michigan-Dearborn
Dearborn MI 48128, U.S.A.
qzhu@umich.edu

ABSTRACT

As the Extensible Markup Language (XML) rapidly establishes itself as the de facto standard for presenting, storing, and exchanging data on the Internet, large volume of XML data and their supporting facilities start to surface. A fast and accurate selectivity estimation mechanism is of practical importance because selectivity estimation plays a fundamental role in XML query optimization. Recently proposed techniques are all based on some forms of structure synopses that could be time-consuming to build and not effective for summarizing complex structure relationships. In this research, we propose an innovative sampling method that can capture the tree structures and intricate relationships among nodes in a simple and effective way. The derived sample tree is stored as a synopsis for selectivity estimation. Extensive experimental results show that, in comparison with the state-of-the-art structure synopses, specifically the TreeSketch and Xseed synopses, our sample tree synopsis applies to a broader range of query types, requires several orders of magnitude less construction time, and generates estimates with considerably better precision for complex datasets.

1. INTRODUCTION

Recently, the Extensible Markup Language (XML) has rapidly established itself as the de facto standard for presenting, storing, and exchanging data on the Internet. XML queries are often expressed as path expressions because of the tree-structured nature of XML data. These path expressions generally consist of linear path expressions that describe linear paths without branches, and branched path expressions that describe branched paths. The latter, commonly known as twig queries, are a very common and general form of XML queries.

In recent years, considerable research efforts [2, 4, 13, 15, 17] have been placed on the design of efficient algorithms for twig query evaluation. Comparatively, much less attention has been paid to the optimization of XML twig queries, which, however, is equally important. The primary task of query optimization is to select efficient query execution plans. The decision is usually based on the selectivity estimation with the rationale that evaluating more selective portions of the queries first may effectively rule out unqualified data.

Of the research work in estimating selectivity of XML path expressions, the majority [16, 18, 19, 24] has focused on the linear path expressions. It is not clear how these approaches can be extended to XML twig queries. Recently, several structure synopses, such as Correlated Suffix Trees [5], StatiX [7], Twig-Xsketch [21], TreeSketch [20], and Xseed [25], have been proposed for twig query selectivity estimation. These synopses generally store some forms of compressed tree structures and simple statistics such as node counts, child node counts, etc. Due to the compression of information, selectivity estimation heavily relies on the statistical assumptions of independence and uniformity. Consequently, they can suffer from poor accuracy when these assumptions are not valid.

Capturing relationships among nodes presents a stern challenge to structure synopses. Twig-Xsketch associates histograms with nodes to capture the distributions of child nodes. TreeSketch clusters nodes based on some predefined similarity measures so that the independence and uniformity assumptions can be reasonably applied. Xseed stores the uniformities of the datasets in the kernel and records the irregularities in the HET (Hyper-Edge Table). Xseed was shown to perform better than TreeSketch [25], which in

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the ACM. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permissions from the publisher, ACM.
EDBT 2009, March 24–26, 2009, Saint Petersburg, Russia.
Copyright 2009 ACM 978-1-60558-422-5/09/0003 ...$5.00

term outperformed the Twig-XSketch [20] in construction speed and estimation accuracy. However, even for the fastest Xseed, the synopsis construction can still be painfully slow for complex datasets. For example, our experiments show that on a computer with 3.4GHz CPU and 1GB RAM, the construction of the Xseed synopsis for the TreeBank dataset could not finish in 4 days. In addition, the maintenance of synopses can also be a concern especially when datasets undergo substantial changes that invalidate the overall optimality of the synopses. A large scale re-clustering may be then required for structure synopses, such as TreeSketch.

While sampling has been applied extensively in relational databases [3, 8, 9, 10, 11, 12, 14], there is virtually no work on sampling in XML databases except for the naive method [24] that randomly draws nodes from individual node sets (i.e., groups of nodes of the same type). Semi-structured XML data exhibit distinct characteristics, such as hierarchical structures and repeating fields, from the relational data, and thus may require a different perspective on sampling. In this research, we present an innovative perspective on sampling XML data. Instead of randomly drawing nodes, we propose to preserve as much the tree structure of the nodes as possible in sampling units. We believe such measure holds the key to the success of an effective sampling scheme. The derived sample, which is in fact a miniature of the data tree, is stored as a synopsis for selectivity estimation. The simplicity of the method manifests itself in construction of the synopsis. For example, in marked contrast to the more than 4 day construction time of both TreeSketch and Xseed, it takes only about 30 seconds to construct a representative sample tree for the TreeBank dataset. Therefore, when an XML database has undergone substantial changes, one can simply regenerate the sample tree with little effort. The sample tree can be evaluated by ordinary query evaluation algorithms for selectivity estimation, while the structure synopses generally require specialized estimation algorithms.

Statistical measures associated with the estimates, such as the confidence probability and intervals, can be derived from our sample tree, which are generally not possible for the structure synopsis approaches. This presents yet another advantage of our method. Extensive experiments have been conducted to study the performance of different methods. The experimental results show that in comparison with the TreeSketch and Xseed methods, our sampling method applies to a broader range of query types, requires several orders of magnitude less construction time, and generates estimates with considerably better precision for complex datasets.

The rest of the paper is organized as follows. Section 2 briefly reviews related research in selectivity estimation of XML twig queries. Section 3 discusses how to sample tree-structured data and deriving selectivity estimation from sample trees. Section 4 compares our sample tree synopsis with two modern structure synopses, namely the TreeSketch and Xseed. Detailed theoretical analyses and experimental results are presented. Finally, Section 5 concludes this paper.

2. RELATED WORK

In the early studies of XML selectivity estimation, much research work [18, 19, 24, 16] has focused on containment joins or linear path expressions. Although technically possible, it is not clear how these techniques can be effectively extended to tackle twig queries.

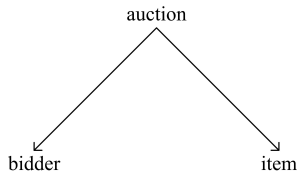

Figure 1: A twig query

There has also been some work dedicated to twig query selectivity estimation. In general, these approaches rely on structure synopses. For instance, Correlated Suffix Trees [5] use pruned subpath trees to summarize the path structures of XML data, while StatiX [7] adopts a one-dimensional histogram to capture the structural relationships of the elements in the XML schemas.

These approaches generally have statistical assumptions, such as independence and uniformity, as their integral parts. Nevertheless, their synopsis construction algorithms do not take these statistical assumptions into consideration [21]. Therefore, it is not clear if these assumptions still remain valid in the constructed synopses.

Polyzotis et al. proposed the Twig-Xsketch [21] and Tree Sketch [20] structure synopses for XML data. Twig-Xsketch is based on the Xsketch synopsis [18, 19], augmented with edge distribution information. It was shown [21] that Twig-Xsketch yields estimates with significantly smaller errors than those derived by Correlated Suffix Trees. By checking a pre-defined similarity measure during clustering so that the independence and uniformity assumptions can be more reasonably applied, TreeSketch was shown to be more accurate in estimation [20] than Twig-Xsketch. On the other hand, clustering with optimality check is computationally complex and time consuming.

Zhang et al. [25] recently proposed the Xseed synopsis to summarize the structural information of XML data. The information is stored in two structures, a kernel, which summarizes the uniform information, and an HET, which records the irregular information. By treating the structural information in a multi-layer manner, the Xseed synopsis is simpler and more accurate than the TreeSketch synopsis. Moreover, Xseed supports recursion by recording "recursion levels" and "recursive path expression" in the synopses. However, although the construction of Xseed is generally faster than that of TreeSketch, it is still time-consuming for complex datasets.

Sampling has been applied to relational databases extensively [3, 8, 9, 10, 11, 12, 14]. Despite the success in the relational paradigm, to the best of our knowledge, only Wang et al.[24] have applied sampling to XML databases for containment join size estimation. His method is not suitable for our application because its samples are drawn on the fly for the specific containment join queries posted, not pre-drawn as a representative of the entire data tree for all possible queries.

3. XML TWIG QUERY SIZE ESTIMATION

3.1 Assumptions and Definitions

An XML document is generally represented by an XML

data tree $\mathcal{T}$. We assume an XML data tree encompasses all the information of the original XML dataset, with attributes, elements, and text values in the dataset represented by nodes and their relationships by edges in the tree. Consequently, queries on an XML dataset can be specified against its XML data tree.

In this research, we consider linear path queries as well as twig queries. In contrast to linear path queries, twig queries are concerned with branched paths. We differentiate two types of twig queries, the regular twig queries and the existential twig queries, as they exhibit different complexities in selectivity estimation. A regular twig query can be specified either pictorially by a twig pattern, or by a declarative query language, such as XQuery. Figure 1 shows a twig pattern that looks for all 3-tuples $(auction, bidder, item)$ such that the "auction" node is the parent of the other two nodes, "bidder" and "item".

The Xpath language is capable of specifying existential twig queries, which differ from regular twig queries in that branches are only treated as existential structural constraints, while their quantitative occurrences ignored. Take an existential twig query auction[/bidder]/item, specified in Xpath, as an example. The branch auction/bidder is treated only as a structural constraint. The query returns all "item" nodes that have an "auction" parent node and at least one "bidder" sibling node. The number of occurrences of the sibling "bidder" nodes is immaterial. Consequently, the formulas for computing query result sizes of a regular twig query and an existential twig query would be quite different. For instance, if we assume there is only 1 "auction" node and it has 4 "bidder" child nodes and 6 "item" child nodes. Then the regular twig query would have $4 \times 6 = 24$ as the answer size while the above existential twig query would have an answer size of 6. In general, selectivity estimation for regular twigs is more difficult than that for existential twigs as the former requires more accurate information about the correlations between branches.

To differentiate the nodes in an XML data tree and in a query pattern, we call the former data nodes and the latter pattern nodes.

3.2 Construction of Sample Synopses

Drawing a representative sample from an XML data tree poses an interesting challenge to database researchers. Naive sampling schemes, such as randomly drawing nodes from a tree, can yield estimates with large variances, as evidenced by random sampling of relations for multi-join size estimation in relational databases. This seemingly complicated problem, however, has a simple solution in XML databases.

The ineffectiveness of random sampling is due to its destruction of correlations among elements, which are neatly captured by the tree structure. We observe that preserving the tree structure and relationships of nodes, as opposed to drawing nodes randomly, holds the key to a successful sampling for tree-structured data. With this principle in mind, we devise a simple, yet effective sampling scheme that preserves the tree structure of the data.

3.2.1 Subtree Sampling

The simplest but perhaps the most effective way to preserve the tree structures and node relationships is to include the entire subtrees in a sample. Subtrees from the data tree form the sampling units of this sampling scheme.

The general idea is to examine the number of data nodes for each tag name starting from the root level. If the number of data nodes for a tag is sufficiently large, a desired fraction of the data nodes are randomly selected using simple random sampling without replacement and then the entire subtrees rooted at these selected data nodes are included, as sampling units, in the sample. Hereafter, we shall call each such set of subtrees to which random sampling is applied a subtree group. If a tag has too few data nodes at the level under study, then all the data nodes for that tag at that level are kept and we move down the tree to check the next level. The paths from the root to the selected subtrees are also included in the sample to preserve the relationships among the sample subtrees.

The above sampling scheme, called subtree sampling scheme, ensures that the sizes of the subtrees in the same subtree group are similar. This is because the root nodes of these subtrees have the same tag name, i.e., they are nodes of the same type. Moreover, these root nodes reside in the same level. Consequently, subtrees in the same subtree group tend to have similar structures, thus similar sizes.

Based on this observation, the sampling fractions of the subtree groups $f_i's$, where f_i is the sampling fraction of the i-th subtree group, can be simply set to f_t, which is the sampling fraction of the whole data tree.

Proof: Let s_i be the size of the i-th subtree, s_o the size of the nodes that do not belong to any subtree group, s_t the size of the data tree, b the size of the budget. Then

$$b = \sum s_i f_i + s_o \tag{1}$$

and

$$s_t = \sum s_i + s_o \tag{2}$$

We divide Equation 1 by Equation 2

$$\frac{b}{s_t} = \frac{\sum s_i f_i + s_o}{\sum s_i + s_o}$$

Since usually $\sum s_i \gg s_o$ and $\sum s_i f_i \gg s_o$, therefore

$$\frac{b}{s_t} \doteq \frac{\sum s_i f_i}{\sum s_i}$$

if we assume all f_i's are equal, then

$$\frac{b}{s_t} \doteq \frac{\sum s_i f_i}{\sum s_i} = \frac{f_i \sum s_i}{\sum s_i} = f_i$$

finally

$$\frac{b}{s_t} = f_t \doteq f_i$$

If the number of nodes n for a tag satisfies the minimum requirement $n \times f_t \geq 1$, we consider it "sufficiently large" because one can draw at least one sample subtree for this tag. If the tag population is not sufficiently large, all of its nodes are kept, otherwise, simple random sampling without replacement is applied to this subtree group. The following examples illustrate this sampling scheme, called subtree sampling.

Example. Suppose we apply subtree sampling to the DBLP data tree shown in Figure 2. In the second level (i.e., the level below the root) of the tree, there are $10,000$ "book"

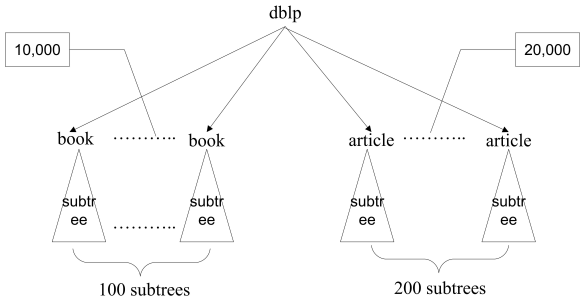

Figure 2: Subtree Sampling Example 1

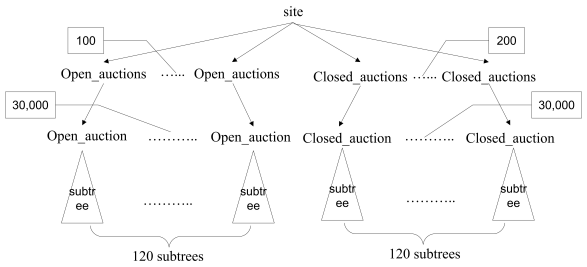

Figure 3: Subtree Sampling Example 2

nodes and 20,000 "article" nodes. Assume the sampling fraction is 1%. Since both tags have sufficiently large numbers of nodes, i.e., $10,000 \times 1\% \geq 1$ and $20,000 \times 1\% \geq 1$, we randomly select 100 "book" nodes and 200 "article" nodes from the second level and include their subtrees as the sample. Note that we also include the paths from the root to the subtrees to preserve the hierarchy.

Example. A more complex scenario arises when we apply subtree sampling to the Xmark data tree shown in Figure 3. Assume the sampling fraction is 0.4%. In the second level of the data tree, there are only 100 "Open_auctions" nodes and 200 "Closed_auctions" nodes. Since the numbers of nodes for both tags at the second level are not sufficiently large, i.e., $100 \times 0.4\% < 1$ and $200 \times 0.4\% < 1$, we move down to examine the third level. In the third level, there are 30,000 "Open_auction" nodes and 30,000 "Closed_auction" nodes. We randomly select 120 nodes for each type and include their subtrees as the sample. Again, the paths from the root to the subtrees are also included in the sample.

Figure 4 describes the Subtree Sampling method in pseudo code. The method takes two parameters, a data tree and its sampling fraction.

3.3 Sample Tree Evaluation Algorithm

It is worth noting that our XML sample trees are just a portion of the original XML data tree. These sample trees differ only in magnitude from the original XML data tree. Therefore, ordinary twig query evaluation methods, such as TwigStack [2] and TJFast [15], can be applied directly to the sample trees synopsis to derive approximate answers to aggregation queries as well as queries that return twig matches. Note that sample trees can be stored in any formats that conform to the requirements of the underlying query evaluation algorithms. For instance, it would be stored as a set of streams if TwigStack [2] is used.

```
SubtreeSampling(T, f)
  Input: XML data tree T, and sampling fraction f
  begin
    sampleNodes=φ;
    tagQueue=φ;
    tagQueue.Enqueue(root.tag);
    while (!taqQueue.empty()) do
      nodeTag=tagQueue.Dequeue();
      if (the number of nodes of nodeTag ×f > 1)
      then
        randomly sample a portion of nodes by f;
        sampleNodes ∪ = the subtrees rooted at the
        sampled nodes and their paths to the root;
      end
      else
        sampleNodes ∪ = all the nodes of nodeTag;
        tagQueue.Enqueue(all the child tags of
        nodeTag);
      end
    end
  end
```

Figure 4: Algorithm SubtreeSampling

3.4 Selectivity Estimation

The sample tree can be used to derive approximate answers for aggregation queries as well as non-aggregation queries. However, for simplicity, we shall restrict ourselves to query cardinality estimation (or selectivity estimation), which is basically a COUNT query. Discussion on other aggregation queries, such as SUM and AVERAGE, are similar.

3.4.1 Estimator

Let n be the total number of subtrees in an XML data tree, and m the number of subtrees sampled. For example, consider the DBLP data tree shown in Figure 2. There are 10,000 and 20,000 subtrees in the book subtree group and article subtree group, respectively, so $n = 10,000+20,000 = 30,000$ and $m = 100 + 200 = 300$. Here, a uniform 1% sampling fraction is applied to each subtree group.

Subtree Sampling selects subtrees at the highest tree level possible and thus is likely to keep related information in individual subtrees. Consequently, it can be expected that a twig match generally comes from a single sample subtree, However, it is possible that some of the matches span multiple sample subtrees. Let B be the maximum number of subtrees that a query match can span. It can be observed easily that B is upper-bounded by the number of root-to-leaf paths in the query.

Let Y be the number of matches in the entire XML data tree. Let y_i be the number of matches that span i sample subtrees, $1 \leq i \leq B$, in the sample tree synopsis. Since there are totally $C(n,i)$ combinations of i-subtrees in the data tree and only $C(m,i)$ such combinations are included in the sample, a reasonable estimate of the number of matches in the data tree is $y_i \times \frac{C(n,i)}{C(m,i)}$. The twig query cardinality estimation formula is given by summing up all the i's:

$$Est = \sum_{1 \leq i \leq B} \frac{C(n,i)}{C(m,i)} \times y_i \quad (3)$$

3.4.2 Statistical Properties of the Estimator

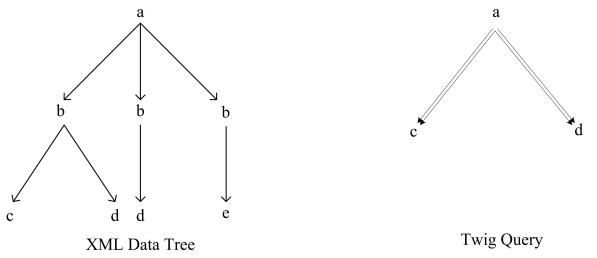

Figure 5: Unbiased Estimator

This section discusses the statistical properties of the estimator.

Unbiasedness. A condition regarding the sample size, $m \geq B$, is required to guarantee that all matches (that span no more than B subtrees) have a chance to be sampled. This condition is generally satisfied naturally as m is usually large while B is small.

THEOREM 1. *If $m \geq B$, Est is an unbiased estimator of the cardinality of the twig query.*

Proof: [sketch]. Let Y_i be the number of query matches in the document that span i subtrees, and $Est_i = y_i \times \frac{C(n,i)}{C(m,i)}$ the estimator for Y_i. We argue that each Est_i is an unbiased estimator of Y_i as follows. Since sample subtrees are picked randomly, all subtrees in the document are equally likely to be selected. Consequently, all i-subtree combinations are equally likely to be selected, each with a probability of $1/C(n,i)$. In each sample tree, there are only $C(m,i)$ such i-subtrees combinations selected. Thus, $E(y_i) = Y_i \times C(m,i)/C(n,i)$. Therefore, $E(Est_i) = Y_i$ and thus $E(Est) = \sum_i Y_i = Y$, which is the total number of query matches in the document.

Example. Consider the XML data tree and a twig query shown in Figure 5. Since the number of root-to-leaf paths in this twig query is 2, then $B \leq 2$. Assume we sample subtrees rooted at "b" nodes at the second level from the XML data tree, and the sample size is 2, i.e., $S = 2$.

There are 3 possible outcomes of the sampling. Case 1: subtrees rooted at the first and the second "b" nodes are sampled. The estimate is calculated as $1 \cdot \frac{C(3,1)}{C(2,1)} + 1 \cdot \frac{C(3,2)}{C(2,2)} = 4.5$. Case 2: subtrees rooted at the first and the third "b" nodes are sampled. The estimate is calculated as $1 \cdot \frac{C(3,1)}{C(2,1)} = 1.5$. Case 3: subtrees rooted at the second and the third "b" nodes are sampled. The estimate in this case is 0. The expected value of the estimator is then $(4.5+1.5+0)/3 = 2$, which is exactly the number of query matches.

Variance. The variance of the estimator Est, denoted as Var(Est), is computed as

$$\begin{aligned} Var(Est) &= \sum_{i=1}^{B}\sum_{j=1}^{B} Cov(Est_i, Est_j) \\ &\leq \sum_{i=1}^{B}\sum_{j=1}^{B} \sqrt{Var(Est_i)Var(Est_j)} \end{aligned} \quad (4)$$

Assume all i-subtree combinations in the sample or the XML data tree are numbered from 1 to $C(m,i)$ or $C(n,i)$, respectively. Let $y_{i,j}$ be the number of query matches found in the jth i-subtree combination in the sample or the data tree. Let $f_i = \frac{C(m,i)}{C(n,i)}$ be the sampling fraction of the i-subtree combinations. Let $\bar{y_i}$ be the sample mean of $y_{i,j}$, i.e., $\bar{y_i} = \sum_{j=1}^{C(m,i)} y_{i,j}/C(m,i)$, and $\bar{Y_i}$ the population mean, i.e., $\bar{Y_i} = \sum_{j=1}^{C(n,i)} y_{i,j}/C(n,i)$. The derivation and property of the variance estimator are stated in the following Theorems. For simplicity, only the results are presented here. Interested readers are referred to the full version of the research paper [**?**] for formal and complete proofs.

THEOREM 2. *An unbiased estimator of the variance of Est_i, $1 \leq i \leq B$, denoted as $var(Est_i)$, is*

$$var(Est_i) = \frac{C(n,i)^2 s^2}{C(m,i)}(1 - f_i) \quad (5)$$

where

$$s^2 = \frac{\sum_{j=1}^{C(m,i)}(y_{i,j} - \bar{y_i})^2}{C(m,i) - 1} \quad (6)$$

As mentioned earlier, we expect that a twig match generally comes from a single subtree. For instance, consider Example 1, where we sample DBLP at the second level, right under the root. No meaningful twig queries that we can think of would require "joining" two subtrees to find a match. Therefore, for queries that have no matches span multiple subtrees, the estimation formula can be simplified to

$$Est = Est_1 = \frac{C(n,1)}{C(m,1)} \times y_1 = \frac{n}{m} \times y_1 \quad (7)$$

Theorem 2 is then simplified to the following theorem, which can also be found in [6]:

THEOREM 3. *Assume no query matches can span multiple sample subtrees. An unbiased estimate of the variance of Est, denoted as var(Est), is $var(Est) = \frac{n^2 s^2}{m}(1 - f_1)$ where $s^2 = \frac{\sum_{j=1}^{m}(y_{1,j} - \bar{y_1})^2}{m-1}$.*

Confidence Interval. The confidence interval of an estimate can be constructed in several ways, for instance, by the Central Limit Theorem [6] and Chebyshev's inequality [22]. If the sample size n is large enough, e.g., > 30, it is usually assumed by the Central Limit Theorem that the estimate Est is normally distributed about the true value, denoted as Y [6]. For a given confidence probability p, let z_p be the $(p+1)/2$th quartile of the cumulative standard normal distribution Φ, that is, $\Phi(z_p) = (p+1)/2$. For example, if the confidence probability p is 95%, then $z_p = 1.96$. For a given confidence probability p, when the normality assumption holds, the associated confidence interval is $Est \pm \epsilon$, where

$$\epsilon = z_p \sqrt{var(Est)} \quad (8)$$

Given a desired confidence probability p, the respective confidence interval is $Est \pm \epsilon$, where

$$\epsilon = (b-a)(\frac{1}{2n}\ln(\frac{2}{1-p}))^{1/2} \quad (9)$$

The Chebyshev's inequality,

$$Pr\{|Est - Y| \geq \epsilon\} \leq \frac{var(Est)}{\epsilon^2} \quad (10)$$

where Y is the true value of the number of matches, gives another way to compute a conservative confidence interval.

Table 1: Dataset Statistics

Dataset	Size (MB)	# of Elements	Depth	Recursion
DBLP	133	3332130	6	free
Xmark	116	1666315	12	light
TreeBank	86	2437666	36	substantial

Table 2: Generality Comparison of the Techniques

Techniques	Parent-Child Only			Ancestor-Descendant		
	LP.	ET.	RT.	LP.	ET.	RT.
TreeSketch	✓	✓	✓	—	—	—
Xseed	✓	✓	—	✓	✓	—
Sampling	✓	✓	✓	✓	✓	✓

The inequality applies to any distribution. For a given confidence probability p, the corresponding confidence interval is $Est \pm \epsilon$, where

$$\epsilon = \sqrt{\frac{var(Est)}{1-p}} \tag{11}$$

4. EXPERIMENTAL RESULTS

This section presents the experimental results of our Subtree Sampling method on both synthetic and real-life XML datasets. We compare its performance with that of TreeSketch [20] and Xseed [25].

While we elaborate on synopsis construction time and estimation accuracy, some general aspects of the methods, such as simplicity, generality, and uncertainty of the estimates, are also discussed. Overall, Subtree Sampling is much simpler, more general, and requires significantly less construction time than the other approaches. TreeSketch and Xseed perform better for simple and uniform XML datasets while Subtree Sampling excels for complex XML datasets.

4.1 Experimental Settings

We have implemented the Subtree Sampling method and Polyzotis [20] and Zhang [25] kindly provided the implementation of TreeSketch and Xseed, respectively. Three well-known XML datasets, DBLP [1], Xmark [23], and TreeBank, were selected for the experiments. Table 1 summarizes the major statistics of these datasets. The experiments were conducted on a dedicated computer with a 3.4GHz CPU and 1GB RAM.

4.2 General Comparisons of the Techniques

Simplicity. Both TreeSketch and Xseed build their synopses by clustering nodes. For complex XML datasets, the construction process can be prohibitively costly. In contrast, our sampling process is simple and straightforward, involving only randomly selecting subtrees.

Specialized query estimation algorithms need to be designed for both TreeSketch and Xseed. On the other hand, any XML query evaluation algorithm is directly applicable to our sample tree synopsis for query selectivity estimation as it is just a miniature of the original data tree.

Generality. The sample tree synopsis can be used for selectivity estimation for all types of queries, i.e., linear path (LP), existential twig (ET), and regular twig (RT) queries. By contrast, TreeSketch is not applicable to queries containing ancestor-descendent relationships, and Xseed does not support regular twig query estimation.

Table 2 summarizes the queries each technique supports.

While we focus here on selectivity estimation, our sample tree synopsis can be directly applied to approximate query answering of aggregation queries as well as non-aggregation queries (i.e. queries that return matches). As for other synopses, significant modifications may be required, if possible at all, for general approximate query answering.

Uncertainty of the Estimates. TreeSketch and Xseed synopses provide only the query size estimates while our sample tree synopsis furnishes statistical uncertainty measures, such as confidence probability and intervals, with the estimates.

Table 3: Construction Time

Dataset	TreeSketch	Xseed	Subtree Sampling
DBLP	$20min$	$25min$	$30sec$
Xmark	$681min$	$1min$	$14sec$
TreeBank	$>4day$	$>4day$	$27sec$
4%TreeBank	$1955min$	$232min$	$1.5sec$

4.3 Construction Time

TreeSketch builds its synopsis in two steps. It first creates an intermediate Count-Stability synopsis that preserves all the information of the original XML dataset in a compact format. The Tree-Sketch synopsis is then built on top of the Count-Stability synopsis by merging similar structures.

The Xseed synopsis consists of two parts, an Xseed kernel and a hyper-edge table (HET). The construction of HET is performed by gradually extracting irregular structures out of the datasets. The HET construction stops when it determines that no further improvement can be made.

Table 3 shows the total construction time of the TreeSketch and our sample tree synopses for three different sizes, 25KB, 50KB, and 100KB. The sizes of Xseed synopses are 15KB for DBLP, 20KB for XMark, and 50KB and 100KB for TreeBank (as explained earlier, the construction stops when no further improvement is possible). As shown in Table 3, TreeSketch and Xseed synopses generally take several orders longer time than the sample tree synopsis to construct. The construction time of the TreeSketch and Xseed synopses largely depends on the complexity of the dataset. This is demonstrated in their significant construction time for the small but complex TreeBank dataset. In fact, the construction of neither synopses could be completed after 4 day's running on a dedicated computer. As a result, we were forced to use a 4% version of the original TreeBank dataset in the experiment, which is merely $3MB$. However, even for such a small dataset, it still took more than a day (1,955 minutes) to build the Tree-Sketch synopsis and around 4 hours (235 minutes) to build the Xseed synopsis. This prohibitively costly construction process could hinder the applications of TreeSketch and Xseed to complex datasets and/or dynamic datasets that undergo frequent changes.

As observed, the entire sampling process took only a very short period of time, for instance, tens of seconds. The sampling time is primarily dependent on the size of the dataset, not its complexity. The quickness of sampling renders this approach particularly suitable for dynamic datasets as samples can be redrawn quickly to reflect the changes.

Table 4: Query Set Statistics

Dataset	Simple Paths		Complex Paths	
	Set 1	Set 2	Set 1	Set 2
DBLP	854	814	776	698
Xmark	927	959	818	857
TreeBank	2167	1717	1721	1468

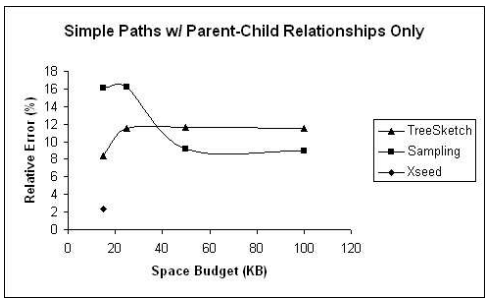

Figure 6: Simple Paths with P/C on DBLP

4.4 Estimation Accuracy

We compare the estimation accuracy of the synopses for different types of queries on three datasets, namely DBLP, Xmark, and TreeBank.

The queries are categorized into two groups, the simple path queries and the complex path queries. The simple path queries include linear path queries and existential twig queries, both of which study the occurrences of linear path patterns, with the latter taking into account the existential condition of specified branches. The complex path queries here refer to regular twig queries, which search for the occurrences of twig patterns that involve multiple branch paths.

In general, the size estimation of simple path queries is easier than that of complex path queries as the latter requires not only the existential but also the quantitative information of the specified branch paths.

For each dataset, these two query types were tested. Each query type has two query sets, one containing only parent-child relationships (Set1) and the other ancestor-descendant relationships (Set2). All the queries are positive in the sense that they return non-empty result sets. The query sets are generated by randomly picking linear paths or twigs from the XML data tree. Table 4 shows the number of queries in each query set.

The error metric we used is the average absolute relative error, which is defined as $RE = avg(100 * |t - e|/t)$, where t and e are the true and estimated sizes of the query result set, respectively.

Performance on the DBLP dataset. As shown in Figure 6, all synopses perform well for simple path queries with parent-child (P/C) relationships only on the DBLP dataset, with Xseed having the best performance. Structure synopses generally record parent-child relationships exactly. If a tag does not have more than one parent tag or all its parent tags have uniform characteristics, structure synopses can yield accurate estimates for linear path queries. Uniformity of the data also helps to estimate existential twig queries accurately. The good performance of Xseed and TreeSketch benefits tremendously from the simplicity and uniformity of the DBLP dataset.

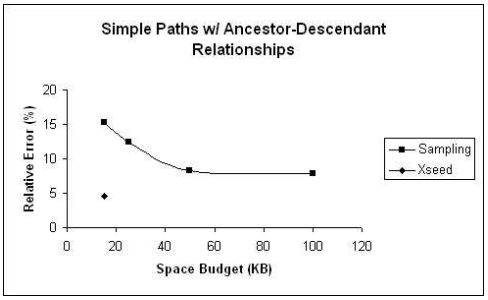

Figure 7: Simple Paths with A/D on DBLP

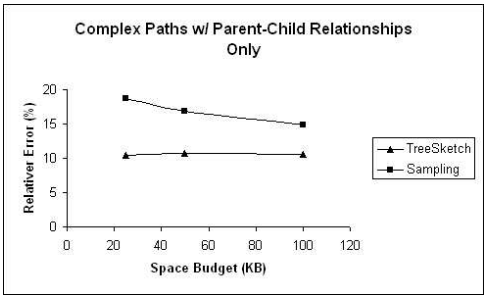

Figure 8: Complex Paths with P/C on DBLP

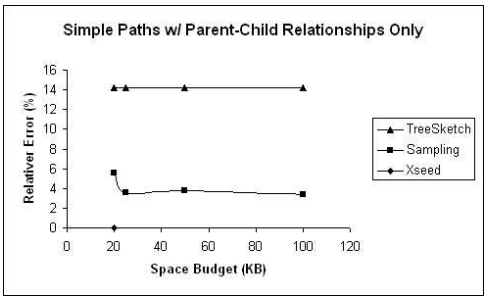

Figure 9: Simple Paths with P/C on Xmark

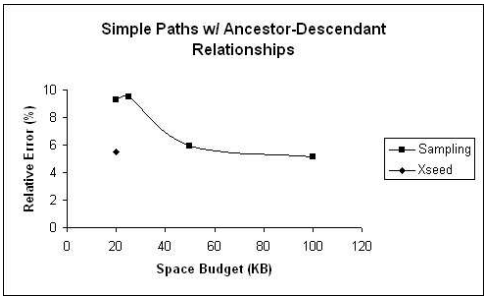

Figure 10: Simple Paths with A/D on Xmark

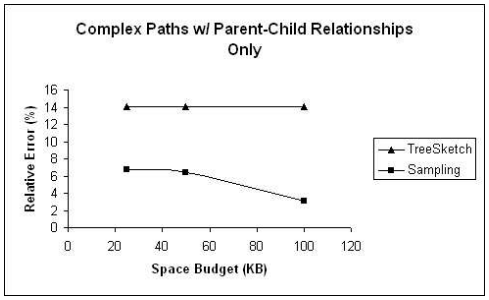

Figure 11: Complex Paths with P/C on Xmark

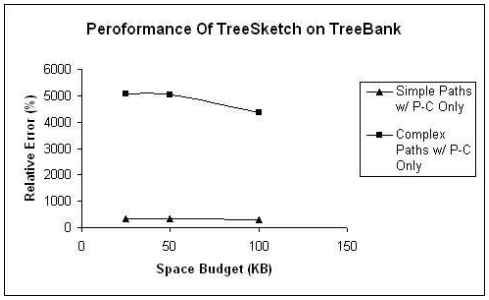

Figure 12: TreeSketch's Ineffectiveness on TreeBank

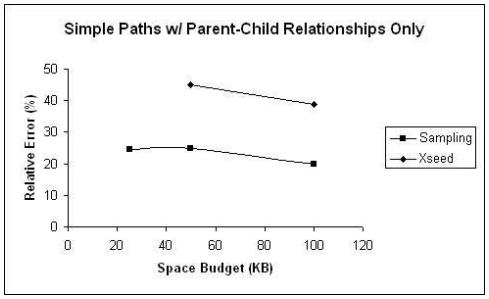

Figure 13: Simple Paths with P/C on TreeBank

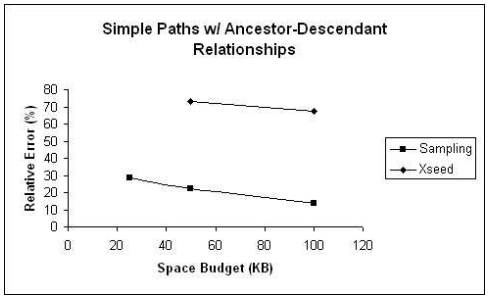

Figure 14: Simple Paths with A/D on TreeBank

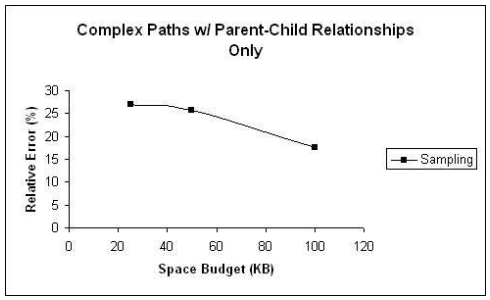

Figure 15: Complex Paths with P/C on TreeBank

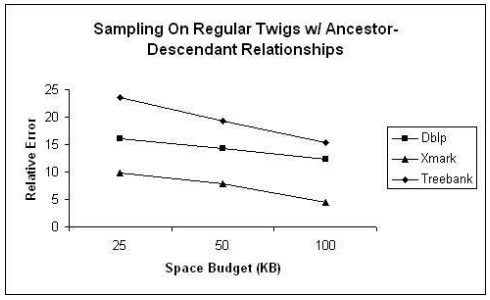

Figure 16: Complex Paths with A/D Relationships

As the sample size increases, the accuracy of Subtree Sampling improves. On the other hand, TreeSketch shows no noticeable performance improvement with the increase of space budget, which might be due to its ineffectiveness in further capturing irregular structural relationships. Consequently, as shown in Figure 6, Subtree Sampling surpasses TreeSketch under larger space budgets for simple path queries.

As TreeSketch doesn't support queries with ancestor-descendant (A/D) relationships, Figure 7 shows the performance of Subtree Sampling and Xseed on simple path queries with A/D relationships. Once again, because of the simplicity of the dataset, Xseed is able to estimate the selectivity accurately and thus outperforms Sampling. On the other hand, with the increase of space budget, the estimation accuracy of sampling improves steadily.

Figure 8 shows the result of complex path queries with P/C relationships only. TreeSketch performs better than Subtree Sampling due to the simplicity and uniformity of the DBLP dataset. Note that the performance of Xseed is missing since Xseed does not support complex path queries.

Performance on the Xmark dataset. Xmark exhibits a little more complex structure than DBLP and has a light degree of recursions. As shown in Figure 9, Xseed still performs the best for simple path queries with P/C relationships only as it employs a special mechanism to handle recursive structures. It supports recursion by recording "recursion levels" and "recursive path expression" in its synopses.

Figure 10 shows that for simple path queries with A/D relationships, Xseed still outperforms Subtree Sampling under small space budgets. With the increase of space budget, however, Subtree Sampling gradually overtakes Xseed.

Figure 11 shows the performance for complex path queries with P/C relationships only on the Xmark dataset. As demonstrated in the figure, Subtree Sampling performed the best. This is mainly due to the increased structural complex-

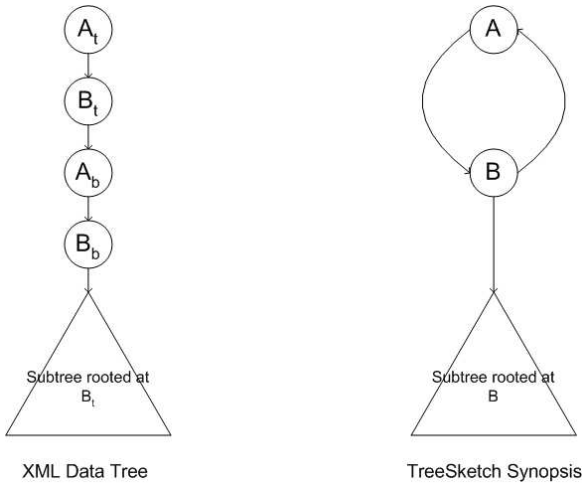

Figure 17: Recursion Poses Problems for TreeSketch

ity in the dataset, including the presence of recursions and a higher structural irregularity that TreeSketch is ineffective to capture.

It is worth mentioning that TreeSketch can have problems dealing with recursive structures. The following example illustrates this point.

Figure 17 shows how recursive structures are clustered to build the TreeSketch synopsis under some space budget. The left side of the figure shows the structure in the XML data tree. For clarity of discussion, two nodes that have the same label name "A" are marked as "A_t" and "A_b", where the subscripts stand for top and bottom, respectively. Similarly, two nodes having the same label name "B" are marked the same way.

Note that nodes A_t and A_b have the same label name and a very similar sub-structure. As a result, they are merged as one single A node in the Tree-Sketch synopsis. In a similar manner, node B_t and B_b are also merged to one single B node in the TreeSketch synopsis, thus creating a loop in the Tree-Sketch synopsis and posing a problem of how this loop should be traversed. In particular, if we allow the queries to contain ancestor-descendant relationships, then there is no way to determine how to traverse the loops. This restricts TreeSketch's applications to queries with only parent-child relationships.

Performance on the TreeBank dataset. As TreeBank exhibits a complex structure and has a high degree of recursions, structure synopses have difficulty effectively summarizing its intricate structural relationships. As far as TreeSketch is concerned, Figure 12 shows it now consistently yields erroneous estimates for queries on TreeBank as it has difficulty dealing with complex XML datasets. The high-degree recursions could cause estimation problems to TreeSketch. The increased depth of the tree can also introduce greater summarization errors since the summarization errors accumulate and propagate in a bottom-up manner. Because of TreeSketch's high estimation error, it will not be further compared with the other two methods.

Figures 13 and 14 show that our sample tree synopsis surpasses Xseed and yields the best performance for simple path queries with P/C and A/D relationships.

As for the complex path queries with P/C relationships only, Figure 15 shows that Subtree Sampling still performs reasonably well. In fact, structural complexity of the queries tend to have little effect on the performance of Subtree Sampling.

Performance on Complex Queries with Ancestor-Descendant Relationships. Since the implementation of TreeSketch doesn't support queries containing ancestor-descendant relationships and Xseed doesn't support regular twig queries, only the performance of Subtree Sampling is shown in Figure 16. Like in other cases, the performance of Subtree Sampling improves with the increase of space budget and there is no significant performance difference between different query sets on the same dataset.

4.4.1 Summary

The construction of structure synopses, specifically the TreeSketch and Xseed synopses in this research, generally requires complicated analysis and expensive processing while the sample tree synopsis is derived through simple random sampling. For complex datasets such as TreeBank, constructing a structure synopsis can be prohibitively time-consuming, whereas regardless of the dataset's complexity, generating a sample tree synopsis is always easy and fast.

As far as selectivity estimation is concerned, a sample tree synopsis can be applied to any type of queries, while TreeSketch is only applicable to queries with P/C relationships and Xseed simple path queries. Moreover, a sample tree synopsis can be applied to aggregation queries, such as SUM, AVG, etc., as well as non-aggregation queries.

The performance of structure synopses generally relies on the independence and uniformity assumptions of the datasets and the simplicity of the query patterns. Xseed performs very well for simple path queries on simple and uniform datasets such as DBLP and XMark. However, as the datasets become more complex, the uniformity and independence assumptions no long hold and thus the performance of structure synopses degrades dramatically. In comparison, the performance of Subtree Sampling is good for uniform datasets as well as complex datasets and for simple path queries as well as complex path queries.

5. CONCLUSIONS

Tree-structured data, notably XML data, pose serious challenge to conventional structural summarization techniques. The intricate structural relationships among nodes are difficult, if not impossible, to measure and model. Since the efficiency of structure synopses largely depends on the summarization of the structural relationships, structure synopses could perform poorly or even erroneously on complex XML datasets. Besides, the construction of these structure synopses is usually slow and costly, especially for complex XML datasets.

Instead of trying to *measure* and *model* the intricate structural relationships, in this research, we propose a sampling method that *preserves* the structural relationships. Compared with conventional summarization techniques, not only is the proposed sampling method significantly simpler both conceptually and practically, it is also outstandingly more effective in dealing with complex XML datasets.

6. REFERENCES

[1] Dblp data set. http://www.informatik.uni-trier.de/ley/db/index.html.

[2] N. Bruno, N. Koudas, and D. Srivastava. Holistic twig joins: optimal xml pattern matching. *Proceedings of the 2002 ACM SIGMOD international conf. on Management of data*, pages 310–321, 2002.

[3] S. Chaudhuri, R. Motwani, and V. Narasayya. On random sampling over joins. *Proc. ACM SIGMOD Conf., June, 1999*, pages 263–274, 1999.

[4] T. Chen, J. Lu, and T. W. Ling. On boosting holism in xml twig pattern matching using structural indexing techniques. *Proceedings of the 2005 ACM SIGMOD international conf. on Management of data*, 2005.

[5] Z. Chen, H. Jagadish, F. Korn, N. Koudas, S. Muthukrishnan, R. T. Ng, and D. Srivastava. Couting twig matches in a tree. *Proceedings of the 17th International Conference on Data Engineering*, pages 595–604, 2001.

[6] W. G. Cochran. *Sampling Techniques*. Wiley, 1977.

[7] J. Freire, J. R. Haritsa, M. Ramanath, P. Roy, and J. Siméon. Statix: making xml count. *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 181–191, 2002.

[8] S. Ganguly, P. Gibbons, Y. Matias, and A. Silberschatz. Bifocal sampling for skew-resistant join size estimation. *Proceedings of the 1996 ACM SIGMOD international conf. on Management of data*, pages 271–281, 1996.

[9] P. Hass, J. Naughton, S. Seshadri, and L. Stokes. Sampling-based estimation of the number of distinct values of an attribute. *Proc. 21st Intl. Conf. on Very Large Data Bases*, pages 311–322, September 1995.

[10] P. Hass, J. Naughton, S. Seshadri, and A. Swami. Fixed-precision estimation of join selectivity. *Proc. 12th ACM Symp. on Principles of Database Systems*, pages 190–201, May 1993.

[11] W.-C. Hou, G. Ozsoyoglu, and B. Taneja. Statistical estimators for relational algebra expression. *Proc. 7th ACM Symp. on Principles of Database Systems*, pages 276–287, 1988.

[12] W.-C. Hou, G. Ozsoyoglu, and B. Taneja. Processing aggregate relational queries with hard time constraints. *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 68–77, June 1989.

[13] H. Jiang, H. Lu, and W. Wang. Efficient processing of xml twig queries with or-predicates. *Proceedings of the 2004 ACM SIGMOD international conf. on Management of data*, 2004.

[14] R. J. Lipton, J. F. Naughton, and D. A. Schneider. Practical selectivity estimation through adaptive sampling. *Proceedings 1990 ACM SIGMOD Intl. Conf. Managment of Data*, pages 1–11, 1990.

[15] J. Lu, T. W. Ling, C.-Y. Chan, and T. Chen. From region encoding to extended dewey: on efficient processing of xml twig pattern matching. *Proceedings of the 31st international conf. on very large data bases*, 2005.

[16] C. Luo, Z. Jiang, W.-C. Hou, F. Yan, and C.-F. Wang. Estimating xml structural join size quickly and economically. *Proc. 22nd Intl. Conf. on Data Engineering*, 2006.

[17] M. M. Moro, Z. Vagena, and V. J. Tsotras. Tree-pattern queries on a lightweight xml processor. *Proceedings of the 31st international conf. on very large data bases*, 2005.

[18] N. Polyzotis and M. Garofalakis. Structure and value synopses for xml data graphs. *Proceedings of the 28th Intl. Conf. on Very Large Data Bases*, 2002.

[19] N. Polyzotis and M. N. Garofalakis. Statistical synopses for graph-structured xml databases. *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 358–369, 2002.

[20] N. Polyzotis, M. N. Garofalakis, and Y. Ioannidis. Approximate xml query answers. *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pages 263–274, 2004.

[21] N. Polyzotis, M. N. Garofalakis, and Y. Ioannidis. Selectivity estimation for xml twigs. *Proceedings of the 20th International Conference on Data Engineering*, 2004.

[22] S. Ross. *Introduction to Probability Models, 2nd Ed.* Academic Press, 1980.

[23] A. Schmidt, F. Waas, M. Kersten, D. Florescu, L. Manolescu, M. J. Carey, and R. Busse. The XML benchmark project. Technical report, CWI, 2001.

[24] W. Wang, H. Jiang, H. Lu, and J. X. Yu. Containment join size estimation: models and methods. *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 358–369, 2003.

[25] N. Zhang, M. T. Ozsu, A. Aboulnaga, and I. F. Ilyas. Xseed: Accurate and fast cardinality estimation for xpath queries. *Proc. 22nd Intl. Conf. on Data Engineering*, 2006.