Counting Relaxed Twig Matches in a Tree

Dongwon Lee[1] and Divesh Srivastava[2]

[1] Penn State University, dongwon@psu.edu
[2] AT&T Labs – Research, divesh@research.att.com

Abstract. We consider the problem of accurately estimating the number of approximate XML answers for a given query, and propose an efficient method that (1) accurately computes selectivity estimates for each relaxed XML query, using a natural generalization of the correlated subpath tree (CST) summary structure, and (2) carefully combines these estimates by analyzing the nature of overlap between the different relaxed twig queries.

1 Introduction

When a query Q submitted by a user is somehow *modified* such that the set of answers returned is a superset of the exact answers to Q, it is called *query relaxation* [5]. In XML, where the schema may be irregular, instead of finding only exact answers for a given query, flexible matching based on query relaxation would facilitate easy and concise querying significantly. In this paper, we view (hierarchically organized) XML documents as node-labeled trees, following [6]. Then, a natural way to query such data is by using small node-labeled trees, referred to as *twigs*, that match portions of the hierarchical data. Query relaxations then modify either the node labeling or the twig structure to enable additional *relaxed twig* matches to be returned. A fundamental problem in this context is to accurately and quickly estimate the number of *relaxed* matches of a twig query against the node-labeled data tree. This problem is relevant for providing users with quick feedback about their query. Another use is in the cost-based optimization for efficiently evaluating such relaxed queries. In this paper, we address this estimation problem.

1.1 Related Work

Two areas in XML literature – query relaxation and selectivity estimation – are evidently relevant to our work.

Query relaxation: Query relaxation has been extensively investigated in both IR and DB areas. More specifically, several authors have proposed approaches that support approximate pattern matching and answer ranking for XML (e.g., [12, 7]) . More recently, [1] proposed different relaxation schemes, including generalizing nodes, deleting nodes, and relaxing edges, and investigated their efficient evaluation. Also, [8] proposed another kind of relaxation based on changing the

order of nodes in a path, and presented various complexity results on query evaluation. However, neither of [1, 8] deal with the selectivity estimation issue for the relaxed queries. In this paper, we consider two relaxation primitives for XML, based on the proposal in [1].

Selectivity estimation: Selectivity estimation for path expressions, more restricted than twigs, has been investigated in Lore [10] and Niagara [2] projects. Selectivity estimation for twigs has been considered in, e.g., [6, 13, 11]. In particular, [6] proposed an estimation method that, given a twig query, first creates a set of query twiglets and estimates the number of matches of each query twiglet using set hashing and combines the query twiglet estimates into an estimate for the twig query using maximal overlap. Our work extends the framework of [6] to support selectivity estimation for the case of relaxed twigs.

2 Background

Definition 1 (Twig Match) *The* twig match *of a twig query $Q = (V_Q, E_Q)$ in a node-labeled data tree $T = (V_T, E_T)$, denoted as $\langle Q(T)\rangle$ or simply $\langle Q\rangle$ when the context is clear, is defined by a mapping: $f : V_Q \rightarrow V_T$ such that if $f(u) = v$ for $u \in V_Q$ and $v \in V_T$, then (1) Label(u) = Label(v), and (2) if $(u, u') \in E_Q$, then $(f(u), f(u')) \in E_T^*$, the transitive closure of E_T, where (u, u') is either a parent-child or ancestor-descendent edge and $(f(u), f(u'))$ preserves the same relationship.* □

Note that the above twig match definition is essentially of a *total match*, where "all" the nodes of a twig query must be matched to data nodes. We observe that by relaxing this constraint, one can flexibly express a variety of match semantics that different XML query languages support. For instance, if we assume that only a subset of nodes S ($\subseteq V_Q$) will be projected at the end (S is termed as *projected nodes*), then a notion of *partial match* can be introduced. Consider the XML data:

```
<a1>
    <b1> <c1/> <d1/> </b1>
    <b2> <c2/> <c3/> <d2/> </b2>
</a1>
```

- *Root Semantics*: Only root node is a projected node. For instance, an XPath [3] query "`//b[c]`" returns $\{(b_1), (b_2)\}$ as a twig match.
- *Leaf Semantics*: Only leaf node is a projected node. For instance, an XPath query "`//b/c`" returns $\{(c_1), (c_2), (c_3)\}$ as a twig match. A twig match in [2], for example, handles the leaf semantics.
- *Total Semantics*: All nodes in a twig are projected nodes. An XQuery [4] query "`FOR $b IN //b, $c IN $b/c RETURN $b,$c`" returns $\{(b_1, c_1), (b_2, c_2), (b_2, c_3)\}$ as a twig match. A twig match proposed in [6] supports the total semantics.

Throughout the rest of the paper, projected nodes in a twig are underlined for distinction (e.g., $/\underline{a}/b/\underline{c}$). A twig query P with projected nodes P_Q is denoted by a triple $P = (V_Q, E_Q, P_Q)$, where $P_Q \subseteq V_Q$. Note that in the projected twig matches, all the mappings, whether projected or not, are relevant, but only the projected matches remain in the final result. In other words, we do not ignore the mapping for non-projected nodes during processing. It is important to note that the "projected nodes" significantly affect the final selectivity of the query.

Query Relaxation for XML Model: In order to explore the set of approximate matches of a query, one must be able to *relax* the query that guarantees that the set of answers to be returned is a "superset" of the set of exact query matches (i.e., no false dismissal). One may think of many ways to relax the given twig query: weakening value predicates, swapping order of nodes, or deleting some nodes in a twig, etc. In this paper, we focus on the following two primitive relaxations, based on the relaxations proposed in [1][3]:

- *Node deletion*: A non-root node v in a twig can be deleted as follows: (1) if v is a leaf, only v is deleted, and (2) if v is an internal node, then v is deleted and children of v become children of v's parent node (using the ancestor-descendant edge). For instance, if the node b is deleted from the twig `/a/b/c` during relaxation, then relaxed twig query becomes `/a//c`.
- *Edge relaxation*: A child edge (denoted by single edge "/") in a twig can be relaxed to a descendant edge (denoted by double edge "//"). Descendant edges cannot be relaxed further.

Then, a *relaxed twig* query $R = (V_R, E_R)$ is a twig obtained by applying one or many of the above relaxations to a twig $Q = (V_Q, E_Q)$. If the function $sel(Q)$ returns a selectivity estimate of a twig Q, then, by definition, $\langle Q \rangle \subseteq \langle R \rangle$[4] and $sel(Q) \leq sel(R)$.

3 Main Idea

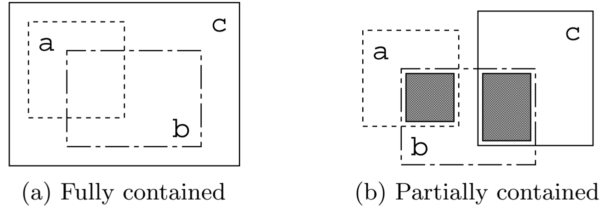
We consider the problem of selectivity estimation in the context of query relaxation in XML. Let us denote a set of relaxed twig queries of a twig query Q as $R_Q = \{R_i \mid R_i \text{ is a relaxed twig of } Q\}$. Then, our goal is to estimate $N = sel(\cup_{R_i \in R_Q} R_i)$, where

$$MAX_{R_i \in R_Q}\{sel(R_i)\} \leq N \leq \sum_{R_i \in R_Q} sel(R_i) \tag{1}$$

Our proposal here is inspired by the approach proposed in [6]. Therefore, as a summary data structure, we use the CST, an *augmented pruned suffix tree*

[3] In fact, [1] achieves deletion of non-leaf nodes using a combination of *subtree promotion* and *leaf node deletion*.

[4] The precise semantics of $\subseteq$ operator under the "projected" model will be defined in Section 3.2.



(a) Fully contained (b) Partially contained

Fig. 1. Rectangles represent matches a, b, and c of the relaxed queries R_a, R_b, and R_c, respectively. Shaded rectangles in (b) are the overlap.

that represents frequency information of small twigs in the data tree. Details of the CST are omitted in the interest of space and interested readers are referred to [6]. In short, our approach consists of two steps: (1) compute $sel(R_i)$, where $R_i \in R_Q$, and (2) combine each $sel(R_i)$ to get $sel(\cup_{R_i \in R_Q} R_i)$ *without counting the same answer multiple times.*

3.1 Computing $sel(R_i)$

To correctly estimate the selectivity of relaxed twig queries, the CST is augmented to include data paths with wildcard ("*") characters. Towards this end, for each root-to-leaf path in the data tree, all *-extended paths are generated and inserted into the CST. For instance, for the path $a.b.c.d$, new paths $a.*.b.c.d$, $a.b.*.c.d$, $a.b.c.*.d$, $a.*.c.d$, $a.b.*.d$, and $a.*.d$ are inserted. In addition, all their suffixes are inserted. To keep the CST space manageable, however, we permit at most one "*" in the *-extended paths in the CST.

Note that relaxed twigs from the original twig query would typically contain descendant edges ("//") when internal node deletion or edge relaxation occurred. Then, selectivity estimates of relaxed twigs can be directly obtained using maximal parsing strategies on the twig query paths to match the "//" appearing in the query with the "*" in the CST data paths. For instance, the selectivity of a relaxed twig $/a//b/c//d$ (i.e., two parent-child edges, a/b and c/d, are relaxed to descendent edges, $a//b$ and $c//d$) can be obtained by locating the path $a.*.b.c.*.d$ from the CST, which would in turn be split into two paths $a.*.b.c$ and $c.*.d$ according to the maximal overlap strategy. Finally, $sel(a.*.b.c.*.d) = \frac{sel(a.*.b.c) \times sel(c.*.d)}{sel(c)}$ by, for instance, the *Pure MO* method in [6].

3.2 Computing $sel(\cup_{R_i \in R_Q} R_i)$

Consider two cases of the three relaxed queries R_a, R_b and R_c depicted in Figure 1. Three estimates $sel(R_a)$, $sel(R_b)$, and $sel(R_c)$ have been already obtained by the method described in Section 3.1. Now, the goal is to compute $sel(R_a \cup R_b \cup R_c)$. In Figure 1(a), matches a or b can be ignored since they are fully contained by c. Thus, $sel(R_a \cup R_b \cup R_c) = sel(R_c)$. However, in Figure 1(b), there is no complete containment relationship among the three matches and thus

$sel(R_a \cup R_b \cup R_c) = sel(R_a) + sel(R_b) + sel(R_c) - sel(\text{overlap})$. Hence, the critical issues are (1) to understand the precise meaning of the overlap and (2) express the overlap in some form that can be estimated using a summary structure like the CST.

Semantics of the Overlap Basic set theory suggests that $sel(R_a \cup R_b \cup R_c) = sel(R_a) + sel(R_b) + sel(R_c) - sel(R_a \cap R_b) - sel(R_b \cap R_c) - sel(R_c \cap R_a) + sel(R_a \cap R_b \cap R_c)$. This formula holds insofar as nodes involved are of the same degree and on the same domain. However, in our case, due to the various relaxations, one relaxed twig may have a different set of (projected) nodes than another one (e.g., $/\underline{a}/b$ and $/a/c//\underline{b}$). To resolve this issue, we introduce the notion of extension-compatibility:

Definition 2 (Extension Compatibility) *Given two sets of projected nodes, P_a and P_b, for relaxed twig queries a and b, if $P_a \subseteq P_b$ or $P_a \supseteq P_b$, then, a and b are said to be* extension-compatible. *Otherwise, they are* extension-incompatible. □

Definition 3 (Redundancy) *Given two twig matches x and y, x is said to be a* redundant *match of y, denoted $x \ll y$, if x can be obtained by projecting out some attributes of y. Otherwise, x is said to be* irredundant. □

Intuitively, redundant matches are not returned as approximate answers, and the selectivity estimate needs to discount for such matches.

Example 1. Consider two twig matches A:$\{(a_1, c_1), (a_2, c_2)\}$ and B:$\{(a_1, b_1, c_1), (a_2, b_2, c_3)\}$. Note that B has one more projected node, b, than A has. Then, $(a_1, c_1)_{\in A}$ is a redundant match of $(a_1, b_1, c_1)_{\in B}$. However, $(a_2, c_2)_{\in A}$ is not a redundant match of $(a_2, b_2, c_3)_{\in B}$. Similarly, none of the matches in B will be a redundant match of any match in A, since B has an additional column. □

Definition 4 (Projected Set Operators) *For two projected twig queries a and b (and their corresponding projected twig matches $\langle a \rangle$ and $\langle b \rangle$, respectively),*

- *$a \Cap b$ (projected intersection) is the set of "redundant" matches in $\langle a \rangle$ and $\langle b \rangle$,*
- *$a \Cup b$ (projected union) is the set of "irredundant" matches in $\langle a \rangle$ or $\langle b \rangle$.* □

Lemma 1. *If twigs a and b are extension-incompatible, then there are no redundant matches between a and b, i.e., $a \Cap b = \emptyset$.* ■

Lemma 2. *For two projected twig queries a and b, the following holds: $a \Cup b = a \cup b - a \Cap b$, where $\cup$ and $-$ are the conventional set union and difference operators.* ■

Corollary 1. For two twig queries a and b, $sel(a \Cup b) = sel(a) + sel(b) - sel(a \Cap b)$ holds. (q.e.d)

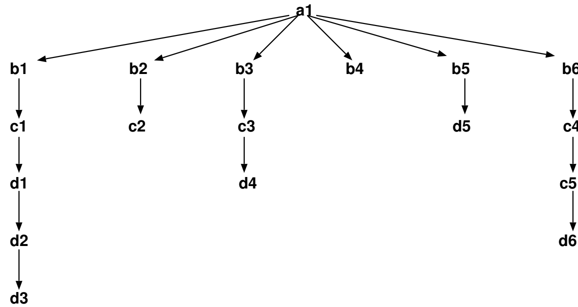


Fig. 2. Example XML data tree D.

Example 2. Overlap between twigs t_1:$/\underline{a}/b$ and t_2:$/a/c//\underline{b}$ is always empty since the two twigs are extension-incompatible, yielding $t_1 \Cap t_2 = \emptyset$. On the other hand, overlap between $/\underline{a}//d/\underline{b}$ and $/\underline{a}/c//\underline{b}$ can be computed by $/\underline{a}//d/\underline{b} \Cap /\underline{a}/c//\underline{b}$ and its selectivity can be non-zero if there happens to be a branch in an XML data that matches the twig $/a/c//d/b$. $\Box$

Let us check if the semantics that we have described so far is consistent with the intuitive meaning of overlap. Figure 2 depicts an example XML tree. Suppose that users ask a simple twig query Q : $/\underline{a}/\underline{b}/\underline{c}/\underline{d}$, and suppose only two relaxed queries are permitted – R_1 : $/\underline{a}/\underline{b}//\underline{c}$ (i.e., edge between b and c is relaxed and node d is removed) and R_2 : $/\underline{a}/\underline{b}//\underline{d}$ (i.e., node c is removed). Users are interested in quickly finding out "the number N of approximate answers that satisfy Q", that is $N = sel(Q \Cup R_1 \Cup R_2)$.

Corresponding matches when these queries are evaluated against D of Figure 2 are shown in Table 1. Note that queries R_1 and R_2 are extension-incompatible due to their projected nodes, $\{a, b, c\}$ and $\{a, b, d\}$, resulting in no overlaps in-between. Then, $N = sel(Q) + sel(R_1) + sel(R_2) - sel(Q \Cap R_1) - sel(Q \Cap R_2) - sel(R_1 \Cap R_2) + sel(Q \Cap R_1 \Cap R_2) = 2+5+6-2-2-0+0 = 9$. When we closely examine the tree D of Figure 2, it is not difficult to see that in fact there are 9 distinct approximate matches for the query Q (with relaxed queries R_1 and R_2).

The general formula involving n such relaxed twig queries is as follows:

$$sel(\sum_{i=1}^{n} R_i) = \sum_{i=1}^{\binom{n}{i}} (-1)^{i-1} sel(\Cap_{j=1}^{i} R_j)$$

In order for the formula to hold under the notions of projected intersection and union, the basic laws such as idempotency, commutativity, associativity, and distributivity must hold. Due to limited space, we simply state that they in fact hold. Detailed proofs can be found in [9].

Overlap Formula One remaining problem is that existing summary data structures, e.g., [6, 2], are unable to handle twigs having the "intersection" operator

Table 1. Projected twig matches of Q, R_1, R_2 and various overlaps against D of Figure 2.

\$a	\$b	\$c	\$d
a_1	b_1	c_1	d_1
a_1	b_3	c_3	d_4

(a) $sel(Q) = 2$

\$a	\$b	\$c
a_1	b_1	c_1
a_1	b_2	c_2
a_1	b_3	c_3
a_1	b_6	c_4
a_1	b_6	c_5

(b) $sel(R_1) = 5$

\$a	\$b	\$d
a_1	b_1	d_1
a_1	b_1	d_2
a_1	b_1	d_3
a_1	b_3	d_4
a_1	b_5	d_5
a_1	b_6	d_6

(c) $sel(R_2) = 6$

\$a	\$b	\$c	\$d
a_1	b_1	c_1	ϵ
a_1	b_3	c_3	ϵ

(d) $sel(Q \Cap R_1) = 2$

\$a	\$b	\$c	\$d
a_1	b_1	ϵ	d_1
a_1	b_3	ϵ	d_4

(e) $sel(Q \Cap R_2) = 2$

$R_1 \Cap R_2 = Q \Cap R_1 \Cap R_2 = \emptyset$

(f) $sel(R_1 \Cap R_2) = sel(Q \Cap R_1 \Cap R_2) = 0$

that are found in $sel(R_i \Cap R_j)$. Therefore, we need to find an alternative twig expression (i.e., overlap formula) that *captures* the twigs involving intersection operator. For instance, observe that any twig matches satisfying the expression $/\underline{a}//d/\underline{b} \Cap /\underline{a}/c//\underline{b}$ must also satisfy $/\underline{a}/c//d/\underline{b}$, and further $sel(/\underline{a}/c//d/\underline{b})$ can be directly obtained from the CST. Therefore, in this sense, $/\underline{a}/c//d/\underline{b}$ is the overlap formula. The characteristic of the overlap formula of two relaxed twigs is that the formula would take the "tighter" condition of two twigs. The algorithm consists of three stages: **split()**, **compare()** and **merge()**. Consider two extension-compatible[5] twigs $R = (V_R, E_R, P_R)$ and $S = (V_S, E_S, P_S)$ that are relaxed from a query Q, where common projected nodes are denoted as: $P_O = P_R \cap P_S$.

1. **Split()**: In this stage, two input twigs are matched against each other and cut into several building blocks. Let us first define some notation. Between two list of characters, the *continuous intersection unit* is the list of common characters that appear in a continuous sequence; i.e., between *abce* and *abcde*, the following seven continuous intersection units are found $\{a, b, c, e, ab, bc, abc\}$. Note *abce* is not such a unit since it is not continuous. Further, the *maximal continuous intersection unit* is the unit that is not part of other units; i.e., e and *abc* are the maximal continuous intersection units.

Now, imagine R and S as lists of characters (ignoring edges / or //). Then, identify all the "maximal continuous intersection units" between R and S and denote them as $\{\alpha, \beta, \gamma, ...\}$. Also, denote a "chunk" of characters (which are not continuous intersection units) as $\{X, Y, Z, ...\}$. Then, scanning from head to tail, identify all of the following five patterns from R and S such that (1) two subsequent patterns share the identical maximal continuous intersection unit (except for the last pattern), and (2) the preference of the patterns (when multiple patterns are possible) is $P1 > P2$, $P3 > P4$, and $P1, P2, P3, P4 > P5$.

[5] If two twigs are extension-incompatible, then the overlap formula would be $F = \emptyset$.

P1 $R = \alpha X \beta$, $S = \alpha Y \beta$ (e.g., $R = /a/u/b//c$, $S = /a//u//d/v/c$, where $\alpha = au$, $X = b$, $Y = dv$, $\beta = c$)
P2 $R = \alpha X$, $S = \alpha Y$ (e.g., $R = /a/b//c$, $S = /a/b//d/u$, where $\alpha = ab, X = c, Y = du$)
P3 $R = \alpha\beta$, $S = \alpha X \beta$ (e.g., $R = /a//c$, $S = /a/b/d//c$, where $\alpha = a, X = bd, \beta = c$)
P4 $R = \alpha$, $S = \alpha X$ (e.g., $R = /a/b$, $S = /a/b/c$, where $\alpha = ab$, $X = c$)
P5 $R = S = \alpha$ (e.g., $R = /a/b$, $S = /a//b$, where $\alpha = ab$)

This step may yield several such patterns from R and S pair. Once all the patterns are identified, each pattern is passed to the next comparison stage.

Example 3. Consider two twigs: $R = /\underline{a}/a/b/c//d/e//f/g$ and $S = /\underline{a}//a//c/u/e/f//h$. Then, after the split() stage, the following three patterns are to be found: (1) **P3**: $\alpha = aa, X = b, \beta = c$, (2) **P1**: $\alpha = c, X = d, Y = u, \beta = ef$, and (3) **P2**: $\alpha = ef, X = g, Y = h$. Note that two neighboring patterns share the same maximal continuous intersection unit. □

2. **Compare()**: In this stage, each pattern is examined individually and its overlap form is composed. Both the intermix of different edge types, / and //, and the existence of the projected nodes make this step non-trivial. To illustrate the complexity, let us consider two twigs: $R = /a/x/b//c$ and $S = /a//y//b$. Then, depending on the projected nodes P, the overlap formula F differs: (1) if $P = \{a\}$ (i.e., root semantics), then $F = /\underline{a}/x/b//y//b//c$, (2) if $P = \{b\}$ (i.e., leaf semantics), then $F = /a//y//a/x/\underline{b}//c$, (3) if $P = \{a, b\}$, then $F = \emptyset$. Below, to improve the readability of the algorithm, we will describe only the case for the "root semantics". The overlap formulas for the other semantics can be found similarly. Let us bring edges back into the consideration, denoted as e_i, which were ignored in the split() stage. For each of the five patterns, the overlap formula F can be stated as follows (symmetric cases are omitted for brevity):

In root semantics:

P1 $R = \alpha\ e_{r1} X\ e_{r2}\ \beta$, $S = \alpha\ e_{s1}\ Y e_{s2}\ \beta$, then

$$F = \begin{cases} \emptyset & \text{if } e_{r1} = e_{s1} = / \\ \emptyset & \text{if } e_{r2} = e_{s2} = / \\ \alpha/X/\beta//Y e_{s2}\beta & \text{if } e_{r1} = e_{r2} = /, e_{s1} = // \\ \alpha/X//Y e_{s2}\beta & \text{if } e_{r1} = /, e_{r2} = e_{s1} = // \\ \alpha//X//Y/\beta & \text{if } e_{r1} = e_{r2} = e_{s1} = //, e_{s2} = / \\ \alpha//X//Y//\beta \Cup \alpha//Y//X//\beta & \text{if } e_{r1} = e_{r2} = e_{s1} = e_{s2} = // \end{cases}$$

P2 $R = \alpha\ e_{r1} X$, $S = \alpha\ e_{s1}\ Y$, then (1) $F = \emptyset$ if $e_{r1} = e_{s1} = /$ (2) $F = \alpha/X//Y$ if $e_{r1} = /, e_{s1} = //$ (3) $F = \alpha/Y//X$ if $e_{r1} = //, e_{s1} = /$ (4) $F = \alpha//X//Y \Cup \alpha//Y//X$ if $e_{r1} = e_{s1} = //$.
P3 If $R = \alpha e_{r1}\beta, S = \alpha e_{s1} X e_{s2}\beta$, then (1) $F = \emptyset$ if $e_{r1} = e_{s1} = /$ (2) $F = \alpha/\beta//X e_{s2}\beta$ if $e_{r1} = /, e_{s1} = //$ (3) $F = S$ if $e_{r1} = //$.
P4 If $R = \alpha, S = \alpha e_{s1} X$, then $F = S$.

Table 2. Different shapes of query sets and degrees of relaxations.

Shape	DBLP	SPROT	Type	DBLP	SPROT
	NumBranch n, Height h			NumRlxQry q, NumRlx r	
PATH	$n = 1$, $2 \leq h \leq 4$	$n = 1$, $2 \leq h \leq 6$	A	$1 \leq q \leq 3$, $1 \leq r \leq 2$	$1 \leq q \leq 3$, $1 \leq r \leq 2$
BS	$3 \leq n \leq 5$, $2 \leq h \leq 3$	$3 \leq n \leq 5$, $2 \leq h \leq 4$	B	$1 \leq q \leq 3$, $2 \leq r \leq 3$	$1 \leq q \leq 3$, $2 \leq r \leq 5$
DS	N/A	$1 \leq n \leq 3$, $4 \leq h \leq 6$	C	$3 \leq q \leq 5$, $1 \leq r \leq 2$	$3 \leq q \leq 5$, $1 \leq r \leq 2$
BAL	$1 \leq n \leq 5$, $2 \leq h \leq 4$	$1 \leq n \leq 5$, $2 \leq h \leq 6$	D	$3 \leq q \leq 5$, $2 \leq r \leq 3$	$3 \leq q \leq 5$, $2 \leq r \leq 5$

P5 If $R = \alpha, S = \alpha$, then $F = \alpha$ such that F keeps the more restricting edge between a pair of edges from R and S.

Example 4. Imagine various forms of relaxed queries from a query $Q : /\underline{a}/b/c/d$. Then, (1) $/\underline{a}//b \Cap /\underline{a}/b//d \equiv /\underline{a}/b//d$, (2) $/\underline{a}/b/c \Cap /\underline{a}//b//d \equiv /\underline{a}/b/c//d$, (3) $/\underline{a}/c \Cap /\underline{a}/b//c \equiv \emptyset$, (4) $/\underline{a}//b/c \Cap /\underline{a}/d//c \equiv /\underline{a}/d//b/c$. □

3. **Merge()**: In this stage, all overlap forms computed in the compare() stage are put together, yielding the final overlap formula. Merging is straightforward: for each two subsequent overlap forms, remove one maximal continuous intersection unit and put the rest together. If one is $\emptyset$, then the final formula is also $\emptyset$.

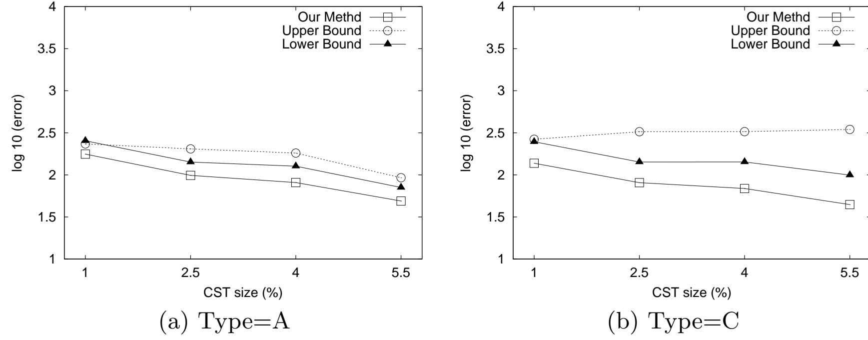
Example 5. Consider R and S in Example 3 again. After the compare() stage, three intermediate overlap forms are to be computed: (1) P3: $a/a/b/c$, (2) P1: $\emptyset$, and (3) P2: $e/f/g//h$. Therefore, the final overlap formula $F = \emptyset$. Also, consider $/\underline{a}/b//c/d \Cap /\underline{a}//c$. After split() and compare() stages, two forms are returned: $/\underline{a}/b//c$ and c/d. Thus, the final overlap formula would be $/\underline{a}/b//c/d$. □

One minor problem in the algorithm is two cases from patterns P1 and P2 which result in a formula having the projected union operator $\Cup$. For instance, $/\underline{a}//\underline{b}//c/d \Cap /\underline{a}/\underline{b}//e = /\underline{a}/\underline{b}//c/d//e \Cup /\underline{a}/\underline{b}//e//c/d$. This case is problematic since like the projected intersection operator, the projected union operator cannot be handled by the summary data structures. If this occurs, we assume (in an ad hoc fashion) that the selectivity of the overlap between two relaxed twigs be the maximum of the selectivities of two twigs: $sel(/\underline{a}//\underline{b}//c/d \Cup /\underline{a}/\underline{b}//e) = MAX\{sel(/\underline{a}//\underline{b}//c/d), sel(/\underline{a}/\underline{b}//e)\}$.

4 Experimental Results

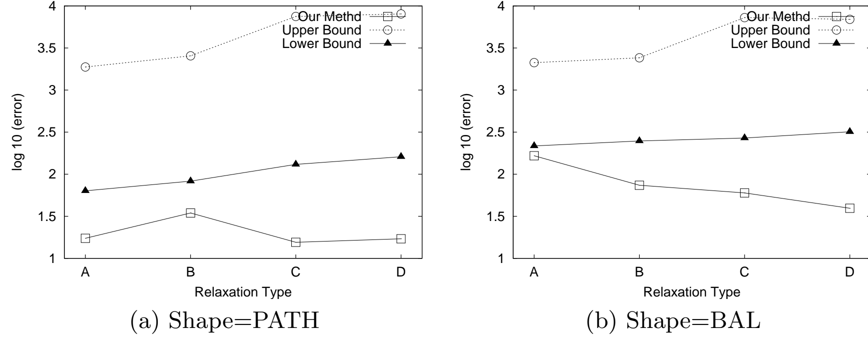
Setup: We used two real XML data sets in our experiments as follows: (1) DBLP[6] is about 10MB, *flat* (i.e., most instances have at most depth 2 or 3) and *regular* (i.e., the types of sub-elements that every root node has are very similar), and (2) SPROT[7] contains annotated protein sequences, and about 5MB. Its structure is rather *irregular* in that the types of sub-elements under the same

[6] ftp://ftp.informatik.uni-trier.de/pub/users/Ley/bib/records.tar.gz
[7] http://www.expasy.ch/sprot



(a) Type=A (b) Type=C

Fig. 3. Average relative squared error of the three methods in $\log_{10}$ scale as the CST space allowed for the estimation increases (Dataset=SPROT, Shape=BAL, Relationship=CHILD).

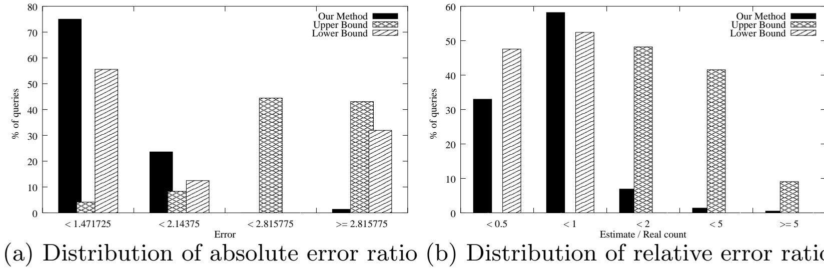


(a) Shape=PATH (b) Shape=BAL

Fig. 4. Average relative squared error of the three methods for query sets with different *shapes* in $\log_{10}$ scale as the relaxation types change (Dataset=DBLP, Relationship=CHILD).

element vary greatly. As to algorithms to test, we implemented three methods (i.e., lowerbound, upperbound, and our proposed methods) in Equation 1 of Section 2. Our implementation is the extension of the MSH algorithm (selectivity estimation method for twigs) and CST summary structure (correlated subpath tree) in [6], which was shown the best among the proposed methods.

We experimented with various forms of query sets (shown in Table 2) with 1,000 queries each: (1) NumBranch (# of branches in a twig) and Height (# of nodes in the path) lead to four shapes of query sets – PATH (trivial path), BS (bushy & shallow), DS (deep & skinny) and BAL (balanced), (2) NumRlxQry (# of relaxed queries per original query) and NumRlx (# of relaxations occurring in each relaxed query) lead to four sets – A, B, C, D, where A roughly being the lightest relaxations and D being the heaviest relaxations, and (3) CHILD with only parent-child relationship and BOTH with both parent-child and ancestor-



(a) Distribution of absolute error ratio (b) Distribution of relative error ratio

Fig. 5. Percentage of queries for the absolute and relative errors (Dataset=SPROT, Shape=DS, Relationship=CHILD, Type=D).

descendant relationships. In total, we have generated query sets: $|\text{data set}| \times |\text{shape}| \times |\text{type}| \times |\text{relationship}| = 2 \times 4 \times 4 \times 2 = 64$.

Following [6], we used *average relative squared error*, defined as: $Error = \frac{1}{|Q|} \sum_{i \in Q} \frac{(Q_i - Q'_i)^2}{Q'_i}$ where Q is a workload of test queries, Q_i is the true selectivity to a query and Q'_i is our estimate.

Results: Figure 3 shows the average relative squared error of all methods for the SPROT data set and the BAL query set as the CST size increases. The results for the DBLP data set exhibit similar patterns (except it requires less space than the SPROT due to its simpler structure) and omitted. As the space increases, the performance of all three methods improves steadily regardless of the types of relaxations. Type B (not shown) is similar to type A and type D (not shown) is similar to type C.

Figure 4 shows the average relative squared error of the three methods for the DBLP data set as the relaxation types change. It again has similar trends as the case for the SPROT data set in Figure 3; that is, our proposal outperforms the other two, where the lowerbound method is the second. For most experiments that we have conducted for the DBLP set, we found that lowerbound method typically closely follows the curve of our proposal while upperbound method is rather far apart. This can be explained as follows. Since the DBLP data set has a relatively shallow and flat schema, even the relaxed queries could not find much more new answers.

Figure 5 shows the different percentages of queries in terms of the distribution of error ratios among three methods. In Figure 5(a), the absolute error ratio for our proposed method is the best among three methods. More importantly, as shown in Figure 5(b), upperbound method tends to over-estimate and lowerbound method tends to under-estimate. This is no surprise since they do not consider "overlap" in their estimation. As the original query tends to have more number of relaxed queries ($3 \leq$NumRlxQry≤ 5) and more number of relaxations occurred in each case ($2 \leq$NumRlx≤ 5), new relaxed queries will inevitably contain more new answers, creating more disjoint answer space. However, the fact that the degree of over-estimation of the upperbound method is far more

severe than that of the under-estimation of lowerbound method suggests that this particular query set (Dataset=SPROT, Shape=DS, Relationship=CHILD, Type=D) has less number of "disjoint" answers than "overlapped" answers.

Due to space limitations, many other results are omitted and can be found in [9]. To summarize, our proposal consistently delivers a good estimate regardless of the factors affecting the query characteristics.

5 Conclusion and Future Work

In this paper, we looked at the problem of estimating the number of approximate answers of an XML twig query, when certain query relaxations are permitted, and proposed an extension to the correlated subpath tree (CST) such that error is minimized by algebraic manipulation of relaxed queries.

Our work is the first to explore the problem of selectivity estimation of tree pattern relaxations, and opens up many interesting directions of future work. How does one optimize the evaluation of relaxed twig pattern queries taking our estimates into account? How can one quickly identify the dominant (i.e., the ones contributing most of the answers) relaxed queries?

References

[1] S. Amer-Yahia, S. Cho, and D. Srivastava. "Tree Pattern Relaxation". In *EDBT*, Prague, Czech Republic, Mar. 2002.
[2] A. Aboulnaga et al. "Estimating the Selectivity of XML Path Expressions for Internet Scale Applications". In *VLDB*, Roma, Italy, Sep. 2001.
[3] A. Berglund et al. "XML Path Language (XPath) 2.0". W3C Working Draft, Nov. 2003. http://www.w3.org/TR/xpath20.
[4] S. Boag et al. "XQuery 1.0: An XML Query Language". W3C Working Draft, Nov. 2003. http://www.w3.org/TR/xquery.
[5] W. W. Chu et al. "CoBase: A Scalable and Extensible Cooperative Information System". *J. Intelligent Information Systems (JIIS)*, 6(2/3):223–259, May 1996.
[6] Z. Chen et al. "Counting Twig Matches in a Tree". In *IEEE ICDE*, Heidelberg, Germany, Apr. 2001.
[7] N. Fuhr and K. Grossjohann. "XIRQL - An Extension of XQL for Information Retrieval". In *ACM SIGIR*, New Orleans, LA, Sep. 2001.
[8] Y. Kanza and Y. Sagiv. "Flexible Queries over Semistructured Data". In *ACM PODS*, Santa Barbara, CA, May 2001.
[9] D. Lee and D. Srivastava. "Counting Relaxed Twig Matches in a Tree". Technical report, UCLA Computer Science Dept., Feb. 2002.
[10] J. McHugh and J. Widom. "Query Optimization for XML". In *VLDB*, Edinburgh, Scotland, Sep. 1999.
[11] N. Polyzotis and M. N. Garofalakis. "Structure and Value Synopses for XML Data Graphs". In *VLDB*, Hong Kong, China, Aug. 2002.
[12] A. Theobald and G. Weikum. "Adding Relevance to XML". In *Int'l Workshop on the Web and Databases (WebDB)*, Dallas, TX, May 2000.
[13] Y. Wu, J. M. Patel, and H. V. Jagadish. "Estimating Answer Sizes for XML Queries". In *EDBT*, Prague, Czech Republic, Mar. 2002.