Mining XML Documents

Laurent Candillier, Ludovic Denoyer, Patrick Gallinari, Marie-Christine Rousset, Alexandre Termier, Anne-Marie Vercoustre

Mining XML documents

Laurent Candillier
GRAppA, domaine universitaire du pont de bois
Université Charles de Gaulle, Lille 3
59653 Villeneuve d'Ascq Cedex, France
Tel: 33 (0)3 20 41 61 78
Fax: 33 (0)3 20 41 67 70
candillier@grappa.univ-lille3.fr

Ludovic Denoyer
LIP6, Laboratoire d'Informatique de Paris 6
Université Pierre et Marie Curie
8 rue du Capitaine Scott
75015 Paris, France
Tel: 33 (0)1 44 27 84 23
Fax: 33 (0)144 27 70 00
luc.denoyer@lip6.fr

Patrick Gallinari
LIP6, Laboratoire d'Informatique de Paris 6,
Université Pierre et Marie Curie
8 rue du Capitaine Scott
75015 Paris, France
Tel: 33 (0)1 44 27 73 70
Fax: 33 (0)1 44 27 70 00
patrick.gallinari@lip6.fr

Marie Christine Rousset
LSR-IMAG,
B.P. 72, 38402 St Martin d'Hères Cedex, France
Tel: 33 (0)4 76 82 72 83
Fax: 33 (0)4 76 82 72 87
Marie-Christine.Rousset@imag.fr

Alexandre Termier
The Institute of Statistical Mathematics
4-6-7 Minami-Azabu, Minato-ku, Tokyo 106-8569, Japan
termier@ism.ac.jp

Anne-Marie Vercoustre,
INRIA,
Domaine de Voluceau, Rocquencourt
B.P. 105, 78153 le Chesnay Cedex, France
Tel: 33 (0)1 39 63 55 88
Fax: 33 (0)1 39 63 58 92
Anne-marie.Vercoustre@inria.fr

Keywords: XML Mining, structured documents, tree-based model, document representation, classification, clustering, frequent patterns, frequent trees, feature selection, stochastic generative model, Bayesian networks.

Mining XML Documents

Abstract

XML documents are becoming ubiquitous because of their rich and flexible format that can be used for a variety of applications. Giving the increasing size of XML collections as information sources, mining techniques that traditionally exist for text collections or databases need to be adapted and new methods to be invented to exploit the particular structure of XML documents. Basically XML documents can be seen as trees, which are well known to be complex structures. This chapter describes various ways of using and simplifying this tree structure to model documents and support efficient mining algorithms.

We focus on three mining tasks: classification and clustering which are standard for text collections; discovering of frequent tree structure which is especially important for heterogeneous collection. This chapter presents some recent approaches and algorithms to support these tasks together with experimental evaluation on a variety of large XML collections.

INTRODUCTION

The widespread use of semi-structured formats like XML for representing data and documents has urged the need to develop tools to efficiently store, access and organize XML corpus. With the development of such structured textual and multimedia document, the document nature is changing. Structured documents usually have a much richer representation than flat ones. They have a logical structure. They are often composed of heterogeneous information sources (e.g. text, image, video, metadata, etc). Another major change with structured documents is the possibility to access document elements or fragments. The development of classifiers for structured content is a new challenge for the Machine Learning (ML) and Information Retrieval (IR) communities. A classifier for structured documents should be able to make use of the different content information sources present in an XML document and to classify both full documents and document parts. It should easily adapt to a variety of different sources and document models (i.e. different Document Type Definitions). It should be able to scale with large document collections.

Handling structured documents for different IR tasks has recently attracted an increasing attention. Many questions are still open for designing such systems so that we are only in the early stages of this development. Most of the work in this new area has concentrated on ad hoc retrieval in the context of the recent INitiative for the Evaluation of XML Retrieval (INEX) launched in 2002. Besides this mainstream of research, some work is also developing around other generic IR problems like clustering and classification for structured documents.

The use of XML format raises a new challenge for document mining, first because of its new complex data structure, second by the two dimensions that can be dealt with: the (semi-) structured dimension and the content (especially text) dimension, and third because of the possible heterogeneity of the documents. Depending on the application or the mining objective, it may be relevant to consider the structure information alone or both the structure and the content of the documents.

XML documents are usually modelled as ordered trees which are regarded as complex structures. Indeed algorithms dealing with tree collections may be very costly when the size of the documents and the collection increases. It is often necessary to *simplify* the document tree model in order to implement efficient algorithms or to adapt scalable existing clustering and classification methods. A common simplification, for example, is to ignore the order of tree siblings, yet some algorithms would take this into account when required by the data.

This chapter describes various tree-based representations of XML documents to support efficiently three mining tasks: frequent pattern extraction, classification and clustering. Frequent pattern extraction from document structure has been studied mostly by the database community, with the objective of clustering large heterogeneous collections of XML documents to support query optimisation. Classification and clustering using document structure, and possibly content, has been studied both by the IR and ML communities.

We first introduce the XML tree-based model. Then we present some advanced algorithms for frequent pattern extraction in XML collections. In the last section we present three flexible classes of document representation that have been used in classification or clustering algorithms. Although we do not cover all the possible representations for XML documents, we show many different working representations that can be derived from the initial complex tree-like structure in order to support XML mining tasks.

TREE-BASED COMPLEX DATA STRUCTURE

XML documents are regarded as semi-structured data since they offer a flexible structure compared to more strictly structured databases. For example, elements in XML documents can be optional or have an unfixed number of occurrences. The document structure can be constrained by the definition of a DTD (Document Type Description), or a document can be just *well formed*. A well formed XML document must conform to the XML grammar, which mostly means that, unlike HTML documents, all the tags must be well parenthesised and that a document must have a single root.

The XML Document Object Model (XML DOM) defines a standard way for accessing and manipulating XML documents. The DOM presents an XML document as a tree-structure (a node tree), with the elements, attributes, and text defined as nodes.

Fig. 1 and Fig.2 present a small XML document describing a movie and its associated tree-structure. This description is very typical of the ones to be used for indexing and retrieving movies, and very different from the ones that would be necessary for describing the frames of the film itself. Note the attribute *lang* in the second title, and the repetition of the elements *title* and *actor*. This document here conforms to the DTD that is referred to in the document model CINEMA (second ligne of the document), not shown here.

```
<?xml version = "1.0" encoding="ISO-8859-1"?>
<!DOCTYPE CINEMA SYSTEM "cinema.dtd">
<movie>
        <header>
                <title>Pierrot le Fou</title>
                <title lang ="en">Pierrot Goes Wild</title>
                <lang>French</lang>
                <year>1965</year>
                <genre>Drama</genre>
        </header>
        <dir>Jean-Luc Godard</dir>
        <photos>Raoul Coutard</photos>

        <tech>
                <color> B&W </color>
                <format>35mm</format>
                <time>110mn</time>
         </tech>
         <actors>
                <actor>Anna Karina</actor>
                <actor>Jean-Paul Belmondo</actor>
                <actor>Graziella Galvani </actor>
        </actor>
        <desc>Pierrot escapes his boring society and travels from Paris to the Mediterranean Sea with Marianne, a girl
        chased by hit-men from Algeria. They lead an unorthodox life, always on the run.
        </desc>
</movie>
```

Fig. 1 An XML document describing the movie "Pierrot le Fou", by Jean-Luc Godard

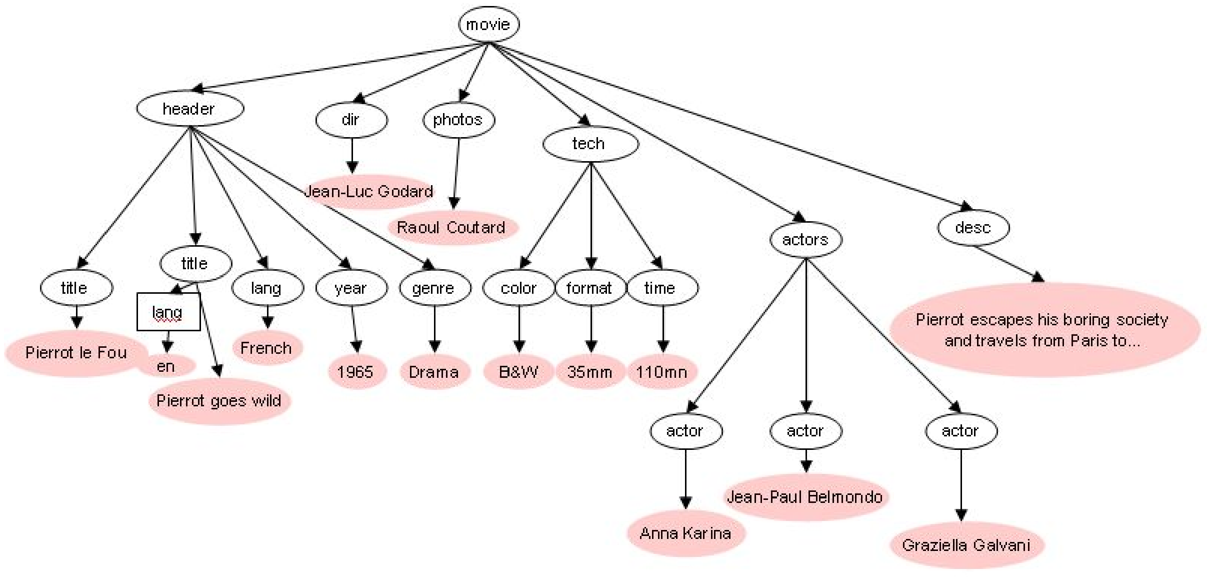


Fig. 2 XML tree corresponding to the document in Fig.1. Nodes are represented by white round boxes, attributes by square boxes and leaf nodes by grey round boxes.

XML documents can be modelled by unranked, ordered labelled trees where labels correspond to XML tags which may or may not carry semantic information. More precisely, when considering only the tree structure:
- each node can have an arbitrary number of children,
- the children of a given node are ordered,

- each node has a label in the vocabulary of the tags.

A labelled node n in a XML document is represented by a couple $n=(s, t)$, where s is the label of n in the structure of the document, and t represents the content of n.
Let $S = \{s_1,...,s_{|S|}\}$ be the set of structural labels, and $\mathcal{T} = \{t_1,...,t_{|\mathcal{T}|}\}$ be the set of possible contents.

An XML document can then be represented by a **labelled tree**. A labelled tree $T = (N, A, root(T))$ is an acyclic connected graph, where N is the set of nodes,
$A \subseteq N \times N$ is a binary relation over N defining the set of edges, and $root(T)$ is a distinguished node called the **root.**
Let $u \in N$ and $v \in N$ be two nodes of a tree. If there exists an edge $(u, v) \in A$, then v is a **child** of u, and u is the **parent** of v.
For two nodes of a tree u and v, if there exists a set of nodes $\{x_1,...,x_p\}$ such that u is the parent of x_1, x_1 is the parent of x_2,..., and x_p is the parent of v, then $\{x_1,...,x_p\}$ is called a **path** from u to v. If there exists a path from u to v in the tree, then v is a **descendant** of u, and u is an **ancestor** of v.
Finally, for two nodes of a tree u and v, if there exists a node x_1 such that both u and v are children of x_1, then u and v are called **sibling** nodes.

A tree is an **attribute tree** if two sibling nodes cannot have the same label (Arimura et al., (2005) describe attribute trees in more detail). This is mostly not the case in XML documents where lists of elements with the same label are quite common. However it is possible to transform the original trees into attribute trees without losing their most canonical structural properties.

Document transformation: Most tree mining algorithms do not directly operate on XML documents. They need a pre-processing step, which takes as input the XML documents and outputs a labelled tree for each document. This labelled tree reflects the tree structure of the original document, where the node labels are the tags of the XML document. Then the documents will be further transformed depending on the document model used by the intended mining task and the specific algorithm. Pre-processing may involve:

- stripping off the textual content of the document when dealing with structure only;
- stripping off the XML attributes when they are not regarded as relevant for the mining task, or transforming them to fit in the tree structure just like XML elements;
- replace some tag labels with equivalent labels, for example if a DTD defines different types of paragraphs (p1, p2, p3), it may be advised to rename them by a common label such as *parag* (this type of transformation requires some semantic knowledge of the DTD or the collection);
- stripping off low levels of the tree which would be irrelevant in some mining tasks, for example *italic* or *bold* elements, or the details of a mathematic formula.
- text processing similar to the one done for flat textual documents, such as removing stop words.

In document collections, content information may be composed of text and images that are associated with the leaves of the trees. In this chapter, we consider only textual parts of documents. The textual content is usually contextually dependent of the logical structure, which means that, even when interested in mining the content of documents, taking the structure into account may have a positive impact on the results. We also consider the case where we are interested only on the structure of the documents, without taking into account their content.

DISCOVERING FREQUENT TREE STRUCTURE

The broad use of XML as an exchange format for data exported from databases results in the availability of huge collections of XML documents in which the labels and their nesting represent the underlying tree schema of the data. Those collections of XML documents are possibly structurally heterogeneous because exported from a mass of different and autonomous data sources. Discovering commonalities between their structures is of primary importance for information integration and classification. In this setting, the focus is not the textual content of the documents but the labelled trees corresponding to their structure.
The discovery of frequent tree patterns from a huge collection of labelled trees is costly but has multiple applications, such as schema extraction from the web (or from frequent user queries like as proposed by (Ji et al., 2005)), automatic creation of DTDs or XML schemas for sub-collections, uniform querying over heterogeneous data, and clustering together data supporting a given frequent tree pattern. Note that the resulting clusters are not necessarily disjoint, which offers different viewpoints on the data and different entry points for querying them.

Definitions and Examples

The most important operation when searching for frequent tree patterns in a collection of tree data is to determine if a tree pattern is included in a tree of the data. The definition used for this **tree inclusion** operation will determine the nature of the patterns that can be discovered, as well as the complexity of the discovery problem.
A tree inclusion definition is based on a tree homomorphism between the tree pattern and the trees of the data (Kilpeläinen, 1992). Different definitions of tree inclusion can be considered according to the preservation or not of (1) the labels (2) the ancestor relation (3) the order of siblings by the homomorphism.

For the first property concerning label preservation, we will only consider homomorphisms preserving labels as most modern approaches use this kind of homomorphism, and because with XML it is important to preserve the labels as they correspond to the tags of the XML documents.

The second property determines the tolerance to "noise" in the nesting of the nodes. If the parent-child relation is preserved, then the paths from root to leaves in the pattern will mimic exactly paths or sub-paths of the original tree. This definition is suitable for relatively homogeneous data (for example data coming from a single source). However, for truly heterogeneous data it is likely that a semantic relation between A and B will be expressed by different structural relationships between A and B within different XML documents written by different organizations. In such cases, the ancestor preserving definition is the only one capable of finding common patterns. Consider for example the trees of Fig. 3 describing the hierarchical relations in a public university in Japan and in France.

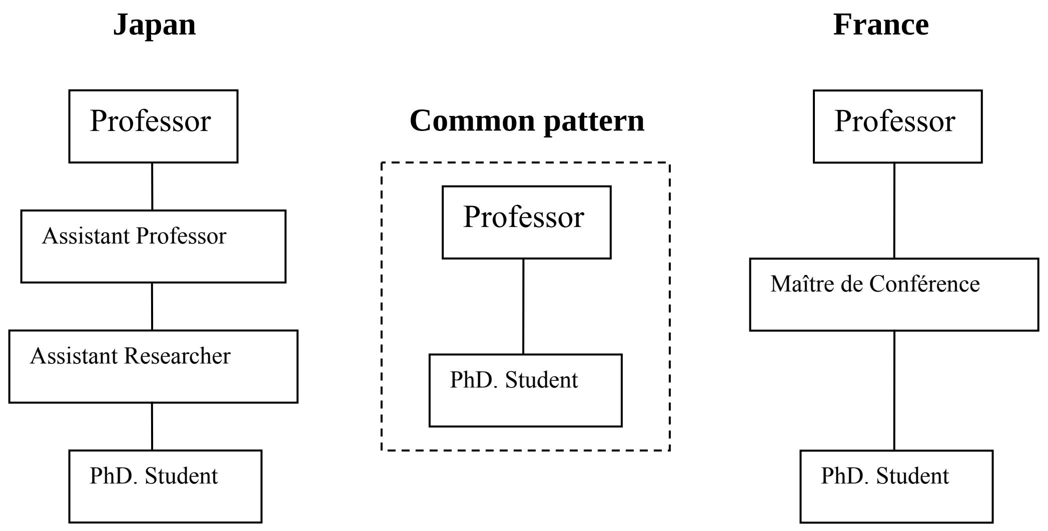


Fig. 3: Academic hierarchical relations in Japan and in France, common pattern between both.

Even if both hierarchies are different, in both cases "Professor" is an ancestor of "PhD Student", so the common pattern can be found. As simple as this pattern is, it shows that there are some common points between both trees.
On the other hand, an inclusion definition based on the preservation of parent-child relation would not have found any common pattern because of the differences between the internal nodes of both trees.

The third property about the order of the siblings has also an influence on the kind of data that it is possible to handle. As XML documents are ordered, one could expect the preservation of the order of siblings in the tree inclusion definition to be mandatory. This may be true for homogeneous collections of data, but with heterogeneous data, different organizations may order the nodes of the documents differently. Consider the two example documents of Fig. 4 about car descriptions: the important point when discovering patterns is that a "Car" has a "Model" and a "Color", no matter their order.

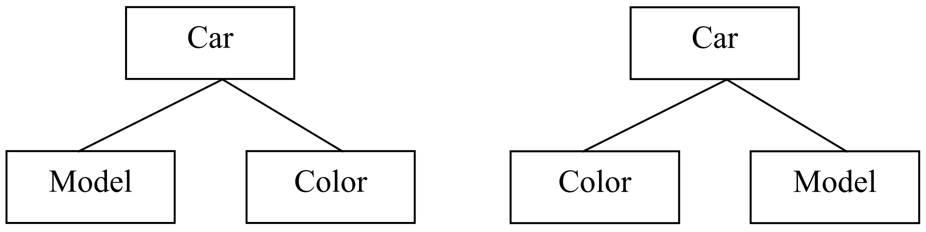


Fig. 4: Two tree structure describing car ads, the same contents is expressed with a different sibling order.

The complexity of testing the tree inclusion varies with the definition used (Kilpeläinen, 1992).

Frequent Tree Discovery Algorithms

Frequent tree discovery is a very computationally intensive task; hence the design of the discovery algorithm is of a central importance. The algorithm must ensure at the same time the quality of the outputs, a fast execution, and avoid consuming too much memory especially when datasets are big. There is currently no perfect algorithm, but many algorithms have been designed and each of them has its strengths and weaknesses. We review the existing algorithms, classifying them according to two main design principles, namely the edge-centric approach and the tile-centric approach.

Basic definitions

Let the data be a set of trees $\{T_1,\ldots,T_N\}$.
A tree T **occurs** in the data if there exists a tree T_i in the data such that T is included in T_i.
A tree T is **frequent** in the data according to an absolute frequency threshold ε if T occurs in more than ε trees of the data. The trees of the data $\{T_{i1},\ldots T_{im}\}$ where T occurs are called the **support** of T.
A tree T is a **closed frequent** tree of the data if T is frequent with support $\{T_{i1},\ldots T_{im}\}$ and T is the biggest tree for this support, i.e. there exist no other frequent tree T' with support $\{T_{i1},\ldots T_{im}\}$ such as T is included in T'.

Edge-centric approaches

Designing a tree mining algorithm from scratch is quite a challenge, and as is often the case with such problems, it is good to start on solid well known ground. A tree can be seen as a set of edges, so a frequent tree can as well be seen as a frequent set of edges.
Finding a frequent set of edges is easy, and since the seminal paper on the Apriori algorithm (Agrawal & Srikant, 1994), there has been a tremendous amount of research on algorithms for discovering frequent itemsets in transactional data (a transaction is a set of elements and an itemset is a set of elements being a subset of one or more transactions). This was the starting point of the TreeFinder algorithm (Termier et al., 2002), which first finds all the frequent set of edges, and then rebuilds the trees from the edges using an operation borrowed from inductive logic programming (ILP), the Least General Generalization (Plotkin, 1970). Though being able to find interesting patterns by application of well know algorithms, the method falls short of completeness in some cases, reducing its practical usability.

However, many other methods for mining structured data have been derived from the very principle of the Apriori algorithm. Inokuchi et al. (2000) presented AGM (Apriori Graph Miner), an algorithm for mining general graphs, based on the *generate and test* principle which is the heart of Apriori. Later on, many tree mining algorithms were designed upon this principle, including specificities for efficient tree mining.

The idea of Apriori to find all the frequent itemsets is to generate candidate itemsets, and to evaluate the frequency of such candidates against the transaction database. If a candidate is frequent, it is flagged as such, and it will also be the base of new, longer candidates. The Apriori algorithm is levelwise: it starts from candidates with only one element, and then processes iteratively candidates with 2,3,...,*n* elements. The efficiency of the search is ensured by the anti-monotony principle, which states that if an itemset I is infrequent, then all the itemsets $I' \supseteq I$ are also infrequent. So when a candidate is evaluated as infrequent, it is not necessary to expand it further, which reduces considerably the search space.

This method can be transposed nicely to trees, by replacing the set elements by tree edges. The first iteration will be to find frequent trees with one edge; the second one will join these trees with one edge to find all frequent trees with two edges, and so on. However, a problem specific to trees arises: a tree can be built by adding edges in many different ways, and so all the different intermediary steps leading to the same tree will be considered by the algorithm, as shown in Fig. 5.

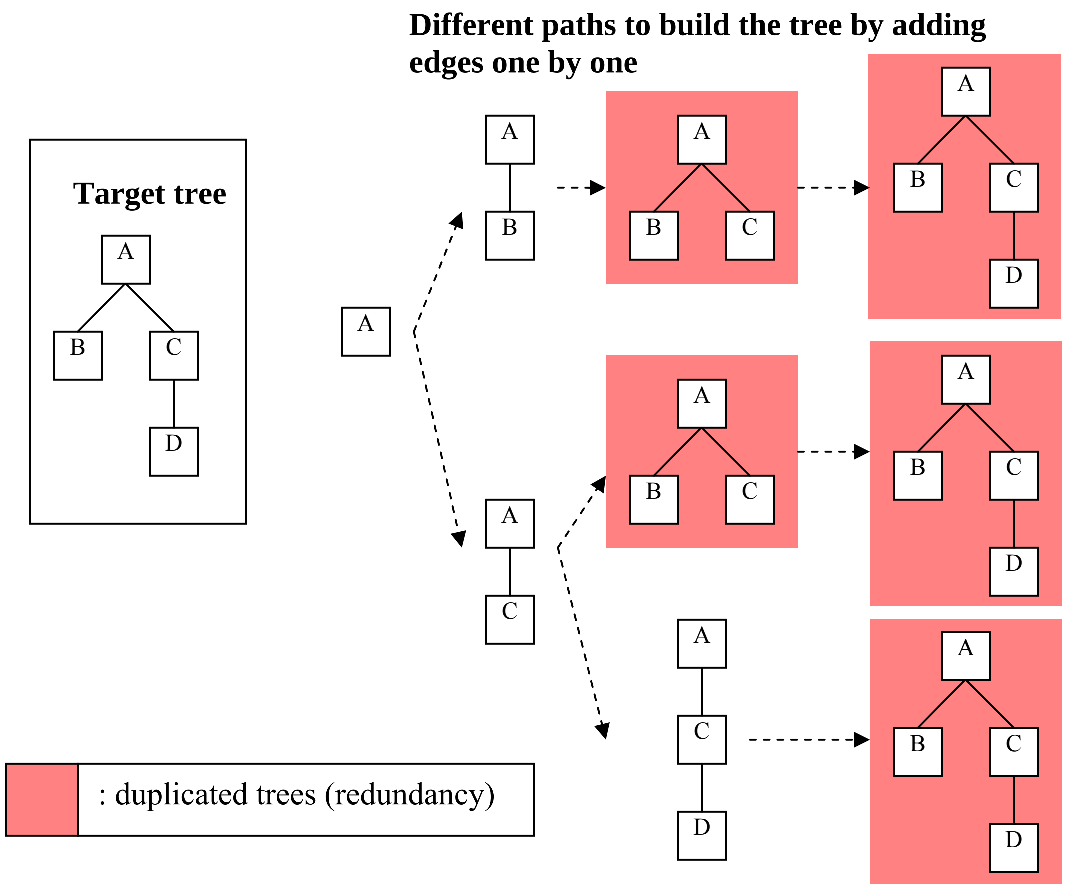

Different paths to build the tree by adding edges one by one
Target tree
A
B
C
D
: duplicated trees (redundancy)

Fig. 5: All the paths to build a single tree by edge adding, and the redundancies induced by creating all these intermediary steps.

This would result in doing a lot of redundant work for each tree. The solution to this problem, found independently by Asai et al.(2002) and Zaki & Aggarwal (2003), is to force the edge expansion to be done in a unique way, along the rightmost branch of the tree. This technique is called *rightmost tree expansion* and is the basis of most frequent tree mining algorithms. Both of the previously cited algorithms mine ordered trees. To be able to mine unordered trees, Asai et al. (2002) and Nijssen & Kok (2003) independently proposed to use *canonical forms*. A canonical form of a tree pattern is a unique representative for all tree patterns that differ only on the order of the siblings.

However, finding all the frequent tree patterns is a very computation-time expensive task. Chi et al. (2004) proposed the CMTreeMiner algorithm that improves the previous algorithms by searching only closed frequent trees, with performance improvements over one order of magnitude. Recently, Arimura & Uno (2005) proposed the CLOATT algorithm for mining closed frequent attribute trees, with a proved output-polynomial complexity.

Tile-centric approach

The previous approaches discover the frequent tree patterns by reconstructing the tree' s edge by edge. However, especially in case of trees with large number of nodes, it can be beneficial to have an approach that expands trees with several edges at a time instead of single edges. Such an approach is represented by the Dryade family of algorithms (Termier et al., 2004; Termier et al., 2005), in which the closed frequent trees of a given step are built by *hooking* at the leaves of the (closed) frequent trees of the previous step full subtrees of depth 1 called *tiles*. In this way, the closed frequent trees are built by increasing levels of depth.

Definition: A *tile* is a closed frequent tree of depth 1.

It can easily be shown that any closed frequent tree can be decomposed into a set of tiles. So the approach used by the Dryade family of algorithms is to first compute all the tiles that are in the data, and then to assemble these tiles together in order to discover the closed frequent trees.

This approach will be illustrated through the DryadeParent algorithm (Termier et al., 2005). This algorithm uses the same tree inclusion definition as CMTreeMiner (the isomorphism preserves the parent-child relation and does not preserve siblings order), but is limited to the discovery of *attribute trees*. The trees of Fig. 6 will be used as data for a running example, with a minimal frequency threshold of 2 trees.

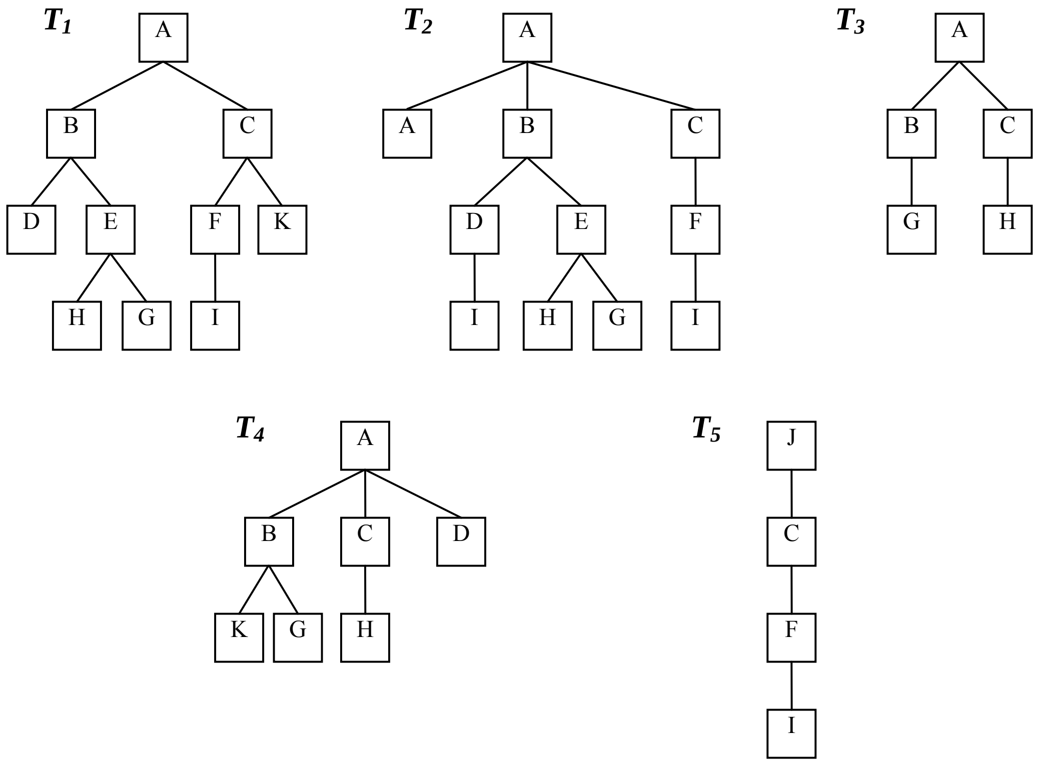
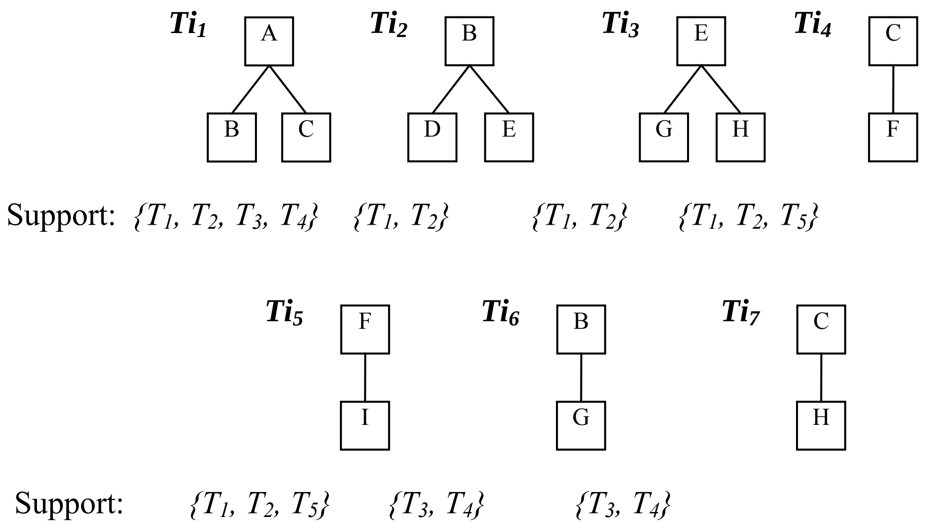


Fig. 6: Example data.

The algorithm can be divided into several steps. The preliminary step is to discover the tiles. Then the iterative part of the algorithm consists in *hooking* together those tiles.

Discovering the tiles: As defined before, the tiles are the closed frequent trees of depth 1. Instead of finding all the tiles, it is simpler to solve the problem of finding all the tiles whose root has label a, for any $a \in S$. This problem boils down to finding all the closed frequent sets of children for the nodes of label a in the data. This problem can easily be solved by using any closed frequent itemset miner. DryadeParent uses the LCM2 algorithm (Uno et al., 2004) which is the fastest algorithm available for this task. By iterating this method on all the labels of *S*, it is then easy to find all the tiles shown in Fig. 7.



Ti1
A
B
C
Ti2
B
D
E
Ti3
E
G
H
Ti4
C
F
Support: {T1, T2, T3, T4} {T1, T2} {T1, T2} {T1, T2, T5}
Ti5
F
I
Ti6
B
G
Ti7
C
H
Support: {T1, T2, T5} {T3, T4} {T3, T4}

Fig. 7: The tiles found in trees of Fig.6 with minimal frequency threshold set to 2.

Hooking the tiles: The previously computed tiles can then be *hooked* together, i.e. a tile whose root has label *a* becomes a subtree of another tile having a leaf of label *a* to build more complex trees. A proper strategy is needed to avoid as much as possible constructing attributes trees that would be found unclosed in a later iteration. The DryadeParent 's strategy consists in constructing attributes trees which are isomorphic to the *k* first depth levels of the patterns, each iteration adding one depth level to the isomorphism.
For this purpose, the first task of DryadeParent is to discover in the tiles those corresponding to the depth levels 0 and 1 of the patterns, the **root tiles**. Some of these tiles can be found immediately as they cannot be hooked on any other tile: they will be the starting point for the first iteration of DryadeParent. This is the case for Ti_1 in the example. For the rest of the root tiles, they can also be used as building blocks for other patterns: they will be used as root of a pattern only when it will become clear that they are not only a building block, to avoid generating unclosed attribute trees. In the example, this is the case for Ti_4, which can be hooked on Ti_1. Only in iteration 2 will this tile be used as a root tile to construct the pattern P_4 (see Fig. 8 for the closed frequent patterns discovered in the data). The computation of the set of tiles that must be hooked to the root tiles for building the next depth level of closed frequent attribute trees is delegated to a closed frequent itemset mining algorithm. They become starting points for the next iteration.

The whole process is shown in Fig. 8. On the root tile Ti_1 (which is also the closed frequent pattern P_1), one can hook the tiles {Ti_2, Ti_4} or the tiles {Ti_6, Ti_7}, the latter leading to the pattern P_2. Note the different supports of the two constructed attribute trees. From the hooking of {Ti_2, Ti_4} on Ti_1, one can then hook the tile Ti_3, leading to the pattern P_3. The tile Ti_4 is not only a building block of P_1, it also has an occurrence which does not appear in P_3 (see tree T_5): it is used as a root tile, and the only possible hooking on it is Ti_5, leading to the pattern P_4.

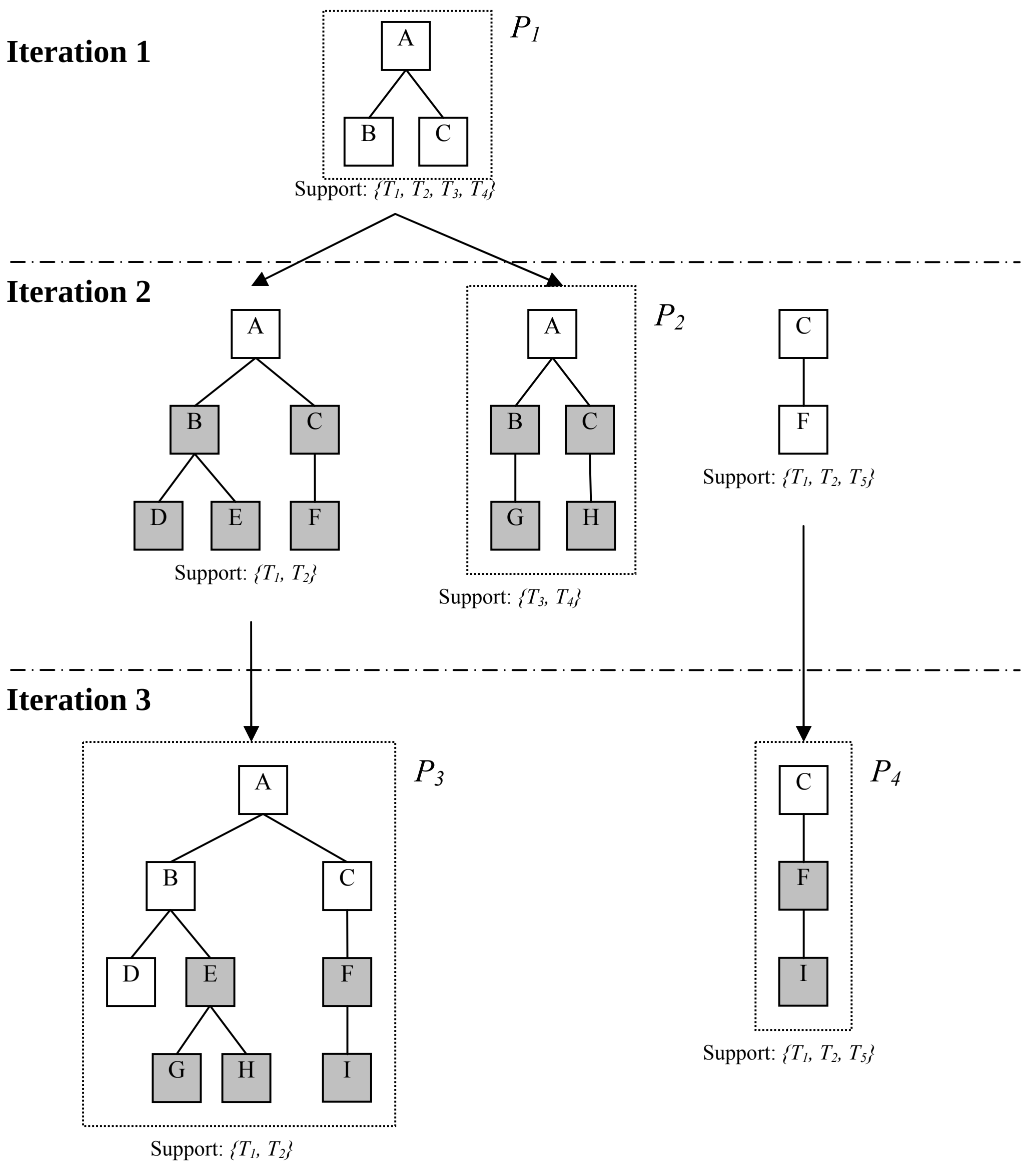


Fig. 8: DryadeParent discovery process.
The closed frequent attribute trees outputted as result are enclosed in dashed boxes.

The soundness and completeness of this hooking mechanism have been proved (Termier, 2004). It has also been shown (Termier, 2005) that DryadeParent has excellent computation-time performances. It over performs by several orders of magnitude CMTreeMiner when the closed frequent trees to be found have a high average branching factor. Concretely, this means that when handling thousands of XML documents, containing a pattern having 100 nodes, DryadeParent will be able to answer in few seconds, sometimes nearly instantly, allowing real-time use in an interactive process. On the other hand, CMTreeMiner will need several minutes to handle the same data, making interactive use problematic.

CLASSIFICATION AND CLUSTERING

In this section, we consider various approaches to XML document classification and clustering, two important mining tasks largely explored for data or textual documents. As for

XML documents, these tasks can be split into sub-tasks that involve the structure only or both the structure and content of the documents.

Classification and clustering are based on a notion of distance. Since XML documents are represented by trees, a natural idea to adapt traditional methods to XML documents would be to use a tree distance, for example the edit distance between two ordered labelled trees proposed by Zang & Shasha (1989), that consists in counting the number of editing operations (add, delete, change the label of a node, etc.) needed to transform a tree into another one. Tree edit distances may differ by the set of editing operations they allow. However, algorithms based on tree edit distances (Chawathe et al., 1996; Nierman & Jagadish, 2002) have a time complexity O(MN), M and N being the number of nodes in the two trees to be compared, which is too high for practical applications. Some approaches therefore replace the original trees by structural summaries (Dalamagas et al., 2004) or s-graphs (Lian et al., 2004) that only retain the intrinsic structure of the tree: for example reducing a list of elements to a single element, or flattening recursive structures. Fig. 9 gives an example of tree summary.

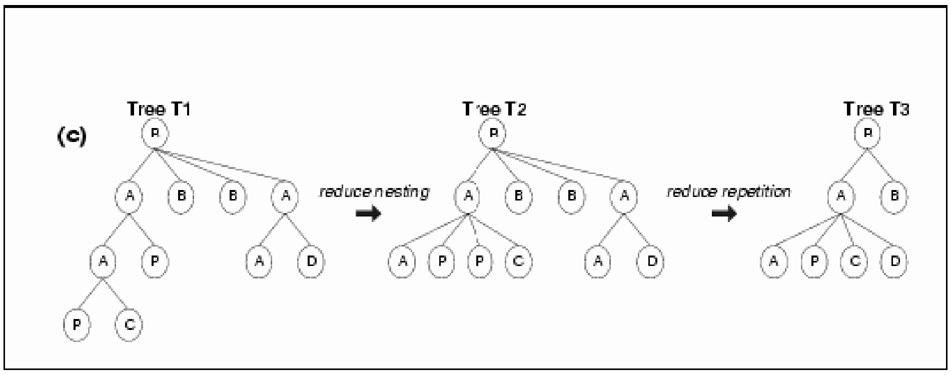


Fig 9. Tree summary.

Other distances have been proposed. Flesca et al. (2002) define a Discrete Fourier transform distance using a linear encoding of the document trees based on the depth-first, left-to-right traversal order. Lian et al. (2004) use a distance based on bit string encodings of the edges of the s-graphs.

However, the above approaches are limited to clustering documents based only on their structure. Besides, computing distances directly between trees may consume a lot of time, and the problem of providing interpretable results remains an open issue. That is why some methods based on different ways for representing XML documents were recently designed.

In the next sections we present three different XML document representations, based on structure or both structure and content, that have been used in combination with different clustering or classification approaches. First we present an attribute-value representation combined with classification and clustering methods based on decision-trees or probabilistic models; second a representation based on document paths, regarded as words, with a k-means like clustering algorithm; finally a Bayesian network model used as a generative process model for classification.

Representation Using Attribute-Value Structure

Candillier et al. (2006) investigate the use of a different kind of representation for the manipulation of XML documents. The idea is to transform the trees into sets of attribute-values pairs, so as to be able to apply various existing methods of classification and clustering on such data, and benefit from their strengths. They propose to construct the following attributes from a set of available XML trees:

- the set of tags labelling the nodes of the trees;
- the set of parent-child and next-sibling relations (whose domain is the set of pairs of tags labelling the nodes);
- the set of distinct paths (including sub-paths), starting from the root (whose domain is the set of finite sequences of tags labelling the nodes).

So they create as many new attributes as distinct features are encountered in the training set. And for each of them, their value for a given document is the number of their occurrences in this document. Finally, they also define as many new attributes as there are absolute distinct node positions in the trees. For every identifier of a node position, the value of the attribute for a document is the arity of the node, which is the count of its child nodes in the document. So the new introduced attributes all take their value into the set of natural numbers.

Such representation could lead to a high number of generated attributes. So the algorithms used to tackle such new datasets should be able to handle many attributes, and to perform feature selection during their learning process. In a classification task, C5 (Quinlan, 1998) is for example well suited. In a clustering task, a subspace clustering algorithm, that is a clustering algorithm able to characterize every distinct cluster on a limited number of attributes (eventually distinct for each cluster), should be used.

Thus, they used SSC (Candillier et al., 2005), a subspace clustering algorithm that has been shown to be effective, and that is able to provide as output an interpretable representation of the clusters found, as a set of rules. They adapted SSC for the clustering and the classification of XML documents, so that the new methods also benefit from the major advantage of producing classifiers that are understandable.

So far, this strategy has been applied to the INEX 2005 collections for mining XML documents, both for the classification task and for the clustering task using the structural description of XML documents alone. The results obtained with such a strategy are very good. In particular, it has been shown to be robust even with noisy data. Besides, the produced classifiers are very understandable. Fig. 10 shows for example the decision tree obtained when classifying a collection of movie description into eleven classes. For instance, the membership to class 8 only depends on the presence of tags named *movie, film, table* and *li*, and the absence of parent-child relation between tags named *p* and *i* in the document.

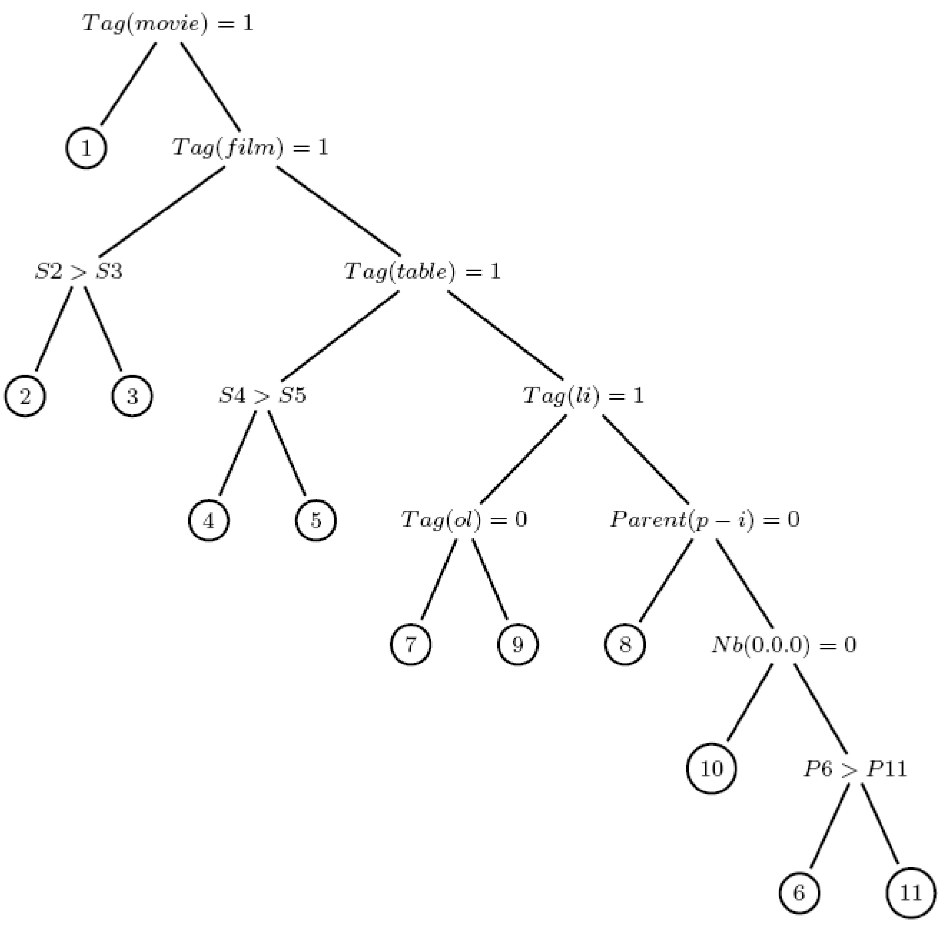


Fig. 10: Example of understandable classifier obtained on an XML data collection by transforming the documents into sets of attribute-values.
Tag(T) refers to the number of occurrences of the given tag T in the document, Parent(A-B) to the number of parent-child relations between tags A and B, Nb(0.0.0) to the arity of the first grand-child node from the root, S to comparisons between models based on the next-sibling relations, and P* to a comparison based on the paths in the document.*

By using such a transformation strategy, part of the information included in the trees is lost, but the data are manipulated more easily and the results are more understandable. However, some differences between trees could be hidden when using such a transformation. Indeed, a swap between two sub-trees of a given tree can lead to a very different tree, although its attribute-value representation proposed here would be very similar to the one of the initial tree. So other types of transformation should be considered when the order of siblings is to be taken in account.

Finally, such a strategy should be generalized in order to perform classification or clustering of XML documents collections using both the structural description and the textual content of the documents. The most direct way to do this would be to consider the textual content of the documents as simple bag-of-words, but more complex transformations could also be considered.

Representing Documents by a Set of Paths

Vercoustre et al. (2006) have been motivated by clustering XML documents based on their structure only or both the structure and content. Like in the previous approach, they use a simplified tree representation to avoid the complexity problems of tree clustering. They define a flexible representation of XML documents based on a subset of their paths, generated according to some criteria, such as the length of the paths, whether they start at the root (root paths), or end at a leaf (leaf ending paths). By considering those sub-paths as words, they can use standard methods for vocabulary reduction (based on their frequency), and simple clustering methods such as K-means that scale well.

Basic Definitions

Let p= $\{x_1,...,x_p\}$ be a path of tree T as defined in section 2;
p is a **root path** if x_1 = root(T); p is a **leaf ending path** if x_p is a leaf of T; p is a **complete path** if it is a root path and a leaf ending path. The **length** of a path is the number of nodes in the path. A path expression is represented by the expression s= $s_1.s_2...s_{p-1}.s_p$ where s_i =label(x_i). Such expressions do not distinguish between two siblings with the same label, which means that, in this approach, two paths are regarded as equivalent of they correspond to the same path expression.

Let u=(s, t) be a node of tree T, where t is the textual content of u. Let w be a word.
u **contains** w if $w \subset t$ or if there exists a path $\{u, \ldots, v\}$, v=(s',t') such that $w \subset$ t'.

If u contains w, and p= $\{x_1,...,u\}$ is a path, we call p' =(p,w} a **text path** and we code it by $s_1.s_2...s_{p-1}.s_p.w$ where s_i =label(x_i).
It means that a text path extends a path p (possibly non terminal) with a word contains in any of the leaves of its subtrees.
If W=$\{w_i \mid w_i \subset t\}$ then $s_1.s_2...s_{p-1}.s_p.w_i$, for $w_i \in$ W, are all the text paths associated to path p.

Using those definitions, we can now introduce a family of representations for an XML document d using the standard vector model, as:
R(d)= $\Sigma_i\ f_i\ p_i$,
for all path $p_i = s_1.s_2...s_{p-1}.s_p$ in d where $m \le |p_i| \le n$, $1 \le m \le n$, m and n two numbers given a priori; f_i is the frequency of the path p_i in d.

When interested by both the structure and the content of documents, it is possible to use both text paths and paths, or text paths only. For specific values of m and n, this model is equivalent to some other models that have been proposed before: for m=n=1, it corresponds to the *naïve* model used by Doucet & Ahonen-Myka (2002), where documents are represented by a bag of tags, or a bag of words and tags. Representing documents by their complete paths has been proposed by Yoon et al. (2001) in their bitmap model, as well as an extension using complete text paths.

Yi and Sundaresan (2000) propose the Structure Vector Model where a document is represented by all its paths of length between 1 and the height h of the document tree. The frequency of terms associated with a path is relative to the subtree associated with that path. The representation developed by Liu et al. (2004) is based on paths of length smaller than L, although they can also fix the level in the tree where the paths must start. It also includes the definitions of leaf-ending paths as well as root-beginning paths, of length less than L.

The motivation for a flexible choice of paths in the document is that some analysis or clustering tasks may be interested in the top part of the tree, the lower parts of the tree, or possibly parts in the middle. An example would be clustering very heterogeneous collections based on the structure, where the partition can be done by looking at the top level elements only. At the opposite end of the spectrum, if one wants to cluster documents based mostly on the text, it could be appropriate to add some limited context just above the text (leaf-ending paths). Another motivation in using paths was to fully represent lists of elements, through their path frequency, as lists are an important feature of XML documents that should be taken into account in some clustering tasks.

By considering those paths as words (with their frequency), it is possible to use standard methods for vocabulary reduction, and simple clustering methods such as K-means. However, clustering algorithms based on the vector model rely on the independence of the various dimensions (modalities) for calculating the distance between the vectors.
Although it is not always verified in practice with words in texts, it usually works fine. In the case where words are paths in the document tree, there is an obvious dependency between embedded sub-paths. To deal with the problem of dependency, one can partition the paths by their length and treat each set of paths as a different variable, using a clustering algorithm such as the one proposed by Celeux et al.(1989 in which the standard Euclidian distance between clusters is replaced by a distance that takes in account the different variables and the modalities within the variables as follows:

$$d(x,y) = \sqrt{\sum_{k=1}^{p}\sum_{j=1}^{m_k}(x_j^k - y_j^k)^2}$$

where p is the number of variables, and m_k is the number of modalities for the variable k.

The approach has been successfully applied to the INEX collections for the structure only tasks. As an output, the clustering algorithm provides not only the set of clusters but discriminate representatives for each cluster that characterize each of them. Since the representatives are paths, it could be interesting to reconstruct sub-trees from those paths in order to provide a more compact representation of the clusters. This should be feasible, at least for root paths.

When considering document content as well as structure, paths are extended with the individual words of the text contained in the terminal node of each path (not necessarily a leaf node). While it works well for relatively small collections, it does not scale well for very large collections and broad trees where the number of paths, especially leaf-ending paths would grow exponentially. Complementary ways of reducing the vocabulary are needed, possibly from relative frequency of words within their specific paths rather than within the document they belong to.

Stochastic Generative Model

Generative models are well-known in the field of Machine Learning. They are used for different applications and particularly for classification and clustering. Generative models allow us to compute the probability of an XML document using a set of parameters describing a stochastic process (that is $P(d/\theta)$ where d is an XML document and θ is a set of parameters). In this part, we describe the family of models proposed by Denoyer & Gallinari (2004). They propose to model the statistical dependencies between the nodes of a semi-structured document using the belief network formalism. These models consider that a document d is the realization of a random vector D whose elements are discrete random variables corresponding to each structural node or content node (text nodes) of the document. Each document is transformed into a belief network, all the networks sharing the same set of parameters. The probability of a document is then computed as the joint probability of the corresponding network and this probability can be computed under different assumptions of statistical dependencies. Each assumption aims at capturing a particular type of structural information, for example the left-sibling information, or the parent information.

This model is able to deal with a very large amount of data. Moreover, its learning complexity is linear with respect to the size of the documents. This type of model has been used both for the categorization and the clustering of XML documents and the authors have proposed extensions that take into account different information content (text, pictures,...) for the multimedia filtering task (Denoyer et al., 2003). A discriminative algorithm has also been developed for the categorization task. On different XML corpora, this model has better performances than state-of-the art models for categorization of textual documents that do not use the structural information (Denoyer & Gallinari, 2004).

For simplification, we describe the model only for textual documents, using the example of Fig. 11. Extensions for multimedia documents are considered by Denoyer et al (2004).

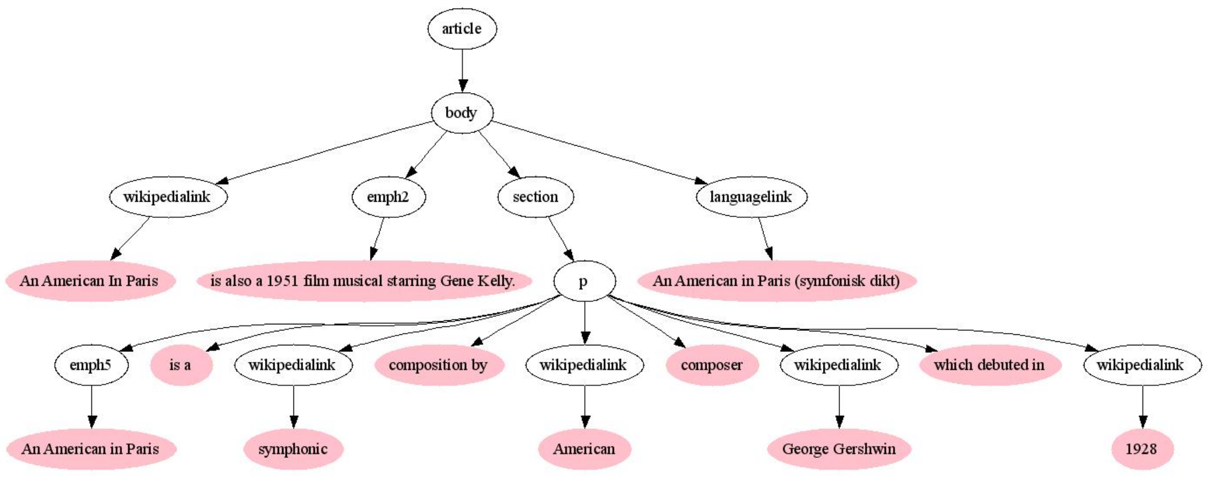


Fig. 11: A tree representation for a structured document composed of an introduction and two sections. White-background nodes and pink/grey background nodes are respectively structural and content nodes.

Modelling documents with Bayesian networks

Let us first introduce some notations:

- Let C be a discrete random variable which represents a class from the set of classes $\mathcal{C}$.
- Let Λ be the set of all the possible labels for a structural node.
- Let V be the set of all the possible words. V^* denotes the set of all possible word sequences, including the empty one.
- Let d be a structured document consisting of a set of features $(s_d^1,\ldots s_d^{|d|},t_d^1,\ldots,t_d^{|d|})$ where s_d^i is the label of the i-th structural node of d ($s_d^i \in \Lambda$), t_d^i is the textual content of this i-th node ($t_d^i \in V^*$) and $|d|$ is the number of structural nodes. d is a realization of a random vector D. In the following, all nodes are supposed to have a unique identifier, indicated here as superscript i.

Bayesian networks offer a suitable framework for modelling the dependencies and relations between the different elements in a structured document. A network model is associated with each document. Since the focus here is on the logical document structure, each network is defined according to the corresponding document structure. For the classification task, the network parameters are learned using all the documents from the same class in the training set. Documents from the same class then share their parameters and there is one set of such parameters for each class.

Different networks could be used for modelling a document, depending on which type of relation one would like to take into account. We only consider here the explicit document structure and we will not try to uncover any hidden structure between the document elements. Some of the natural relations which could then be modelled are: “is a descendant of” in the document tree, “is a sibling of”, “is a successor of”, given a preorder visit of the document tree, and combinations of these different possibilities. Fig. 12 and 13 give two examples of document models encapsulating different relations for the document tree in Fig. 11. In the simplest one (Fig. 12), the network structure is similar to the document tree structure, as it only encodes the ‘is a descendant of’ relation. The second model (Fig. 13) makes use of a *Tree Augmented Network* (TAN) at each level of the tree and takes into account an ordering relation between structural siblings and subtrees. As usual there is a trade-off between complexity and efficiency. Tests performed with different models did not show a clear superiority of one model over the others with respect to the classification performances. For simplicity, from now on, we then consider tree-like Bayesian networks. The network structure is built from the document tree, but need not be identical to this tree.

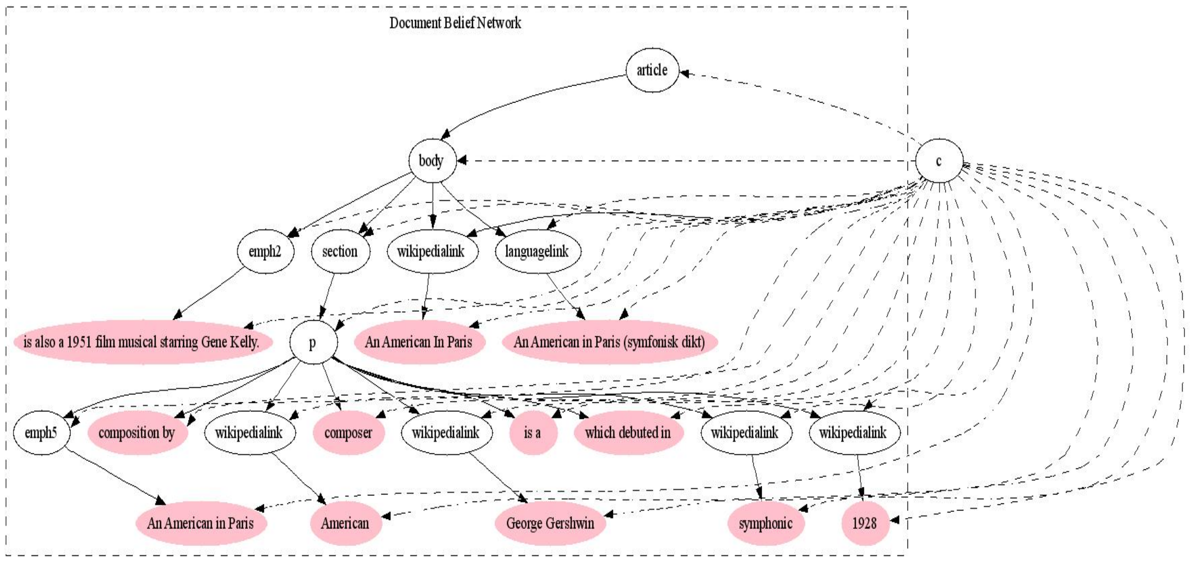


Fig. 12: The final Bayesian network encoding "is a descendant of" relation.

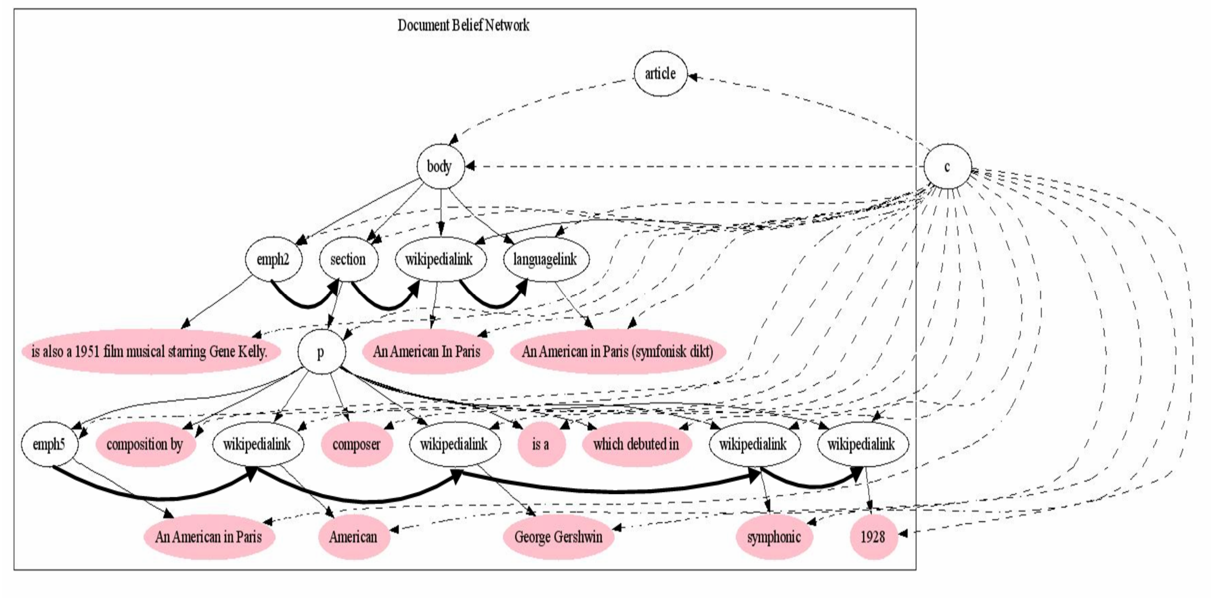


Fig. 13: The final Bayesian network making use of a TAN network at each level of the tree.

A Tree-like model for structured document classification

For this model, we make the following assumptions:

- There are two types of variables corresponding to structure and content nodes.
- Each structure node may have zero or many structure sub-nodes and zero or one content node.
- Each feature of the document depends on the class *c* we are interested in.
- Each structural variable s_d^i depends on its parent $pa(s_d^i)$ in the document network.
- Each content variable t_d^i depends only on its structural variable.

The generative process for the model corresponds to a recursive application of the following process: at each structural node *s*, one chooses a number of structural sub-nodes, which could be zero, and the length of the textual part if any. Sub-nodes labels and words are then sampled from their respective distribution which depends on *s* and the document class. The document depth could be another parameter of the model. Document length and depth distributions are omitted in the model since the corresponding terms fall out for the classification problems considered here.

Using such a network, we can write the joint content and structure probability that document d belongs to class c:

$$P(d,c) = P(c)\underbrace{\left(\prod_{i=1}^{|d|} P(s_d^i \mid pa(s_d^i),c)\right)}_{(a)}\underbrace{\left(\prod_{i=1}^{|d|} P(t_d^i \mid s_d^i,c)\right)}_{(b)} \qquad (1)$$

where (a) and (b) respectively correspond to **structural** and **textual probabilities.** Structural probabilities $P(s_d^i|pa(s_d^i),c)$ can be directly estimated from data using some smooth estimator.

Since t_d^i is defined on the infinite set V^*, we shall make additional hypothesis for estimating the textual probabilities $P(t_d^i|s_d^i,c)$. In the following, we use a Naive Bayes model for text fragments, but this is not a major option and other models could do as well. Let us define t_d^i as the sequence of words $t_d^i=(w_{d,1}^i,....,w_{d,|t_d^i|}^i)$ where $w_{d,k}^i \in V$ and $|t_d^i|$ is the number of word occurrences i.e. the length of t_d^i. Using Naive Bayes for the textual probability, the joint probability for this model is then:

$$P(d,c) = P(c)\left(\prod_{i=1}^{|d|} P(s_d^i \mid pa(s_d^i),c)\right)\left(\prod_{i=1}^{|d|}\prod_{j=1}^{/t^{i,d}} P(w_j^{i,d} / s_d^i,c)\right) \qquad (2)$$

Learning

In order to estimate the joint probability of each document and each class, the model parameters must be learned from a training set of documents. We do not describe here the learning algorithm which is fully explained in (Denoyer & Gallinari, 2004).

Experiments

Many experiments have been made with this model on different corpora (INEX, WebKB, NetProtect,WIPO). Denoyer & Gallinari (2004) for more details on these collections. Table 1 gives the results of different models on three XML corpora :

- the INEX corpus composed of about 12,000 XML documents that describe scientific articles (18 categories)
- the WIPO corpus composed of 10,900 XML documents that describe patents (15 categories), and
- the WebKB corpus composed of 8282 XHTML documents (7 categories).

The different methods are compared using a classic F1 score (micro and macro):

- NB is the Naive Bayes method on flat documents,

- RB is the model proposed here
- SVM TF-IDF is the classic SVM method for classification of flat documents using TF-IDF vectors, and
- Fisher RB is a kernel method that allows us to use our Belief network model with a support vector machine.

The results in Table 1 show that the RB model improves the baseline models.

	Micro-F1	Macro-F1
NB	0.59	0.605
RB model	**0.619**	**0.622**
SVM TF-IDF	0.534	0.564
Fisher RB	**0.661**	**0.668**

INEX

	Micro-F1	Macro-F1
NB	0.801	0.706
RB model	**0.827**	**0.743**
SVM TF-IDF	0.737	0.651
Fisher RB	**0.823**	**0.738**

WebKB

	Micro-F1	Macro-F1
NB	0.662	0.565
RB model	**0.677**	**0.604**
SVM TF-IDF	0.822	0.71
Fisher RB	**0.862**	**0.715**

WIPO

Table 1. Results of the RB model on different XML textual corpora

Future trends for the stochastic generative model

We have presented a generative model for structured documents. It is based on Bayesian networks and allows modeling the structure and the content of documents. It has been tested for the classical task of whole document classification. Experiments show that the model behaves well on a variety of situations. Further investigations are needed for analyzing its behavior on document fragments classification. The model could also be modified for learning implicit relations between document elements besides using the explicit structure. An interesting aspect of the generative model is that it could be used for other tasks relevant to IR. It could serve as a basis for clustering structured documents. The natural solution is to consider a mixture of Bayesian network models where parameters do depend on the mixture component instead of the class as it is the case here. Schema-mapping and automatic document structuring are new tasks that are currently being investigated in the database and IR communities. The potential of the model for performing inference on document parts when information is missing in the document will be helpful for this type of application. Preliminaries experiments about automatic document structuring are described by Denoyer et al. (2004, July).

CONCLUSION

XML is becoming a standard in many applications because of its universal and powerful tree structure. On the internet for example, unstructured documents are being replaced by such structured documents, so that approaches that have been designed to tackle internet resources need to be revisited in order to take advantage of the new structured nature of the documents.

The tree structure of XML documents can be seen as information by itself. When searching for the origin of a document for example, looking at its tree structure can be sufficient because different sources may use different structures to generate their documents. Clustering XML documents using their structure only can help methods designed to handle homogeneous XML collections work also on heterogeneous collections.
But taking the structure of the documents into account may also have a positive impact on the results even if we are only interested in classifying the documents according to their content. The organisation of a document may indeed differ from one context to another. The structure of a document can for example help distinguish an article concerning history to another one about science. Moreover, generic models that combine structure and content may help put the right context or weight on smaller parts of the documents.

On the other hand, many existing approaches designed to handle data represented by trees suffer from high complexities, limiting their use to small volumes of data. Hopefully, as we have shown in this chapter, some transformations of XML tree structures can be used to simplify their representations, still preserving some of their interesting structural properties, and thus providing new ways to efficiently manage high volumes of such data.

As a summary, since XML collections will become more and more important, and since their tree structure can help improve ML and IR tasks on such data, a good compromise has to be found when designing a new method for XML, so that the information contained in the structure is used but does not affect too much its complexity.

Another important challenge concerns the output provided by such methods. In that field, we have highlighted some methods that can exhibit the resulting tree patterns, classifiers or cluster representatives, and therefore can support analysts in mining tasks.

REFERENCES

Agrawal, R. & Srikant, R. (1994, September). Fast Algorithms for Mining Association Rules in Large Databases. In *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB'94*, Santiago de Chile**,** Chile, pp 487-499.

Arimura, H. & Uno, T. (2005). An Output-Polynomial Time Algorithm for Mining Frequent Closed Attribute Trees. In Kramer, S. & Bernhard Pfahringer, B. (Eds.): *Proceedings of the 15th International Conference on Inductive Logic Programming, ILP'05*, Bonn, Germany, LNCA (3625), pp 1-19, Springer.

Asai, T., Abe, K., Kawasoe, S., Arimura, H., Sakamoto, H. &, Arikawa, S. (2002, April). Efficient Substructure Discovery from Large Semi-structured Data. In *Proceedings of the 2nd SIAM International Conference on Data Mining, ICDM'02,* Arlington, VA, USA, pp 158-174.

Candillier, L., Tellier, I., & Torre, F. (2006). Transforming XML trees for efficient classification and clustering. In *Proceedings of the 4th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX'05*, Schloss Dagstuhl, Germany, LNCS (3977), Springer.

Candillier, L., Tellier, I., Torre, F., & Bousquet, O. (2005). SSC: Statistical Subspace Clustering. In Perner, P. and Imiya, A. (Eds): *Proceedings of the 4th International Conference on Machine Learning and Data Mining in Pattern Recognition, MLDM'05*, Leipzig, Germany, LNCS (3587), pp 100-109, Springer.

Costa, G., Manco, G., Ortale, R. & Tagarelli, A. (2004). A Tree-Based Approach to Clustering XML Documents by Structure. In *Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases, PKDD'04*, Pisa, Italy, LNAI (3202), pp 137–148, Springer.

Celeux, G., Diday, E., Govaert, G., Lechevallier, Y., and Ralambondrainy, H. (1989). *Classification Automatique des Données, Environnement statistique et informatique*. Dunod informatique, Bordas, Paris.

Chi, Y., Yang, Y., Xia, Y. & Muntz, R. R. (2004, May). CMTreeMiner: Mining both closed and maximal frequent subtrees. In *Proceedings of the 8th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining, PAKDD'04*, Sydney, LNCS (3056), Springer.

Dalamagas, T., Cheng, T., Winkel, K.-J. & Sellis, T. K. (2004, May).Clustering In *Proceedings of the 3rd Helenic Conferenceon Methods and Applications of Artificial Intelligence*, *SETN'04*, Samos, Greece, LNCS (3025), pp 112–121, Springer.

Denoyer, L., Vittaut, J.-N., Gallinari, P., Brunesseaux, S. & Brunesseaux, S. (2003). Structured Multimedia Document Classification. In *Proceedings of the ACM Symposium on Document Engineering*, Grenoble, France, pp.153-160, ACM.

Denoyer, L. & Gallinari, P. (2004). Bayesian Network Model for Semi-Structured Document Classification. Information Processing and Management. In *Information Processing and Management: an International Journal, Special issue: Bayesian networks and information retrieval*, Vol. 40 (5), pp 807 – 827.

Denoyer, L., Wisniewski, G., & Gallinari, P. (2004, July). Document structure matching for heterogeneous corpora. In *Workshop on XML and Information Retrieval*, *SIGIR'04,* Sheffield.

Doucet, A. & Ahonen-Myka, H. (2002, December). Naïve clustering of a large XML document collection. In *Proceedings of the 1st ERCIM Workshop of the Initiative for the Evaluation of XML Retrieval, INEX'02*, pp. 81–88, Schloss Dagsuhl, Germany.

Flesca, S., Manco, G., Masciari, E., Pontieri, L. & Pugliese, A. (2002). Detecting structural similarities between xml documents. In *Proceedings of the 5th International Workshop on the Web and Databases, WebDB'02*, Madison.

Inokuchi, A., Washio, T. & Motoda, H.. (2000) An Apriori-based Algorithm for Mining Frequent Substructures from Graph Data. In *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*. LNCS (1910) pp. 13 – 23.

Ji, Z., Wei, W., Han, L., Sheng, Z. (2005) X-warehouse: building query pattern-driven data. In *International World Wide Web Conference (Special interest tracks and posters).* pp. 896-897

Kilpeläinen, P. (1992). *Tree Matching Problems with Applications to Structured Text Databases*. PhD thesis, University of Helsinki, November. TR A-1992-6.

Lian, W., Cheung, D. W., Mamoulis, N. & Yiu, S.-M. (2004). An Efficient and Scalable Algorithm for Clustering XML Documents by Structure. In *IEEE Transactions and Knowledge Data Engineering,* 16(1): 82-96.

Liu, J., Wang, J. T. L., Hsu, W. & Herbert, K. G.(2004). XML Clustering by Principal Component Analysis. In *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence, ICTAI'04*, pp 658–662.

Nierman, A. & Jagadish, H.V.(2002). Evaluating Structural Similarity in XML Documents. In *Proceedings of the 5th International Workshop on the Web and Databases, WebDB'02*, Madison.

Nijssen, S. & Kok, J.N. (2003). Efficient discovery of frequent unordered trees. In *Proceedings of the first International Workshop on Mining Graphs, Trees and Sequences, MGTS'03.*

Plotkin, G. (1970) A note on inductive generalisation. In *Machine Intelligence*, 5:153-163.

Quinlan, R. (1998). *Data mining tools see5 and c5.0.* Technical Report, RuleQuest Research.

Termier, A., Rousset, M.-C., & Sebag, M. (2002). Treefinder: a first step towards xml data mining. In *Proceedings of the IEEE International Conference on Data Mining, ICDM'02*, Japan, pp. 450-457, IEEE Computer Society.

Termier, A., Rousset, M.-C., & Sebag, M. (2004). DRYADE: a new approach for discovering closed frequent trees in heterogeneous tree databases. In *Proceedings of the* 4th IEEE International Conference on Data Mining, ICDM'04*,* Brighton, UK , pp. 543–546, IEEE Computer Society.

Termier, A., Rousset, M.-C., Sebag, M., Ohara, K., Washio, T., & Motoda, H. (2005). Efficient mining of high branching factor attribute trees. In *Proceedings of the 5th IEEE International Conference on Data Mining, ICDM' 05*, Houston, U.S.A, IEEE Press.

Uno, T., Kiyomiet, M. & Arimura, H. (2004, November). LCM v2.0: Efficient mining algorithms for frequent/closed/maximal itemsets. In *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, FIMI'04,* Brighton, UK.

Vercoustre, A.-M., FEGAS, M., Gul, S. & Lechevallier, Y. (2006). A flexible structured-based representation for XML document mining. In *Proceedings of the 4th International*

Workshop of the Initiative for the Evaluation of XML Retrieval, INEX'05, Schloss Dagstuhl, Germany, LNCS (3977), Springer.

Yi, J. & Sundaresan, N. (2000). A classifier for semi-structured documents. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD'00*, pp 340-344, New York, NY, USA, ACM Press.

Yoon, J.P., V. Raghavan, Chakilam, V. & Kerschberg, L. (2001). BitCube: A Three-Dimensional Bitmap Indexing for XML Documents. In *Journal of Intelligent Information Systems*, 17(2-3): 241-254.

Zaki, M.J. & Aggarwal, C. (2003). XRules: An effective structural classifier for XML data. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp 316-325, 2003.

Zang, K. & Shasha, D. (1989). Simple fast algorithms for the editing distance between trees and related problems. In *SIAM Journal of Computing*, 18:1245-1262.