Evaluating Tree Pattern Similarity for Content-based Routing Systems

Raphaël Chand, Pascal Felber

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Evaluating Tree Pattern Similarity for Content-based Routing Systems

Raphaël Chand — Pascal Felber

N° 5891

Avril 2006

Thème COM

Rapport de recherche

N 0249-6399

Evaluating Tree Pattern Similarity for Content-based Routing Systems

Raphaël Chand , Pascal Felber[*]

Thème COM — Systèmes communicants
Projets Mascotte

Rapport de recherche n° 5891 — Avril 2006 — 27 pages

Abstract: With the advent of XML as the de facto language for data interchange, scalable distribution of data to large populations of consumers remains an important challenge. Content-based publish/subscribe systems offer a convenient abstraction for data producer and consumers, as most of the complexity related to addressing and routing is encapsulated within the network infrastructure. Data consumers typically specify their subscriptions using some XML pattern specification language (e.g., XPath), while producers publish content without prior knowledge of the recipients, if any. A novel approach to content-based routing consists in organizing consumers with similar interests in peer-to-peer *semantic communities* inside which XML documents are propagated. In order to build semantic communities and connect peers that share common interests with each other, one needs to evaluate the similarity between their subscriptions. In this paper, we specifically address this problem and we propose novel algorithms to compute the similarity of seemingly unrelated tree patterns by taking advantage of information derived from the XML document types, such as valid combinations of elements, or conjunctions and disjunctions on their occurrence. These results are of interest in their own right, and can prove useful in other domains, such as approximate XML queries involving tree patterns. Results from a prototype implementation validate the effectiveness of our approach.

Key-words: Data management, Content Routing, XML, Tree Pattern Matching, Semantic Overlays.

[*] University of Neuchâtel, Neuchâtel, Switzerland, *pascal.felber@unine.ch*

Evaluation de la similarité entre filtres arborescents pour les systèmes à routage basé sur le contenu

Résumé : Avec l'avènement de XML comme language standard d'échange de données, la distribution extensible de données à de larges populations de consommateurs reste un défi important. Les systèmes de type publication/abonnement dont le routage est basé sur le contenu offrent une abstraction commode pour les producteurs et consommateurs de données; en effet, la complexité afférante à l'adressage et au routage est en grande partie encapsulée à l'intérieur de l'infrastructure du réseau. Typiquement, les consommateurs de données spécifient leurs souscriptions en utilisant un language de filtrage XML (e.g., XPath), tandis que les producteurs publient du contenu sans connaissance à priori des destinataires, s'il y en a. Une approche novatrice consiste à organiser les consommateurs partageant des intérêts similaires dans des communautés sémantiques pair-à-pair, au sein desquelless les documents XML sont propagés. Afin de construire ces communautés sémantiques et connecter les pairs partageant des intérêts communs, il est nécessaire d'évaluer la similarité entre leurs souscriptions. Dans cet article, nous nous intéressons à ce problème en particulier , et nous proposons des algorithmes novateurs pour calculer la similarité entre filtres arborescents apparemment sans rapports, en tirant parti de l'information dérivée des types des documents XML, tels que les combinaisons valides d'éléments, ou les conjonctions et disjonctions de leurs occurences. Ces résultats relèvent d'un intérêt à la fois pragmatique et théorique. Bien que décrits dans le contexte publication/abonnement, ils peuvent être utilisés pour resoudre différents problèmes de réseaux ou de gestion de données. Les résultats expérimentaux découlant d'une implémentation prototype valident l'efficacité de notre approche.

Mots-clés : Gestion de données, Routage basé sur le contenu, XML, Filtrage arborescent, Réseaux recouvrants sémantiques.

1 Introduction

XML (eXtensible Markup Language) [1] 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 has become 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 [2]). 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 the consumers. 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., [3, 4, 5]) to match each incoming document against every known subscription. The use of summarization techniques (e.g., subscription aggregation [6, 7]) alleviates those issues, but at the cost of significant control message overhead or a loss of routing accuracy.

We have recently proposed an original approach to XML content routing [8] that does not rely on dedicated network of content routers, nor on complex filtering and forwarding algorithms. Instead, producers and consumers are organized in a peer-to-peer network that self-adapts upon peer arrival, departure, or failure. The underlying idea is to connect peers with similar interests so as to form semantic communities and use an extremely simple forwarding algorithm: a peer propagates to its neighbors every incoming message that matches its interest while other messages are essentially discarded. Therefore, messages are quickly spread inside the community of interested consumers and vanish as soon as they reach its boundaries.

Obviously, the price to pay for this simplicity is that routing may not be perfectly accurate, in the sense that some consumers may receive some messages that do not match their interests (false positives), or fail to receive relevant messages (false negatives). One can, however, limit the scope of this problem by organizing the peers according to adequate *proximity metrics*, i.e., by creating semantic communities that correctly map to the interests of the consumers.

In [8], we have proposed a proximity metric based on the notion of subscription containment: we say that a subscription p *contains* another subscription q, or $q \sqsubseteq p$, if and only if any message m that matches q also matches p (note that this relation is transitive and defines a partial order). By organizing consumers in a tree topology according to subscription containment, one can build a content routing network that produces *no* false negatives and very few false positives. This approach does, however, suffer from two important drawbacks: its poor applicability to subscriptions with little or no containment relationships; and its tree topologies that may be fragile with dynamic consumer populations. These limitations can be alleviated with a more general proximity metric that creates semantic communities with graph topologies instead of trees. The actual challenge lay 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.

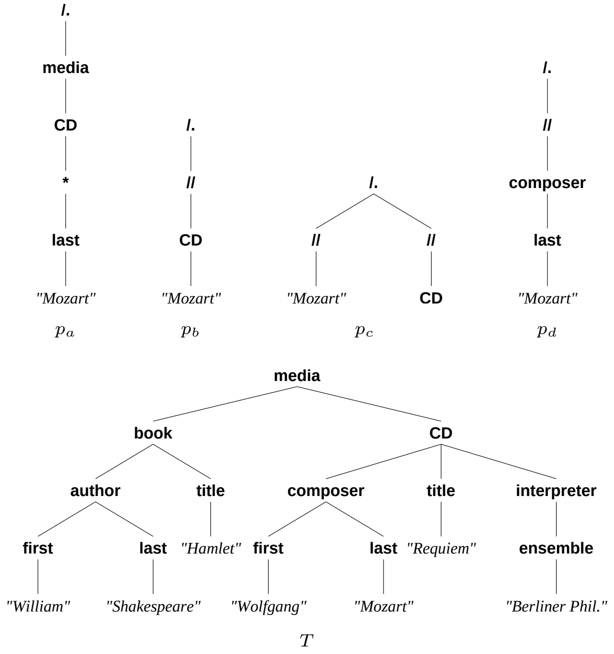


Figure 1: Example tree patterns (top) and XML document tree (bottom).

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 XPath expressions in terms of filtering error on XML documents, i.e., the error that would be induced when filtering XML documents against one expression instead of the other. Obviously, consumers with highly similar subscriptions are expected to be good neighbors in the overlay network.

Briefly, our *tree pattern similarity* problem can be stated as follows: given a tree pattern p and a set of tree patterns $\mathcal{S}$, find $q \in \mathcal{S}$ such that filtering XML documents against q instead of p induces the minimal error.

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 grand-child 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 conversely, 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 grand-child 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.* □

So far, work on tree pattern similarity has partially addressed the problem of proximity without exploiting the constraints expressed in XML type descriptors [7]. 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 the XML document types, such as valid combinations of elements, or conjunctions and disjunctions on their occurrence. We illustrate the benefits of our proximity metric for content routing and evaluate its accuracy. We would like to stress that the usefulness of our results on tree pattern similarity 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 notion of tree pattern expansion 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 the paper.

2 Problem Formulation

2.1 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 [7]. 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. Tree patterns can also define simple conditions and predicates on the values associated with the elements and attributes of XML documents. These constraints are typically specified in terms of equality ($=$, $\neq$), order relation ($>$, $<$, $\geq$, $\leq$), or string containment (prefix, suffix, substring). We denote by $Pred(v)$ the set of predicates associated with node v. Some examples of tree patterns are depicted in Figure 1 (for instance, tree pattern p_a specifies value predicate $last$ = "Mozart"). 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 *satisfies* $Subtree(v, p)$ 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 *satisfies* p, 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.

We say that a tree pattern q is *contained* in another tree pattern p, denoted by $q \sqsubseteq p$, if and only if for any XML tree T, if T matches q then T also matches p. We refer to p as the *container pattern* and q as the *contained pattern*. We say that p and q are *equivalent*, denoted by $p \equiv q$, if $p \sqsubseteq q$ and $q \sqsubseteq p$.

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., [9, 10]).

XML documents are optionally accompanied by a *document type definition* (DTD) [1] that defines the legal building blocks and the structure of valid XML documents. A DTD is typically represented as an extended context free grammar [11] and defined by a tuple $D = (E, P, r)$, where E is a set of *element types*, P is a set of *production rules* that define element types, and r is a distinguished element type called the *root type*. For each element type $e \in E$, $P(e)$ is a regular expression $\alpha ::= S \mid e' \mid \epsilon \mid \alpha|\alpha \mid \alpha, \alpha \mid \alpha*$, where S denotes the string type (PCDATA), $e' \in E$ is an element type, ϵ is the empty word, and "|", ",", and "*" denote disjunction, concatenation, and the Kleene closure, respectively. Production rules can also use the symbols "?" (zero or one) and "+" (one or more) as a syntactic facility for specifying the cardinality of regular expressions. A DTD containing some element type that is defined in terms of itself, directly or indirectly, is said to be *recursive*. We do not take into account attributes and ID/IDREF, as they are not relevant for computing tree pattern similarity, nor the ordering imposed by concatenation. Note that the type information used by our algorithms could also be derived from an XML schema [12].

```
<!ELEMENT a (b?,c?,d?)>
<!ELEMENT b (e+,f?,(g|h)?)>
<!ELEMENT e (k|m)+>
<!ELEMENT k (#PCDATA)>
<!ELEMENT m (#PCDATA)>
<!ELEMENT f (#PCDATA)>
<!ELEMENT g (#PCDATA)>
<!ELEMENT h ((k|m)+|n*)>
<!ELEMENT c (f|o)+>
<!ELEMENT o (n*)>
<!ELEMENT n (#PCDATA)>
<!ELEMENT d (e+,(p|q)?)>
<!ELEMENT p (#PCDATA)>
<!ELEMENT q (k|m)+>
```

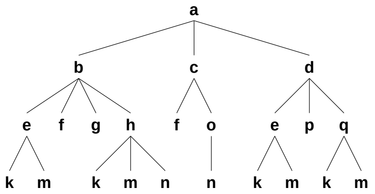


Figure 2: Sample DTD (left) and its graph representation (right).

Example 2.1 *Figure 2 shows a sample DTD, compatible with the XML document of Example 1.1 (we have replaced tag names by synthetic labels for conciseness), together with a* *graph representation of the hierarchical organization of element types. Note that the graph is actually a tree because the DTD is not recursive; edge directions have hence been omitted. The root element type is "a", which has three optional children: "b", "c", and "d". Element "b" has at least one child "e", an optional child "f", and an optional child whose type is either "g" or "h". Element "c" has a non-empty set of children of type "f" or "o". Element "d" has at least one child "e", as well as an optional child whose type is either "p" or "q". Elements "e" and "q" have a non-empty set of children of type "k" or "m". Element "h" has either a non-empty set of children of type "k" or "m", or an optional set of children of type "n". Element "o" has an optional set of children of type "n". The remaining elements are of string type.* □

2.2 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 $\mathcal{S}$ be the universe of all possible subscriptions. We define the similarity between p and q, denoted by $(p \sim q)$, as a function of $\mathcal{S}^2 \mapsto [0,1]$ that returns the probability that an XML document T matching p also matches q. We define the proximity between p and q, denoted by $(p \approx q)$, as $\frac{(p \sim q)+(q \sim p)}{2}$. Note that the proximity relation is symmetric while similarity is not. It directly follows from these definitions that $p \sqsubseteq q \Rightarrow (p \sim q) = 1$ and $p \equiv q \Rightarrow (p \approx q) = 1$.

Given a tree pattern p and a set of tree pattern subscriptions S, we want to find $q \in S$ such that:

$$\forall q' \in S, q' \neq q : (p \approx q) \geq (p \approx q')$$

Obviously, the similarity and proximity relations are valid for a finite set of XML documents or, more generally, for all documents of a given type D.

3 Tree Pattern Expansions

In order to compute the similarity of two tree patterns p and q, we first determine the class of XML documents that match p. This is achieved by building the so-called "expansion" of p with respect to document type D. We then use this expansion to evaluate the probability that the same type of documents match q.

Accurate evaluation of tree pattern similarity requires a good understanding of the XML documents on which these patterns apply. Although we only need to know the *structure* of valid documents, we can take advantage of additional knowledge about *correlations* between the elements, as well as their *frequency distribution.* This information can be extracted from DTDs and from historical data, maintained in the form of a compact synopsis and updated as XML documents stream by.

3.1 Extracting Correlations from DTDs

In addition to the structure of valid XML documents, DTDs allow us to derive two types of correlations between elements:

1. Constraints on the cardinality of single elements, expressed using regular expression operators: "?" for zero or one, "*" for zero or more, "+" for one or more, no operator for exactly one.
2. Constraints expressed as choices in groups of elements, expressed using the "|" disjunction operator: $(a|b)$ means that element a or b may appear, but not both.

We say that two elements a and b are in *opposition* when only one of them may appear in a given context, as in $(a|b)$. By extension, we say that an element is in opposition with itself if it may appear at most once, as in (a) or $(a?)$.[1]

We use the following rules and simplifying assumptions when parsing a DTD to analyze element correlations:

- $(a?)$ or (a) imply that element a is in opposition with itself.
- $(a\#)\&$ is always equivalent to $a*$, where # and & are two *different* regular expression operators. For example, $(a+)?$ is equivalent to $a*$. Thus, element a is *not* in opposition with itself.
- $(a|b)*$ is equivalent to $(a*, b*)$ except for the ordering. Elements a and b are *not* in opposition.
- $(a|b)?$ is equivalent to $(a?|b?)$. Elements a and b are in opposition.
- $(a|b)+$ is *not* equivalent to $(a+|b+)$ nor to $(a+, b+)$. Elements a and b are *not* in opposition.
- $(a|b)$ is a regular disjunction and elements a and b are in opposition.

The rules exposed above trivially extend to more than two elements and to compound elements. Note that the opposition relation is symmetric and transitive.

Example 3.1 *The DTD of Figure 2 allows us to derive the following constraints: Under element "a", every element ("b", "c", and "d") is in opposition with itself. Under element "b", "f" is in opposition with itself while "g" and "h" are in opposition with each other. Under element "d", "e" is in opposition with itself while "p" and "q" are in opposition with each other.*

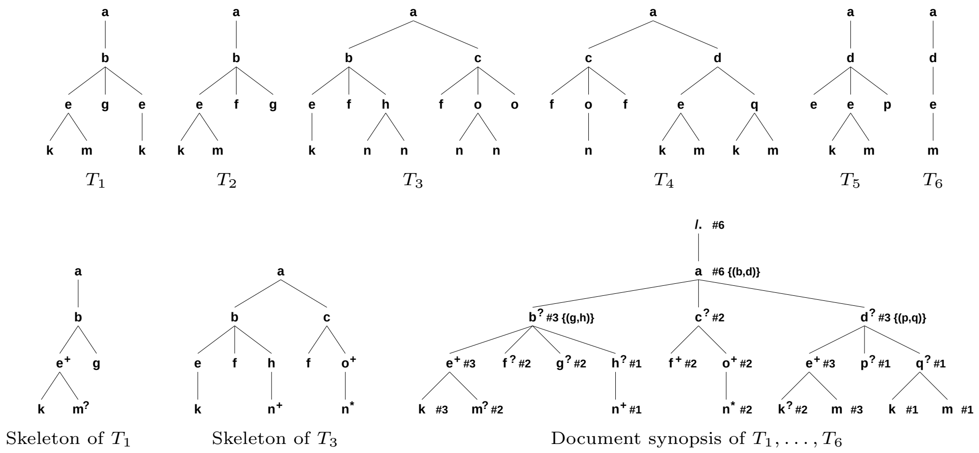
We can also observe that elements "b", "c", "d", "g", "h", "p", and "q" cannot appear more than once in an XML document. Only element "a" is mandatory. □

[1] To better understand this definition, note that (a) is equivalent to $(a|a)$ and $(a?)$ is equivalent to $(a|a|\epsilon)$.

3.2 Extracting Distribution from History

The frequency distribution of elements is useful to increase the accuracy of our similarity metric, as it allows to estimate the probability that a given element will appear in a future XML document. DTDs do not include frequency information and, without additional information, we assume that every valid element appears equally often in a given context (uniformity assumption).

Remember that we are interested in computing the similarity between tree patterns for routing XML documents. We can therefore presuppose that we have access to some historical data about documents that we have seen in the past. Assuming that historical documents are representative of the complete collection of XML documents, we can use this information to accurately estimate the distribution of future documents. It is, obviously, infeasible to maintain the complete history H of XML documents. Therefore, we maintain a concise *document synopsis* of H on-line, as documents are streaming by, and we use this synopsis as an approximation of the distribution.


a b e g e k m k
a b e f g k m
a b c e f h f o o k n n n n
a c d f o f e q n k m k m
a d e e p k m
a d e m
a b e+ g k m?
a b c e f h f o+ k n+ n*
/. #6
a #6 {(b,d)}
b? #3 {(g,h)} c? #2 d? #3 {(p,q)}
e+ #3 f? #2 g? #2 h? #1 f+ #2 o+ #2 e+ #3 p? #1 q? #1
k #3 m? #2 n+ #1 n* #2 k? #2 m #3 k #1 m #1

T_1 T_2 T_3 T_4 T_5 T_6

Skeleton of T_1 Skeleton of T_3 Document synopsis of $T_1, \ldots, T_6$

Figure 3: Example XML documents (top) and the corresponding skeleton trees and document synopsis (bottom).

Our synopsis is similar to the document tree synopsis proposed in [7], but includes additional information for deriving constraints (cardinality and correlations). The synopsis for H, denoted by H_S, captures path statistics for documents in H and is built incrementally. The document synopsis has essentially the same structure as an XML tree, except for two differences. First, the root node of H_S has the special label "/.". Second, each non-root node t in H_S has a additional information associated with it: a *frequency*, denoted by $freq(t)$; a *cardinality*, denoted by $card(t)$; and a list of *conflict groups*, denoted by $Conf(t)$.

Intuitively, if "$l_1/l_2/\cdots/l_n$" is the sequence of labels on nodes along the path from the root to t (excluding the label for the root), then $freq(t)$ represents the number of documents in H that contain a path "$l_1/l_2/\cdots/l_n$" originating at the root. The frequency for the root node of H_S is set to $|H|$, the number of documents in H. Note that the frequency of a node is always smaller or equal to that of its parent. The cardinality $card(t)$ represents the number of instances of t that have been observed under its parent element in the documents of H: we have always seen zero or one occurrence if $card(t) =$ "?"; zero or more occurrences if $card(t) =$ "*"; one or more occurrences if $card(t) =$ "+"; exactly one occurrence if $card(t) = \epsilon$ (no cardinality). Finally, conflict groups $Conf(t)$ specify sets of children, if any, that have never been observed together under the same parent element.

H_S is incrementally maintained as XML documents stream by. For each document T, we first construct a *skeleton tree* T_s in which each node has at most one child with a given label. T_s is built from T by traversing nodes in a top-down fashion and coalescing a set of children of a node in T if they have the same label. The nodes in T_s also have an optional cardinality defined as follows: when children of a non-coalesced node are coalesced, the resulting node has cardinality "+"; children of a coalesced node have cardinality "?" if they appear once under some of the original nodes and not under the others, "*" if they appear more than once under some of the original nodes and not under some of the others, "+" if they appear more than once under some of the original nodes and at least once under the others, and no cardinality if they appear once under each of the original nodes. Clearly, we can construct T_s in a single pass over document T.

Next, we use T_s to update our document synopsis H_S. 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 first increment $freq(t')$ by 1. We then update $card(t')$ as follows (x and y correspond to either combinations of t and t'):

$$card(t') = \begin{cases} card(t) & \text{if } t' \text{ has just been added to } H_S \\ & \text{or } card(t) = card(t') \\ + & \text{if } card(x) = + \text{ and } card(y) = \epsilon, \\ ? & \text{if } card(x) = \,?\, \text{ and } card(y) = \epsilon, \\ * & \text{otherwise} \end{cases}$$

Finally, we update conflict groups $Conf(t')$ using the following rules: for any child c of t that does not appear under t', and for any child c' of t' that does not appear under t, we add (c, c') to $Conf(t')$; for any two children c and c' of t, we remove (c, c') from $Conf(t')$ if it exists. Note that, in practice, most of the conflict groups are empty or small, and their size tends to decrease as H grows.

We shall mention that it is possible to further compress H_S as proposed in [13, 7] by merging nodes with the lowest frequencies and storing, with each merged node, the average of the original frequencies.

Example 3.2 *Figure 3 shows several XML documents $T_1, \ldots, T_6$ compliant with the DTD of Figure 2, the skeleton trees of T_1 and T_3, and the resulting document synopsis. We can* *observe how the cardinality constraints (represented by an optional cardinality symbol next to the labels) appear when coalescing nodes in the skeleton trees: for instance, element "e" appears multiple times under "b" and element m is optional under "e" in T_1; element "n" appears zero or more times under "o" in T_3. Obviously, cardinality information is just an approximation based on observation and may differ from the actual constraints expressed in the DTD.*

The document synopsis contains information about cardinality, frequencies, and conflicts (respectively represented by an optional cardinality symbol, a number prefixed by "#", and an optional list of sets of conflicting children). The frequency of a path from the root to node t is given by $freq(t)$, and the probability that such a path is encountered in an XML document is given by $\frac{freq(t)}{|H|}$. For instance, the probability of an occurrence of path "/a/b/h" is $\frac{1}{6}$ (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.*

The cardinality and conflict information associated with the nodes of H_S are an approximation of the constraints expressed in the DTD and are expected to become more accurate when the size of H grows. For instance, the synopsis fails to identify the cardinality of the children of element "q" (only T_4 contains "q") and incorrectly reports a conflict between element "b" and "d" (they never appear together in the original documents). □

3.3 Constructing Tree Pattern Expansions

Using the information derived from DTDs and/or document histories, we can now construct the so-called *expansion* of a tree pattern. Informally, a tree pattern expansion is a data structure that represents the constraints on the documents that match the tree pattern. This structure allows us to model the class of documents that a tree pattern specifies and to accurately determine the similarity with other tree patterns (recall that, by definition, two patterns are similar if the same documents match both).

The expansion E_p of a tree pattern p is a tree where each node n has the following attributes:

- a label $label(n)$, which is a *valid* element name, i.e., not "*" or "//";
- a probability $0 < prob(n) \leq 1$;
- a class $class(n) \in \mathbf{N}$;
- a set of predicates $Pred(n)$ on the value associated with n, if any;
- a list $Conf(n)$ of *conflict groups* composed of nodes among the children of n.

E_p has an artificial root node $root(E_p)$ with label "/.". Roughly speaking, E_p is a compact representation of (some of) the possible documents that match tree pattern p. The label of a node n in E_p refers to the name of an element in a document that matches p.

Its probability refers to the likeliness that an element with name $label(n)$ that verifies the predicates of n occurs, given that all ancestor nodes have occurred. Its class identifies the node in p that it refers to: two nodes in E_p with the same parent and the same class do not occur at the same time (at least, this is not explicitly specified by p). The list of conflict groups determine the children of n that are in *opposition* according to the DTD or H_S. Each conflict group contains a set of children that are in opposition with each other. Finally, the predicates describe constraints on the element's values or attributes.

The computation of the node probabilities is based on H_S, if available; otherwise, we assume a uniform distribution of element types among those allowed by the DTD.

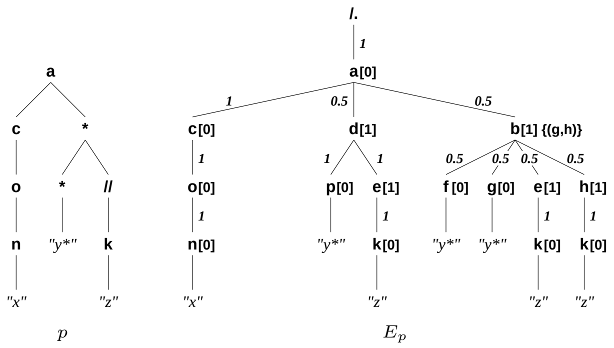


Figure 4: Sample tree pattern subscription (left) and its expansion with respect to DTD of Figure 2 (right).

Example 3.3 *Figure 4 shows a tree pattern subscription p, compliant with the DTD of Figure 2, and its expansion E_p. The expansion was built without the help of a synopsis H_S and probabilities are thus uniform.*

Elements "b" and "d" are the two possible occurrences under "a" of the first "" in p (element "c" is not a valid occurrence because it does not have a descendant labeled "k" in the DTD). Under our uniformity assumption, both nodes have an equal probability of occurring (*0.5*). They refer to the same node in p and thus have the same class. In contrast, element "c" in E_p has a different class as it refers to a different node in p.*

Elements "f" and "g" are the two possible occurrences under "b" of the second "" in p, with probability* 0.5*, while elements "e" and "h" are the two possible occurrences of node "//" in p, also with probability* 0.5*. Hence, path "/a/b/e" has probability* 0.25 *(*$= 1 \times 0.5 \times 0.5$*) of occurring in a document that matches p. Conflict groups at node "b" in E_p specify that children "g" and "h" cannot appear simultaneously in the same document. Finally, element "p" is the only possible occurrence (probability* 1*) under "d" of the second "*" in p. As elements "p" and "q" are in opposition under "d", element "e" becomes the only possible occurrence under "d" of node "//" in p.* □

We can now describe formally the *expand* function, shown in Algorithm 1, which builds the expansion E_p of a subscription p. The expand function works recursively on the nodes of p. It takes as input a node u of p and a node n of E_p, such that the path with element names $root(E_p) \rightarrow n$ is a possible occurrence of $root(p) \rightarrow parent(u)$. A call to $expand(u, n)$ determines the possible trees that start with $root(E_p) \rightarrow n$ and are possible occurrences of the pattern $\{root(p) \rightarrow u\} \cup Subtree(u, p)$.

Algorithm 1 Recursive expand function: $expand(u, n)$

```
if $depth(n) \geq \Delta$ or $prob(root(E_p) \rightarrow n) \leq \tau$ then
  return true                                          {Avoid infinite recursion}
end if
$matches = 0$                                          {Number of branches matching u}
$class\text{-}id = \#classes(n)$                       {New class number (> any child of n)}
if $u \neq //$ and $u \neq *$ then
  if $u$ is a valid child of $n$ then
    create child node $n_c$ of $n$
    $label(n_c) \leftarrow label(u)$
    $Pred(n_c) \leftarrow Pred(u)$
    $prob(n_c) \leftarrow 1$                           {Node must match}
    $class(n_c) \leftarrow class\text{-}id$
    if $u$ is leaf or $\bigwedge_{u_i \in Children(u)} expand(u_i, n_c) = $ true then
      $matches \leftarrow 1$                           {Branch matches tree pattern}
    else
      delete $Subtree(n_c, E_p)$                       {No match: delete new subtree}
    end if
  end if
else if $u = *$ then
  for all valid child $n_i$ of $n$ do
    create child node $n_c$ of $n$
    $label(n_c) \leftarrow label(n_i)$
    $Pred(n_c) \leftarrow Pred(u)$
    $class(n_c) \leftarrow class\text{-}id$
    adjust $prob(n_c)$                                 {Node matches with some probability}
    if $u$ is leaf or $\bigwedge_{u_i \in Children(u)} expand(u_i, n_c) = $ true then
      $matches \leftarrow matches + 1$                 {Branch matches tree pattern}
    else
      delete $Subtree(n_c, E_p)$                       {No match: delete new subtree}
    end if
  end for
else if $u = //$ then
  if $expand(child(u), n) = $ true then                {Node u has exactly one child}
    $matches \leftarrow 1$                             {// matches empty path}
  end if
  for all valid child $n_i$ of $n$ do
    create child node $n_c$ of $n$
    $label(n_c) \leftarrow label(n_i)$
    $class(n_c) \leftarrow class\text{-}id$
    adjust $prob(n_c)$                                 {Node matches with some probability}
    if $expand(u, n_c) = $ true then
      $matches \leftarrow matches + 1$                 {// matches path of length $\geq 1$}
    else
      delete $Subtree(n_c, E_p)$                       {No match: delete new subtree}
    end if
  end for
end if
return $matches > 0$                                   {true $\Leftrightarrow$ branch matches tree pattern}
```

The function returns *true* if it has managed to find a valid expansion of u rooted at n, and *false* otherwise. One can note that, if a tree pattern p contains one or more ancestor operator ("//") and the DTD is recursive, then E_p can be infinite; in that case, we truncate the expansion to a predefined depth Δ or to a minimum leaf probability[2] τ (lines 1–3).

Now, we have to find all children n_c of n such that $root(E_p) \rightarrow n_c$ is a possible occurrence of $root(p) \rightarrow u$. It follows from the definition of E_p that, if $label(u) \neq$ "*" and $label(u) \neq$ "//", then a single node n_c is built in E_p and appended as child of n (lines 6–18). It has label $label(u)$, the same predicates as u, and probability 1: indeed, node n_c is the only possible occurrence of node u. Its class is such that no other child of n has the same class.[3] We then proceed recursively with each child u_i of u to find the possible occurrences of $root(p) \rightarrow u_i$. If some u_i could not be expanded, then there exists no valid document that starts with $root(E_p) \rightarrow n_c$ and match the pattern $\{root(p) \rightarrow u\} \cup Subtree(u, p)$; therefore, we delete the newly inserted node.

If $label(u) =$ "*", then a set of nodes $N = \{n_i \in children(n)\}$ of the same class is appended as n's children (lines 19–31). Each node n_i corresponds to a possible occurrence of node u that can be recursively expanded (as before, n_i is deleted if expansion does not succeed). The set of valid children is determined using the DTD, if available, or the synopsis H_S otherwise. The probability associated to n_i is computed as follows: if we have a synopsis, then the probabilities are adjusted to match the distribution of H_S; otherwise, each n_i has equal probability $\frac{1}{|N|}$. Probabilities are always adjusted so that the sum of the probabilities of siblings of the same class is equal to 1.

Finally, if $label(u) =$ "//", then we first try to map u with an empty path by recursively calling *expand* on the child of u (by definition, element "//" must have exactly one child) and n. Then, we try to map u with a non-empty path by recursively calling *expand* on u and the children of n; this allows us to map u to arbitrarily long paths (lines 32–47). Probabilities and classes are set as in the previous case.

Conflict groups have been omitted from the algorithm for clarity. They are added to a node n after all the children of n have been expanded (post-order traversal). Information about children of n that are in opposition is copied from the DTD, if available, or from synopsis H_S otherwise, to $Conf(n)$. Note that the construction of conflict groups may result in some subtrees being deleted: if node $n_i \in Children(n)$ with $prob(n_i) = 1$ and belongs to a conflict group, then the subtrees rooted at the other nodes n_j of the same conflict group must be deleted (n_i *must* occur and, hence, n_j *cannot* occur). If two nodes n_i and n_j with probability 1 appear in the same conflict group, then the whole subtree rooted at the lowest common ancestor of n_i and n_j that has a probability smaller than 1 is invalid and must be deleted.

[2] The function $prob(r \rightarrow n)$ returns the probability of path $r \rightarrow n$ to occur in an XML document, computed as the product of the probabilities of all the nodes on the path from r to n.

[3] The function $\#classes(n)$ returns the number of distinct classes among the children of n. As classes are numbered from 0, we can use $\#classes(n)$ as the next class identifier to associate with new children of n.

4 Computing Tree Pattern Similarity

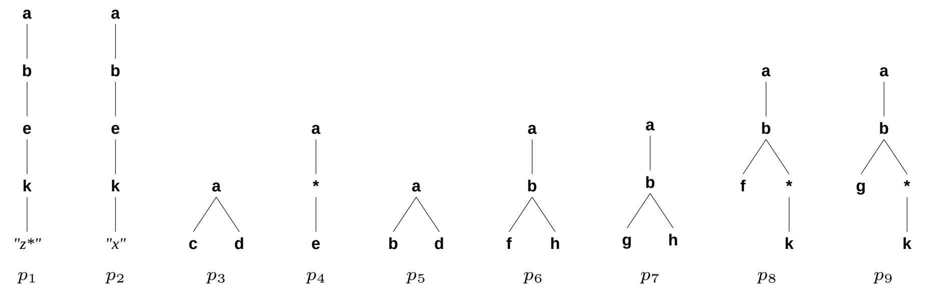


Figure 5: Example tree pattern subscriptions.

4.1 General Principle

The similarity function $p \sim q$ works recursively on the nodes of q and the nodes of p's expansion, E_p. Recall that $p \sim q$ returns the probability that a document matching p also matches q, and is *not* symmetric.

Consider a document T that matches p: we want to evaluate the probability that it also matches q. The idea is to use p's expansion instead of p. Indeed, E_p represents all the possible structures and content predicates that a document matching p is expected to have, together with their associated probabilities. Hence, the principle is to find paths in E_p that are equivalent to, or contained in, paths of q (we say that the paths in E_p *match* those in q).

If the whole tree pattern q has matching paths in E_p, then $p \sqsubseteq q$ and $p \sim q = 1$. When there are several possible paths in E_p to chose from, we choose the one with the highest probability. The conjunction of different paths in q is computed as the product of the probabilities of their respective occurrences in E_p.

We first look for the longest paths in q that have a matching path in E_p. If E_p matches only the initial portion of a path in q, the remaining part may still occur in an XML document and we estimate the probability of the missing portion using heuristics.

We then compute the probability of the conjunction of individual paths. To distinct paths of q must correspond distinct paths in E_p. By construction of the subscription expansion, this implies that the nodes of two matching paths in E_p that have the same parent must belong to different classes. If that is not the case, we say that there is a *collision* (a single path of E_p matches two distinct paths of q). Intuitively, collisions decrease the similarity between p and q, although they do not prevent a document from matching both p and q. Therefore, we associate a penalty $\gamma \ll 1$ with each collision when computing similarity.

Example 4.1 *Consider tree patterns $p_1, \ldots, p_9$ in Figure 5 and p in Figure 4. The similarity between every pair of tree patterns $p \sim p_i$ is computed as follows:*

$p \sim p_1 = 0.25$ *(*$= 1 \times 0.5 \times 0.5 \times 1 \times 1$*). Indeed, "/a/b/e/k" is the only path in E_p that matches p_1 (note that string predicate "z" matches "z*"). Hence, the similarity between p_1 and p is computed as the probability of path "/a/b/e/k" in E_p.*

$p \sim p_2 = 0$ *(*$= 1 \times 0.5 \times 0.5 \times 1 \times 0$*). Although path "/a/b/e/k" in E_p matches p_2, the string predicate associated with "k" does not match.*

$p \sim p_3 = 0.5$ *(*$= 1 \times (1 \times 0.5)$*). Paths "/a/c" and "/a/d" in p_3 have matching paths in E_p. As elements "c" and "d" belong to different classes, they can occur independently in an XML document. The similarity is computed as the probability that both paths occur, i.e., the product of their respective probabilities.*

$p \sim p_4 = 0.75$ *(*$= 1 \times (0.5 \times 1 + 0.5 \times 0.5)$*). Path "/a/*/e" in p_4 has two matching paths in E_p: "/a/d/e" and "/a/b/e". The similarity is computed as the sum of the probabilities of each path.*

$p \sim p_5 = 0.25\gamma$ *(*$= 1 \times (0.5 \times 0.5 \times \gamma)$*), with $0 < \gamma \ll 1$. Paths "/a/b" and "/a/d" in p_5 have matching paths in E_p, but they share the same "*" element in p ("b" and "d" belong to the same class in E_p). Hence, we have a collision and the similarity is computed as the product of the probabilities of each path combined with a penalty γ.*

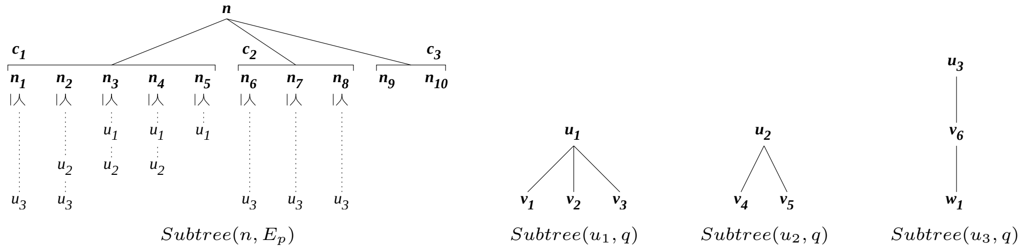
$p \sim p_6 = 0.125$ *(*$= 1 \times 0.5 \times (0.5 \times 0.5)$*). Path "/a/b" occurs with probability* 0.5 *in E_p. Elements "f" and "h" belong to different classes and can occur independently. Hence, the similarity is computed as the product of individual probabilities.*

$p \sim p_7 = 0$ *(*$= 1 \times 0.5 \times 0$*). Path "/a/b" occurs with probability* 0.5 *in E_p, but elements "g" and "h" cannot occur simultaneously, as indicated by the conflict groups of node "b" in E_p. Therefore, similarity is null (note that p_7 is* not *a valid expression with respect to the DTD).*

$p \sim p_8 = 0.25$ *(*$= 1 \times 0.5 \times (0.5 \times (0.5 \times 1 + 0.5 \times 1))$*). Path "/a/b" occurs with probability* 0.5 *in E_p, and element "f" with probability* 0.5*. Path "*/k" in p has two matching paths in E_p ("e/k" and "h/k", with "e" and "h" belonging to the same class); thus, we add their probabilities.*

$p \sim p_9 = 0.125$ *(*$= 1 \times 0.5 \times (0.5 \times 0.5)$*). This case is similar to $p \sim p_8$, except that nodes "g" and "h" are in opposition. Consequently, only element "e" in E_p can match the "*" in p_9. This example illustrate the importance of conflict groups for improving accuracy: we would have otherwise computed the same similarity as for $p \sim p_8$.*

□



$Subtree(n, E_p)$ $Subtree(u_1, q)$ $Subtree(u_2, q)$ $Subtree(u_3, q)$

Figure 6: Illustration of the computation of $q \sim p$ on subtrees of the tree pattern expansion E_p (left) and the tree pattern q (right). We have indicated between each node n_j the set of matching nodes u_i (for instance, n_3 matches u_1 and u_2).

4.2 Similarity Function

We can now describe formally the similarity function. We compute $p \sim q$ using the recursive function $sim(n, U)$, where n is a node of E_p and $U = \{u_1, \ldots, u_n\}$ is a set of node in q. Although the number of situations that we have to deal with is rather large (combinations of classes, conflict, collisions, etc.), in practice very few apply at a time when comparing two tree patterns and the similarity function can be implemented very efficiently.

The principle of the computation is simple, but it requires to introduce some additional notation. Let $N = \{n_1, \ldots, n_m\}$ be the children of n and $C = \{c_1, \ldots, c_l\}$ be the set of classes among N. We define $Nodes_{c_j}^{\preceq u_i}$ as the set of nodes in class c_j that match u_i; $Classes^{\preceq u_i}$ as the set of classes containing at least one node matching u_i; a *match combination* P_k as an n-tuple of non-empty $Nodes_{c_j}^{\preceq u_i}$, where c_j is the matching class for node u_i corresponding to combination P_k (the set of all such c_j is denoted by $Classes^{P_k}$); and $\mathcal{P}$ is the set of all valid match combinations.

$$\begin{aligned}
Nodes_{c_j}^{\preceq u_i} &= \{n_k \mid class(n_k) = c_j \wedge n_k \preceq u_i\} \\
Classes^{\preceq u_i} &= \{c_j \mid Nodes_{c_j}^{\preceq u_i} \neq \emptyset\} \\
P_k &= (Nodes_{c_j}^{\preceq u_i})_{i=1,\ldots,n} \\
\mathcal{P} &= \{P_k\}
\end{aligned}$$

Example 4.2 *Consider the example of Figure 6. We have, for instance:*

$$\begin{aligned}
Nodes_{c_1}^{\preceq u_1} &= \{n_3, n_4, n_5\}\\
Nodes_{c_2}^{\preceq u_3} &= \{n_6, n_7, n_8\}\\
Classes^{\preceq u_1} &= \{c_1\}\\
Classes^{\preceq u_3} &= \{c_1, c_2\}\\
P_1 &= (\underbrace{\{n_3, n_4, n_5\}}_{\preceq u_1}, \underbrace{\{n_2, n_3, n_4\}}_{\preceq u_2}, \underbrace{\{n_1, n_2\}}_{\preceq u_3})\\
P_2 &= (\{n_3, n_4, n_5\}, \{n_2, n_3, n_4\}, \{n_6, n_7, n_8\})\\
\mathcal{P} &= \{P_1, P_2\}
\end{aligned}$$

□

Intuitively, each match combination corresponds to a mapping where each node in q has some matching nodes in a given class in p. Note that each u_i *must* have at least one class with some matching nodes; if that is not the case, we estimate the probability of the occurrence of $Subtree(u_i, q)$ using heuristics. Also, each match combination is *redundant* with the others, in the sense that the occurrence of one combination does not change the likelihood of the others.

We now formalize the collisions that occur between the nodes of a class. Let $\mathcal{A}_{c_j}^{P} = \{A_k\}$ be a partition of U, of minimal cardinality, where A_k is a set of nodes $\{u_i\}$ with a matching node in class c_j and such that the intersections $Nodes_{c_j}^{\cap A_k} = \cap_{u_i \in A_k} Nodes_{c_j}^{\preceq u_i}$ of the matching nodes for each set $A_k \in \mathcal{A}_{c_j}^{P}$ are *non-empty, disjoint* sets. The reasoning behind the partitioning is that collisions happen when the matching nodes for some u_i do not intersect with the matching nodes for another u_j. Hence, $|\mathcal{A}|$ has to be as small as possible in order to minimize the number of collisions. When several nodes n_i are in opposition with each other, we compute the similarity using each of the nodes in turn and keep the maximum value.

Example 4.3 *Consider again Figure 6. One partition for P_1 of Example 4.2 is $\mathcal{A}_{c_1}^{P_1} = \{\{u_1, u_2\}, \{u_3\}\}$. Indeed, $Nodes_{c_1}^{\preceq u_1} \cap Nodes_{c_1}^{\preceq u_2} = \{n_3, n_4, n_5\} \cap \{n_2, n_3, n_4\} = \{n_3, n_4\}$ and $Nodes_{c_1}^{\preceq u_3} = \{n_1, n_2\}$ are non-empty disjoint sets. Another possible partition is $\mathcal{A}'^{P_1}_{c_1} = \{\{u_1\}, \{u_2, u_3\}\}$. For combination P_2, we have only one valid partition $\mathcal{A}_{c_1}^{P_2} = \{\{u_1, u_2\}\}$.*
□

The similarity function $sim(n, U)$ can then be defined as:

$$sim(n,U) = \underbrace{\max_{P\in\mathcal{P}}}_{\substack{\text{redundant}\\ \text{combinations}}} \left(\underbrace{\prod_{c_j\in Classes^P}}_{\substack{\text{conjunctions of}\\ \text{patterns in } q}} \underbrace{\gamma^{|A|-1}\prod_{A_k\in\mathcal{A}}}_{\text{conflicts}} \underbrace{\sum_{n_i\in Nodes_{c_j}^{\cap A_k}}}_{\substack{\text{disjunctions of}\\ \text{patterns in } p}} prob(n_i)\cdot f(n_i,A_k)\right)$$

The function $f(n,A)$ returns the probability that a document matching $Subtree(n,p)$ also matches $n \rightarrow \bigcup_{u_i\in A}\bigcup_{v_j\in Children(u_i)} Subtree(v_j,p)$. It is defined recursively as follows:

$$f(n,A) = \begin{cases} 1\ , \text{ if } \forall u_i \in A, Children(u_i) = \emptyset \\ sim(n, \bigcup_{u_i\in A}\bigcup_{v_j\in Children(u_i)} v_j)\ , \text{ otherwise} \end{cases}$$

When A contains some node(s) u_i with label "//", we try to map it to an empty path and to a node with any label, and we keep the maximum value.

The similarity between tree patterns p and q is computed as $sim(root(E_p), \{root(q)\})$.

Example 4.4 *We can now compute $sim(n,u_1,u_2,u_3)$ for the example of Figure 6. We have two valid match combinations P_1 and P_2. Considering partitions $\mathcal{A}_{c_1}^{P_1}$ and $\mathcal{A}_{c_1}^{P_2}$, we have:*

$$\begin{aligned}
&sim(n,u_1,u_2,u_3) = \max(sim_{P_1}, sim_{P_2})\\
&sim_{P_1} = \gamma\ \cdot\\
&\quad \big(prob(n_3)\cdot f(n_3,\{u_1,u_2\}) + prob(n_4)\cdot f(n_4,\{u_1,u_2\})\big)\ \cdot\\
&\quad \big(prob(n_1)\cdot f(n_3,\{u_3\}) + prob(n_2)\cdot f(n_2,\{u_3\})\big)\\
&sim_{P_2} =\\
&\quad \big(prob(n_3)\cdot f(n_3,\{u_1,u_2\}) + prob(n_4)\cdot f(n_4,\{u_1,u_2\})\big)\ \cdot\\
&\quad \big(prob(n_6)\cdot f(n_6,\{u_3\}) + prob(n_7)\cdot f(n_7,\{u_3\})\ +\ prob(n_8)\cdot f(n_8,\{u_3\})\big)
\end{aligned}$$

□

The worst case complexity of the similarity function is rather high and directly depends on the number of classes in E_p, the branching factor in q, and the number of "//" and "*" nodes in p and q. In most cases, however, the computation is simply a product of sums and can be performed very efficiently (typically less than 1 ms, as will be discussed shortly).

5 Evaluation

We now present the simulations that we conducted to test the behavior and the efficiency of our similarity metric for XML documents and XPath expressions.

5.1 Parameters of the Experiments

The tree patterns considered in this paper can be 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) and xCBL (Common Business Library) Order, which contain 123 and 569 elements, respectively. For each DTD, we have generated sets of random XPath expressions of various sizes $n = 1,000 \rightarrow 10,000$, which we call *search set* and denote by $S = \{p_1, \cdots, p_n\}$.

We have generated our data set of XML documents with IBM's XML Generator [14] tool, using a uniform distribution for selecting element tag names. The data set, which we denote by D, consists of $1,000$ random documents with 108 tag pairs on average and up to 10 levels. We have a generated a *distinct* set of $1,000$ documents with the same characteristics to construct the synopsis H_S.

We have experimented with four different variants of our proximity metric:

- **DTD:** we use the DTD, but do not exploit conflicts between elements; we do not use a synopsis of past documents.
- **Synopsis:** we do not use the DTD and do not exploit conflicts; we use a synopsis of past documents.
- **DTD+CG:** we use the DTD and exploit conflicts between elements; we do not use a synopsis of past documents.
- **DTD+CG+Synopsis:** we use the DTD and exploit conflicts between elements; we use a synopsis of past documents.

5.2 Experimental Setup

Given a tree pattern p and a set of tree pattern subscriptions S, we denote by p's first *substitute* the tree pattern $q \in S$ such that:

$$\forall q' \in S, q' \neq q : (p \approx q) \geq (p \approx q')$$

By extension, p's k^{th} substitute is the tree pattern ranked at position k from p in terms of proximity.

For each XML document T, we determine the tree patterns in S that it matches. For each tree pattern $p \in S$, we define $Matches(p) = \{T \in D \mid T \text{ matches } p\}$. We can now define the false positive and false negative ratios resulting from the substitution of p by q as:

$$FP(p \to q) = \frac{|Matches(q) \setminus Matches(p)|}{|\overline{Matches(p)}|}$$

$$FN(p \to q) = \frac{|Matches(p) \setminus Matches(q)|}{|Matches(p)|}$$

where $\overline{Matches(p)} = D \setminus Matches(p)$ is the set of documents in D that do not match p. In other words, $FP(p \to q)$ is the portion of documents that match q and not p, out of the set of documents that do not match p. $FN(p \to q)$ is the portion of documents that match p and not q out of the set of documents that match p. Intuitively, suppose that we substitute p by q to filter the documents in D. Then, $FP(p \to q)$ is the percentage of irrelevant documents that would be wrongly identified as matches, while $FN(p \to q)$ is the percentage of relevant documents that would fail to be identified as matches. These concepts are graphically illustrated in Figure 7. Note that, if $q \sqsubseteq p$, then $FP(p \to q) = 0$ and $FN(q \to p) = 0$.

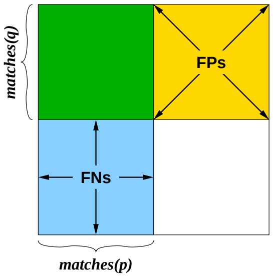


Figure 7: Quantifying false positives and false negatives.

Finally, we define the filtering error, i.e., the precision loss resulting from filtering documents against q instead of q, as:

$$FE(p \to q) = \frac{FP(p \to q) + FN(p \to q)}{2}$$

5.3 Results

For each DTD, each size of search set, and each variant of the proximity metric, we have computed the average filtering error of the k^{th} substitute as previously explained. Figure 8 shows the average filtering error of the first substitute as a function of the size of the search

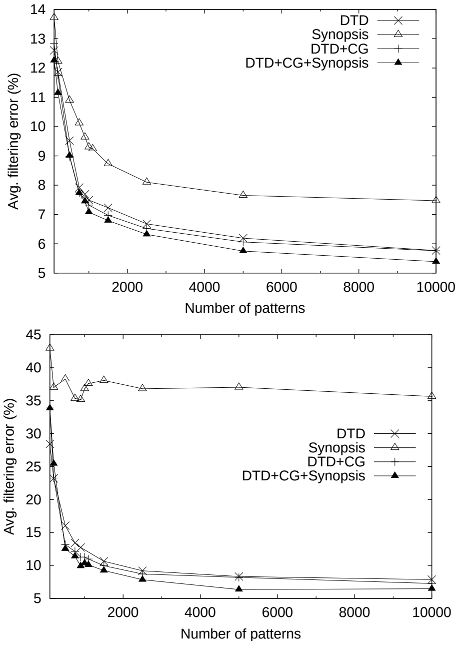


Figure 8: Average filtering error of first substitute for the NITF (top) and xCBL Order (bottom) DTDs.

set. One can notice that, on average, the best substitute found by the proximity metric for a tree pattern is very accurate. The average filtering error decreases with the size of the search set because we have more candidates to choose from. In particular, the xCBL Order DTD produces unsatisfactory results for small search sets because it includes a large number of elements; hence, many tree patterns are required to find good first substitutes.

Unsurprisingly, the accuracy of our proximity metric improves significantly when we use more information for computing subscription expansions and evaluating similarity. The best accuracy is achieved by combining the DTD, conflict groups, and the synopsis. When the DTD is not known, the average filtering error remains high. Indeed, the synopsis does not contain all the possible structures of XML documents (we would need a far larger set of XML documents to build an accurate synopsis). This phenomenon is more acute with the xCBL Order DTD due to its higher diversity.

Figure 9 shows the average filtering error of the k^{th} substitute as a function of its rank k for a search set of $10,000$ tree patterns when using the DTD, correlation groups, and the synopsis. We observe that the average filtering error of the k^{th} substitute increases *consistently* with k, which means that our proximity metric is indeed able to find the best

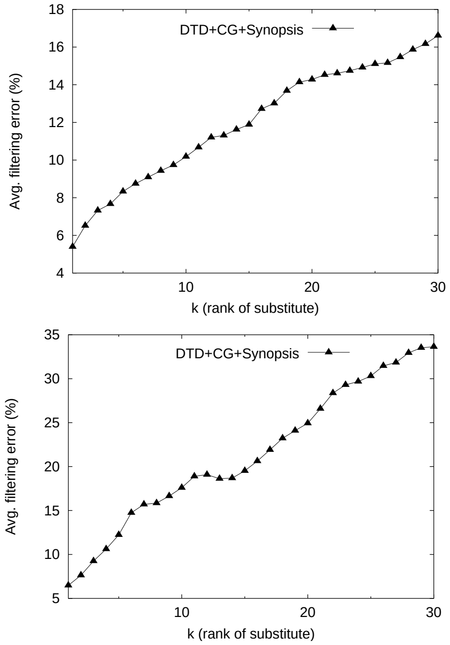


Figure 9: Average filtering error of k^{th} substitute for the NITF (top) and xCBL Order (bottom) DTDs.

substitutes, in the correct order. The only singularity appears for the xCBL Order DTD with the 13^{th} substitute, which has a slightly lower filtering error than the 12^{th} substitute. Again, this is probably due to the large number of elements in the DTD that decrease accuracy, but the difference between the filtering errors of both substitutes is so small that this imprecision remains insignificant.

An evaluation of the routing accuracy of our peer-to-peer publish/subscribe system, configured to use the (symmetric) proximity metric described in this paper for creating semantic communities, is given in [15]. Results demonstrate that the proximity metric based on tree pattern similarity enables us to build a well balanced and robust network topology that succeeds in delivering documents to almost all interested consumers with only a relatively small fraction of false positives.

In term of processing efficiency, we measured an average time of approximately 700 ms for finding the best substitute among $1,000$ tree patterns using the most complex variant of our proximity metric (DTD+CG+Synopsis), i.e., 0.7 ms per tree pattern. We observed a linear slowdown for larger populations. The algorithms were implemented in C++ and compiled using GNU C++ version 2.96. Experiments were conducted on a 1.5 GHz Intel Pentium IV machine with 512 MB of main memory running Linux 2.4.18.

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 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., [16, 17, 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 document 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., [19, 20, 21, 22]). Some forms of tree patterns have also been studied as queries for XML data [9, 23]. In particular, 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.

Our work builds upon some of the results of earlier research on tree pattern aggregation [7], where the objective is to combine several patterns into one smaller, but less precise, aggregate pattern. A document synopsis is used to compute the selectivity of tree patterns and choose the aggregate pattern that results in the minimal loss in selectivity. This work does not specifically address the problem of tree pattern similarity and does not take into account document types, as we did extensively here.

7 Conclusion

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 types and histories, such as cardinalities, conflicts, and frequency distributions. The principle of similarity computation is based on the notion of tree pattern expansion, a data structure that faithfully represents the class of XML documents that match a given tree pattern. Using the expansion, we can precisely determine whether the same XML documents match another tree pattern, and hence quantify the similarity between both patterns. Results from 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] W3C, “Extensible Markup Language (XML) 1.1,” http://www.w3.org/TR/xml11, Feb. 2004.

[2] ——, “XML Path Language (XPath) 1.0,” http://www.w3.org/TR/xpath, Nov. 1999.

[3] M. Altinel and M. Franklin, “Efficient Filtering of XML Documents for Selective Dissemination of Information,” in *Proceedings of VLDB*, Sept. 2000.

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

[5] 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.

[6] A. Carzaniga, D. Rosenblum, and A. Wolf, “Design and Evaluation of a Wide-Area Event Notification Service,” *ACM Transactions on Computer Systems*, vol. 19, no. 3, pp. 332–383, 2001.

[7] 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.

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

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

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

[11] P. Kilpelainen and D. Wood, “SGML and Exceptions,” in *Proceedings of PODP*, Sept. 1996.

[12] W3C, “XML Schema,” http://www.w3.org/TR/xmlschema-0/, Oct. 2004.

[13] A. Aboulnaga, A. Alameldeen, and J. Naughton, “Estimating the Selectivity of XML Path Expressions for Internet Scale Applications,” in *Proceedings of VLDB*, Sept. 2001.

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

[15] R. Chand, "Large scale diffusion of information in Publish/Subscribe systems," Ph.D. dissertation, Université de Nice, Sophia Antipolis, France, Sept. 2005.

[16] D. Shasha and K. Zhang, "Simple Fast Algorithms for the Editing Distance between Trees and Related Problems," *SIAM Journal on Computing*, vol. 18, no. 6, pp. 1245–1262, 1989.

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

[18] S. Chawathe and H. Garcia-Molina, "Meaningful Change Detection in Structured Data," in *Proceedings of SIGMOD*, May 1997.

[19] D. Shasha and K. Zhang, "Approximate Tree Pattern Matching," in *Pattern Matching Algorithms.* Oxford University Press, 1997, pp. 341–371.

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

[21] Y. Kanza and Y. Sagiv, "Flexible Queries Over Semistructured Data," in *Proceedings of PODS*, May 2001.

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

[23] P. Wood, "Minimizing Simple XPath Expressions," in *Proceedings of WebDB*, May 2001.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399