Hierarchical Clustering of XML Documents Focused on Structural Components

Abstract

Clustering XML documents by structure is the task of grouping them by common structural components. Hitherto, this has been accomplished by looking at the occurrence of one preestablished type of structural components in the structures of the XML documents. However, the a-priori chosen structural components may not be the most appropriate for effective clustering. Moreover, it is likely the resulting clusters exhibit a certain extent of inner structural inhomogeneity, because of uncaught differences in the structures of the XML documents, due to further neglected forms of structural components.

To overcome these limitations, a new hierarchical approach is proposed, that allows to consider (if necessary) multiple forms of structural components to isolate structurally-homogeneous clusters of XML documents. At each level of the resulting hierarchy, clusters are divided by considering some type of structural components (unaddressed at the preceding levels), that still differentiate the structures of the XML documents.

In addition, each cluster in the hierarchy is summarized through a novel technique, that provides a clear and differentiated understanding of its structural properties.

A comparative evaluation over both real and synthetic XML data proves that the devised approach outperforms established competitors in effectiveness and scalability. Cluster summarization is also shown to be very representative.

Keywords: Data mining; Semi-structured Data and XML; XML clustering; XML transactional representation; XML cluster representative.

1. Introduction

XML is a popular model for data representation, in which two main types of information coexist, i.e., pure content and logical structure. The latter is a valuable support for better information discovery and management, being helpful to explain the nested content.

Conventional approaches to information handling are not meant for exploiting the structural information of XML data [20, 45], being either devoted to the management of highly structured data, such as relational databases, or too focused on the textual nature of the data, such as in the case of information retrieval techniques. From this perspective, XML data is a challenging research domain [16], that calls for suitable methods for information handling [3, 14].

The analysis of the structural information enables more effective processing of XML data [13, 14, 18, 34], since it allows both to understand the spectrum of queries answerable by the available XML documents (i.e. the type of information as well as its organization) and to identify XML documents with similar structures as sources of somehow related contents.

In particular, clustering XML documents by their structural features is useful in several applicative contexts. For example, the detection of structural similarities among documents can help to recognize different sources providing the same kind of information [5]. Also, it can support the extraction of (schema or DTD) structures from a collection of XML documents [22, 35], by enabling the identification of more accurate structures from structurally-homogeneous subsets of the original collection. Yet, the structural analysis of Web sites benefits from the identification of similar documents, conforming to a particular structure, which can serve as the input for wrappers working on structurally similar Web pages [4, 21]. Additionally, the re-organization of the XML documents on the basis of their structure is crucial for efficient storage as well as XML query formulation and optimization. Indeed, if the XML documents are stored using the relational database technology [15, 19, 42], their structural clustering attenuates the resulting fragmentation and, thus, reduces the number of join operations required to retrieve information from the fragmented data [26]. If instead the XML documents are natively stored as semi-structured documents [27, 30, 32, 38], their separation into structurally-homogeneous clusters helps in devising indexing techniques for such documents, thus improving the construction of query plans [17].

Grouping structurally similar XML documents is problematic for the following reasons.

Foremost, XML documents can share various forms of common structural components (ranging from simple nodes and edges, to paths [17], subtrees [25], s-graphs [26] and summaries [26]) and, generally, there is no prior knowledge about the most appropriate one to be considered for effective clustering. This highlights the need for setting the task of clustering XML documents by structure in a more general framework, that allows both to accommodate disparate types of structural components and to assess the impact of their discriminative power on clustering effectiveness.

The choice of one type of structural components should fit the structural peculiarities of the available XML documents, since it influences both the effectiveness and efficiency of clustering. This latter aspect is unexplored in previous works, such as [16, 17, 25, 26], that instead exploit various preestablished forms of structural components. We believe that this is a major limitation, since the chosen structural components do not necessarily accord with the structures of the XML data across the various applicative domains. In such cases, valuable relationships of structural similarity among the XML documents can be missed or underestimated, with a consequent degrade of clustering effectiveness.

In addition, focusing only on one form of structural components may not suffice to properly separate the available XML documents. Therein, a careful investigation of the resulting clusters is likely to reveal an extent of intra-cluster inhomogeneity, that may be due to some uncaught differences in the structures of the XML documents in the same clusters, ascribable to further unconsidered forms of structural components. This is highly undesirable in the foresaid applicative settings.

To address the highlighted issues, our contribution is threefold.

We explore the potential of hierarchical clustering as a framework that allows to consider, if necessary, various forms of structural components, with which to form a hierarchy of nested and progressively-purer clusters. Precisely, a hierarchical clustering approach is designed, in which clusters explain, at each level of the hierarchy, how the XML documents can be divided with respect to certain structural components (of the type considered at that level), that differentiate their structures. The explanation is refined at the next level, where another type of structural components is used to further divide the individual clusters from the above level into child clusters, that reveal meaningful and previously uncaught structural differences.

The devised clustering scheme accommodates all types of tree-like forms of structural components, and allows the user to specify the most appropriate one for the separation of the XML documents at each level of the cluster hierarchy.

Also, in order to gain an understanding into the structural properties of the XML documents within the individual clusters in the hierarchy, a new summarization method is developed that associates each cluster with a set of highly representative substructures. These provide a clear and differentiated understanding of the structural information within a cluster, in terms of the structural components addressed at the level of that cluster in the hierarchy.

A comparative evaluation over both real and artificial XML data reveals that the proposed approach outperforms established competitors in effectiveness and scalability. Cluster summarization is also shown to be very representative.

The paper proceeds as follows. Section 2 introduces notation and preliminaries. Section 3 covers the clustering approach. Section 4 discusses cluster summarization. Section 5 presents an intensive experimental evaluation. Section 6 overviews some related works from the literature. Finally, section 7 concludes and highlights major directions of future research.

2. Preliminaries

The notation used throughout the paper as well as some basic concepts are introduced below. The structure of XML documents without references can be modeled in terms of *rooted ordered labeled trees*, that represent the hierarchical relationships among the document elements (i.e., nodes).

Definition 2.1. ***XML Tree***. *An* XML tree *is a rooted, labeled, ordered tree, represented as a tuple* $\mathbf{t} = (r_{\mathbf{t}}, \mathbf{V}_{\mathbf{t}}, \mathbf{E}_{\mathbf{t}}, \lambda_{\mathbf{t}})$, *whose individual components have the following meaning:*

- $\mathbf{V_t}$ *is a set of nodes;*
- $\mathbf{E_t} \subseteq \mathbf{V_t} \times \mathbf{V_t}$ *is a set of edges, catching the parent-child relationships between nodes of* $\mathbf{t}$*;*
- $r_\mathbf{t} \in \mathbf{V_t}$ *is the root node of* $\mathbf{t}$*, i.e. the only node with no entering edges;*
- Σ *is an alphabet of node tags (i.e., labels);*
- $\lambda_t : \mathbf{V_t} \mapsto \Sigma$ *is a node labeling function.*

In the above definition, the elements of XML documents and their attributes are not distinguished: both are mapped to nodes in the corresponding XML-tree representation.

Let n_i and n_j be two nodes from $\mathbf{V_t}$. n_i is the parent of n_j (and, dually, n_j is a child of n_i), if $(n_i, n_j) \in \mathbf{E_t}$. This type of parent-child hierarchical relationship is represented as $n_i \rightarrow n_j$. Instead, if there is a path from n_i to n_j of any positive length p (representing intermediate edges), n_i is an ancestor of n_j, whereas n_j is a descendant of n_i. The ancestor-descendant hierarchical relationship is indicated as $n_i \xrightarrow{p} n_j$: clearly, if $p = 1$, the ancestor-descendant relationship reduces to the parent-child relationship $n_i \rightarrow n_j$. The set of all paths from $r_\mathbf{t}$ to any node n in $\mathbf{V_t}$ is denoted as *paths*($\mathbf{t}$), i.e., $\mathit{paths}(\mathbf{t}) = \{r_\mathbf{t} \xrightarrow{p} n | r_\mathbf{t}, n \in \mathbf{V_t}, p \geq 1\}$.

Nodes in $\mathbf{V_t}$ divide into two disjoint subsets: the set $\mathbf{L_t}$ of *leaves* and the set $\mathbf{V_t} - \mathbf{L_t}$ of *inner nodes*. An inner node has at least one child. There is a (predefined) left-to-right ordering among the siblings of each inner node [1]. A leaf is instead a node with no children. The height of $\mathbf{t}$, denoted as *height*($\mathbf{t}$), is the maximum number of intermediate edges between $r_\mathbf{t}$ and any leaf from $\mathbf{L_t}$, i.e., $\mathit{height}(\mathbf{t}) = max_p\{r_\mathbf{t} \xrightarrow{p} l | l \in \mathbf{L_t}\}$.

All nodes from $\mathbf{V_t}$ can be numbered according to their position in the preorder depth-first traversal of $\mathbf{t}$. Assume that function $\mathit{num}_\mathbf{t} : \mathbf{V_t} \mapsto \mathbf{N}$ maps nodes to their numbering and that m_i and m_j are two sibling nodes from $\mathbf{V_t}$. Notation $\mathit{num}_\mathbf{t}(m_i) < \mathit{num}_\mathbf{t}(m_j)$ indicates that m_i precedes m_j in the (predefined) sibling ordering.

Tree-like structures are also used to represent generic structural patterns occurring across a collection of XML trees (such as individual nodes, edges as well as paths).

Definition 2.2. ***Substructure***. *Let* $\mathbf{t}$ *and* $\mathbf{s}$ *be two XML trees.* $\mathbf{s}$ *is a* substructure *of* $\mathbf{t}$*, if there exists a total function* $\varphi : \mathbf{V_s} \rightarrow \mathbf{V_t}$*, that satisfies the following conditions for each* $n, n_i, n_j \in \mathbf{V_s}$*:*

- $(n_i, n_j) \in \mathbf{E_s}$ *iff* $\varphi(n_i) \xrightarrow{p} \varphi(n_j)$ *in* $\mathbf{t}$ *with* $p \geq 1$*;*
- $\mathit{num}_\mathbf{s}(n_i) < \mathit{num}_\mathbf{s}(n_j)$ *iff* $\mathit{num}_\mathbf{t}(\varphi(n_i)) < \mathit{num}_\mathbf{t}(\varphi(n_j))$*;*

- $\lambda_{\mathbf{s}}(n) = \lambda_{\mathbf{t}}\left[\varphi(n)\right]$.

The mapping φ preserves node labels and hierarchical relationships. In this latter regard, depending on the value of p, two definitions of substructures can be distinguished. In the simplest case $p = 1$ and a substructure $\mathbf{s}$ is simply an *induced* tree pattern that matches a contiguous portion of $\mathbf{t}$, since φ maps the parent-child edges of $\mathbf{s}$ onto parent-child edges of t. This is indicated as $\mathbf{s} \sqsubseteq \mathbf{t}$. A more general definition follows when $p \geq 1$ [48]. In such a case, $\mathbf{s}$ matches not necessarily contiguous portions of $\mathbf{t}$, since φ summarizes hierarchical relationships by mapping parent-child edges of $\mathbf{s}$ into either parent-child or ancestor-descendant edges of $\mathbf{t}$. This is denoted as $\mathbf{s} \preceq \mathbf{t}$ and $\mathbf{s}$ is also said to be an *embedded* tree pattern of $\mathbf{t}$. The summarization method in sec. 4.1 subsumes a cluster of XML trees with a set of frequent (and representative) embedded substructures. Many of these substructures would not be found as induced substructures, since the embedded substructures are intermixed with unfrequent (and unrepresentative) substructures in the XML trees.

Hereafter, the notions of substructure, (structural) component and tree pattern are used as synonyms.

Clustering by structure aims to divide a collection $\mathcal{D} = \{\mathbf{t}_1, \ldots, \mathbf{t}_N\}$ of N XML trees to form a partition $\mathcal{P} = \{\mathcal{C}_1, \ldots, \mathcal{C}_K\}$ of nonempty clusters such that $\mathcal{C}_i \subseteq \mathcal{D}$ and $\mathcal{C}_i \cap \mathcal{C}_j = \emptyset$ for all $i, j = 1, \ldots, K$ with $i \neq j$. The clustering process generally attempts to maximize the degree of structural homogeneity exhibited by the XML trees in the same cluster and to minimize the extent of structural homogeneity between XML trees within distinct clusters.

In this paper, we develop a clustering approach that produces a hierarchy of nested clusters. The approach differs from conventional hierarchical methods in two respects.

First, it allows to account for various forms of structural components that differentiate the available XML trees. Such structural patterns may be suggested by domain analysts, fixed classification criteria, or in-depth customized analysis.

Second, it is a multi-stage divisive method. At stage i, the clusters sited at level i of the resulting hierarchy are formed by separating the individual clusters from level $i-1$ (i.e., obtained at stage $i-1$) with respect to some form of structural components, that has not yet been considered up to stage $i-1$. This enforces a progressively increasing degree of intra-cluster structural-homogeneity along the paths from root to the leaves of the cluster hierarchy.

All types of tree-like components can be handled for clustering at each stage. Notation $\mathcal{T}^{(i)}$ denotes the collection of components addressed at stage i. Example definitions of $\mathcal{T}^{(i)}$ include:

- The selection of one-node substructures modeling the individual nodes in the XML trees, i.e.,

$$\mathcal{T}^{(i)} \quad = \quad \{\mathbf{s} : \mathbf{V_s} = \{r_{\mathbf{s}}\}, \exists \mathbf{t} \in \mathcal{D}, \mathbf{s} \sqsubseteq \mathbf{t}\}$$

In such a case, the generic substructure $\mathbf{s}$ consists only of its root $r_{\mathbf{s}}$, that matches some corresponding node of an XML tree $\mathbf{t}$ in $\mathcal{D}$.

- The selection of one-edge substructures modeling the parent-child relationships in the XML trees, i.e.,

$$\mathcal{T}^{(i)} = \{\mathbf{s} : \mathbf{V_s} = \{r_{\mathbf{s}}, n\}, \mathbf{E_s} = \{(r_{\mathbf{s}}, n)\}, \exists \mathbf{t} \in \mathcal{D}, \mathbf{s} \sqsubseteq \mathbf{t}\}$$

  Here, the individual substructure $\mathbf{s}$ consists only of one edge $(r_{\mathbf{s}}, n)$, that matches some corresponding parent-child edge of an XML tree $\mathbf{t}$ in $\mathcal{D}$.

- The selection of one-path substructures modeling the distinct root-to-node paths in the XML trees, i.e.,

$$\mathcal{T}^{(i)} = \{\mathbf{s} : \ |paths(\mathbf{s})| = 1, \exists \mathbf{t} \in \mathcal{D}, \mathbf{s} \sqsubseteq \mathbf{t}\}$$

  Each $\mathbf{s}$ in the above set $\mathcal{T}^{(i)}$ is hence linear, i.e., consists of only one path matching some (root-to-node or root-to-leaf) path of an XML tree $\mathbf{t}$ in $\mathcal{D}$.

Interestingly, more sophisticated definitions of $\mathcal{T}^{(i)}$ may involve induced or embedded subtrees.

The components chosen at any clustering stage i enable the projection of the XML trees into a high-dimensional space, wherein the occurrence of the individual substructures within each XML tree is explicitly represented. More precisely, the XML trees can be modeled as transactions over a feature space $\mathcal{S}^{(i)} = \{\mathcal{F}_{\mathbf{s}} | \mathbf{s} \in \mathcal{T}^{(i)}\}$. Here, the generic feature $\mathcal{F}_{\mathbf{s}}$ is a boolean attribute, that indicates the presence/absence of the related component $\mathbf{s}$ of $\mathcal{T}^{(i)}$ in the individual XML trees.

Let $\mathbf{x}^{(\mathbf{t})}$ be the high-dimensional transactional representation over $\mathcal{S}^{(i)}$ of an XML tree $\mathbf{t}$. The value of each attribute $\mathcal{F}_{\mathbf{s}}$ in the context of $\mathbf{x}^{(\mathbf{t})}$ is true if $\mathbf{s}$ is a substructure of $\mathbf{t}$, otherwise it is false. Hence, $\mathbf{x}^{(\mathbf{t})}$ can be modeled as a proper subset of $\mathcal{S}^{(i)}$, namely $\mathbf{x}^{(\mathbf{t})} = \{\mathcal{F}_{\mathbf{s}} \in \mathcal{S}^{(i)} | \mathbf{s} \sqsubseteq \mathbf{t}\}$, with the meaning that the features explicitly present in $\mathbf{x}^{(\mathbf{t})}$ take value true, whereas the others assume value false.

At each stage i, the set $\mathbf{D}^{(i)} = \{\mathbf{x}^{(\mathbf{t})} | \mathbf{t} \in \mathcal{D}\}$ comprises all transactions over $\mathcal{S}^{(i)}$, that correspond to XML trees of $\mathcal{D}$. For convenience, we introduce a mapping $t(\cdot) : 2^{\mathbf{D}^{(i)}} \rightarrow 2^{\mathcal{D}}$, that projects a set $\mathcal{C}$ of transactions over $\mathcal{S}^{(i)}$ to their corresponding XML trees in $\mathcal{D}$, i.e., $t(\mathcal{C}) = \{\mathbf{t} | \mathbf{x}^{(\mathbf{t})} \in \mathcal{C}\}$.

The devised hierarchical approach to clustering XML documents by structure benefits from the transactional representation. Indeed, the cost for testing the presence of the selected components within the transactions related to the XML trees is, at each stage, independent of the structural complexity of the same components. Nonetheless, at each stage, the transactional representation involves the non-trivial discovery of meaningful clusters in large-scale databases of high-dimensional transactions.

From this perspective, our approach reformulates the original problem of grouping $\mathcal{D}$ by structure as that of progressively finding structurally-purer clusters in the transactional representations of $\mathcal{D}$ over various types of structural components across multiple stages.

This is accomplished through a hierarchical clustering process described in the next section.

3. Hierarchical Clustering of XML Trees

The scheme of the hierarchical clustering process is sketched in fig. 1. The Generate-Hierarchy procedure preliminarily requires that the end user incorporates (at line 1) valuable domain knowledge and application semantics into the hierarchical clustering process. This is accomplished by establishing the most appropriate set of structural components $\mathcal{T}^{(i)}$ to address at each stage i of clustering (i.e. at each level of the cluster hierarchy) as well as the overall number m of clustering stages (i.e., levels in the cluster hierarchy).

At each clustering stage i, $\mathcal{P}$ represents the partition containing all clusters of XML trees sited at level $i-1$ of the cluster hierarchy. Initially, when $i=1$, $\mathcal{P}$ includes a single cluster, which coincides with the whole data set $\mathcal{D}$ of XML trees (line 4). Instead, $\mathcal{P}^{(i)}$ is the partition produced by Generate-Clusters at stage i (line 14). Notice that $\mathcal{P}$ includes clusters of XML trees, whereas $\mathcal{P}^{(i)}$ contain clusters of transactions over the feature space $\mathcal{S}^{(i)}$. At the end of stage i, $\mathcal{P}^{(i)}$ is mapped to $\mathcal{P}$(at line 18), which thus become the last level of the cluster hierarchy to be partitioned in the subsequent stage $i+1$.

For the sake of clarity, the above distinction between clusters of XML trees and clusters of transactions corresponding to XML trees is reflected in the pseudo code of fig. 1 by an appropriate notation: $\mathcal{C}$ and $\mathbf{C}$ indicate clusters, respectively, of the former and the latter type.

The body of each clustering stage i (lines 6-19) consists of two phases: separation (lines 6-14) and summarization (lines 15-17).

At the generic stage i, each cluster $\mathcal{C}$ within $\mathcal{P}$ is divided by means of the Generate-Clusters procedure, which is covered in sec. 3.1. Here, it suffices to anticipate that Generate-Clusters partitions (at line 9) a cluster $\mathcal{C}$ of XML trees into an appropriate number of child clusters, which contain transactions over the feature space $\mathcal{S}^{(i)}$. Together, these child clusters form the partition $\mathcal{R}$ of the foresaid cluster $\mathcal{C}$. At this point, each child cluster $\mathbf{C}$ in $\mathcal{R}$ is associated (lines 10-12) with its siblings $\overline{\mathbf{C}} = \mathcal{R} - \mathbf{C}$ (for the purpose of cluster summarization) and $\mathcal{R}$ is then added (at line 13) to the partition $\mathcal{P}^{(i)}$ that is being constructed. The separation of clusters in $\mathcal{P}$ is reiterated (between lines 6-14) until each cluster in $\mathcal{P}$ is individually partitioned.

Summarization takes place after cluster separation. Each cluster $\mathbf{C}$ within $\mathcal{P}^{(i)}$ is summarized (lines 15-17) trough MineRep. The latter is a procedure, covered in sec. 4, that associates $\mathbf{C}$ with a set *Rep*($\mathbf{C}$) of representative substructures, subsuming the structural information within $\mathbf{C}$.

To this point, the partition $\mathcal{P}$ (at line 18) includes all clusters of XML trees sited at level i of the cluster hierarchy and Generate-Hierarchy proceeds

```
GENERATE-HIERARCHY($\mathcal{D}$)
  Input: a set $\mathcal{D} = \{\mathbf{t}_1, \ldots, \mathbf{t}_N\}$ of XML trees;
  Output: a set $\cup_i \mathcal{P}^{(i)}$ of multiple cluster partitions;
  1: let $\mathcal{T}^{(i)}$ be the set of structural components at stage $i = 1, \ldots, m$;
  2: let $\mathcal{S}^{(i)} \leftarrow \{\mathcal{F}_\mathbf{s} | \mathbf{s} \in \mathcal{T}^{(i)}\}$ be the feature space at stage $i = 1, \ldots, m$;
  3: let $i \leftarrow 1$;
  4: let $\mathcal{P} \leftarrow \{\mathcal{D}\}$;
  5: while $i \leq m$ do
  6:   while $\mathcal{P} \neq \emptyset$ do
  7:     let $\mathcal{C}$ be a cluster in $\mathcal{P}$;
  8:     $\mathcal{P} \leftarrow \mathcal{P} - \mathcal{C}$;
  9:     $\mathcal{R} \leftarrow$ GENERATE-CLUSTERS($\mathcal{C}, \mathcal{S}^{(i)}$);
  10:      for each $\mathbf{C} \in \mathcal{R}$ do
  11:        let $\overline{\mathbf{C}} \leftarrow \mathcal{R} - \{\mathbf{C}\}$ be the set of siblings of $\mathbf{C}$;
  12:      end for
  13:      $\mathcal{P}^{(i)} \leftarrow \mathcal{P}^{(i)} \cup \mathcal{R}$;
  14:    end while
  15:    for each $\mathbf{C} \in \mathcal{P}^{(i)}$ do
  16:      $Rep(\mathbf{C}) \leftarrow$ MINEREP($\mathbf{C}, \overline{\mathbf{C}}, \alpha$);
  17:    end for
  18:    $\mathcal{P} \leftarrow \cup_{\mathbf{C} \in \mathcal{P}^{(i)}} t(\mathbf{C})$;
  19:    $i \leftarrow i + 1$;
  20:  end while
  21:  RETURN $\cup_i \mathcal{P}^{(i)}$;
```

Figure 1: The hierarchical clustering process

to partition them at the subsequent level $i+1$ with respect to the structural components $\mathcal{T}^{(i+1)}$.

The choice of a distinct set of structural components at each stage guarantees a progressively increasing degree of structural homogeneity. This is due to the fact the XML trees corresponding to the transactions within the generic cluster of $\mathcal{P}^{(i)}$ (that are already homogeneous according to the previously considered sets of structural components $\mathcal{T}^{(j)}$ with $j = 1, \ldots i$) can still be separated by isolating groups of such XML trees, which are strongly discriminated by meaningful co-occurrences of the (previously unconsidered) structural patterns in $\mathcal{T}^{(i+1)}$. Obviously, this also implies a significant differentiation in the representatives of clusters at different stages. Indeed, at each distinct stage, representatives provide a summarization of the tree structures within the corresponding clusters in terms of (a combination of) the structural components considered a that particular stage. Hence, the representative of a child cluster highlights local patterns of structural homogeneity, that are not caught by the representative of the parent cluster.

3.1. Cluster generation

The basic idea behind cluster separation at each stage consists in projecting an input set $\mathcal{D}$ of XML trees into a high-dimensional feature space $\mathcal{S}$, in which to isolate homogeneous groups of transactions sharing discriminatory co-occurrences of structural features.

Finding clusters in the high-dimensional feature space $\mathcal{S}$ is problematic for various reasons [8]. Primarily, transactions tend to form different clusters on distinct subsets of features, which penalizes the effectiveness of clustering and exacerbates its time requirements. Secondarily, poor scalability with both the size and the dimensionality of transactions is usually a major limitation. Yet, an underestimation (resp. overestimation) of the number of child clusters to isolate in the input set $\mathcal{D}$ misses (resp. uncovers) actual (resp. artificial) groups.

To fit the peculiarities of the transactional setting, the XML trees within the input set $\mathcal{D}$ are separated through the Generate-Clusters algorithm proposed in [8]. The latter is an effective and parameter-free technique for transactional clustering, that automatically partitions $\mathcal{D}$ into an appropriate number of child clusters.

The fundamentals of Generate-Clusters are reviewed below. Instead, a discussion on the convergence of Generate-Clusters along with a comparative analysis of its empirical behavior against a wide variety of established competitors can be found in [8].

The general scheme of the Generate-Clusters algorithm is reported in fig. 2. Generate-Clusters initially maps (lines L1- L2) the input set $\mathcal{D}$ of XML trees to a space of clustering features in $\mathcal{S}$. This yields the transactional representation $\mathbf{D}$. The algorithm starts with a partition $\mathcal{P}$ containing a single cluster corresponding to the whole transactional dataset $\mathbf{D}$ (line L3). The core of the algorithm is the body of the loop between lines L4-L17. Within the loop, an attempt to generate a new cluster is performed by (*i*) choosing a candidate node (corresponding to a cluster with low quality) to split (line L6); (*ii*) splitting the candidate cluster into two child clusters (line L7); and (*iii*) evaluating whether the splitting allows a new partition exhibit better quality than the original partition (lines L8-L15). If this is the case, the loop can be stopped (line L12) and the partition is updated, by replacing the candidate cluster with the new child clusters (line L10). Viceversa, child clusters are discarded and a new candidate cluster is considered for splitting.

The Partition-Cluster procedure at line L7 iteratively evaluates, for each transaction $\mathbf{x^{(t)}} \in \mathcal{C}_i \cup \mathcal{C}$, whether a membership reassignment improves the degree of structural homogeneity of the two clusters. The contribution of $\mathbf{x^{(t)}}$ to structural homogeneity is evaluated in two cases: both in the case that $\mathbf{x^{(t)}}$ is maintained in its original cluster of membership and in the case that $\mathbf{x^{(t)}}$ is moved to the other cluster. If moving $\mathbf{x^{(t)}}$ causes an improvement in the structural homogeneity, then the swap is accepted.

The local quality *Quality*$(\mathcal{C})$ of a cluster $\mathcal{C}$ is a key component of Generate-Clusters and measures the degree of structural homogeneity within $\mathcal{C}$. More precisely, *Quality*$(\mathcal{C})$ is defined as the gain in feature strength with respect to the whole transactional dataset $\mathbf{D}$, i.e.,

$$\mathit{Quality}(\mathcal{C}) = \Pr(\mathcal{C}) \sum_{\mathcal{F} \in \mathcal{S}_\mathcal{C}} \left[\Pr(\mathcal{F}|\mathcal{C})^2 - \Pr(\mathcal{F}|\mathbf{D})^2\right]$$

where $\Pr(\mathcal{F}|\mathcal{C})^2$ corresponds to the relative strength of $\mathcal{F}$ within $\mathcal{C}$, whereas

$\Pr(\mathcal{C})$ represents the relative strength of $\mathcal{C}$. These two factors work in contraposition: singleton clusters exhibit strong features in a sparse region, whereas highly populated clusters exhibit weaker features in a dense region. The above formula finds an interpretation in terms of subspace clustering. Features exhibiting a high occurrence frequency with respect to the occurrence frequency in the whole dataset $\mathbf{D}$, define a subset of relevant features, as opposed to low-occurrence features which are indeed irrelevant for the purpose of clustering. Thus, clusters exhibit high quality whenever a subset of relevant features occurs, whose frequency is significantly higher than in the whole dataset $\mathbf{D}$.

Differently from the PARTITION-CLUSTER procedure, where the improvement in quality is attempted locally to a cluster, the STABILIZE-CLUSTERS procedure tries to increase the global partition quality $\mathit{Quality}(\mathcal{P})$. This is accomplished by finding, for each transaction, the most suitable cluster among the ones available in the partition. The quality $\mathit{Quality}(\mathcal{P})$ of a partition $\mathcal{P}$ is meant to measure both the homogeneity of clusters and their compactness. Viewed in this respect, partition quality is defined as follows

$$\mathit{Quality}(\mathcal{P}) = \sum_{\mathcal{C}\in\mathcal{P}} \Pr(\mathcal{C})\,\mathit{Quality}(\mathcal{C})$$

Notice that the component $\mathit{Quality}(\mathcal{C})$ is already proportional to the contribution $\Pr(\mathcal{C})$. As a result, in the overall partition quality, the contribution of each cluster is weighted by $\Pr(\mathcal{C})^2$. This weighting has a major effect in the GENERATE-CLUSTERS procedure: splitting in very small clusters is penalized. Indeed, the generated clusters are added to the partition only if their contribution is really worth.

```
GENERATE-CLUSTERS($\mathcal{D}, \mathcal{S}$)
  Input: A set $\mathcal{D} = \{\mathbf{t}_1, \ldots, \mathbf{t}_N\}$ of XML trees;
         a set of clustering features $\mathcal{S}$;
  Output: A partition $\mathcal{P} = \{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ of clusters of transactions
          corresponding to XML trees;
  L1:  let $\mathbf{x}^{(\mathbf{t}_i)} \leftarrow \{\mathcal{F}_\mathbf{s} \in \mathcal{S} | \mathbf{s} \sqsubseteq \mathbf{t}_i\}$ for each $i = 1, \ldots, N$;
  L2:  let $\mathbf{D} \leftarrow \{\mathbf{x}^{(\mathbf{t}_i)} \subseteq \mathcal{S} | \mathbf{t}_i \in \mathcal{D}\}$;
  L3:  let $\mathcal{P} \leftarrow \{\mathbf{D}\}$;
  L4:  repeat
  L5:    Generate a new cluster $\mathcal{C}$ of transactions, initially empty;
  L6:    for each cluster $\mathcal{C}_i \in \mathcal{P}$ do
  L7:      PARTITION-CLUSTER($\mathcal{C}_i, \mathcal{C}$);
  L8:      $\mathcal{P}' \leftarrow \mathcal{P} \cup \{\mathcal{C}\}$;
  L9:      if $\mathit{Quality}(\mathcal{P}) < \mathit{Quality}(\mathcal{P}')$ then
  L10:       $\mathcal{P} \leftarrow \mathcal{P}'$;
  L11:       STABILIZE-CLUSTERS($\mathcal{P}$);
  L12:       break
  L13:     else
  L14:       Restore all $\mathbf{x}^{(\mathbf{t}_j)} \in \mathcal{C}$ into $\mathcal{C}_i$;
  L15:     end if
  L16:   end for
  L17: until no further cluster $\mathcal{C}$ can be generated
  L18: RETURN $\mathcal{P}$;
```

Figure 2: The GENERATE-CLUSTERS scheme

4. Cluster Summarization

The representative of a cluster of XML trees is a set of highly representative XML tree patterns, that satisfy the following two conditions. Foremost, each XML tree pattern $\mathbf{s}$ must appear as a substructure of the XML trees in $\mathcal{C}$ with an occurrence frequency $\Pr(\mathbf{s}|\mathcal{C})$, that is higher than the frequency $\Pr(\mathbf{s}|\mathcal{C} \cup \overline{\mathcal{C}})$ with which $\mathbf{s}$ occurs throughout $\mathcal{C} \cup \overline{\mathcal{C}}$, where $\overline{\mathcal{C}}$ is the set of siblings of $\mathcal{C}$ in the cluster hierarchy formed by GENERATE-CLUSTERS. Also, there must be a strong degree of correlation between $\mathbf{s}$ and $\mathcal{C}$, which guarantees that the XML tree pattern is representative of certain structural properties from the cluster.

Cluster summarization relies on an Apriori-based, pattern-growth strategy. At any stage i, the latter initially considers the elementary features within $\mathcal{S}^{(i)}$ as the most basic XML tree patterns for a cluster. Such patterns are progressively combined into composite tree patterns. To avoid combinatory explosion, only two types of combined tree pattern are admitted, i.e., *parent-child* and *sibling* tree patterns.

Definition 4.1. ***Parent-child tree pattern***. *A* parent-child *tree pattern is an arrangement of two basic tree patterns, in which one of the two tree patterns is rooted at some leaf node of the other tree pattern. Let* $\mathbf{s}_i$ *and* $\mathbf{s}_j$ *be two generic tree patterns. Also, assume that* l *is some leaf node of* $\mathbf{s}_i$*. The operator* $\mathbf{s}_i \lhd_l \mathbf{s}_j$ *defines a new parent-child tree pattern* $\mathbf{s}$*, such that* $|\mathbf{V}_\mathbf{s}| = |\mathbf{V}_{\mathbf{s}_i}| + |\mathbf{V}_{\mathbf{s}_j}|$ *and* $|\mathbf{E}_\mathbf{s}| = |\mathbf{E}_{\mathbf{s}_i}| + |\mathbf{E}_{\mathbf{s}_j}| + 1$*, wherein the root* $r_{\mathbf{s}_j}$ *of* $\mathbf{s}_j$ *is a child of* l*. Formally, if* $l \in \mathbf{V}_{\mathbf{s}_i}$*,* $\mathbf{s}_i \lhd_l \mathbf{s}_j$ *defines a tree pattern* $\mathbf{s}$ *such that there exist two mappings* $\varphi_i : \mathbf{V}_{\mathbf{s}_i} \mapsto \mathbf{V}_\mathbf{s}$ *and* $\varphi_j : \mathbf{V}_{\mathbf{s}_j} \mapsto \mathbf{V}_\mathbf{s}$ *satisfying the following conditions for each* $h \in \{i, j\}$ *(wherever subscript* h *figures):*

- $\varphi_i(r_{\mathbf{s}_i}) = r_\mathbf{s}$ *(i.e.* $r_{\mathbf{s}_i}$ *matches* $r_\mathbf{s}$*)*
- $\forall n \in \mathbf{V}_{\mathbf{s}_i}$ *and* $\forall n' \in \mathbf{V}_{\mathbf{s}_j}$, $\varphi_i(n) \neq \varphi_j(n')$;
- $\forall n \in \mathbf{V}_{\mathbf{s}_h}$, $\lambda_{\mathbf{s}_h}(n) = \lambda_\mathbf{s}(\varphi_h(n))$;
- $\forall n, n' \in \mathbf{V}_{\mathbf{s}_h}$, $(n, n') \in \mathbf{E}_{\mathbf{s}_h}$ *iff* $(\varphi_h(n), \varphi_h(n')) \in \mathbf{E}_\mathbf{s}$;
- $\forall (n, n'), (n, n'') \in \mathbf{E}_{\mathbf{s}_h}$, $num_{\mathbf{s}_h}(n') < num_{\mathbf{s}_h}(n'')$ *iff* $num_\mathbf{s}(\varphi(n')) < num_\mathbf{s}(\varphi(n''))$;
- $(\varphi_i(l), \varphi_j(r_{\mathbf{s}_j})) \in \mathbf{E}_\mathbf{s}$.

Given any two tree patterns $\mathbf{s}_i$ *and* $\mathbf{s}_j$*, the set of all possible parent-child tree patterns in which the root of* $\mathbf{s}_j$ *is a child of the individual leaves of* $\mathbf{s}_i$ *is denoted as*

$$\mathbf{s}_i \lhd \mathbf{s}_j = \bigcup_{l \in \mathbf{L}_{\mathbf{s}_i}} \{\mathbf{s}_i \lhd_l \mathbf{s}_j\}$$

where $\mathbf{L}_{\mathbf{s}_i}$ *represents the set of leaves of* $\mathbf{s}_i$*.*

A parent-child tree pattern is a vertical arrangement of two component tree patterns. Instead, a sibling tree pattern follows from an horizontal arrangement of its components.

Definition 4.2. ***Sibling tree pattern***. *Given two tree patterns with a same label at their roots, a* sibling *tree pattern is a composite structure, in which the two tree patterns are merged under the same root label. Let* $\mathbf{s}_i$ *and* $\mathbf{s}_j$ *be two tree patterns such that* $\lambda_{\mathbf{s}_i}(r_{\mathbf{s}_i}) = \lambda_{\mathbf{s}_j}(r_{\mathbf{s}_j})$. *The operator* $\mathbf{s}_i \wedge \mathbf{s}_j$ *defines a sibling tree pattern* $\mathbf{s}$*, such that there exist two mappings* $\varphi_i : \mathbf{V}_{\mathbf{s}_i} \mapsto \mathbf{V}_{\mathbf{s}}$ *and* $\varphi_j : \mathbf{V}_{\mathbf{s}_j} \mapsto \mathbf{V}_{\mathbf{s}}$ *satisfying the following conditions for each* $h \in \{i, j\}$ *(wherever subscript h figures):*

- $\varphi_i(r_{\mathbf{s}_i}) = \varphi_j(r_{\mathbf{s}_j}) = r_{\mathbf{s}}$;
- $\forall n \in \mathbf{V}_{\mathbf{s}_h}$, $\lambda_{\mathbf{s}_h}(n) = \lambda_{\mathbf{s}}(\varphi_h(n))$;
- $\forall n, n' \in \mathbf{V}_{\mathbf{s}_h}$, $(n, n') \in \mathbf{E}_{\mathbf{s}_h}$ *iff* $(\varphi_h(n), \varphi_h(n')) \in \mathbf{E}_{\mathbf{s}}$;
- $\forall (n, n'), (n, n'') \in \mathbf{E}_{\mathbf{s}_h}$, $num_{\mathbf{s}_h}(n') < num_{\mathbf{s}_h}(n'')$ *iff* $num_{\mathbf{s}}(\varphi(n')) < num_{\mathbf{s}}(\varphi(n''))$ *for each* $h \in \{i, j\}$.

4.1. Mining Representative XML Tree Patterns

MineRep, in fig. 3, is an Apriori-based pattern-growth technique to mine a set of representative substructures for a cluster $\mathcal{C}$. Such substructures are obtained via progressive combinations of the elementary structures in $\mathcal{C}$.

MineRep receives three input parameters, namely the cluster $\mathcal{C}$ to be summarized, the set $\overline{\mathcal{C}}$ of all siblings of $\mathcal{C}$ (as defined at line 11 of fig. 1) and a significance threshold α. The procedure starts (at line M3) by considering a space $\mathcal{S}_{\mathcal{C}}$ of features, whose occurrence frequency in cluster $\mathcal{C}$ is higher than in the whole partition $\mathcal{C} \cup \overline{\mathcal{C}}$. These features are inherently characteristic of $\mathcal{C}$ and, according to the definition of cluster quality in sec. 3, are directly provided by Generate-Clusters, without having to be re-computed. Notably, focusing only on such features from the beginning strongly prunes the space of candidates.

The elementary structures from the feature space $\mathcal{S}_{\mathcal{C}}$ are considered (at line M4) as candidate tree patterns. Each such a candidate $\mathbf{s}$ is associated (lines M5-M7) with a bit list $\mathcal{B}(\mathbf{s})$, that keeps trace of the transactions in $\mathcal{C} \cup \overline{\mathcal{C}}$, that exhibit the corresponding feature $\mathcal{F}_{\mathbf{s}}$. Bit lists are a fundamental tool in *vertical mining* methods [41, 47, 49], that expedites frequency counting for candidate patterns. Here, this mechanism enables the fast estimation of the bit list related to any candidate (combined) tree pattern, which is henceforth obtained through the intersection (at line N4 of the Candidate-Generation sub-procedure in fig. 4) of the bit lists associated with its constituents, without costly re-scans of all the available transactions.

The generic bit list $\mathcal{B}(\mathbf{s})$, in fig. 3, is represented as a set of transactions for convenience: the transactions in $\mathcal{B}(\mathbf{s})$ exhibit $\mathcal{F}_{\mathbf{s}}$, whereas the others do not include $\mathcal{F}_{\mathbf{s}}$.

At the heart of MineRep is a loop (lines M8-M20), that distils representative tree patterns from $C^{(k)}$ and generates more-complex (parent-child and sibling) candidates via combinations of representative tree patterns. The loop halts when no more candidates can be generated.

At any generic iteration k, MineRep computes the occurrence frequencies $\Pr(\mathbf{s}|\mathcal{C})$ and $\Pr(\mathbf{s}|\overline{\mathcal{C}})$ of each candidate $\mathbf{s}$ from $C^{(k)}$ within, respectively, $\mathcal{C}$ and $\overline{\mathcal{C}}$. This is accomplished by looking at the transactions in the associated bit list $\mathcal{B}(\mathbf{s})$. The latter, by construction (at line N4 of the Candidate-Generation scheme in fig. 4), includes all those transactions in $\mathcal{C} \cup \overline{\mathcal{C}}$ that exhibit all and only the features in $\mathbf{s}$. However, at any iteration $k \geq 2$ (tested at line M10), such transactions cannot be directly exploited to compute the occurrence frequencies. This is due to the fact that $\mathbf{s}$ is not necessarily an embedded substructure of all the XML trees associated to the transactions in $\mathbf{s}$, because of the possibility that, in some of these trees, the features of $\mathbf{s}$ originate distinct structural combinations. Therefore, $\mathcal{B}(\mathbf{s})$ is inspected to identify (at line M11) the subset $\mathcal{D}(\mathbf{s})$ of transactions corresponding to XML trees, that do not actually include $\mathbf{s}$ as an embedded substructure. Therein, the test on the height of the XML trees within $\mathcal{B}(\mathbf{s})$ strongly reduces the overall number of such trees, that is actually necessary to inspect in order to check the embedded inclusion of candidate $\mathbf{s}$. $\mathcal{D}(\mathbf{s})$ is then used to rectify $\mathcal{B}(\mathbf{s})$ (at line M12), which enables the computation of $\Pr(\mathbf{s}|\mathcal{C})$ and $\Pr(\mathbf{s}|\mathcal{C} \cup \overline{\mathcal{C}})$ (at lines M14 and M15) from the bit list.

Actually, the identification of $\mathcal{D}(\mathbf{s})$ is optimized, since checking the embedded inclusion of a candidate $\mathbf{s}$ in an XML tree $\mathbf{t}$ requires time $O(|\mathbf{V}_{\mathbf{s}}||\mathbf{V}_{\mathbf{t}}|)$. Precisely, a preprocessing phase of Generate-Hierarchy in fig. 1 (which is not formalized to avoid cluttering discussion) also associates the available XML trees with revised s-graphs [26]. These are explicit representations of the parent-child and ancestor-descendant hierarchical relationships within the XML trees. Such representations are used to avoid the expensive test (at line M11) on the embedding of a substructure $\mathbf{s}$ in an XML tree $\mathbf{t}$, whenever all edges of $\mathbf{s}$ are not included in the revised s-graph associated to $\mathbf{t}$.

The representative tree patterns are distilled in $L^{(k)}$ (at line M17) from the set $C^{(k)}$ of candidates, by choosing the ones that satisfy the following two conditions. First, the occurrence frequency of each representative tree pattern $\mathbf{s}$ must be higher in $\mathcal{C}$ than in $\mathcal{C} \cup \overline{\mathcal{C}}$, i.e., it must hold that $\Pr(\mathbf{s}|\mathcal{C}) > \Pr(\mathbf{s}|\mathcal{C} \cup \overline{\mathcal{C}})$. Both $\Pr(\mathbf{s}|\mathcal{C})$ and $\Pr(\mathbf{s}|\mathcal{C} \cup \overline{\mathcal{C}})$ are computed (respectively at lines M14 and M15) after the pruning phase. Second, there must be a strong degree of correlation between $\mathbf{s}$ and $\mathcal{C}$. This is useful to establish whether the occurrence of $\mathbf{s}$ in $\mathcal{C}$ is statistically relevant and, hence, structurally representative. Statistical hypothesis testing is used for this purpose, as it is discussed in sec. 4.3. The resulting set $L^{(k)}$ of representative tree patterns provides the basic structures for the generation of candidate tree patterns at the subsequent iteration $k + 1$.

MineRep halts when $C^{(k)}$ is empty and, hence, no more representative tree patterns can be discovered. In such a case MineRep returns (line M22) all of

```
MineRep($\mathcal{C}, \overline{\mathcal{C}}, \alpha$)
 Input: a set $\mathcal{C} = \{\mathbf{x}^{(\mathbf{t}_1)}, \ldots, \mathbf{x}^{(\mathbf{t}_h)}\}$ of XML trees in transactional
     form;
     the set $\overline{\mathcal{C}}$ of siblings of $\mathcal{C}$ as defined at line 11 of fig. 1;
     a significance threshold $\alpha$;
 Output: a set $\mathcal{R}$ of representative XML structures;
 M1:  $\mathcal{R} \leftarrow \emptyset$;
 M2:  $k \leftarrow 1$;
 M3:  let $\mathcal{S}_{\mathcal{C}} \leftarrow \{\mathcal{F}_{\mathbf{s}} | \exists \mathbf{x}^{(\mathbf{t})} \in \mathcal{C}, \mathcal{F}_{\mathbf{s}} \in \mathbf{x}^{(\mathbf{t})}, \Pr(\mathcal{F}_{\mathbf{s}}|\mathcal{C}) > \Pr(\mathcal{F}_{\mathbf{s}}|\mathcal{C} \cup \overline{\mathcal{C}})\}$;
 M4:  let $C^{(k)} \leftarrow \{\mathbf{s} | \mathcal{F}_{\mathbf{s}} \in \mathcal{S}_{\mathcal{C}}\}$;
 M5:  for each $\mathbf{s} \in C^{(k)}$ do
 M6:    let $\mathcal{B}(\mathbf{s}) \leftarrow \{\mathbf{x}^{(\mathbf{t})} \in \mathcal{C} \cup \overline{\mathcal{C}} | \mathcal{F}_{\mathbf{s}} \in \mathbf{x}^{(\mathbf{t})}\}$;
 M7:  end for
 M8:  while ($C^{(k)} \neq \emptyset$) do
 M9:    for each $\mathbf{s} \in C^{(k)}$ do
 M10:     if ($k > 1$) then
 M11:       $\mathcal{D}(\mathbf{s}) \leftarrow \{\mathbf{x}^{(\mathbf{t})} \in \mathcal{B}(\mathbf{s}) | height(\mathbf{t}) < height(\mathbf{s})\} \cup \{\mathbf{x}^{(\mathbf{t})} \in$
            $\mathcal{B}(\mathbf{s}) | height(\mathbf{t}) \geq height(\mathbf{s}), \mathbf{s} \not\preceq \mathbf{t}\}$;
 M12:       $\mathcal{B}(\mathbf{s}) \leftarrow \mathcal{B}(\mathbf{s}) - \mathcal{D}(\mathbf{s})$;
 M13:     end if
 M14:     $\Pr(\mathbf{s}|\mathcal{C}) \leftarrow \frac{|\{\mathbf{x}^{(\mathbf{t})} \in \mathcal{B}(\mathbf{s}) \cap \mathcal{C}\}|}{|\mathcal{C}|}$;
 M15:     $\Pr(\mathbf{s}|\mathcal{C} \cup \overline{\mathcal{C}}) \leftarrow \frac{|\{\mathbf{x}^{(\mathbf{t})} \in \mathcal{B}(\mathbf{s})\}|}{|\mathcal{C} \cup \overline{\mathcal{C}}|}$;
 M16:   end for
 M17:   $L^{(k)} \leftarrow \{\mathbf{s} \in C^{(k)} | \Pr(\mathbf{s}|\mathcal{C}) > \Pr(\mathbf{s}|\mathcal{C} \cup \overline{\mathcal{C}}), \chi^2(\mathbf{s}, \mathcal{C}) > \tau_\alpha\}$;
 M18:   $k \leftarrow k + 1$;
 M19:   $C^{(k)} \leftarrow$ Candidate-Generation($L^{(k-1)}, \mathcal{C}$);
 M20: end while
 M21: $\mathcal{R} \leftarrow \cup_k L^{(k)}$;
 M22: return $\mathcal{R}$;
```

Figure 3: The cluster summarization procedure

the XML tree patterns, that were found in the previous steps to be strongly discriminatory of the structural properties of cluster $\mathcal{C}$. The candidate generation phase as well as the exploitation of statistical hypothesis testing for the identification of representative substructures are analyzed in the following sections.

4.2. Candidate generation

The Candidate-Generation sub-procedure, reported in fig. 4, initially identifies (at line N2) the maximum height $H_{\mathcal{C}}$ of the XML trees within cluster $\mathcal{C}$. $H_{\mathcal{C}}$ is used to prune (at lines N7-N11) combined candidates with an unnecessarily large height.

Candidate-Generation then combines (lines N3-N21) each pair of distinct tree patterns $\mathbf{s}_i$ and $\mathbf{s}_j$ from L into further candidate (parent-child or sibling) tree patterns.

The bit list $\mathcal{E}$ (at line N4) is a common estimation of the actual bit lists associated with all candidates obtainable through any admissible combination of $\mathbf{s}_i$ and $\mathbf{s}_j$. $\mathcal{B}$ shall be refined into the actual bit lists of such candidates by the MineRep procedure of fig. 3 (at lines M11 and M12).

The set T (at line N5) contains all parent-child tree patterns obtainable from $\mathbf{s}_i$ and $\mathbf{s}_j$. The candidate generation strategy soon prunes T (lines N6-N12): the

```
CANDIDATE-GENERATION($L, \mathcal{C}$)
  Input: a set $L$ of discriminative tree patterns;
         a set $\mathcal{C}$ of XML trees;
  Output: a set $C$ of candidate combined tree patterns;
  N1:  $C \leftarrow \emptyset$;
  N2:  $H_{\mathcal{C}} \leftarrow max\{height(\mathbf{t}) | \mathbf{x}^{(\mathbf{t})} \in \mathcal{C}\}$;
  N3:  for each $\mathbf{s}_i, \mathbf{s}_j \in L$ do
  N4:      $\mathcal{E} \leftarrow \mathcal{B}(\mathbf{s}_i) \cap \mathcal{B}(\mathbf{s}_j)$;
  N5:      $T \leftarrow \mathbf{s}_i \triangleleft \mathbf{s}_j \ \cup \ \mathbf{s}_j \triangleleft \mathbf{s}_i$;
  N6:      for each $\mathbf{s} \in T$ do
  N7:          if ($height(\mathbf{s}) \leq H_{\mathcal{C}}$) then
  N8:              $\mathcal{B}(\mathbf{s}) \leftarrow \mathcal{E}$;
  N9:          else
  N10:             $T \leftarrow T - \{\mathbf{s}\}$;
  N11:         end if
  N12:     end for
  N13:     $C \leftarrow C \cup T$;
  N14:     if ($|\mathbf{V}_{\mathbf{s}_i}| > 1$) and ($|\mathbf{V}_{\mathbf{s}_j}| > 1$) then
  N15:         if ($\lambda_{\mathbf{s}_i}(r_{\mathbf{s}_i}) = \lambda_{\mathbf{s}_j}(r_{\mathbf{s}_j})$) then
  N16:             $\mathbf{s} \leftarrow \mathbf{s}_i \wedge \mathbf{s}_j$;
  N17:             $\mathcal{B}(\mathbf{s}) \leftarrow \mathcal{E}$;
  N18:             $C \leftarrow C \cup \{\mathbf{s}\}$;
  N19:         end if
  N20:     end if
  N21: end for
  N22: return $C$;
```

Figure 4: The candidate generation sub-procedure

height of each candidate $\mathbf{s}$ in T is tested (lines N7-N11) against the maximum height $H_{\mathcal{C}}$ of the XML trees in cluster $\mathcal{C}$. If *height*($\mathbf{s}$) does not exceed $H_{\mathcal{C}}$, $\mathbf{s}$ is left in T and $\mathcal{E}$ is set as the estimated bit list of $\mathbf{s}$. By the contrary, if *height*($\mathbf{s}$) exceeds $H_{\mathcal{C}}$, $\mathbf{s}$ is removed from T (at line N10). The resulting T is added to the ongoing set C of candidates (at line N13).

At this point, CANDIDATE-GENERATION considers the sibling pattern obtainable from $\mathbf{s}_i$ and $\mathbf{s}_j$, if both are not tree-like representations of individual nodes (tested at line N14) and share a common root label (tested at line N15). The candidate is associated (at line N17) with the estimation $\mathcal{E}$ of its actual bit list and, then, is added to C (at line N18).

Once all pairs of representative tree patterns are considered for combination, CANDIDATE-GENERATION returns (at line N22) the set C of new candidate tree patterns.

4.3. Representativeness of Candidate Tree Patterns

As to the use of statistical hypothesis testing in MINEREP (at line M17), the non-parametric chi-square test is used to establish whether the representativeness of a candidate tree pattern $\mathbf{s}$ is statistically grounded. This involves a decision between two alternative hypotheses: a *null* hypothesis according to which the occurrence of $\mathbf{s}$ in $\mathcal{C}$ is a consequence of chance (and, thus, the representativeness of $\mathbf{s}$ must necessarily be considered as statistically groundless) and an *alternative* hypothesis, according to which the occurrence of $\mathbf{s}$ in $\mathcal{C}$ is statistically relevant (and, hence, $\mathbf{s}$ must be representative of some corresponding structural properties). To make a proper decision between the two alternative hypotheses, the following four statistics are considered:

- $n_{\mathbf{s}\mathcal{C}}$, the number of (transactions corresponding to) XML trees in $\mathcal{C}$ that contain $\mathbf{s}$ as an embedded substructure, i.e., $n_{\mathbf{s}\mathcal{C}} = |\{\mathbf{x}^{(\mathbf{t})} \in \mathcal{B}(\mathbf{s}) \cap \mathcal{C}\}|$.
- $n_{\mathbf{s}\neg\mathcal{C}}$, the number of (transactions corresponding to) XML trees in any cluster other than $\mathcal{C}$, that contain $\mathbf{s}$, i.e., $n_{\mathbf{s}\neg\mathcal{C}} = |\{\mathbf{x}^{(\mathbf{t})} \in \mathcal{B}(\mathbf{s}) \cap \overline{\mathcal{C}}\}|$ (recall that $\overline{\mathcal{C}}$ is the set of siblings of $\mathcal{C}$).
- $n_{\neg\mathbf{s}\mathcal{C}}$, the number of (transactions corresponding to) XML trees in $\mathcal{C}$, that do not contain $\mathbf{s}$. The value of $n_{\neg\mathbf{s}\mathcal{C}}$ is computed through two alternative definitions, according to the nature of $\mathbf{s}$. Precisely, $n_{\neg\mathbf{s}\mathcal{C}} = |\{\mathbf{x}^{(\mathbf{t})} \in \mathcal{D}(\mathbf{s}) \cap \mathcal{C}\}|$ if $\mathbf{s}$ is a combined tree pattern resulting at any iteration $k \geq 2$. Otherwise, $n_{\neg\mathbf{s}\mathcal{C}}$ follows from complementing $n_{\mathbf{s}\mathcal{C}}$, i.e., $n_{\neg\mathbf{s}\mathcal{C}} = |\{\mathbf{x}^{(\mathbf{t})} \in [\mathcal{C} - (\mathcal{B}(\mathbf{s}) \cap \mathcal{C})]\}|$, since $\mathcal{D}(\mathbf{s}) = \emptyset$ for elementary components at iteration $k = 1$.
- $n_{\neg\mathbf{s}\neg\mathcal{C}}$, the number of (transactions corresponding to) XML trees within all clusters but $\mathcal{C}$, that do not contain $\mathbf{s}$. Again, $n_{\neg\mathbf{s}\neg\mathcal{C}} = |\{\mathbf{x}^{(\mathbf{t})} \in \mathcal{D}(\mathbf{s}) \cap \overline{\mathcal{C}}\}|$ if $\mathbf{s}$ is a combined tree pattern (at any iteration $k \geq 2$). Otherwise, $n_{\neg\mathbf{s}\neg\mathcal{C}}$ follows from complementing $n_{\mathbf{s}\mathcal{C}}$, i.e., $n_{\neg\mathbf{s}\neg\mathcal{C}} = |\{\mathbf{x}^{(\mathbf{t})} \in [\overline{\mathcal{C}} - (\mathcal{B}(\mathbf{s}) \cap \overline{\mathcal{C}})]\}|$ (at iteration $k = 1$).

The above statistics enable the computation of two marginal totals, namely, the overall numbers $n_{\mathbf{s}}$ and $n_{\neg\mathbf{s}}$ of XML trees in $\mathcal{C} \cup \overline{\mathcal{C}}$ that contain, respectively, do not contain $\mathbf{s}$ as an embedded substructure. Marginal totals, in turn, allow to compute the expected values of the foregoing statistics, respectively denoted as $\overline{n}_{\mathbf{s}\mathcal{C}}$, $\overline{n}_{\mathbf{s}\neg\mathcal{C}}$, $\overline{n}_{\neg\mathbf{s}\mathcal{C}}$, $\overline{n}_{\neg\mathbf{s}\neg\mathcal{C}}$, that represent those values that would be expected if there was no meaningful correlation between $\mathbf{s}$ and $\mathcal{C}$, i.e., if $\mathbf{s}$ occurred in $\mathcal{C}$ by chance. Precisely, $\overline{n}_{\mathbf{s}\mathcal{C}} = \frac{n_{\mathbf{s}}|\mathcal{C}|}{|\mathcal{C} \cup \overline{\mathcal{C}}|}$, $\overline{n}_{\neg\mathbf{s}\mathcal{C}} = \frac{n_{\neg\mathbf{s}}|\mathcal{C}|}{|\mathcal{C} \cup \overline{\mathcal{C}}|}$, $\overline{n}_{\mathbf{s}\neg\mathcal{C}} = \frac{n_{\mathbf{s}}|\overline{\mathcal{C}}|}{|\mathcal{C} \cup \overline{\mathcal{C}}|}$, $\overline{n}_{\neg\mathbf{s}\neg\mathcal{C}} = \frac{n_{\neg\mathbf{s}}|\overline{\mathcal{C}}|}{|\mathcal{C} \cup \overline{\mathcal{C}}|}$. Given both the observed and expected values of the statistics, it is possible to compute the value of the following test-statistic:

$$\chi^2(\mathbf{s}, \mathcal{C}) = \sum_{\mathbf{s}' \in \{\mathbf{s}, \neg\mathbf{s}\}, \mathcal{C}' \in \{\mathcal{C}, \neg\mathcal{C}\}} \frac{(n_{\mathbf{s}'\mathcal{C}'} - \overline{n}_{\mathbf{s}'\mathcal{C}'})^2}{\overline{n}_{\mathbf{s}'\mathcal{C}'}}$$

The null hypothesis is rejected in favor of a statistically relevant occurrence of $\mathbf{s}$ in $\mathcal{C}$ if the difference between observed and expected statistics is high, i.e. if $\chi^2(\mathbf{s}, \mathcal{C}) > \tau_\alpha$, where τ_α is the threshold for the chi-square distribution with one degree of freedom at a significance level α. In statistical terms, α is the probability of a so-called *type I* error, i.e., the probability of rejecting the null hypothesis, when it is instead true. Setting α to low values is useful for a twofold purpose: it increases the significance level (i.e. the probabilistic robustness) of the test and, also, it accordingly limits the number of representative substructures that can be distilled (at line M17 of fig. 3) from the current set of candidates (which, in turn, reduces the number of candidates attainable from such representative).

5. Evaluation

In this section, the behavior of the devised clustering approach is investigated through an empirical evaluation with three main objectives.

1. *The assessment of clustering quality.* This involves measuring the effectiveness of clustering, i.e., verifying whether the clusters, at any level of the hierarchy produced by the algorithm, correspond to groups of XML documents that are structurally-homogeneous with respect to the type of structural patterns considered at that level in the hierarchy.
2. *The assessment of cluster-summarization.* This encompasses evaluating the extent with which a set of substructures actually subsumes the structural properties of the XML documents of a cluster.
3. *Performance comparison.* Multi-stage clustering is compared against state-of-the art competitors [16] to verify whether it retains similar or even better levels of effectiveness and scalability.

All experiments were conducted on a Windows machine, with an Intel Itanium processor, 2Gb of memory and 2Ghz of clock speed.

5.1. Data Sets

Standard benchmark data sets were employed for a direct comparison against the competitors. These data sets are described below.

5.1.1. Real data

We choose three real-world data sets, characterized by imbalanced distributions of the classes of XML documents.

DBLP is a bibliographic archive of scientific publications on computer science (`http://dblp.unitrier.de/xml/`). The archive is available as one very large XML file with a diversified structure. The whole file is decomposed into 479,426 XML documents corresponding to as many scientific publications. These individually belong to one of 8 classes: `article` (173,630 documents), `proceedings` (4,764 documents), `mastersThesis` (5 documents), `incollection` (1,379 documents), `inproceedings` (298,413 documents), `book` (1,125 documents), `www` (38 documents), `phdthesis` (72 documents). The individual classes of XML documents exhibit differentiated structures, despite some overlap among certain document tags (such as *title*, *author*, *year* and *pages*), that occur in (nearly) all of the XML documents.

The *Sigmod* collection groups 988 documents complying to three different class DTDs: `IndexTermsPage`, `OrdinaryIssue` and `Proceedings`. These classes contain, respectively, 920, 51 and 17 XML documents. Such classes have diversified structures, despite the occurrence of some overlapping tags, such as *volume*, *number*, *authors*, *title* and *year*.

Real is a collection of 649 XML documents assembled and used in [17], that encompasses the following 5 classes:

- Astronomy, 217 documents extracted from an XML-based metadata repository, that describes an archive of publications owned by the *Astronomical Data Center* at NASA/GSFC.

- Forum, 264 documents concerning messages sent by users of a Web forum.

- News, 64 documents concerning press news from all over the world, daily collected by *PR Web*, a company providing free online press release distribution.

- Wrapper, 53 documents representing wrapper programs for Web sites, obtained through the *Lixto* system [4].

- Sigmod, a selection of 51 documents (from the whole Sigmod collection), concerning issues of SIGMOD Record. These documents were obtained from the XML version of the ACM SIGMOD Web site produced within the *Araneus* project [12].

The distribution of tags in *Real* is heterogeneous, due to the complexity of the class DTDs and to the semantic differences among the documents. *Real* allows to evaluate the behavior of classification models on XML data assembled from multiple real-world sources. This is especially interesting, since in such cases it is hard to assemble training sets representative of all structural characteristics with which to isolate the individual document classes (i.e. sources).

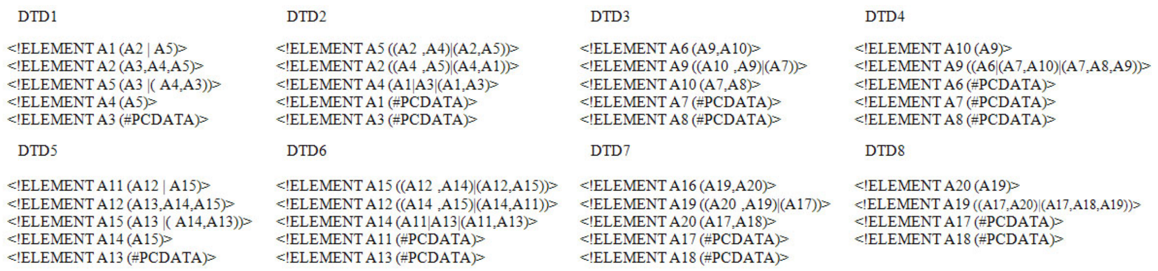
5.1.2. Synthetic Data Sets

Four synthetic data sets were generated from as many collections of DTSs.

The first synthesized data set, referred to as *Synth1*, comprises 1000 XML documents produced from a collection of 10 heterogeneous DTDs (illustrated in fig. 6 of [16]), that were individually used to generate 100 XML documents. These DTDs exhibit strong structural differences and, thus, can be neatly separated by most clustering algorithms.

A finer evaluation can be obtained by investigating the behavior of the compared algorithms on a collection of XML documents, that are very similar to one another from a structural point of view. To perform such a test, a second synthesized data set, referred to as *Synth2* and consisting of 3000 XML documents, was assembled from 3 homogeneous DTDs (illustrated in fig. 7 of [16]), individually used to generate 1000 XML documents. Experiments over *Synth2* clearly highlight the ability of the competitors at operating in extremely-challenging applicative-settings, wherein the XML documents share multiple forms of structural patterns.

Synth3 consists of 1400 synthesized documents individually belonging to one of 7 distinct class DTDs. These classes represent a challenging domain for the approaches to structural classification, since they were suitably designed in [17] to overlap in at most 30% of their element definitions. This implies a commonality of nodes, edges and even paths in the documents conforming to the different classes.



```
DTD1
<!ELEMENT A1 (A2 | A5)>
<!ELEMENT A2 (A3,A4,A5)>
<!ELEMENT A5 (A3 |( A4,A3))>
<!ELEMENT A4 (A5)>
<!ELEMENT A3 (#PCDATA)>

DTD2
<!ELEMENT A5 ((A2 ,A4)|(A2,A5))>
<!ELEMENT A2 ((A4 ,A5)|(A4,A1))>
<!ELEMENT A4 (A1|A3|(A1,A3)>
<!ELEMENT A1 (#PCDATA)>
<!ELEMENT A3 (#PCDATA)>

DTD3
<!ELEMENT A6 (A9,A10)>
<!ELEMENT A9 ((A10 ,A9)|(A7))>
<!ELEMENT A10 (A7,A8)>
<!ELEMENT A7 (#PCDATA)>
<!ELEMENT A8 (#PCDATA)>

DTD4
<!ELEMENT A10 (A9)>
<!ELEMENT A9 ((A6|(A7,A10)|(A7,A8,A9))>
<!ELEMENT A6 (#PCDATA)>
<!ELEMENT A7 (#PCDATA)>
<!ELEMENT A8 (#PCDATA)>

DTD5
<!ELEMENT A11 (A12 | A15)>
<!ELEMENT A12 (A13,A14,A15)>
<!ELEMENT A15 (A13 |( A14,A13))>
<!ELEMENT A14 (A15)>
<!ELEMENT A13 (#PCDATA)>

DTD6
<!ELEMENT A15 ((A12 ,A14)|(A12,A15))>
<!ELEMENT A12 ((A14 ,A15)|(A14,A11))>
<!ELEMENT A14 (A11|A13|(A11,A13)>
<!ELEMENT A11 (#PCDATA)>
<!ELEMENT A13 (#PCDATA)>

DTD7
<!ELEMENT A16 (A19,A20)>
<!ELEMENT A19 ((A20 ,A19)|(A17))>
<!ELEMENT A20 (A17,A18)>
<!ELEMENT A17 (#PCDATA)>
<!ELEMENT A18 (#PCDATA)>

DTD8
<!ELEMENT A20 (A19)>
<!ELEMENT A19 ((A17,A20)|(A17,A18,A19))>
<!ELEMENT A17 (#PCDATA)>
<!ELEMENT A18 (#PCDATA)>
```

Figure 5: DTDs for the *Synth4* data set

Additionally, *Synth4* comprises 800 documents complying to the DTDs of fig. 5. There are 8 separate classes. Each class groups 100 XML documents. In particular, `DTD1` and `DTD2` share nodes `A1`, ..., `A5`, whereas `DTD3` and `DTD4` share nodes `A6`, ..., `A10`. Instead `DTD5` and `DTD6` share nodes `A11`, ..., `A15`, whereas `DTD7` and `DTD8` share nodes `A17`, ..., `A20` (node `A16` is specific of `DTD7`). These DTDs capture substantial similarities and differences: they exhibit different paths, despite sharing some common edges. Furthermore, the XML documents in `DTD4` can be further split, since their trees can exhibit paths ending in node `A6`. Also, node frequencies in `DTD4` can substantially differ, thus differentiating this DTD from the others even at a node level.

Synthetic XML data was generated by means of the XML data generator described in [24]. The latter essentially accepts an input DTD and produces a set of conforming documents, on the basis of suitable statistical models governing the occurrences of elements marked by operators $*$, ?, $|$, and $+$. The generation process was constrained as in [16]. Precisely, the maximum number of occurrences of a child node in the context of its parent node was fixed to 6. The number of repetitions is, hence, randomly chosen in the interval $[0, 6]$. The maximum depth of the synthetic XML trees was set to 7.

5.2. Competitors

The behavior of the proposed hierarchical clustering method was compared over the chosen data sets against a selection of state-of-the-art competitors, namely *SGrace* [26], *XRep* [17], *XProj* [16], the Chawathe's algorithm [10] and the Dalamagas et al.'s approach [25]. In particular, the results of the comparisons against *SGrace* [26] and *XRep* [17] are obtained by exploiting our implementations of such competitors. Instead, the comparison against *XProj* is indirect, i.e. based on the results reported in [16], since it was unfortunately not possible to obtain an executable version of the algorithm from its authors. Notwithstanding, the adoption of the same data sets used in [16], namely *Sigmod*, *Synth1* and *Synth2*, still enables a meaningful comparison against both *XProj* as well as the competitors chosen in [16] for the evaluation of *XProj*, i.e., the Chawathe's algorithm [10] and the Dalamagas et al.'s approach [25], whose performances are reported from [16] too.

5.3. Evaluation of Clustering Effectiveness

Three main methods for assessing clustering effectiveness are described in [28, 31], namely

- *external criteria*, when clustering results are evaluated according to a prespecified structure, that corresponds to a meaningful explanation of the data at hand;
- *internal criteria*, when clustering results are evaluated in terms of the quantities that are computable from the available data;
- *relative criteria*, when evaluation takes place in comparison with other clustering schemes.

It is worth recalling here that the adoption of external criteria helps to understand clustering results and, hence, the adequacy of a clustering algorithm. Indeed, a predefined structure can be interpreted as the explanation of the available data by means of some hypotheses. As a consequence, the correspondence of a cluster to one of such predefined structures implies the interpretation of the cluster according to the corresponding hypothesis. For these reasons, we resort to external criteria to evaluate clustering effectiveness. More specifically, we investigate the behavior of Generate-Hierarchy over collections of XML documents with known class labels and analyze the correspondence between the discovered and hypothesized structures. In particular, class labels serve as ground truth for natural data classes. Therefore, by matching the discovered partitions against the actual data classes, we measure the effectiveness of both Generate-Hierarchy and the competing state-of-the-art algorithms at discovering natural clusters.

Let $\mathcal{D}$ be a set of XML documents and $\mathbf{C}_1, \ldots, \mathbf{C}_t$ be the true classes of such documents. The individual classes $\mathbf{C}_j$, with $j = 1, \ldots, t$, are disjoint subsets of XML documents, that represent a true partition of $\mathcal{D}$ (i.e., $\mathcal{D} = \cup_i \mathbf{C}_i$ and $\mathbf{C}_i \cap \mathbf{C}_j = \emptyset$ for each $i \neq j$).

Assume that $\mathcal{P}^{(l)}$ is the set of cluster leaves produced at stage l (i.e., sited at level l of the cluster hierarchy) by Generate-Hierarchy. $\mathcal{P}^{(l)}$ together with all possible cluster leaves sited at any preceding level $l' < l$ form a partition of $\mathcal{D}$. The effectiveness of Generate-Hierarchy is assessed in terms of two traditional measures from the field of information retrieval, namely, average precision (i.e. exactness) and recall (i.e. completeness) [2].

We first introduce precision and recall for evaluating effectiveness at the first level of the cluster hierarchy, i.e., when all cluster leaves are sited at level $l = 1$. Then, we refine such definitions of precision and recall in order to evaluate clustering effectiveness at any level $l > 1$ (in which case possible cluster leaves at any preceding level $l' < l$ of the cluster hierarchy must be accounted for).

The partition $\mathcal{P}^{(1)} = \{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ at the first level of the cluster hierarchy can be summarized into a contingency table m, where columns represent discovered clusters and rows represent true classes. Each entry m_{ij} indicates the number of transactions (corresponding to XML documents in $\mathcal{D}$), that are assigned to cluster $\mathcal{C}_j$, with $1 \leq j \leq k$, and actually belong to class $\mathbf{C}_i$, with $1 \leq i \leq t$. The table provides an immediate visual description of the degree of agreement between the results yielded by GENERATE-CLUSTERS and the actual class partition. Also, the table permits a quantitative evaluation of such a degree of correspondence, that can be finely caught through precision and recall. Intuitively, each cluster $\mathcal{C}_j$ corresponds to the class $\mathbf{C}_i$ that is best represented in $\mathcal{C}_j$, i.e., such that m_{ij} is maximal. For any cluster $\mathcal{C}_j$, the index $h(j)$ of the class with maximal m_{ij} is defined as $h(j) = max_i\ m_{ij}$. Precision $P(\mathcal{C}_j)$ and recall $R(\mathcal{C}_j)$ for cluster $\mathcal{C}_j$ are defined as follows:

$$P(\mathcal{C}_j) = \frac{|\{\mathbf{x^{(t)}} \in \mathcal{C}_j | \mathbf{t} \in \mathbf{C}_{h(j)}\}|}{|\mathcal{C}_j|},\ R(\mathcal{C}_j) = \frac{|\{\mathbf{x^{(t)}} \in \mathcal{C}_j | \mathbf{t} \in \mathbf{C}_{h(j)}\}|}{|\mathbf{C}_{h(j)}|}$$

Hence, the average precision $P^{(1)}$ and recall $R^{(1)}$ for partition $\mathcal{P}^{(1)}$ can be defined as

$$P^{(1)} = \frac{1}{|\mathcal{P}^{(1)}|} \sum_{\mathcal{C} \in \mathcal{P}^{(1)}} P(\mathcal{C}),\ R^{(1)} = \frac{1}{|\mathcal{P}^{(1)}|} \sum_{\mathcal{C} \in \mathcal{P}^{(1)}} R(\mathcal{C})$$

The definitions of $P^{(l)}$ and $R^{(l)}$ for a partition with cluster leaves at some level $l > 1$ of the cluster hierarchy require a slight adaptation. The latter is due to account for possible cluster leaves (if any) sited at the preceding levels of the cluster hierarchy. Let $\mathcal{L}^{(l)}$ be the set of all cluster leaves sited at any level $l' < l$ in the cluster hierarchy. The partition of the original set $\mathcal{D}$ of XML documents is hence $\mathcal{P}^{(l)} \cup \mathcal{L}^{(l)}$. Therefore, the definitions of $P^{(l)}$ and $R^{(l)}$ remain as the ones of $P^{(1)}$ and $R^{(1)}$, with the exception that summation and averaging involve $\mathcal{P}^{(l)} \cup \mathcal{L}^{(l)}$ rather that only $\mathcal{P}^{(l)}$.

5.4. Single-Stage Clustering

GENERATE-HIERARCHY effectiveness is evaluated in multiple steps. One interesting aspect is to comparatively investigate the effectiveness of GENERATE-HIERARCHY when the latter produces one partitioning of the original collection of XML documents. This allows to understand the effectiveness of GENERATE-CLUSTERS at separating the available XML documents with respect to the occurrence of one form of structural pattern within their tree structures.

Table 1 shows the comparative evaluation of GENERATE-HIERARCHY, when it is applied in single-stage modality to consider document nodes as discriminatory substructures. Notably, nodes are the structural components that, in principle, are expected to originate the worst clustering performance, being the least discriminatory type of substructures. Notwithstanding, the observed values of average precision and recall for GENERATE-HIERARCHY are maximum over each data set but *DBLP*.

The behavior over *DBLP* is essentially due to the fact that GENERATE-HIERARCHY splits the original `article`, `inproceedings` and `proceedings` classes into subclasses, on the basis of the presence/absence of certain nodes within the tree structures of their XML documents. As an example, the `proceedings` class is divided into two subclasses: the tree structures of the XML documents within one such a subclass exhibit certain nodes (such as, e.g., *isbn*, *volume* and *url*), that do not belong to tree structures of the XML documents in the other subclass. Clearly, the division of the foresaid classes into subclasses results in a number of clusters larger than the actual number of classes and an average recall lowered to 0.92.

Moreover, the overlap among some nodes of XML documents from different classes contributes to a negligible error (actually less than 0.01) in average precision. As a matter of fact, two clusters group XML documents from different classes. More precisely, one cluster contains XML documents originally belonging to the `inproceedings`, `incollection` and `www` classes, which overlap on the *title* and *url* nodes. The other cluster instead includes XML documents from the `phdthesis`, `masterthesis`, `book` and `www` classes, which mainly share the *author*, *title* and *year* nodes (actually, the very few documents from the `www` class overlap only on the *title* node). Apart from these two clusters, all other clusters produced by GENERATE-HIERARCHY exhibit maximum precision.

SGrace and *XRep* did not successfully complete their respective tests on *DBLP*. This is indicated by means of symbol $-$ in the corresponding entries of table 1.

As far as clustering effectiveness over the *Synth2* collection is concerned, notice that *XProj* also achieves the same maximum precision and recall as GENERATE-HIERARCHY. However, it is worth to underline that the performance of GENERATE-HIERARCHY is measured over $3,000$ XML documents, whereas the performances of *XProj* and the Dalamagas et al.'s approach (whose behavior is suboptimal) were obtained in [16] over a much smaller *Synth2* collection, consisting of 300 XML documents. Additionally, the sensitivity analysis of *XProj* (fig. 4 and table 4 in [16]) reveals that an improper setting of important input parameters, such as the sequence length as well as the minimum and maximum support-thresholds, lowers precision over the reduced *Synth2* collection down to nearly 0.92%. Notice that, in this test, the performance of GENERATE-HIERARCHY over the *Sigmod* data set refers to the identification of the tree clusters at the first level (beneath the root) of the hierarchy in fig. 7.

Table 1 also confirms that GENERATE-HIERARCHY is generally capable to autonomously discover the actual number of clusters in the XML data, even when poorly discriminatory substructures are taken into account. Cluster number is instead a problematic input parameter of the chosen competitors.

Efficiency is another strong point of GENERATE-HIERARCHY.

By looking at table 1, one can notice that the overall running time taken by GENERATE-HIERARCHY is up to three orders of magnitude less than the running time of its competitors. We emphasize that all running times related to GENERATE-HIERARCHY are actually the sum of two contributions: the time for the prior enumeration of the individual nodes in the underlying collection of XML documents plus the time required for cluster generation and summarization.

Unfortunately, the running times concerning *XProj*, the Chawathe's algorithm and the Dalamagas et al.'s approach were not reported in [16]. The unavailability of this information is indicated by means of symbol $-$ in the

Collection	N. of Docs	Classes	Method	Clusters	Avg Precision	Avg Recall	Time (s)
DBLP	479,426	8	Generate-Hierarchy	11	0.99	0.92	2,301.63
			S-Grace	-	-	-	-
			XRep	-	-	-	-
Sigmod	998	3	Generate-Hierarchy	**3**	**1**	**1**	**4.91**
			S-Grace	3	0.67	0.88	69.11
			XRep	3	1	1	200.42
Real	649	5	Generate-Hierarchy	**5**	**1**	**1**	**20.48**
			S-Grace	5	1	1	21.98
			XRep	5	1	1	479.52
Synth1	1,000	10	Generate-Hierarchy	**10**	**1**	**1**	**13.32**
			S-Grace	10	1	1	53.23
			XRep	11	0.91	0.96	331.38
			Chawathe's algorithm [10]	12	0.83	0.96	-
			Dalamagas et al.'s algorithm [25]	11	1	0.98	-
			XProj [16]	10	1	1	-
Synth2	3,000	3	Generate-Hierarchy	**3**	**1**	**1**	**7.53**
			S-Grace	3	0.99	0.99	1,351.43
			XRep	3	0.67	0.93	2,411.72
	300	3	*Dalamagas et al.'s algorithm*	3	0.78	0.78	-
			XProj	3	1	1	-
Synth3	1,400	7	Generate-Hierarchy	**7**	**1**	**1**	**5.44**
			S-Grace	7	1	1	137.35
			XRep	7	0.85	0.97	228.15

Table 1: Comparative evaluation of separability and homogeneity of Generate-Hierarchy in single-stage modality

corresponding entries of table 1.

5.5. Multi-Stage Clustering

A second aspect to investigate is the behavior of Generate-Hierarchy, when multiple forms of structural patterns are considered in some meaningful sequence to progressively partition the available XML documents. In the following tests, Generate-Hierarchy is exploited to form a cluster hierarchy, in which nodes are considered for partitioning at the first level of the hierarchy (below the root), edges are accounted for at the second level and root-to-leaf paths are addressed at the third level.

Fig. 6 shows the hierarchy produced by Generate-Hierarchy over *Synth4*. Notice that `DTD4` is separated from `DTD3` at the node level (i.e., the first level beneath the root) of the cluster hierarchy. `DTD4` is further split into two child clusters at the edge level (i.e. the second level of the hierarchy), according to whether or not edges include (`A9`, `A6`). Also, the XML trees with such an edge can be further split at the path level (i.e., the third level the hierarchy) according to whether or not they contain the path from `A10` to `A6`. By the contrary, `DTD8` does not behave similarly, since there is no such a node like `A6` that differentiates the XML trees in the class.

The behavior of Generate-Hierarchy is confirmed by experimenting over Sigmod. As it has been anticipated earlier, this data set consists of documents complying with three different DTDs. In particular, the distribution of the documents is unbalanced, since one of the DTDs, `IndexTermsPage`, contains 920 documents. Fig. 7 shows that Generate-Hierarchy separates all documents complying to different DTDs at the node level (the first level beneath the root) in the hierarchy. Also, the documents in the class related to `IndexTermsPage` are further split, according to whether or not they contain the optional elements described in the DTD (mainly `category`, `content`, `term` and `categoryAndSubjectDescriptorsTuple`). In particular, the separation of the

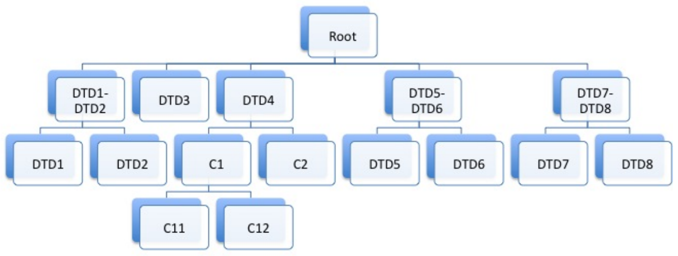


Figure 6: Cluster hierarchy for the *Synth4* data set

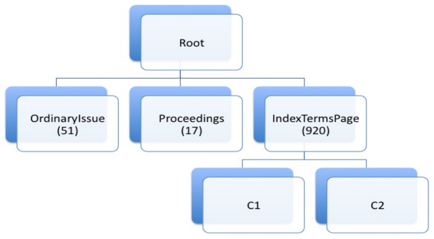


Figure 7: Cluster hierarchy for the *Sigmod* data set

`IndexTermsPage` class leads to two subclasses $\mathtt{C}_1$ and $\mathtt{C}_2$, that can be described by two DTDs, both subsumed by `IndexTermsPage`. $\mathtt{C}_1$ is a subclass of 437 documents, in which the optional elements of `IndexTermsPage` are absent. Instead, the remaining 483 documents (in which the foresaid optional elements occur) are assembled into subclass $\mathtt{C}_2$. The latter finely isolates the stronger relationships of resemblance induced by the additional pieces of structures, which correspond to edges at the level of $\mathtt{C}_1$ and $\mathtt{C}_2$ in the cluster hierarchy of fig. 7.

Collection	**N. of Docs**	**Classes**	**Clusters**	**Avg Precision**	**Avg Recall**	**Time**
Synth4	800	8	**10**	**1**	**0.81**	**3.68s**
Sigmod	998	3	**4**	**1**	**0.54**	**8.31s**

Table 2: Evaluation of Generate-Hierarchy in the multi-stage modality

The results of multi-stage clustering over both *Synth4* and *Sigmod* are summarized in table 2. Notably, Generate-Hierarchy must form a number of clusters that is necessarily larger than the actual number of classes, in order to better explain (i.e., isolate) the various structural properties of the underlying XML documents. Therefore, a higher degree of intra-cluster homogeneity from the structural viewpoint comes with optimal precision and a lowered recall. However, a reduction in recall after multiple stages of clustering is irrelevant in practice. Indeed, recall is already maximal for the clusters at the first level of the hierarchy (as in the case of *Sigmod*) and becomes actually uninformative for the clusters beneath, whose purpose is only to isolate forms of structural patterns in the underlying XML trees that are uncaught at the preceding levels. Again, the running times reported in table 2 are the sum of two contributions: the time for the prior enumeration of the substructures taken into account at each level of the resulting cluster hierarchy (respectively, the set of all nodes, the set of all edges as well as the set of all root-to-leaf paths in the underlying collection of XML documents) plus the time required for cluster generation and summarization.

5.6. Cluster Summarization

The evaluation of the representativeness of cluster summarization is inspired to an idea originally proposed in [16] for a different purpose, i.e., measuring the structural homogeneity of a set of intermediate clusters obtained while partitioning a collection of XML documents. Let $\mathbf{t}$ be an XML tree and $\mathcal{R}$ a set of substructures. The representativeness $\gamma(\mathcal{R}, \mathbf{t})$ of $\mathcal{R}$ with respect to $\mathbf{t}$ is the fraction of nodes in $\mathbf{t}$ matched by the embedded substructures of $\mathcal{R}$:

$$\gamma(\mathcal{R}, \mathbf{t}) = \frac{|\cup_{\mathbf{s}\in\mathcal{R},\mathbf{s}\preceq\mathbf{t}} \{n | n \in \mathbf{V}_{\mathbf{s}\mapsto\mathbf{t}}\}|}{|\mathbf{V}_{\mathbf{t}}|}$$

where, $\mathbf{V}_{\mathbf{t}}$ is the set of nodes of the XML tree $\mathbf{t}$ and $\mathbf{V}_{\mathbf{s}\mapsto\mathbf{t}}$ is instead the subset of nodes in $\mathbf{t}$ matched by the nodes of $\mathbf{s}$. Representativeness can be easily generalized to sets (i.e., clusters) of XML trees. Let $\mathcal{C}$ be a generic cluster of XML trees and $\mathit{Rep}(\mathcal{C})$ some set of substructures from $\mathcal{C}$. The representativeness $\Gamma\left[\mathit{Rep}(\mathcal{C})\right]$ of $\mathit{Rep}(\mathcal{C})$ with respect to $\mathcal{C}$ can be defined as the average representativeness of $\mathit{Rep}(\mathcal{C})$ with respect to the individual XML trees in $\mathcal{C}$:

$$\Gamma\left[\mathit{Rep}(\mathcal{C})\right] = \frac{1}{|\mathcal{C}|}\sum_{\mathbf{t}\in\mathcal{C}} \gamma(\mathit{Rep}(\mathcal{C}), \mathbf{t})$$

A connection between cluster representativeness and structural homogeneity of XML document collections explains unexpectedly low Γ values.

If $\mathcal{C}$ includes structures that do not frequently occur within $\mathcal{C}\cup\overline{\mathcal{C}}$, as it instead happens with very homogeneous collections of documents (e.g., *Synth2*), cluster representativeness reflects the degree of intra-cluster structural homogeneity of $\mathcal{C}$. Therefore, $\Gamma\left[\mathit{Rep}(\mathcal{C})\right]$ is low if cluster $\mathcal{C}$ includes structurally heterogeneous XML trees and, hence, MineRep (fig. 3) is unable to extract representative substructures. On the contrary, $\Gamma\left[\mathit{Rep}(\mathcal{C})\right]$ is high if $\mathcal{C}$ contains structurally homogeneous XML trees, whose nodes are matched in a high percentage by the substructures extracted by MineRep.

Γ has instead a different interpretation, whenever several structures frequently occur in both $\mathcal{C}$ and $\mathcal{C} \cup \overline{\mathcal{C}}$ with the same frequency (which happens, e.g., when $\mathcal{C}$ and $\overline{\mathcal{C}}$ are a partition of a very structurally homogeneous collection of documents). In such a case, MineRep does not consider such structures as elementary substructures for the construction of $\mathit{Rep}(\mathcal{C})$, since their occurrence

Real	*Synth1*	*Synth2*	*Synth3*	*Synth4*	*Sigmod*	*DBLP*
0.96	**0.95**	**0.45**	**0.8**	**0.72**	**0.67**	**0.91**

Figure 8: Average Γ values over the chosen data sets

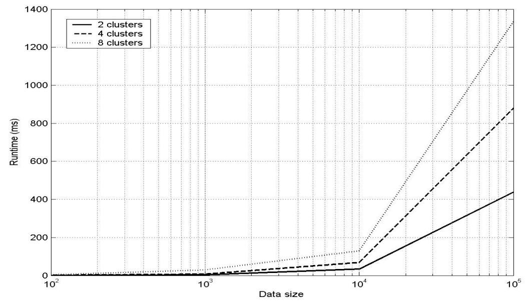


Figure 9: Scalability (in ms) over synthetic data with increasing size

frequency within $\mathcal{C}$ is not greater (at line M3 of fig. 3) then the occurrence frequency within $\mathcal{C} \cup \overline{\mathcal{C}}$. In these cases, a low value of $\Gamma\left[Rep(\mathcal{C})\right]$ ceases to reflect the degree of structural homogeneity of $\mathcal{C}$ (that still remains high, being $\mathcal{C} \cup \overline{\mathcal{C}}$ very homogeneous) and becomes indicative of the small percentage of nodes of the XML trees within $\mathcal{C}$ (matched by $Rep(\mathcal{C})$), that actually discriminate $\mathcal{C}$.

The table in fig. 8 shows the average Γ values over all data sets, computed by averaging the Γ values associated to the leaves of the cluster hierarchies produced by GENERATE-HIERARCHY in sec. 5.4 and 5.5. In all tests, the statistical significance α of fig. 3 for the representative substructures is set to 0.05 and, hence, the corresponding threshold τ_α is 3.841.

As it can be noticed, *Synth1* exhibits the maximum value of Γ among the synthetic data sets and, by contrast, *Synth2* exhibits the lowest. Indeed, documents in *Synth2* share several features and, thus, are very homogeneous. According to the above interpretation of low Γ values, the representative tree patterns only cover those small fragments, that really discriminate among clusters.

5.7. Scalability

To evaluate the scalability of GENERATE-HIERARCHY, we used the DTDs for *Synth1* and produced respectively 100, $1,000$, $10,000$ and $100,000$ documents with 2, 4, and 8 clusters. The results reported in fig. 9 reveal that the algorithm scales linearly in the number of both documents and clusters to process large-scale databases of XML trees.

An attempt was made to study the scalability of *SGrace* and *XRep* for comparison. However, such competitors revealed unable to successfully complete the heaviest tests involving $10,000$ and $100,000$ XML documents. Nonetheless, one can still gain an insight into the limited scalability of *SGrace* and *XRep* by considering the running times in table 1 taken to process the largest data collections (such as *Synth2*).

Another comparison can be made with respect to the scalability of *XProj* reported in [16]. The proposed clustering method outperforms *XProj*, since its scalability is orders of magnitude higher than the one of *XProj*. This result is strengthened by two additional considerations. The scalability of XProj was tested over replicated XML data sets, which would further expedite Generate-Hierarchy. Moreover, Generate-Hierarchy requires one input value, i.e., the significance threshold, whose typical settings are well known (i.e., 95% or 99%). On the contrary, XProj requires, for each data set, a complex and specific parameter-tuning.

6. Related works

Clustering XML documents by structure has its root in the problem of inferring structural similarities for the purpose of comparing semi-structured data.

In this section, we overview some seminal works available from the literature, that are most closely related to ours. Emphasis is on reviewing major results, concerning the evaluation of structural similarity, clustering and summarization.

6.1. Structural similarity

The evaluation of structural similarity has been investigated from different perspectives, e.g., in the context of change detection [11, 23, 46], or with the purpose of characterizing a document with respect to a given DTD [6].

The approach in [36] instead measures structural similarity via an XML-aware edit distance and applies a standard hierarchical clustering algorithm to evaluate how closely cluster documents correspond to their respective DTDs.

Apart from their effectiveness in the targeted applicative domains, most of these methods are based on the concept of tree edit distance [39, 40, 43, 44, 50] and use graph-matching algorithms to calculate a (minimum cost) edit script that contains the updates necessary to transform a document into another. From a computational point of view, computing tree edit distances turns out to be highly time-expensive, being at least quadratic in the number of elements between any two XML documents, since the similarity of each pair of tags (or element names) of two XML documents must be considered. Another disadvantage of the attempts at detecting structural similarity through tree edit distance is that its computation basically consists of insertion, removal and update operations, that are performed on the individual nodes of the XML trees.

A remedy to this latter issue was introduced in [9, 23], which presents heuristic tree-editing algorithms capable to perform move and copies of whole subtree. This is especially relevant when the size of the involved subtrees is large. However, the quadratic complexity of the tree edit distance still remains.

To overcome the issues concerning tree-edit distance, an incremental clustering technique is proposed in [33] for grouping XML documents on the basis of their structural similarity. The structure of the generic XML document is represented according to the so-called *level structure*, which preserves the names of the document elements along with their occurrences and levels in the tree structure. Clusters of XML documents are also represented by means of level structures. Precisely, each level of the structure associated to a cluster contains a collection of elements at the same level for all documents in the cluster. The *level similarity* is then used to measure the structural similarity between the level structures of two objects (an object can either a single XML documents or a cluster of XML documents). Level similarity measures the occurrences of common elements in each corresponding level. Elements at different levels are assigned different weights, due to the assumption that the documents with structural differences in the higher levels are likelier to belong to distinct clusters. Hierarchical relationships of elements are also considered by counting occurrences of common elements sharing common ancestors.

The detection of structural similarities can draw upon tree matching techniques. For instance, the problem of matching two trees is addressed in [37] by constructing an association graph based on the graph-theoretic concept of connectivity. The authors show that finding maximal cliques in the association graph is equivalent to finding maximal subtree isomorphisms.

A rather different approach has been recently proposed in [24]. Here, the structure of an XML document is represented as a time series, in which each occurrence of a tag corresponds to an impulse and the degree of similarity among documents is computed by analyzing the frequencies of the corresponding Fourier transform. The overall cost of this method is $O(N\,logN)$, where N here denotes the size of the larger document.

An approximation of structural similarity based on the shingles technique is proposed in [7] The idea is to reduce the paths in a document to hash values, which can thus be compared to those of another document using set union and set intersection operators. Although this approach is more efficient than [24], in the worst case it is not linear.

The approach in [29] extracts the structural information (either tags, pairwise, paths, or family order) from an XML document and then exploits the entropy between the structural data as a measure of similarity. This approach has linear complexity $O(N)$, where N is the number of tags in both documents.

Unlike the foregoing works, in the proposed approach, we do not directly compute pairwise structural similarity between XML trees. Rather, a cluster of XML documents is projected over a high-dimensional feature space, wherein the presence/absence of certain structural components within the individual XML tree is explicitly represented. The transactions corresponding to the XML trees in the original cluster is then partitioned into child clusters, by isolating groups of such transactions sharing discriminatory co-occurrences of (combinations of) structural components.

6.2. Clustering by structure and summarization

Both hierarchical and partitioning clustering technique have been developed for clustering XML documents by structure.

Hierarchical clustering has been largely adopted [17, 21, 25, 26], because of the high quality of its results.

XRep [17] is an adaptation of the agglomerative hierarchical algorithm. Initially, each XML tree is placed in its own cluster. The algorithm then walks into an iterative step, in which the least dissimilar clusters are merged. Cluster merging halts when an optimal partition (i.e. a partition whose intra-distance within clusters is minimized and inter-distance between clusters is maximized) is reached. *XRep* is parametric to the notions of *distance measure* and *cluster representative*. The distance between two (clusters of) XML trees is measured as the proportion of non-overlapping root-to-leaf paths in their associated representatives. The representative of a cluster of XML trees is an XML tree which effectively synthesizes the most relevant structural features of the XML trees in the cluster. A cluster representative is essentially obtained as the outcome of a proper overlapping among the XML documents in a cluster, which is accomplished through suitable tree *matching*, *merging* and *pruning* techniques.

XClust [21] is a schema integration strategy that uses agglomerative clustering to reconcile similar DTDs in the same clusters. The similarity between two DTDs is computed by evaluating the similarity between the elements of the corresponding DTD trees. This is accomplished by exploiting a computationally expensive method, that takes into account the semantic similarity of two elements (i.e., the similarity between the names, constraints and ancestors of the two elements) as well as their child- and leaf-context similarity. Moreover, to simplify the computation of similarity in the case of DTD elements with auxiliary AND-OR nodes, the original DTDs need be suitably simplified, which causes some loss of structural information.

S-GRACE [26] is a hierarchical clustering algorithm, that groups XML documents according to the structural information in their associated *s-graphs*. The notion of s-graph denotes a minimal summary of edge containment in one or multiple XML trees. More precisely, given a cluster of XML trees, the associated s-graph is a directed graph whose nodes and edges correspond, respectively, to the nodes (as well as attributes) and parent-child edges of the XML trees in the cluster. The distance between two clusters is defined as the the proportion of non-overlapping edges in the associated s-graphs.

A methodology devoted to the hierarchical clustering of XML documents by structure is presented in [25]. The idea is summarize the original XML trees into suitable summaries, that maintain the original structural relationships of the XML trees and still reduce their redundancy (i.e., the repetition and nesting of nodes). The removal of redundancy allows to more accurately evaluate the similarity between the structural summaries of any two XML trees. A new metric is proposed for this purpose, that is defined on the tree edit distance of the structural summaries. The latter is efficiently computed by means of a suitable dynamic-programming algorithm, that allows node replacements and restricts node insertion and deletion to leaf nodes. Single-link hierarchical-clustering is the reference scheme for grouping the (structural summaries of the) XML trees. The most appropriate clustering level for the single-link hierarchies is determined by exploiting a C-index.

A major criticism to the proposals in [17, 21, 25, 26] is that the adoption of a hierarchical clustering scheme makes such approaches impractical when processing large-scale databases of XML trees, being basically the time requirement of hierarchical clustering $O(N^2)$, with N representing the size of the available collection $\mathcal{D}$ of XML trees to be clustered (e.g., the time complexity of *S-GRACE* is $O(NV^2 + M^3)$ where V and M are, respectively, the number of distinct nodes and s-graphs in $\mathcal{D}$).

As far as cluster summarization is concerned, the representative introduced in [17] actually catches all structural properties in a cluster of XML trees. However, it is computationally expensive, both in time and space. In particular, its time complexity is proportional to the product of the number of nodes in the representatives associated to the two least dissimilar clusters to be merged in the hierarchy.

A main limitation with the notion of s-graph [26] is the loose-grained similarity which occurs. Indeed, two XML trees can share the same prototype s-graph and still have significant structural differences, such as in the hierarchical relationship between nodes. This has an undesirable effect, i.e., that structurally heterogeneous XML trees might be placed within a same cluster, with a consequent degrade of clustering quality.

No emphasis is paid in [25] on providing an intelligible description of the discovered single-link clusters. These correspond to the remaining connected components of an original graph formed through the pairwise connection of the structural summaries and the comprehension of their properties requires further in-depth customized investigations.

A similar difficulty at interpreting the outcome of clustering is faced with [21], that does not provide any summarization of the obtained results.

XProj [16] is a partitioning method that initially divides the available XML documents into k random clusters. Each such a cluster is equipped with a representative, i.e., a collection of substructures with a fixed number n of nodes, that frequently occur in the cluster. Henceforth, the algorithm reiterates the relocation of the XML documents. These are individually reassigned to the clusters with the best matching representative substructures. Each single relocation is followed by the re-computation of the frequent representative substructures within the individual clusters. Relocation eventually halts when either the average intra-cluster structural cohesiveness does not significantly increase or a (pre-specified) maximum number of relocations has been performed.

A main shortcoming of *XProj* is the presence of many input parameters that require a careful tuning, namely the number k of clusters, the size n of the frequent substructures in a cluster representative and the minim frequency threshold for the substructures themselves.

Also, in *XProj*, the notion of cluster representative is functional to some degree of cohesiveness of the intermediate clusters of XML documents. It is not meant for providing an understanding of their structural properties. This justifies two strong approximations: the inclusion in the representatives of structures only with a fixed size n (with a consequent loss of structural information from the corresponding clusters), that may also be unrepresentative (i.e., such that their edge sequences are subsequences of the edge representations of the XML documents in the clusters, although the structures themselves are not substructures of the XML trees).

In contrast to previous work, the devised approach does not rely on fixed reference structures (such as summaries and s-graphs) to partition XML documents. Rather, it takes into account any number of tree-like substructures occurring in the XML documents at hand to guide the clustering process in the derivation of a hierarchy of nested clusters.

Another advantage of our approach is that it does not require a complex parameter tuning: testing the statistical representativeness of the substructures during cluster summarization involves the specification of an input significance threshold, whose usual settings are well-known values, i.e., 95%, or 99%.

Finally, multi-stage clustering efficiently can be scale to process very large databases of XML documents and provides an intelligible understanding of the structural properties of the clusters in the hierarchy.

7. Conclusions and Further Work

Two aspects must be taken into account in the process of clustering XML documents by structure.

Primarily, looking for the occurrence of pre-specified forms of structural components in the structures of the XML documents may be ineffective, when such forms are not the most appropriate ones or do not accord with the structural peculiarities of the XML documents themselves.

Secondarily, the XML documents can share multiple types of structural components. In such cases, it is likely that focusing on one type of structural components may produce clusters of XML documents, whose structures still exhibit meaningful differences due to further neglected forms of structural components.

To overcome the above limitations, a new approach to clustering XML documents by structure was proposed. The approach is hierarchical and, thus, capable (if necessary) to progressively separate a collection of XML documents, by looking (along the paths from the root to the leaves of the resulting cluster hierarchy) at the occurrence in their structures of various user-supplied types of structural components. Essentially, at any level of the hierarchy, each cluster is divided by considering some specific type of structural components (unaddressed at the preceding levels), that still differentiate the structures of the XML documents in that cluster.

Furthermore, each cluster in the hierarchy is also summarized through a novel technique, that provides a clear and differentiated understanding of its structural properties.

A comparative evaluation proved that the devised approach outperforms established competitors in effectiveness, scalability. Cluster summarization was also empirically proved to be very representative.

Ongoing research aims to incorporate the analysis of content features for effective clustering. This is especially useful in those applicative domains in which the XML documents share an undifferentiated structure.

References

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, 2000.

[2] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

[3] R. A. Baeza-Yates, N. Fuhr, and Y.S. Andamaarek. Special issue on xml retrieval. *ACM Transactions on Information Systems*, 24(4), 2006.

[4] R. Baumgartener, S. Flesca, and G. Gottlob. Visual web information extraction with lixto. In *Proc. of Int. Conf. on Very Large Databases (VLDB)*, pages 119 – 128, 2001.

[5] S. Bergamaschi, S. Castano, and M. Vincini. Semantic integration of semistructured and structured data sources. *SIGMOD Record*, 28(1):54–59, 1999.

[6] E. Bertino, G. Guerrini, and M. Mesiti. A matching algorithm for measuring the structural similarity between an xml document and a dtd and its applications. *Information Systems*, 29(1):23 – 46, 2004.

[7] D. Buttler. A short survey of document structure similarity algorithms. In *Proc. of Int. Conf. on Internet Computing*, 2004.

[8] E. Cesario, G. Manco, and R. Ortale. Top-down parameter-free clustering of high-dimensional categorical data. *IEEE Transanction on Knowledge and Data Engineering*, 19(12):1607 – 1624, 2007.

[9] S. Chawathe and H. Garcia-Molina. Meaningful change detection in structured data. *ACM SIGMOD Record*, 26(2):26–37, 1997.

[10] S. S. Chawathe. Comparing hierarchical data in external memory. In *Proc. of Inf. Conf. on Very Large Databases (VLDB)*, pages 90 – 101, 1999.

[11] G. Cobena, S. Abiteboul, and A. Marian. Detecting changes in xml document. In *Proc. of Int. Conf. on Data Engeneering (ICDE)*, pages 41 – 52, 2002.

[12] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *Proc. of Int. Conf. on Very Large Databases (VLDB)*, pages 109–118, 2001.

[13] L. Denoyer and P. Gallinari. Report on the xml mining track at inex 2007: Categorization and clustering of xml documents. *ACM SIGIR Forum*, 42(1):22–28, 2008.

[14] L. Denoyer and P. Gallinari. Overview of the inex 2008 xml mining track. In *Advances in Focused Retrieval*, pages 401–411, 2009.

[15] A. Deutsch, M. Fernandez, and D. Suciu. Storing semistructured data with stored. In *Proc. of ACM SIGMOD Int. Conf. on Management of Data (SIGMOD)*, pages 431 – 442, 1999.

[16] C. C. Aggarwal et al. Xproj: A framework for projected structural clustering of xml documents. In *Proc. of SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, pages 46 – 55, 2007.

[17] G. Costa et al. A tree-based approach to clustering xml documents by structure. In *Proc. of Int. Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, pages 137 – 148, 2004.

[18] G. Demartin et al. Report on the xml mining track at inex 2008: Categorization and clustering of xml documents. *ACM SIGIR Forum*, 43(1):17–36, 2009.

[19] J. Shanmugasundaram et al. Relational databases for querying xml documents: Limitations and opportunities. In *Proc. of Int. Conf. on Very Large Databases (VLDB)*, pages 302 – 314, 1999.

[20] K. Abe et al. Efficient substructure discovery from large semi-structured data. In *Proc. of Siam Conf. on Data Mining (SDM)*, pages 158 – 174, 2002.

[21] M. L. Lee et al. Xclust: Clustering xml schemas for effective integration. In *Procs. ACM Conf. on Information and Knowledge Management (CIKM)*, pages 292 – 299, 2002.

[22] M. N. Garofalakis et al. Xtract: A system for extracting document type descriptors from xml documents. In *Proc. of Int. Conf. on Management of Data (SIGMOD)*, pages 165 – 176, 2000.

[23] S. Chawathe et al. Change detection in hierarchically structured information. In *Proc. of Int. Conf. on Management of Data (SIGMOD)*, pages 493 – 504, 1996.

[24] S. Flesca et al. Fast detection of xml structural similarity. *IEEE Transanction on Knowledge and Data Engineering*, 17(2):160 – 175, 2005.

[25] T. Dalamagas et al. A methodology for clustering xml documents by structure. *Information Systems*, 31(3):187 – 228, 2006.

[26] W. Lian et al. An efficient and scalable algorithm for clustering xml documents by structure. *IEEE Transanction on Knowledge and Data Engineering*, 16(1):82 – 96, 2004.

[27] T. Fiebig, S. Helmer, C. Kanne, G. Moerkotte, J. Neumann, R. Schielle, and T. Westmann. Anatomy of a native xml base management system. *VLDB Journal*, 11(4):292–314, 2002.

[28] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. Cluster validity methods. *SIGMOD Record*, 31(1-2), 2002.

[29] S. Helmer. Measuring the structural similarity of semistructured documents using entropy. In *Proc. of Int. Conf. on Very Large Databases (VLDB)*, pages 1022 – 1032, 2007.

[30] H. Jagadish, S. Al-Khalifa, A. Chapman, L. Lakshmanan, A. Nierman, S. Paparizos, J. Patel, D. Srivastava, N. Wiwatwattana, Y. Wu, and C. Yu. Timber: A native xml database. *VLDB Journal*, 11(4):274–291, 2002.

[31] A.K. Jain and R.C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, 1988.

[32] J. Mchugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A database management system for semistructured data. *ACM SIGMOD Record*, 26(3):54–66, 1997.

[33] R. Nayak. Fast and effective clustering of xml data using structural information. *Knowledge and Information Systems*, 14:197–215, 2008.

[34] R. Nayak, C. M. De Vries, S. Kutty, S. Geva, L. Denoyer, and P. Gallinari. Overview of the inex 2009 xml mining track: Clustering and classification of xml documents. In *Focused Retrieval and Evaluation*, pages 366–378, 2010.

[35] S. Nestorov, S. Abiteboul, and R. Motwani. Extracting schema from semistructured data. In *Proc. of ACM SIGMOD Int. Conf. on Management of Data (SIGMOD)*, pages 295 – 306, 1998.

[36] A. Nierman and H. V. Jagadish. Evaluating structural similarity in xml documents. In *Proc. of Int. Workshop on the Web and Databases (WebDB)*, pages 61–66, 2002.

[37] M. Pelillo, K. Siddiqi, and S. W. Zucker. Matching hierarchical structures using association graphs. *IEEE Transactions on Pattern Analisys and Machine Intelligence*, 21(11):1105–1119, 1999.

[38] H. Schoning. Tamino - a dbms designed for xml. In *Proc. of Int. Conf. on Data Engineering (ICDE)*, pages 149–154, 2001.

[39] S. Selkow. The tree-to-tree editing problem. *Information Processing Letters*, 6(6):184–186, 1977.

[40] D. Shasha and K. Zhang. *Pattern Matching in Strings, Trees, and Arrays.* Oxford University Press, 1995.

[41] P. Shenoy, J.R. Haritsa, S. Sudarshan, G. Bhalotia, M. Bawa, and D. Shah. Turbo-charging vertical mining of large databases. *ACM SIGMOD Record*, 29(2):22 – 33, 2000.

[42] T. Shimura, M. Yoshikawa, and S. Uemura. Storage and retrieval of xml documents using object-relational databases. In *Proc. of Int. Conf. on Database and Expert Systems Applications (DEXA)*, pages 206 – 217, 1999.

[43] K.C. Tai. The tree-to-tree correction problem. *Journal of the ACM*, 26(3):422–433, 1979.

[44] J. Wang, K. Zhang, K. Jeong, and D. Shasha. A system for approximate tree matching. *IEEE Transanction on Knowledge and Data Engineering*, 6(4):559–571, 1994.

[45] K. Wang and H. Liu. Discovering typical structures of documents: A road map approach. In *Proc. of ACM SIGIR Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 146 – 154, 1998.

[46] Y. Wang, D.J. DeWitt, and J. Cai. X-diff: A fast change detection algorithm for xml documents. In *Proc. of Int. Conf. on Data Engeneering (ICDE)*, pages 519 – 530, 2003.

[47] M. J. Zaki. Scalable algorithms for association mining. *IEEE Transanction on Knowledge and Data Engineering*, 12(3):327 – 390, 2000.

[48] M. J. Zaki. Efficiently mining frequent trees in a forest: Algorithms and applications. *IEEE Transanction on Knowledge and Data Engineering*, 17(8):1021 – 1035, 2005.

[49] M. J. Zaki and K. Gouda. Fast vertical mining using diffsets. In *Proc. of SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, pages 326 – 335, 2003.

[50] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6):1245 – 1262, 1989.