Tree-Pattern Similarity Estimation for Scalable Content-based Routing

Raphaël Chand *
University of Geneva, Switzerland
raphael.chand@cui.unige.ch

Pascal Felber
University of Neuchâtel, Switzerland
pascal.felber@unine.ch

Minos Garofalakis
Intel Research Berkeley, USA
minos.garofalakis@intel.com

Abstract

With the advent of XML as the de facto language for data publishing and exchange, scalable distribution of XML data to large, dynamic populations of consumers remains an important challenge. Content-based publish/subscribe systems offer a convenient design paradigm, as most of the complexity related to addressing and routing is encapsulated within the network infrastructure. To indicate the type of content that they are interested in, data consumers typically specify their subscriptions using a tree-pattern specification language (an important subset of XPath), while producers publish XML content without prior knowledge of any potential recipients. Discovering semantic communities *of consumers with similar interests is an important requirement for scalable content-based systems: such "semantic clusters" of consumers play a critical role in the design of effective content-routing protocols and architectures. The fundamental problem underlying the discovery of such semantic communities lay in effectively evaluating the* similarity *of different tree-pattern subscriptions based on the observed document stream. In this paper, we propose a general framework and algorithmic tools for estimating different tree-pattern similarity metrics over continuous streams of XML documents. In a nutshell, our approach relies on continuously maintaining a novel, concise synopsis structure over the observed document stream that allows us to accurately estimate the fraction of documents satisfying various boolean combinations of different tree-pattern subscriptions. To effectively capture different branching and correlation patterns within a limited amount of space, our techniques use ideas from hash-based sampling in a novel manner that exploits the hierarchical structure of our document synopsis. Experimental results with various XML data streams verify the effectiveness of our approach.*

1. Introduction

XML (eXtensible Markup Language) [24] has become the dominant standard for data encoding and exchange. Given the rapid growth of XML traffic on the Internet, the effective and efficient delivery of XML documents is an important issue. As a consequence, there is growing interest in the area of XML *content-based filtering and routing*, which addresses the problem of effectively directing high volumes of XML-document traffic to interested consumers based on document *contents*.

Unlike conventional routing, where packets are routed based on a limited, fixed set of attributes (e.g., source/destination IP addresses and port numbers), content-based publish/subscribe systems route messages on the basis of their content and the interests of the message consumers. Consumers typically specify *subscriptions*, indicating the type of XML content that they are interested in, using some XML pattern specification language (e.g., XPath [23]). For each incoming XML document, a *content-based router* matches the document contents against the set of subscriptions to identify and route the document to the (sub)set of interested consumers. Therefore, the "destination" of an XML document is generally unknown to the data producer and is computed *dynamically* based on the document contents and the active set of subscriptions.

Traditional content routing systems are usually based on a fixed infrastructure of reliable brokers that filter and route documents on behalf of producers and consumers (e.g., [6]). This routing process is a complex and time-consuming operation, as it often requires the maintenance of large routing tables on each router and the execution of complex filtering algorithms (e.g., [1, 5, 12]) to match each incoming document against every known subscription. The use of summarization techniques (e.g., subscription aggregation [3, 4]) alleviates those issues, but at the cost of significant control message overhead or a loss of routing accuracy.

Another approach to XML content routing consists in gathering consumers with similar interests so as to form *semantic communities*. Thereafter, messages can be quickly disseminated within a community without incurring significant filtering cost [7]. Obviously, for such techniques to be efficient, one need to organize consumers according to adequate *proximity metrics*, i.e., by creating semantic communities that correctly map to the interests of the consumers. One such metric is that of containment: a subscription p *contains* another subscription q, or $q \sqsubseteq p$, if and only if any message m that matches q also matches p. However, this metric is not adequate for building semantic communities

*This work was done while the author was at INRIA Sophia Antipolis, France.

as it is asymmetric and evaluates to a boolean value, hence producing tree (inclusion-based) topologies instead of clusters. The challenge lies in the design of a proximity metric that can determine with high accuracy whether distinct tree pattern subscriptions are likely to represent the same set of documents, and hence should be part of the same semantic community.

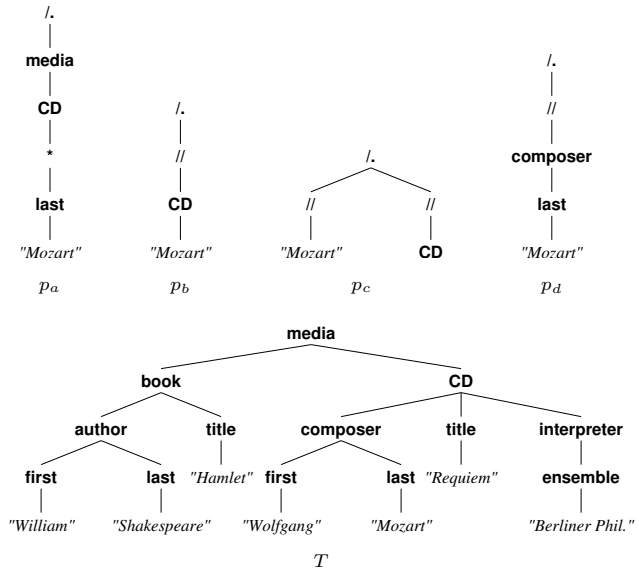

Figure 1: Sample tree patterns (top) and XML tree (bottom).

In this paper, we specifically address the problem of computing the similarity between sets of tree patterns (which do not have containment relationships in the general case). The objective is to evaluate the proximity between two given tree patterns (e.g., XPath expressions) in the context of XML filtering, i.e., the probability that both patterns yield the same result when applied against a given XML document. Obviously, consumers with highly similar subscriptions are good candidates for being part of the same semantic community.

Example 1.1 *Consider the two tree pattern subscriptions p_a and p_b shown in Figure 1: p_a specifies documents with a root element labeled "media" that has a child labeled "CD", which in turn has a grandchild labeled "last" with a sub-element labeled "Mozart"; p_b specifies documents that have an element labeled "CD" (at any depth) with a sub-element labeled "Mozart". Here, the node labeled "*" (wildcard) represents any label, while the node labeled "//" (descendant) represents some (possibly empty) path.*

The XML document T shown in Figure 1 matches p_a but not p_b because the sub-element labeled "Mozart" in T does not have a parent element labeled "CD". As a matter of fact, it is unlikely that any XML document with the same type descriptor (DTD) as T matches p_b because the name of the CD's author is expected two levels deeper than specified in the pattern. A document matching p_a is thus unlikely to match p_b (and vice versa), and the patterns have therefore low similarity.

Pattern p_c specifies documents that have an element labeled "CD" and an element labeled "Mozart" (both can appear at any depth). The XML document T matches p_c and it trivially appears that p_c contains p_a—any document that matches p_a also matches p_c—but the converse is not true: a wide range of XML documents can match p_c but not p_a. In particular, "Mozart" doesn't need to be the composer of a CD, but could be for instance the title of a book. Therefore, while there is some similarity between p_a and p_c, these patterns are clearly not equivalent.

Pattern p_d specifies documents that have an element labeled "composer" (at any depth) with a child labeled "last" and a grandchild labeled "Mozart". Formally, there is no *containment relationship between p_a and p_d although document T matches both. Taking into account the XML document type and assuming that T shows all valid elements (i.e., other XML documents of the same type will have the same structure, with variations only in the cardinality of the elements and the values at the leaves), then any document that matches p_a must also match p_d and conversely: the "*" in p_a* must *correspond to "composer" while the "//" in p_d* must *correspond to the path "media/CD". Therefore, both patterns are equivalent with respect to T and XML documents of the same type.* □

The similarity between two tree patterns p and q can be evaluated based on their selectivity. For instance, one can estimate the probability $P(p|q)$ that p matches a document T given that q matches T. Existing solutions cannot be directly adapted for our problem, because they only address single-path selectivity [8], only support exact match in tree patterns (no "*" or "//") [9], or do not accurately estimate conjunctions [4].[1]

In contrast, the main focus of this paper is to accurately evaluate the similarity of seemingly unrelated tree patterns (e.g., patterns p_a and p_d in Figure 1) using information derived from a possibly infinite stream of XML documents. Given the large volume of observed XML data, the key challenge is to incrementally maintain a concise, yet accurate *synopsis* that allows us to effectively capture the correlations of pattern occurrences across documents in the stream; for instance, to estimate the fraction of documents containing both patterns p and q, rather than only p or q. To that end, we propose a novel compact XML stream synopsis that can accurately capture such cross-pattern correlations using stream-sampling techniques that exploit the hierarchical nature of XML documents. Our synopsis can be maintained incrementally over the document stream and employs new pruning algorithms for maintaining an accurate picture of the document correlations within a given space budget.

[1] Note that support for conjunctions is necessary to accurately estimate $P(p|q) = P(p \wedge q)/P(q)$.

We also present a simple recursive algorithm for estimating tree-pattern selectivities and similarities over our proposed synopsis. We should stress that the usefulness of our results is not limited to content-based routing, but also extends to other application domains, such as approximate XML queries involving tree patterns.

The rest of the paper is organized as follows. We first formulate the problem in Section 2. We introduce the synopsis structure in Section 3 and describe how it is used to compute the similarity of tree pattern in Section 4. We evaluate the effectiveness of our algorithms in Section 5 and discuss related work in Section 6. Finally, Section 7 concludes.

2. Problem Formulation

Definitions. We use a subset of XPath for expressing tree-structured XML queries. A *tree pattern* is an unordered node-labeled tree that specifies constraints on the content and the structure of an XML document. In this paper, we mostly reuse the terminology and notation introduced in [4]. The set of nodes of a tree pattern p is denoted by $Nodes(p)$, where each node $v \in Nodes(p)$ has a label, $label(v)$, which can either be a tag name, a "*" (wildcard that can correspond to any tag), or a "//" (the descendant operator). A descendant operator must have exactly one child that is either a regular node or a "*". We define a partial ordering $\preceq$ on node labels such that if a and a' are tag names, then (1) $a \preceq * \preceq //$ and (2) $a \preceq a'$ if and only if $a = a'$.

The root node $root(p)$ has a special label "/.". We use $Subtree(v, p)$ to denote the subtree of p rooted at v, referred to as a *sub-pattern* of p, $Children(v)$ to denote the set of children of v, $parent(v)$ to denote the parent of v, and $root(p) \rightarrow v$ to denote the path from the root of p to node v. XML documents are represented as node-labeled trees, referred to as *XML trees*. The notation for $Nodes$, $Subtree$, $Children$, $parent$, $label$, and $root$ also applies to XML trees.

Let T be a node-labeled XML tree with $t \in Nodes(T)$, and p be a tree pattern with $v \in Nodes(p) \setminus root(p)$. We say that T *matches* or *satisfi esSubtree*(v, p) at node t, denoted by $(T, t) \models Subtree(v, p)$, if the following conditions hold: (1) if $label(v)$ is a tag, then t has a child node t' labeled $label(v)$ such that for each child node v' of v, $(T, t') \models Subtree(v', p)$; (2) if $label(v) =$ "*", then t has a child node t' labeled with an arbitrary tag such that for each child node v' of v, $(T, t') \models Subtree(v', p)$; and (3) if $label(v) =$ "//", then t has a descendant node t' (possibly $t' = t$) such that for each child v' of v, $(T, t') \models Subtree(v', p)$.

Let T be an XML tree with $t_r = root(T)$, and p be a tree pattern with $v_r = root(p)$. We say that T *matches* or *satisfi esp*, denoted by $T \models p$, if and only if the following conditions hold for each child node v of v_r: (1) if $label(v)$ is a tag a, then t_r is labeled with a and for each child node v' of v, $(T, t_r) \models Subtree(v', p)$; (2) if $label(v) =$ "*", then t_r may have any label and for each child node v' of v, $(T, t_r) \models Subtree(v', p)$; (3) if $label(v) =$ "//", then t_r has a descendant node t' (possibly $t' = t_r$) such that $T' \models p'$, where T' is the subtree rooted at t', and p' is identical to $Subtree(v, p)$ except that "/." is the label for the root node v (instead of $label(v)$). The reason for treating v_r differently from the rest of the nodes of p is illustrated by p_c in Figure 1: the node labeled "CD" may appear anywhere in the XML document, including at the root, and it may or not be an ancestor of the node labeled "Mozart". This cannot be expressed without our special root label "/." as tree patterns do not allow a union operator.

It is worth mentioning that our tree patterns are graph representations of a class of XPath expressions, which are similar to the tree patterns that have been studied for XML queries (e.g., [2, 27]).

Problem Statement. We can now state the *tree pattern similarity* problem that we address in this paper as follows. Consider two tree pattern subscriptions p and q. Let $\mathcal{S}$ and $\mathcal{D}$ be the universe of all valid subscriptions and XML documents, respectively. We want to estimate the *similarity* between p and q, denoted by $(p \sim q)$ and defined as a function of $\mathcal{S}^2 \mapsto [0, 1]$ that returns the probability that p matches the same subset of XML documents from $\mathcal{D}$ as q does. Depending on the proximity metric that we use to estimate similarity, this relation may or may not be symmetric.

3. Summarizing the Document Stream

Given the large volume of streaming XML content, it is clearly infeasible to maintain an accurate distribution of all documents — the space requirements would be linear in the size of the stream. Instead, our approach is to maintain a concise, yet accurate *synopsis* that effectively captures the path distribution of the XML document stream, and allows us to estimate the fractions of documents satisfying different patterns (i.e., tree pattern selectivities). Estimating pattern similarity metrics makes the problem even more complex: essentially, our methods need some space-efficient means of capturing *cross-pattern correlations* across documents in the stream. In other words, given two tree patterns p and q, our synopsis should allow us to effectively capture potential correlations in the occurrence of p, q in the streaming documents. Note that, especially for "similar" patterns, such correlations will be quite strong and naive independence assumptions are likely to fail miserably.[2]

3.1. Building the Synopsis

As mentioned above, it is simply impossible to maintain an accurate distribution for the complete XML-document

[2] In the presence of DTDs, we can obtain additional structural information about the documents to enhance our synopsis. To simplify the exposition, in this paper, we assume no DTD information is available.

history H, in order to obtain accurate similarity estimates for our tree patterns. Instead, our approach is to approximate H by a concise *document synopsis* structure, H_S, that is built *on-line* as XML documents stream by. Our document synopsis essentially has the same structure as an XML tree, except for two key differences. First, the root node of H_S has the special label "/.". Second, each non-root node t in H_S is associated with a *matching set*, denoted by $S(t)$, that tracks the collection of documents that contain the given node.

Intuitively, if $l_1/l_2/\cdots/l_n$ is the sequence of tag names on nodes along the path from the root to t (excluding the label for the root), then $S(t)$ represents the set of document identifiers in H that contain a path "$l_1/l_2/\cdots/l_n$" originating at the root. The purpose of the matching set information stored within synopsis nodes is to capture cross-pattern correlations that can be used during our selectivity estimation process (Section 4). It is clear, however, that the size of this information is in the order of the size of the stream, so it needs to be effectively compressed — we discuss different compression schemes later in this section.

H_S is incrementally maintained as XML documents stream by. We briefly describe a straightforward maintenance technique here, as the details may vary depending on the specifics of the matching set representation. For each arriving document T, we first construct the *skeleton tree* T_s for document T. In the skeleton tree T_s, each node has at most one child with a given tag. T_s is built from T by simply coalescing two children of a node in T if they share a common tag. Clearly, by traversing nodes in T in a top-down fashion and coalescing child nodes with common tags, we can construct T_s from T in a single pass (using an event-based XML parser). Next, we use T_s to update our synopsis H_S as follows. For each path in T_s ending with node t, let t' be the last node on the corresponding unique path in H_S (we add missing nodes if the path does not yet exist in H_S). We add the document identifier for T to $S(t')$.

Example 3.1 *Figure 2 shows several XML documents $T_1, \ldots, T_6$, the skeleton trees of T_1 and T_3, and the resulting document synopsis. The synopsis contains information about uncompressed matching sets (represented by a set of document identifiers). The frequency of a path from the root to node t is given by $|S(t)|$, and the probability that such a path is encountered in an XML document is given by $|S(t)|/|H|$. For instance, the probability of an occurrence of path "/a/b/h" is $1/6$ (it only appears in one of the 6 documents); we can also observe that elements "b" and "d" are more frequent than "c". The probability that an XML document matches a tree pattern with more than one branch, or with "*" and "//" nodes, will be discussed later.* □

3.2. Compressing Matching Set Information

Given that we cannot afford to store information whose size is in the order of the stream length, we need effective methods of summarizing matching set information in our synopsis nodes. We discuss three different alternatives for matching set compression in what follows.

Counters. In its simplest form, a matching set can be summarized as a simple counter that indicates the number of documents that contain the corresponding synopsis node. This simple frequency-based approach (also proposed in [4]) may work reasonably for estimating the selectivity of single tree patterns; however, it fails to effectively capture cross-pattern correlations since it relies on naive independence assumptions to handle conjunctions (i.e., branching points in the patterns). Consider, for instance, the tree pattern $p =$"a[b][d]" (a node "a" with two descendants "b" and "d") and the synopsis of Figure 2. Using counters, one would estimate the probabilities of paths "a/b" and "a/d" as $1/2$, and multiply them to compute the probability of p as $1/4$; the correct value is 0 because elements "b" and "d" are mutually exclusive in the XML documents. Similarly, the probability of tree pattern $p =$"a[c/f][c/o]" would be under-evaluated as $1/9$; its correct value is $1/3$ because of the co-occurrence of elements "f" and "o" in the XML documents. The loss of precision can grow larger with more complex tree patterns.

Fixed-Size Sample Sets. A second simple strategy consists in using sampling over the document stream to ensure that our synopsis only stores information about a (fixed-size) random subset of the streaming documents. To ensure that a uniform sample of documents is kept in the sets, the probability of adding a document in the synopsis decreases with its position in the stream. More precisely, using Vitter's reservoir-sampling scheme [22], given a desired sample size of s, the k^{th} document in the stream is included in the synopsis with probability $\min\{1, s/k\}$; if there is no sample slot available at the root node of the synopsis (i.e., $k > s$), a document identifier chosen uniformly at random from the current sample is removed from all synopsis nodes to make room for the new selected document. This strategy guarantees that the paths from a (fixed-size) uniform random sample of the document stream are used to maintain the synopsis. The drawbacks of this approach is that it relies on fairly coarse, document-level sampling, with a sampling rate decided uniformly across all synopsis nodes. This typically implies that only information from a small number of documents and paths is maintained, resulting in poor estimates. The sampling space is also not uniformly utilized across synopsis nodes, with most nodes holding only a few elements. Furthermore, due to the coarse-grain sampling, the given space budget for the synopsis may very well be under-utilized at times, since new arrivals may cause several

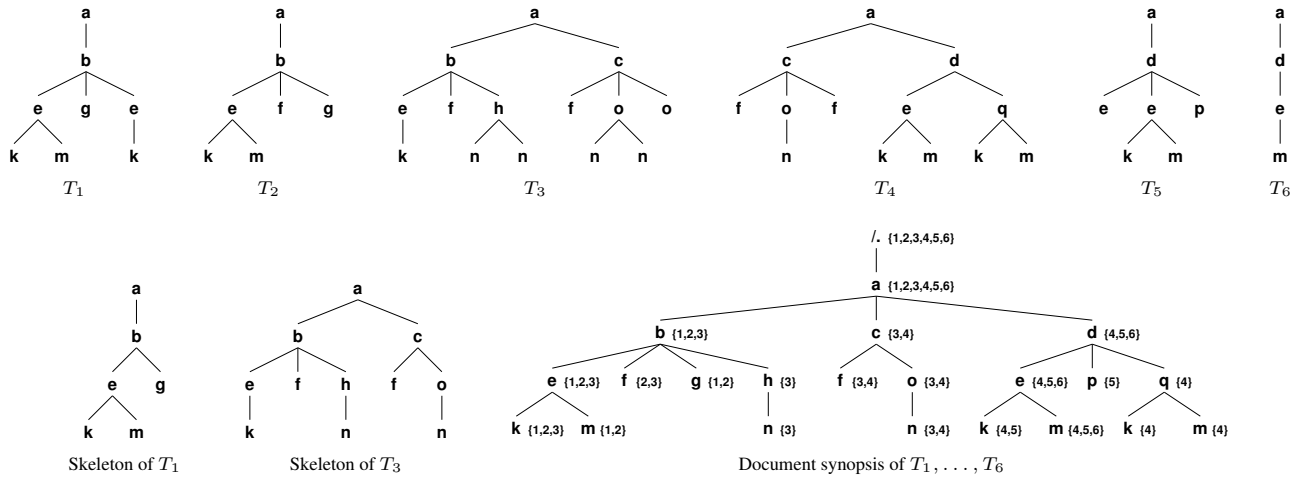

Figure 2: Example XML documents (top) and the corresponding skeleton trees and document synopsis (bottom).

nodes in the synopsis to be deleted (when their matching set becomes empty).

Per-Node Hash Samples. Our final, and more sophisticated, scheme for compressing matching sets in our document synopsis uses Gibbons' *distinct sampling* technique [15] for maintaining a sample of distinct elements (of a given target size) over a stream. Briefly, the main idea is to employ a hash function $h()$ that maps element identifiers onto a logarithmic range of *levels*, such that each element maps to a level $\geq l$ with probability $1/2^l$; that is, $\mathbf{Prob}[h(x) \geq l] = \frac{1}{2^l}$. Thus, every element in the stream maps to a level ≥ 0, approximately $1/2$ of the elements map to a level ≥ 1, and so on. Given a target sample size s, the idea is to start with level $= 0$ (i.e., sampling every element with probability 1); whenever the sample size exceeds s, set the current level to (level $+1$) and sub-sample the current set of sampled elements to keep only the ones mapping to the current level and above (reducing the size of the hash sample by approximately $1/2$). In other words, the current level determines the (appropriate power-of-two) sampling probability for an element in the stream.

Our final compression method essentially maintains such a bounded-size hash-sample signature of the $S(t)$ sets at each synopsis node t. More specifically, for each incoming document T, every root-to-leaf path in the document skeleton T_s is mapped to a (unique) synopsis path, and the document identifier of T is inserted in the hash sample for the *last node* of that synopsis path.

It is important to note that updating $S()$ only at the final node of an incoming path is sufficient, since the full matching set of a parent node in the document synopsis is guaranteed to contain those of its children. A hash sample of the full matching set at a node t in the synopsis can be computed by recursively unioning the hash samples across all descendants of t. Unioning two hash samples is a simple operation: letting l_1 and l_2 denote the current levels of the two samples, the idea is to set the level of the output sample to $l = \max\{l_1, l_2\}$, and sub-sample the higher probability (lower-level) sample down to level l; an additional sub-sampling step with level $= l + 1$ may be needed if the size of the output exceeds the space budget s. Similar schemes can be used for effectively estimating other expressions (e.g., intersection, cardinality) over the matching sets based on the maintained hash samples; more details can be found in [15, 14]. These algorithms form a key part of our selectivity-estimation procedure.

Unlike our earlier document-level sampling scheme, per-node hash samples allow us to sample document identifiers at a much finer granularity, essentially sampling the identifiers "hitting" individual nodes in the synopsis based on a per-node space budget. In general, this implies better quality samples (and corresponding selectivity/similarity estimates) that can effectively exploit the available space budget for the document synopsis. At the same time, even though the synopsis factorizes common paths in the document stream to store the corresponding matching set samples, the volume of these paths could cause the size of the synopsis to grow well beyond our available space budget. Thus, effective techniques for synopsis pruning become critical — we address this issue next.

3.3. Pruning the Document Synopsis

To keep the space requirements of our document synopsis under control, we now propose different techniques for pruning the synopsis (i.e., removing nodes) while minimizing the loss of precision during selectivity evaluation. These techniques should be combined during synopsis maintenance to ensure that the size of the synopsis stays within the available space budget.

Merging Same-Label Nodes. Our first pruning operation merges same-label nodes that have similar matching sets, unioning their corresponding hash-sample summaries. To minimize precision loss, same-label node pairs (say, t

and t') are selected in decreasing order of their (estimated) matching set similarity, i.e., the ratio $\frac{|S(t)\cap S(t')|}{|S(t)\cup S(t')|}$ as estimated using the hash-sample summaries in t and t'. The final, merged node has the same label as t and t', and its hash sample is computed as the intersection of the samples of t and t' (i.e., $S(t) \cap S(t')$). In general, this operation transforms the document synopsis tree into a DAG, since the merged nodes can have different parents located at different levels of the original synopsis. Also, note that the final merged node satisfies the inclusion property for the resulting parent-child matching sets: a node matches a subset of the documents that its parents match.

Our node-merging scheme only considers merging either same-label leaf nodes or same-label non-leaf nodes that share the same children (i.e., their children have already been merged). That way, we can merge complete subtrees in a bottom-up pass starting from the leaves, without losing structural information (i.e., introducing false label paths in the synopsis). Note that, if the matching sets of merged nodes are identical, the compression is essentially lossless. Figure 3 (right subtree) illustrates an example merge of same-label leaf nodes on the synopsis of Figure 2.

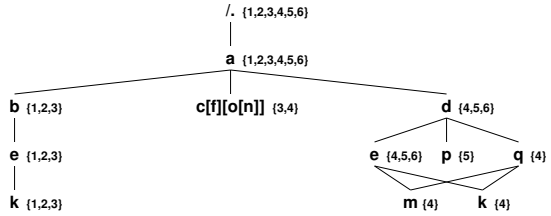

Figure 3: Compressed synopsis after merging same-label nodes (right subtree), folding nodes (middle subtree), and deleting nodes (right subtree).

Folding Leaf Nodes. Our second compression technique is based on discovering a parent-child pair of synopsis nodes (say, t_p, t), where the child node t is a leaf and has a matching set $S(t)$ that is similar to that of its parent, i.e., the (estimated) similarity ratio $\frac{|S(t)\cap S(t_p)|}{|S(t)\cup S(t_p)|}$ is large. Note that such pairs could be quite common, e.g., for children nodes that are mandatory according to the underlying document schema. Our compression scheme "folds" such leaf children t into the parent node t_p. Letting $l(t)$, $l(t_p)$ denote the labels of t and t_p, respectively, we define the label of the resulting folded node as $l(t_p)[l(t)]$ to denote the nesting of the child node previously under t_p. We also set the matching set hash sample for the resulting node equal to the union of the hash samples for t and t_p. Note that, in general, this operation can result in folded synopsis nodes with labels nested at several levels (essentially, representing synopsis subtrees), and the matching sets at the folded nodes capture the set of document identifiers that share all the paths in these subtrees. Thus, our folding scheme allows us to effectively compress the matching set information for entire subtrees appearing in similar subsets of documents in the stream. An example nested label is depicted in Figure 3 (middle subtree).

As previously, we apply folding operations across parent-leaf child pairs in decreasing order of their set-similarity ratio. In the case of synopsis leaves with multiple parents (which may result from node merges), we estimate the overall similarity score as the average of the similarity ratios across all parent nodes, and fold the leaf node into all its parents in the synopsis.

Deleting Low-Cardinality Nodes. Our final (and simplest) technique for synopsis pruning is to remove leaf nodes that have little influence on selectivity estimation. To that end, we choose candidate leaf nodes t that have small matching sets; that is, the count $|S(t)|$ is small. (In the case of sample summaries, this count can be easily estimated from the maintained sample.) By repeating this process, entire parts of the synopsis tree can be pruned. An example of low-frequency node deletions over the synopsis of Figure 2 is shown in Figure 3 (left subtree).

Note that, of the three pruning techniques presented here, leaf-node deletions are the primary means for controlling synopsis size in the case of counter summaries. With more detailed, sampling-based information stored in synopsis nodes, our first two pruning methods can provide significant space savings by compressing samples across nodes.

4. Estimating Tree Pattern Selectivity

We now describe how the *selectivity* of p, i.e., the probability $P(p)$ that a document matches p, is computed using the synopsis. The algorithm works independently of how matching sets are stored in the synopsis (as complete sets or as summaries), given that we can compute their union, intersection, and cardinality.

The Selectivity Algorithm. Let p be a tree pattern. The selectivity function SEL (Algorithm 1) parses recursively nodes of p against nodes of the synopsis H_S. When executed with the root nodes of H_S and p, the function returns the set of document identifiers that satisfy p — or an approximation thereof.

Let u be a node of p and v a node of the synopsis such that v *matches* u. Let $Children(u) = \{u_1 \cdots u_n\}$ be the children of u in p and $Children(v) = \{v_1 \cdots v_m\}$ the children of v in the synopsis. The root nodes of the synopsis and p are denoted by r_s and r_p, respectively (note that r_p is the node with label "/." introduced in Section 2). Let $S(v)$ be the set of documents that contain path $r_s \rightarrow v$.

Intuitively, the algorithm tries to locate paths in H_S that satisfy root-to-leaf paths of p (line 4), taking their unions and, upon branching, the intersection over all children in p (line 9). The $\preceq$ operator (line 1) allows a "*" node in p to match any label in H_S. The process is slightly more

complex when encountering a "//" node, because we try to map it to paths of length 0 (line 12) or ≥ 1 (line 13).

Some subtle modifications must be performed to the algorithm when representing matching sets as counters (see Section 3). Essentially, we replace respectively the union, intersection, and cardinality of sets by the maximum, product, and value of counters.

Algorithm 1 Recursive selectivity function: $\text{SEL}(v, u)$

1: **if** $label(v) \not\preceq label(u)$ **then**
2: $\quad \text{SEL}(v, u) = \varnothing$
3: **else if** u is a leaf **then**
4: $\quad \text{SEL}(v, u) = S(v)$
5: **else if** $label(u) \neq //$ **then**
6: $\quad$ **if** v is a leaf **then**
7: $\quad\quad \text{SEL}(v, u) = \varnothing$
8: $\quad$ **else**
9: $\quad\quad \text{SEL}(v, u) = \bigcap_{u' \in Children(u)} \bigcup_{v' \in Children(v)} \text{SEL}(v', u')$
10: $\quad$ **end if**
11: **else** $\{label(u) = //\}$
12: $\quad S_0 = \bigcap_{u' \in Children(u)} \text{SEL}(v, u')$
13: $\quad S_{\geq 1} = \bigcup_{v' \in Children(v)} \text{SEL}(v', u)$
14: $\quad \text{SEL}(v, u) = S_0 \cup S_1$
15: **end if**

Algorithm 2 Selectivity function: $P(p)$

1: $\text{P}(p) = |\text{SEL}(r_s, r_p)|/|S(r_s)|$

Note that, for tree patterns that contain "//" nodes, a pair (v, u) of nodes from H_S and p can be evaluated more than once. Although not shown in the algorithm, one can trivially store the results of previous evaluations (e.g., in a 2-dimensional array) to avoid redundant computations and guarantee a quadratic time complexity in $O(|H_S| \cdot |p|)$ time, where $|H_S|$ and $|p|$ are the number of nodes in the synopsis and in the tree pattern, respectively.

To estimate the selectivity of a tree pattern p, function P (Algorithm 2) computes the ratio of the cardinality of the set returned by SEL, invoked on the root nodes of H_S and p, to the number of documents added to the synopsis.

Proximity Metrics. Using our selectivity function, we can now evaluate the similarity between a pair of tree patterns p and q using a given proximity metric. We describe below three specific metrics that, we believe, make sense for the estimation of $(p \sim q)$.

The first metric is the *conditional probability* of p given q, computed as:

$$M_1(p, q) = P(p|q) = \frac{P(p \wedge q)}{P(q)}.$$

To estimate $P(p \wedge q)$, we construct a new tree by simply merging the root nodes of p and q.

Our second metric is the mean of the conditional probabilities of p and q, computed as:

$$M_2(p, q) = \frac{P(p|q) + P(q|p)}{2} = \frac{P(p \wedge q) \cdot (P(p)^{-1} + P(q)^{-1})}{2}.$$

The third metric that we consider is the ratio of the joint probability to the union probability, computed as:

$$M_3(p, q) = \frac{P(p \wedge q)}{P(p \vee q)} = \frac{P(p \wedge q)}{P(p) + P(q) - P(p \wedge q)}.$$

Note that the M_2 and M_3 metrics are symmetric, i.e., $M_{\{2,3\}}(p, q) = M_{\{2,3\}}(q, p)$, while M_1 is not.

5. Evaluation

We now present the performance study that we conducted to verify the effectiveness of our tree pattern similarity evaluation algorithm.

5.1. Experimental Setup

Data sets. We have generated sets of XML documents with IBM's XML Generator [13] tool, using a uniform distribution for selecting element tag names. Each set, which we denote by D, consists of $10,000$ random documents with approximately 100 tag pairs on average and up to 10 levels.

The tree patterns considered in this paper are expressed using a subset of the standard XPath language. We have generated realistic subscription workloads using a custom XPath generator that takes a Document Type Descriptor (DTD) as input and creates a set of valid XPath expressions based on several parameters that control: the maximum height h of the tree patterns; the probabilities p_* and $p_{//}$ of having a wildcard ("*") and descendant ("//") operators at a node of a tree pattern; the probability p_λ of having more than one child at a given node; and the skew θ of the Zipf distribution used for selecting element tag names. For our experiments, we have generated sets of distinct tree patterns of various sizes, with $h = 10$, $p_* = 0.1$, $p_{//} = 0.1$, $p_\lambda = 0.1$, and $\theta = 1$.

We have experimented with two different DTDs: NITF (News Industry Text Format) [10] and xCBL Order [26], which contain 123 and 569 elements, respectively. For each DTD, we have generated two different sets of XPath expressions. The first set, denoted by S_P, contains $1,000$ positive queries, i.e., each expression matches at least one document in D. The second set, denoted by S_N, contains $1,000$ expressions that match no documents in D.

For the NITF (xCBL) DTD, a tree pattern in S_P matches 8.27% (36.17%) of the documents in D, on average. The most selective pattern matches 0.01% (0.01%) of the documents in D, and the least selective 84.85% (100%).

Synopsis. We constructed the synopsis H_S from the set of documents D. We have experimented with the three different representations of the *matching sets* stored at the synopsis nodes that we presented earlier, namely: *Counters*, *Sets*, and *Hashes*. For the latter two, we have experimented with sets and hashes of different maximum sizes (i.e., the maximum number of entries that can be stored in a matching set at a given node).

We have also experimented with different degrees of compression of the synopsis H_S when using *Hashes*. We have pruned the synopsis using the techniques presented in Section 3. We refer to the compressed synopsis as H_S^c. The compression degree, α, is computed as: $\alpha = |H_S^c|/|H_S|$,

where $|H_S|$ is defined as the sum of the number of nodes, the number of edges, the number of labels, and the total number of entries of all hashes in H_S. Each of these elements can fit in a 32-bit integer (e.g., edges can be represented as node indices, labels can be stored as hashed strings, etc.).

It is important to note that the compression degree is not computed relative to the original set of documents D from which the synopsis is built. Indeed, we are interested in evaluating the synopsis compression techniques and study the loss of precision when using H_S^c instead of H_S. The original synopsis H_S is already a compacted representation of D, since it factorizes common paths in the documents. This compaction ratio is not interesting because it depends almost exclusively on the kind of documents used for the experiments. For instance, the DBLP XML document [11] has $7,991,221$ tag nodes that can be stored in a 137-nodes synopsis, i.e., with a compaction ratio of 0.0017%. With our data sets, we had average document compaction ratios of 36.3% for the NITF DTD and 0.082% for the xCBL DTD.

The different experimental parameters are summarized in Table 1.

Symbol	Name	Value range
D	Data set	NITF, xCBL
S	Tree patterns set	S_P, S_N
MS	Matching sets format	*Counters*, *Sets*, *Hashes*
h	Maximum hash size	$50 < h < 10,000$
k	Maximum set size	$50 < k < 10,000$
α	Compression ratio	$0 \leq \alpha \leq 1$

Table 1: Experimental parameters

Proximity metrics. To evaluate the three proximity metrics M_1, M_2, and M_3 introduced in Section 4, we have computed the subset D_p of documents of D that match each tree pattern p. Thereafter, the exact values of $P(p)$ and $P(p \wedge q)$ have been computed as $|D_p|/|D|$ and $|D_p \cap D_q|/|D|$, respectively.

Error metrics. We have measured the accuracy of our selectivity function using the same error metrics as in [9, 8]. Let $P(p)$ be the exact value for the selectivity of p, and $P'(p)$ our estimate. For a positive query p, we define the absolute relative error of the estimation of its selectivity as: $|P'(p) - P(p)|/P(p)$. We then define the average absolute relative error to quantify the accuracy of positive queries as:

$$E_{rel} = \frac{1}{|S_P|} \sum_{p \in S_P} \frac{|P'(p) - P(p)|}{P(p)}$$

For negative queries, we use the root mean square error to quantify the accuracy of their estimated selectivity:

$$E_{sqr} = \sqrt{\frac{1}{|S_N|} \sum_{p \in S_N} (P'(p) - P(p))^2}$$

For each of the proximity metrics M_i, we have measured the accuracy of the estimated tree pattern similarity as follows (M_i is the exact value, M_i' is our estimate):

$$E_{rel}(M_i) = \frac{1}{|S_P|^2} \sum_{p,q \in S_P} \frac{|M_i'(p,q) - M_i(p,q)|}{M_i(p,q)}$$

5.2 Results

Selectivity. We have studied the average absolute relative error of positive queries, for each representation of the matching sets, when varying the maximum hash size h and set size k. Note that h and k are upper bounds and may not be representative of the total number of entries in matching sets. Further, for a given value of $k = h$, the synopsis based on sets is typically smaller than when using hashes because the former stores information about at most k documents.

Results are shown in Figure 4. One can observe that *Hashes* clearly outperforms the other approaches and is less sensitive to the DTD. Unsurprisingly, the error decreases with the maximum size of hashes and sets. A hash size of $1'000$ entries is sufficient to reach a relative error of less than 5%. Even a very small hash size of 250 entries (less than 1 kB) produces an error as low as 10%. Note that the accuracy of *Counters* is constant as it does not use variable-size structures to represent matching sets.

We have then studied the root mean square error to quantify the accuracy of negative queries. Results are shown in Figure 5 (note that *Sets* and *Hashes* are not shown for the xCBL DTD as they produced no error for negative queries). One can see that all three methods most always identify negative queries and return a selectivity of 0 (the real selectivity). Again, *Hashes* outperforms the other approaches.

To better analyze the space requirements of the synopsis, we have analyzed the accuracy of the selectivity estimation as a function of the total size of the synopsis $|H_S|$ (computed as described above). The results, shown in Figure 6 for the xCBL DTD, enable us to perform a fairer evaluation of the three approaches. Obviously, *Counters* have the lowest space requirements ($3'567$, which should typically be multiplied by 4 to obtain the size in bytes), but limited accuracy. *Hashes* have better accuracy than *Sets* for a given space budget. For instance, the relative error reaches 5% for a size of less that $150'000$ (approximately 600 kB) while *Sets* need 4 times more space for this level of precision.

Proximity metrics. We have measured the average absolute relative error of the estimated similarity with each of the proximity metrics over a set of $5,000$ random pairs of tree patterns in S_P, for each representation of matching sets, and when varying the maximum size of hashes and sets. Results are shown in Figures 7, 8, and 9. One can observe that all three metrics produce similar results, thus highlighting the consistency of our selectivity estimation algorithm. Obviously, accuracy is slightly lower than when estimating selectivity because errors add up when computing similarity. Again, *Hashes* consistently outperforms the other approaches and produce good estimations with relatively small hashes (approximately 5% error for $h = 2'500$).

Compressed synopsis. We have finally measured the accuracy of the estimated selectivity on a compressed synop-

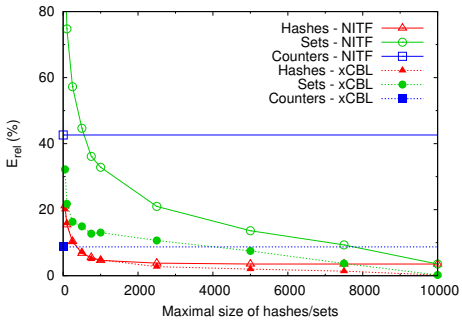

Figure 4: Average absolute relative error of positive queries.

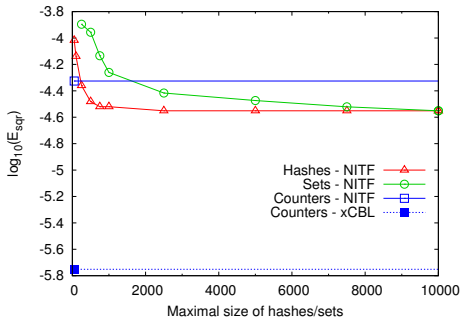

Figure 5: Log_{10} of the root mean square error of negative queries.

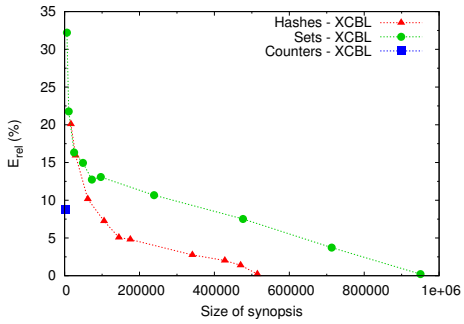

Figure 6: E_{rel} as a function of the total size of the synopsis.

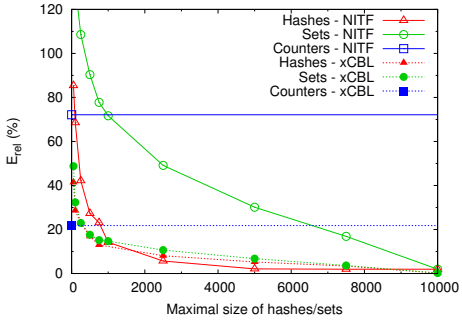

Figure 7: Average absolute relative error of proximity metric $M_1(p,q) = P(p|q)$.

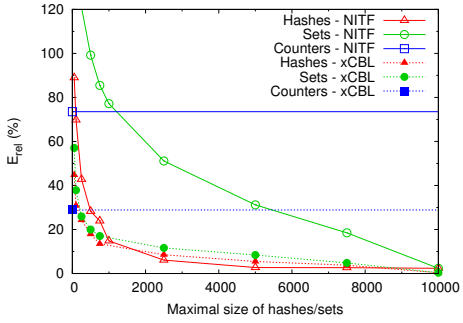

Figure 8: Average absolute relative error of proximity metric $M_2(p,q) = (P(p|q)+P(q|p))/2$.

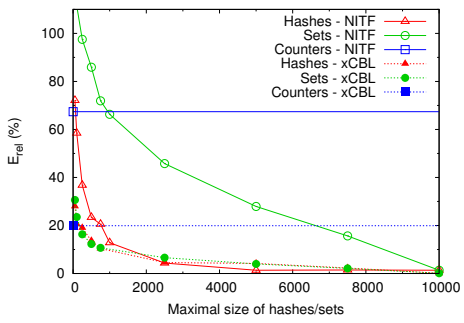

Figure 9: Average absolute relative error of proximity metric $M_3(p,q) = P(p \wedge q)/P(p \vee q)$.

sis, for different ratios of compression α. For this experiment, we only considered the *Hashes* approach and we fixed the hash size k to a value of $1,000$ entries. Compression techniques have been applied as follows: first, folding leaf nodes with the same matching set as their parents (lossless compression); then, folding and deleting low-cardinality nodes; finally, merging same-label nodes. We have obtained good overall results in our experiments by applying pruning operations in this order.

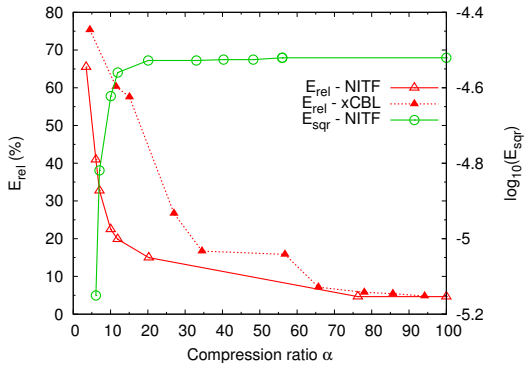

Figure 10: E_{rel} and E_{sqr} as a function of the compression ratio of the synopsis.

Figure 10 illustrates the average absolute relative error of positive queries E_{rel} and the root mean square error of negatives queries E_{sqr} (note that the xCBL DTD produced no error for negative queries). As expected, the error of positive queries decreases when α increases. The more compressed the synopsis, the more mistakes the selectivity function does. For a synopsis compressed to 20% of its original size, the relative error remains reasonably low (14.98%).

For negative queries, surprisingly, the error increases with the compression ratio, but still remains extremely low. The most logical explanation is that a highly compressed synopsis has lost many of its initial nodes and, hence, the selectivity function is less likely to contain paths that lead to erroneously accepting a negative query.

6. Related Work

The similarity between *data trees* has been extensively studied as a technique for linking data items in different databases that correspond to the same real world objects. The most widely used approach consists in computing the "edit distance" between two trees, i.e., the minimum cost sequence of edit operations (node insertion, node deletion, and label change) that transform one tree into the other (e.g., [20, 18]). In contrast, we study the similarity of XML *query trees*, where similarity is not defined in terms of structural resemblance, but according to the set of documents that match these queries. To the best of our knowledge, our work is the first to study this problem.

Query transformations have been proposed in the context of approximate matching. The idea is to rewrite queries for faster evaluation or to take into account the variability among XML data conforming to the same schema (e.g., [21, 16, 17, 19]). Some forms of tree patterns have also been studied as queries for XML data [2, 25]. In partic-

ular, minimization algorithms for these patterns have been developed in order to optimize pattern queries. These problems differ significantly from ours and the techniques proposed to address them have little relevance here.

Other related work deals with the problem of XPath selectivity estimation in XML databases. Chen et al. [8] propose the use of Pruned Suffix Tree (PST) summaries, where a trie is used to encode the paths in the document and special min-hash signatures within each node attempt to capture branching correlations. Their selectivity estimation problem deals with the number of *elements in a single document* that are discovered by an XPath expression; instead, we focus on pattern selectivities and similarities at the granularity of *entire documents containing the pattern(s)*. This makes our summarization and estimation problems very different. For instance, the parent-child inclusion properties for our hash samples do not apply to the min-hash signatures of [8].

7. Conclusions

We have studied the problem of *tree pattern similarity*, an important concept for building scalable XML distribution networks. We have proposed algorithms for accurately evaluating the similarity between tree patterns by taking into account information derived from document histories, such as correlations and frequency distributions, and using different proximity metrics. The principle of similarity computation relies on incrementally maintaining a novel, concise synopsis structure over the observed document stream that allows us to accurately estimate the fraction of documents satisfying different boolean combinations of tree-patterns. Our techniques use ideas from hash-based sampling in a novel manner that exploits the hierarchical structure of our document synopsis. Results from an experimental evaluation demonstrate that our similarity metric is very accurate and consistent. Although the algorithms presented in this paper have been designed for creating semantic communities in peer-to-peer content-based routing systems, they are of interest in their own right and can prove useful in other domains, such as approximate XML queries.

References

[1] M. Altinel and M. Franklin. Efficient Filtering of XML Documents for Selective Dissemination of Information. In *Proceedings of VLDB*, Sept. 2000.

[2] S. Amer-Yahia, S. Cho, L. Lakshmanan, and D. Srivastava. Minimization of Tree Pattern Queries. In *Proceedings of SIGMOD*, May 2001.

[3] A. Carzaniga, D. Rosenblum, and A. Wolf. Design and Evaluation of a Wide-Area Event Notification Service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001.

[4] C.-Y. Chan, W. Fan, P. Felber, M. Garofalakis, and R. Rastogi. Tree Pattern Aggregation for Scalable XML Data Dissemination. In *Proceedings of VLDB*, Aug. 2002.

[5] C.-Y. Chan, P. Felber, M. Garofalakis, and R. Rastogi. Efficient Filtering of XML Documents with XPath Expressions. In *Proceedings of ICDE*, Feb. 2002.

[6] R. Chand and P. Felber. XNet: A Reliable Content-Based Publish/Subscribe System. In *SRDS 2004, 23rd Symposium on Reliable Distributed Systems*, Florianopolis, Brazil, Oct. 2004.

[7] R. Chand and P. Felber. Semantic Peer-to-Peer Overlays for Publish/Subscribe Networks. In *Proceedings of Euro-Par 2005*, Sept. 2005.

[8] Z. Chen, H. Jagadish, F. Korn, N. Koudas, S. Muthukrishnan, R. Ng, and D. Srivastava. Counting twig matches in a tree. In *Proceedings of ICDE*, pages 595–604, 2001.

[9] Z. Chen, F. Korn, N. Koudas, and S. Muthukrishnan. Selectivity estimation for boolean queries. In *Proceedings of PODS*, pages 216–225, 2000.

[10] I. P. T. Council. News Industry Text Format.

[11] DBLP. DBLP XML records.

[12] Y. Diao, P. Fischer, M. Franklin, and R. To. YFilter: Efficient and Scalable Filtering of XML Documents. In *Proceedings of ICDE*, San Jose, CA, Feb. 2002.

[13] A. Diaz and D. Lovell. *XML Generator*. http://www.alphaworks.ibm.com/tech/xmlgenerator, Sept. 1999.

[14] S. Ganguly, M. Garofalakis, and R. Rastogi. "Processing Set Expressions over Continuous Update Streams". In *Proceedings of SIGMOD*, San Diego, California, June 2003.

[15] P. B. Gibbons. "Distinct Sampling for Highly-Accurate Answers to Distinct Values Queries and Event Reports". In *Proceedings of VLDB*, Roma, Italy, Sept. 2001.

[16] Y. Kanza, W. Nutt, and Y. Sagiv. Queries with Incomplete Answers over Semistructured Data. In *Proceedings of PODS*, May 1999.

[17] Y. Kanza and Y. Sagiv. Flexible Queries Over Semistructured Data. In *Proceedings of PODS*, May 2001.

[18] P. Klein. Computing the Edit-Distance between Unrooted Ordered Trees. In *Proceedings of the 6th European Symposium on Algorithms*, Aug. 1998.

[19] T. Schlieder. Schema-Driven Evaluation of Approximate Tree-Pattern Queries. In *Proceedings of EDBT*, Mar. 2002.

[20] D. Shasha and K. Zhang. Simple Fast Algorithms for the Editing Distance between Trees and Related Problems. *SIAM Journal on Computing*, 18(6):1245–1262, 1989.

[21] D. Shasha and K. Zhang. Approximate Tree Pattern Matching. In *Pattern Matching Algorithms*, pages 341–371. Oxford University Press, 1997.

[22] J. S. Vitter. "Random Sampling with a Reservoir". *ACM Trans. on Math. Software*, 11(1):37–57, Mar. 1985.

[23] W3C. XML Path Language (XPath) 1.0, Nov. 1999.

[24] W3C. Extensible Markup Language (XML) 1.1, Feb. 2004.

[25] P. Wood. Minimizing Simple XPath Expressions. In *Proceedings of WebDB*, May 2001.

[26] xCBL. XML Common Business Library.

[27] L. Yang, M. Lee, and W. Hsu. Efficient Mining of XML Query Patterns for Caching. In *Proceedings of VLDB*, Sept. 2003.