A general framework for XML Document Clustering

Francesco De Francesca[2], Gianluca Gordano[2], Giuseppe Manco[1],
Riccardo Ortale[2], and Andrea Tagarelli[2]

[1] ICAR-CNR – Institute of Italian National Research Council
Via Bucci 41c, 87036 Rende (CS), Italy
e-mail: manco@icar.cnr.it
[2] DEIS, University of Calabria
Via Bucci 41c, 87036 Rende (CS), Italy
e-mail: {ortale, tagarelli}@si.deis.unical.it

Abstract. A novel methodology for clustering XML documents is discussed. The underlying idea is grouping documents which exhibit structural similarities. To this purpose, a suitable technique for identifying meaningful matchings among the nodes of two XML document trees is investigated. The proposed technique also allows to associate to each set of related documents a single prototype XML document, i.e. a representative subsuming the most relevant features of the documents in the set. Suitable techniques for both building and refining cluster-specific representatives are analyzed. Some initial experimental results show the effectiveness of our approach.

1 Introduction

The increasing relevance of the Web as a mean for sharing information around the world has posed several new interesting issues to the computer science research community. The traditional approaches to information handling are ineffective in the new context: they are mainly devoted to the management of highly structured information, like relational databases, whereas Web data are semistructured and encoded using different formats (HTML, XML, email messages and so on).

In such context, we address the problem of clustering structurally similar Web documents, and in particular XML documents. This problem has several interesting applications, related, e.g., to the management of Web data. For example, the detection of structural similarities among documents can help in solving the problem of recognizing different sources providing the same kind of information [2], or in the structural analysis of a Web site. A further interesting application domain is related to query processing in semistructured data. Grouping semistructured documents according to their structural homogeneity can help in devising indexing techniques for such documents, thus improving the construction of query plans.

The problem of measuring the similarity among structured documents has been addressed in the literature mainly from the point of view of change detection. Indeed, several methods for detecting the similarity of XML documents [5, 4, 3] have been recently proposed. However, most of these methods are based on tree/graph-matching algorithms and they are more concerned with the changes occurring within the contents of the documents rather than their structural similarities. [7] proposes a fast and effective technique for detecting structural similarities between XML documents by representing the structure of an XML document as a time series in which each occurrence of a tag corresponds to an impulse. The degree of similarity between documents is computed by analyzing the frequencies of the corresponding Fourier transform.

The problem of XML document clustering is tackled in [8] as an equivalent problem of graph partitioning, in which each node represents a document and edges represent links between documents. The aim is to partition such a graph in order to visualize it on a density map which shows clusters of similar documents, with a color corresponding to the density of documents in each square (pixel) of the map. A classic partitional clustering technique is exploited in [6]. Here, a XML document is represented in a vector-space model by using three different kind of features: textual features (i.e., words), tag-based features and their combination.

The detection of structural similarities can draw upon tree matching techniques. For instance, in [11] the identification of matchings between two relational structures is reduced to the equivalent problem of finding a maximal clique within an association graph. However, it is not clear how such an approach can be applied to hierarchically organized graphs such as trees, since in these cases maximal cliques do not preserve the structural ordering.

In this paper we propose a novel methodology for clustering XML documents, focusing on the notion of *XML cluster representative*, i.e., a prototype XML document subsuming the most relevant features of the set of XML documents within the cluster. In particular, we devise a technique to compute a representative of a set of XML documents, which is capable of capturing all the structural specificities within the represented documents.

To this purpose, the notion of *structural matching* between the trees associated to two XML documents is exploited. Structural matchings allow to both identify the structural similarities between two XML documents and to build a representative around these similarities. We also investigate the exploitation of *merging* and *pruning* strategies for refining XML document trees into effective cluster representatives.

2 Preliminaries on tree matching

Depending on the specific application domain, the notion of tree matching can be defined in a variety of ways. Here it is exploited in the context of XML trees, i.e. trees resulting from a hierarchy of XML tags, to the purpose of highlighting *common skeletons*. Given two XML documents, a common skeleton is a substructure belonging to both the XML trees: precisely, it is defined as a collection of tags which all exhibit the same name, depth level and parent node. Some definitions at the basis of our approach are provided next.

A tree t is a tuple $t = \langle r_t, V_t, E_t, \delta_t \rangle$ where $V_t \subseteq \mathbb{N}$ is the set of nodes, $E_t \subseteq V_t \times V_t$ is the set of edges, r_t is the root node of t, and $\delta_t : N_t \mapsto \Sigma$ is a node labelling function where Σ is an alphabet of node labels. In addition, we denote by $depth_t(v)$ the depth of a node v in t.

Definition 1 (meaningful matching). *Given two XML trees t_1 and t_2, and $v \in V_{t_1}$, $w \in V_{t_2}$, a* meaningful matching *(henceforth called $m(v, w)$) holds if the following conditions inductively hold:*

- *$v = r_{t_1}$, $w = r_{t_2}$ and $\delta_{t_1}(v) = \delta_{t_2}(w)$;*
- *otherwise, $\delta_{t_1}(v) = \delta_{t_2}(w)$, $depth_{t_1}(v) = depth_{t_2}(w)$ and $m(a, b)$ holds, where $(a, v) \in E_{t_1}$ and $(b, w) \in E_{t_2}$.*

□

Notice that, in general, multiple matchings may occur when a node in a tree has a meaningful matching with more than one node of a different tree. Formally, given two trees t_1 e t_2, a node $v \in V_{t_1}$ has a *multiple matching* if $\exists w', w'' \in V_{t_2}$ s. t. both $m(v, w')$ and $m(v, w'')$ hold.

The definition below introduces a criterion exploited to overcome the multiple matching problem. The idea is avoiding to take into account those meaningful matchings which are not effectively indicative of strong structural similarities. The relevance of a meaningful matching is evaluated through the weighting function $w_m : V \times V \mapsto \mathbb{N}$, for a set V of nodes.

Definition 2 (match weighting function). *Given two nodes $v \in V_{t_1}$ e $w \in V_{t_2}$, the weight associated with $m(v, w)$ is computed as:*

$$w_m(v,w) = \begin{cases} 0 & \text{if } \delta_{t_1}(v) \neq \delta_{t_2}(w) \\ 1 + sum_{v,w} & \text{otherwise} \end{cases}$$

□

where $sum_{v,w}$ is recursively defined as the sum of the relevance degrees associated to the best matchings between the descendants of v and w. Function w_m weights the prominence of a meaningful matching in such a way that the higher its value, the more relevant the matching itself.

The set of meaningful matchings represent the common paths between two trees. A tree containing only these common paths is defined as a *matching tree*, since it resembles the intersection of the original trees.

Definition 3 (matching tree). *A tree t is a* matching tree *(henceforth called t_m) for two trees t_1, t_2 if $\forall v \in V_t$, $\exists! v_1 \in V_{t_1}$ and $\exists! v_2 \in V_{t_2}$ s. t. there holds $m(v_1, v_2)$. A matching tree t_m is* optimal *if its size is maximal, i.e., if for each matching tree $u_m \neq t_m$, we have $|t_m| \geq |u_m|$.* □

3 Problem Statement

Suitable clustering algorithms for semistructured documents were extensively studied in the current literature [10]. Hierarchical methods are widely known as providing clusters with a better quality, especially for text datasets [1, 12]. Fig. 1 shows *XRep*, an adaptation of an agglomerative hierarchical algorithm to our problem. Initially each XML tree (derived by parsing the corresponding XML document) is placed in its own cluster, and a matrix containing the pair-wise tree distance is computed. Next, the algorithm walks into an iterative step in which the least dissimilar clusters (evaluated on the basis of their cluster representatives) are merged. As a consequence, the distance matrix is updated to reflect this merge operation. The overall process is stopped when an optimal partition (i.e., a partition whose intra-distance within clusters is minimized and inter-distance between clusters is maximized) is reached.

We address the problem of clustering XML documents in a parametric way. More precisely, the general schema of XRep algorithm is parametric w.r.t. the concepts of distance measure and cluster representative. Viewed in this respect, we need to investigate the conditions for the optimality of a cluster representative, once a suitable distance measure is given.

Intuitively, a representative of a cluster of XML documents is an XML document which effectively reflects all the structural contents within the original cluster. The notion of representative in our application domain can be formalized as follows:

Definition 4. *Given a domain $\mathcal{U}$, equipped with a distance function $d : \mathcal{U} \times \mathcal{U} \mapsto \mathbb{R}$, and a set $\mathcal{S} = \{t_1, \ldots, t_n\} \subseteq \mathcal{U}$ of XML document trees, the representative of $\mathcal{S}$ is the tree that minimizes the sum of the distances:*

$$rep(\mathcal{S}) = min_{t \in \mathcal{U}} \sum_{i=1}^{n} d(t_i, t)$$

Input: a set $\mathcal{D} = \{d_1, \ldots, d_n\}$ of XML documents;
Output: a cluster partition $\mathcal{P} = \{C_1, \ldots, C_k\}$ of $\mathcal{D}$.
Method:
parse XML documents in $\mathcal{D}$ to derive the set $\mathcal{S} = \{t_1, \ldots, t_n\}$ of XML trees;
let initially $\mathcal{P} = \{\{t_1\}, \ldots, \{t_n\}\}$, and assume the representative $r_i = t_i$ for each t_i;
compute a tree-distance matrix M_d, where $M_d(i, j) = d(t_i, t_j)$;
repeat
choose clusters C_i and C_j such that $d(C_i, C_j)$ is minimized;
compute the representative for cluster $C = C_i \cup C_j$;
$\mathcal{P} := \mathcal{P} - \{C_i, C_j\} \cup C$, update M_d;
until $\mathcal{P}$ **has maximal quality**;

Fig. 1. The XRep algorithm for clustering of XML documents

□

The core of the *XRep* algorithm is the computation of the XML cluster representative through three main steps (each of which will be discussed in the next section):

1. Tree matching – This step suitably combines the information related to all meaningful matchings between the XML trees in order to build an optimal matching tree.
2. Tree merging – The optimal matching tree is successively grown into a merge tree by adding to it the uncommon substructures within the original trees.
3. Pruning of merge tree – The focus here is pruning merge tree according to a strategy which aims at removing the least frequent nodes: this allows to obtain a representative as an XML tree.

The notion of proximity, between two patterns drawn from the same feature space, is essential to the definition of a cluster. We mainly focus on suitably adapting a distance measure originally conceived to deal with strings, namely *Edit distance*.

Edit distance between trees computes the minimum-cost sequence of operations required to convert one given tree to another [13]. We adopt the definition of tree edit distance proposed in [9]. Given two trees t_1 and t_2, the edit operations are the insertion of a node $w \in V_{t_2}$ in t_1, and the deletion of leaf nodes from t_1.

The idea of exploiting other distances, such as those originally conceived to deal with binary-valued features, is also appealing. The standard *Jaccard similarity* [10] is defined as

$$s_J(x, y) = \frac{a_{11}}{a_{11} + a_{10} + a_{01}}$$

where a_{11} is the number of features occurring within both x and y (i.e., number of features that equal 1), a_{10} (resp. a_{01}) is the number of features occurring only within x (resp. y).

In the context of proximity among XML trees, a number of intuitive way of defining the Jaccard similarity can be provided. For example, a first measure can be straightforwardly defined when the feature space represent the set of labels associated with the nodes in a tree. An alternative (and more refined) definition is given by taking into account the paths in the trees rather than only the node labels. Precisely,

$$s_J(t_i, t_j) = \frac{\varphi_i \cdot \varphi_j}{\varphi_i \cdot \varphi_i + \varphi_j \cdot \varphi_j - \varphi_i \cdot \varphi_j}$$

where φ_i denotes the path vector of a tree t_i, i.e. the generic element $\varphi_i[j]$ is equal to 1 if there exists a path from the root of t_i to the node v_j, 0 otherwise. The dot product between φ_i and φ_j represents

the number of common paths both in (i.e., it corresponds to the coefficient a_{11}), while the dot product between φ_i (resp. φ_j) and itself indicates the number of paths in t_i (resp. t_j).

A definition of *Jaccard distance* can be derived from 1's complement of s_J, i.e. $d_J = 1 - s_J$.

4 Cluster Representative

4.1 XML tree matching

We propose a dynamic-programming algorithm for building the optimal matching tree (henceforth called t_m^*) from two XML trees. The idea is to identify meaningful matchings among the nodes of two XML trees on a *per level* basis. At each level, all meaningful matchings are detected and weighted by means of function w_m. Weights take into account the relevance of meaningful matchings and, as a consequence, allow to overcome multiple matchings.

The bottom-up approach to the construction of t_m^* consists of the following steps: *i*) matching matrix computation, *ii*) removal of multiple matchings, *i*) optimal matching tree construction.

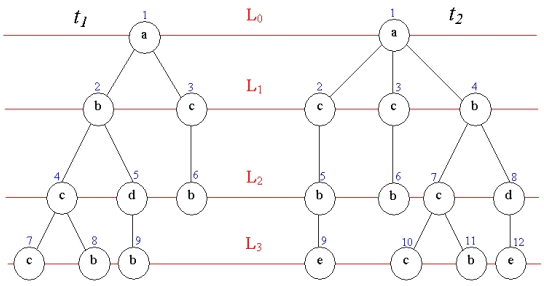

Fig. 2. Tree node enumeration.

Matching matrix computation. Given two XML trees $t_1 = \langle r_{t_1}, V_{t_1}, E_{t_1}, \delta_{t_1} \rangle$ and $t_2 = \langle r_{t_2}, V_{t_2}, E_{t_2}, \delta_{t_2} \rangle$, all meaningful matchings among nodes at the same level within both trees are captured by a matching matrix M_m. M_m has $|V_{t_1}|$ rows and $|V_{t_2}|$ columns. Fig. 2 shows that nodes in a XML tree are numbered on a *per level* basis. This determines the particular block structure of matrix M_m. For each level k, a sub-matrix $M_m(k)$ collects the meaningful matchings among the sets of $|V_{t_1}(k)|$ and $|V_{t_2}(k)|$ nodes at level k respectively belonging to t_1 and t_2.

The computation starts from the last level of the XML trees and iterates until the roots r_{t_1} and r_{t_2} are reached. At the generic iteration, it verifies if there are nodes at the same level in both t_1 and t_2, which exhibit meaningful matchings. In these cases the weight corresponding to each such meaningful matching becomes the value of the entry associated to the matching nodes. Any pair of nodes with no meaningful matching is assigned an entry value 0. Once completed, the computation of a matching matrix guarantees that every meaningful matching has been taken into account (Fig. 3 (a)).

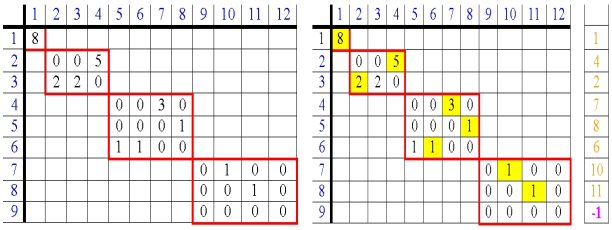

Fig. 3. XML tree matching: (a) matching matrix, (b) marked matching matrix and marking vector

Removal of multiple matchings. A multiple matching is denoted by the presence of at least two non-zero values in some row and/or column of M_m. A node $v_i \in V_{t_1}$ corresponds to the i-th row in M_m. In general, v_i can match with a number of nodes belonging to t_2. Let $J_{v_i} = \{j_1, \ldots, j_h\}$ be the set of column indexes corresponding to these nodes. Formally, v_i exhibits a multiple matching if $|J_{v_i}| > 1$.

Multiple matchings can be eliminated by simply choosing the best matching for each node $v_i \in V_{t_1}$: such a matching corresponds to the column index $j^*_{v_i} = \arg\max_{j_1,\ldots,j_h}\{M_m(i, j_1), \ldots, M_m(i, j_h)\}$. This technique allows the construction of a *marking vector* $V_m = \{j^*_{v_1}, \ldots, j^*_{v_n}\}$, whose generic i-th entry indicates the best matching node in t_2 for v_i (both nodes are obviously taken at the same level). Fig. 3 (b) shows that $V_m[i] = -1$ for all nodes $v_i \in V_{t_1}$ with no meaningful matchings.

Optimal matching tree construction. An optimal matching tree is efficiently built from vector V_m: it suffices that all nodes $v_i \in V_{t_1}$ such that $V_m[i] = -1$ are removed from t_1. These nodes exhibit no meaningful matchings and, as a consequence, are not taken into account.

4.2 Merging of XML trees

This step aims at building an approximation of the actual cluster representative. Starting from t_2 and the above vector V_m, a merge tree $t_{1,2}$ is suitably formed in order to include nodes which reveal to be either common or uncommon to both trees t_1 and t_2: each node is taken into account at most once. The inclusion of non-matching nodes in $t_{1,2}$ is described next. Let v_i be a node in t_1 with no meaningful matchings, i.e. such that $V_m[i] = -1$. If the parent node of v_i matches with a node $w_j \in V_{t_2}$, i.e. $V_m[\pi(i)] = j$, v_i appears in $t_{1,2}$ as a child of w_j.

Merging the documents in a cluster. Both the techniques for building an optimal matching tree and a merge tree are conceived to work with only two XML documents. We show that this does not determine a loss of generality, since merge trees are associative and, as a consequence, can be associated even to clusters consisting of more than two XML documents. Practically, a first merge tree t is constructed starting from any two XML documents in a given cluster. A new merge tree t' is successively built from t and any other XML document in the cluster not yet taken into account. Such a task is reiterated until all cluster documents are processed. Finally, suitable pruning operations make the resulting merge tree become an effective cluster representative.

The following two theorems show that the merging process is associative with respect to both the Jaccard and Edit distances.

Let t_1, t_2, and t_3 be three XML trees. Their merging permutations provide three different representatives r', r'' and r''', defined respectively as $r' = merge(t_1, merge(t_2, t_3)))$, $r'' = merge(t_2, merge(t_1, t_3)))$, and $r''' = merge(t_3, merge(t_1, t_2)))$.

Theorem 1. $d_J(t_i, r') = d_J(t_i, r'') = d_J(t_i, r'''),\ i = 1, 2, 3.$

Proof. Assume $tag(t)$ denotes the set of tags within a generic tree t. The definition of merge tree implies that $tag(merge(t_2, t_3)) = tag(t_2) \cup tag(t_3)$, $tag(merge(t_1, t_3)) = tag(t_1) \cup tag(t_3)$, $tag(merge(t_1, t_2)) = tag(t_1) \cup tag(t_2)$. Therefore we have that: $tag(r') = tag(t_1) \cup tag(merge(t_2, t_3)) = tag(t_1) \cup (tag(t_2) \cup tag(t_3))$. Since the union operator is associative, we find that $tag(r') = tag(t_1) \cup tag(t_2) \cup tag(t_3) = tag(r'') = tag(r''')$. This means that a merge tree is associative with respect to Jaccard distance. □

Theorem 2. $d_E(t_i, r') = d_E(t_i, r'') = d_E(t_i, r'''),\ i = 1, 2, 3.$

Proof. Since $d_E(t_i, r') = |r'| - |t_i|$, $i = 1, 2, 3$, we have to show that $|r'| = |r''| = |r'''|$ to prove that a merge tree is associative with respect to Edit distance.

Let $t^*_m(t_1, t_2)$ be an optimal matching tree between trees t_1 e t_2. Since $|merge(t_2, t_3)| = |t_2| + |t_3| - |M^*(t_2, t_3)|$, we find that:

$$\begin{aligned}
|r'| &= merge(t_1, merge(t_2, t_3)) \\
&= |t_1| + |merge(t_2, t_3)| - |t^*_m(t_2, t_3)| \\
&= |t_1| + |t_2| + |t_3| - |t^*_m(t_2, t_3)| - |t^*_m(t_1, t_2)| - |t^*_m(t_1, t_3)| + |t^*_m(t^*_m(t_1, t_2), t^*_m(t_1, t_3))| \\
&= |t_1| + |t_2| + |t_3| - |t^*_m(t_2, t_3)| - |t^*_m(t_1, t_2)| - |t^*_m(t_1, t_3)| + |t^*_m(t_1, t_2, t_3)|
\end{aligned}$$

As the same result can be obtained for both r'' and r''', it holds that $|r'| = |r''| = |r'''|$. □

Updating the frequencies of the nodes in a merge tree. The process of merging any two XML trees t_1 and t_2 requires the frequencies of the nodes in the resulting merge tree to be updated. To this purpose, the frequency of a node within an original XML document, such as either t_1 or t_2, is assumed to be 1. A node z belonging to a merge tree has a frequency $freq(z) = freq(v) + freq(w)$, if z corresponds to a meaningful matching between nodes $v \in V_{t_1}$ and $w \in V_{t_2}$. Otherwise it holds that $freq(z) = 0$.

It is worth noticing that a frequency value is associated to the path from the root to the specific node under investigation. In fact, by the definition of meaningful matching, two nodes match if the paths leading from the node themselves to their respective roots are identical. This means that any node in a merge tree cannot have a frequency value greater than that of its parent node. Node frequencies are exploited to suitably prune a merge tree.

4.3 Representative computation

Pruning is at the basis of the refining process which turns a merge tree into an effective cluster representative. Precisely, leaf nodes are inspected for removal from the merge tree one at a time.

Any performed cut minimizes the distance between the refined merge tree and the original document trees in the cluster. The removal of a leaf node from the refined merge tree is reiterated until its distance from the original cluster trees cannot be further decreased. Fig. 4 shows that the pruning technique for building a cluster representative is parametric w.r.t. the notion of distance.

Note that at each iteration the vector of leaf nodes is sorted in increasing order w.r.t. node frequencies. It turns out that such a sorting is essential for an efficient pruning. Intuitively, removal of a node with associated a frequency below $n/2$, where n is the number of documents in a given cluster, leads to a better distance between the representative and more than a half of documents in the cluster.

Input: a cluster $\mathcal{C} = \{t_1, \ldots, t_n\}$ of XML trees, a tree $t = \langle r_t, V_t, E_t, \rangle$ associated with $\mathcal{C}$;
Output: a tree $r = \langle r_r, V_r, E_r, \rangle$, with $r_r = r_t$, $V_r \subseteq N$, $E_r \subseteq E$.
Method:
let initially $r := t$ be the representative of cluster $\mathcal{C}$;
compute $d_0 := \sum_{i=1}^{n} d(t_i, r)$;
let $\mathcal{L} \subseteq N_r$ be the set of leaf nodes in r;
repeat
for each leaf node $v_l \in \mathcal{L}$
compute $r^{(l)} =$ `remove_leaf`(r, v_l);
$l^* = \arg\min_{v_l} [\sum_{i=1}^{n} d(t_i, r^{(l)})]$;
if $(d^* < d_0)$
$V_r := V_r - \{n_l^*\}$;
until $d^* > d_0$;
return r;

Function `remove_leaf` $(r = \langle r_t, V_t, E_t \rangle, v_l)$: t';
$N_{t'} := V_t - \{v_l\}$;
$E_{t'} := V_t - \{e_{lj}\}$ where e_{lj} is the edge $(v_l, v_j) \in E_t$;
return $t' = \langle r_t, V_{t'}, E_{t'} \rangle$;

Fig. 4. Pruning of merge tree for representative computation

5 Evaluation

5.1 Complexity of the approach

The computational cost for the process of building a merge tree (which also involves the previous construction of an optimal matching tree) from two XML document trees, t_1 and t_2, is $\mathcal{O}(hl^3)$, where h represents the number of levels within t_1 and l indicates the maximum number of nodes at any level.

The removal of multiple matchings consists in finding a maximum value in each diagonal block within the matching matrix. Specifically, the cost of eliminating multiple matchings is $\mathcal{O}(rc)$ for any sub-matrix $M_m(k)$ with dimensions $|r| \times |c|$, where r and c represent sets of nodes at level k respectively within trees t_1 and t_2.

Finally, the cost of the whole phase necessary for pruning a merge tree t is $\mathcal{O}(|V_t| \cdot |\mathcal{L}|)$. In details, $\mathcal{O}(|\mathcal{L}|)$ is the cost for each iteration and $|V_t|$ is the maximum number of iterations. It is worth noticing that such a phase directly follows that of building a merge tree. Moreover, its overall computational cost can be reduced by adopting suitable optimizations which, in turn, strictly depend on the specific notion of distance exploited.

5.2 Experimental results

In this section we describe two different kinds of experiments carried out to measure the effectiveness of our approach to clustering XML documents. Tests were performed on synthetic data. Precisely, all XML documents were created by an automatic generator, which extracts multiple documents in conformity with a user-specified DTD [7]. The phase of document generation relies on various statistic models, which together describe the differences within a class of XML documents from a same DTD. Specifically, what makes documents of a same class differ from one another is the presence of operators $*, ?, |, +$ in the element definitions of the original DTD. The idea behind the above generator is associating each occurrence of these non-deterministic operators with a specific parameter modelling a random generation process. In particular, operator $*$ is associated with a stochastic variable representing the length of a

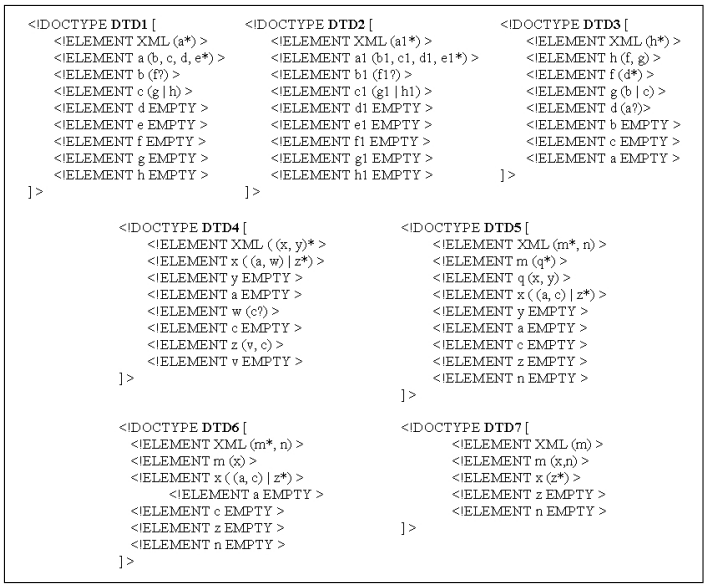

Fig. 5. DTDs used to generate test documents

sequence. Both operators ? and | correspond to a Bernoulli trial with a fixed success probability. In all experiments, Bernoulli variables were randomly chosen from a uniform distribution, whereas any stochastic variable were modelled by a log-normal function whose mean and standard deviation were randomly extracted from a log-normal function distribution with expected values equal to 10 and 2, respectively.

Two different kinds of experiments were performed. Though conceived to address different facets of the effectiveness of the proposed technique, both the experimental settings compare our approach with a traditional agglomerative hierarchical algorithm, namely UPGMA (Unweighted Pair-Group Method using averages). UPGMA computes the similarity between an element and a cluster as the arithmetic average of the similarities between the element and all the objects in the cluster. If the element under investigation is also a cluster, the similarity is taken as the average similarity of all the pairs of objects within the two clusters. The resulting hierarchies are evaluated at each level.

The first set of experiments was carried out to measure the accuracy of our approach in identifying homogeneous groups of XML documents on the basis of their structural similarities. To this purpose, we exploited a synthetic dataset consisting of 60 XML documents, automatically built from 5 distinct DTDs: Fig. 5 shows the exploited document classes respectively labelled DTD1,...,DTD5. These document classes are such that any pair of DTDs exhibits, at the same time, considerable differences and similarities. For instance, DTD1 and DTD2, are characterized by a very similar structure, though having no tag in common.

The second set of experiments was performed on a dataset built around 7 different DTDs. Precisely, 5 DTDs were borrowed from the previous series of tests, while two entirely new document classes, namely DTD6 and DTD7, were added. Fig. 5 shows both these documents classes. The peculiarity of both DTD6 and DTD7 consists in having a structure which is very similar to that of DTD5. 100 XML documents

documents	classes	distance	algorithm	clusters	precision	recall	F-measure
300	5	Edit	UPGMA	5	0.6	0.9	0.72
300	5	Edit	XRep	5	0.785	0.93	0.85
700	7	Edit	UPGMA	11	0.43	0.785	0.63
700	7	Edit	XRep	11	0.53	0.83	0.71

Table 1. Quality results

were automatically generated for each of the above DTDs. The purpose of this series of tests was to evaluate the accuracy with which our approach distinguishes among different degrees of similarity due to a number of similar document classes.

The results obtained were evaluated by measuring the standard *precision* and *recall* measures. These were combined to provide values of the standard *F-measure* [1]. As we can see in Table 1, our approach works better than classical UPGMA algorithm. In particular, we can observe how XRep exhibits better values of recall, whereas it outperforms UPGMA from the precision viewpoint. In other terms, XRep guarantees a significant gain in accuracy w.r.t. the classical UPGMA hierarchical algorithm. However, the main advantage of XRep is more intuitive cluster description, based on our notion of representative.

6 Conclusion and further work

We presented a novel methodology for clustering XML documents, focusing on a notion of *XML cluster representative* which is capable of capturing all the structural specificities within a collection of XML documents. By exploiting the tree nature of XML documents, we provided suitable criteria of structural tree-matching and merging. In particular, we proposed a dynamic-programming algorithm for building an optimal matching tree from two XML trees by means of the detection of meaningful node-matchings. We also studied properties of trees on associativity to extend merging strategies to sets of XML trees.

Our technique, based on an agglomerative hierarchical clustering algorithm, was tested in some initial experiments showing a high degree of effectiveness. However, more experiments performed on massive datasets need to be evaluated.

Notwithstanding, several issues have still to be addressed. For instance, in order to guarantee more flexibility and efficiency to our approach, we have to study other clustering schemas such as partitional ones. Furthermore, the task of pruning merge trees can be optimized according to the specific notion of distance exploited.

Also, the notion of representative can be relaxed to involve a more flexible structure. Indeed, a single representative can be too limited to include all the relevant features of a group of documents. To this purpose, strategies based on frequent pattern discovery seem to be more suitable. In details, the notion of representative can be enhanced to include a forest of (not necessarily connected) frequent substructures.

References

1. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press Books. Addison Wesley, 1999.
2. S. Bergamaschi, S. Castano, and M. Vincini. Semantic integration of semistructured and structured data sources. *SIGMOD Record*, 28(1):54–59, 1999.

3. E. Bertino, G. Guerrini, and M. Mesiti. Measuring the stuctural similarity among XML documents and DTDs. Technical report, Tech. Report DISI-TR-02-02, Department of Computer Science, University of Genova, 2002. Available at `http://www.disi.unige.it/person/MesitiM`.
4. S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom. Change detection in hierarchically structured information. In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD '96)*, pages 493–504, Montreal, Quebec, June 1996.
5. G. Cobena, S. Abiteboul, and A. Marian. Detecting changes in XML document. In *Proc. 18th Int. Conf. on Data Engineering (ICDE '02)*, 2002.
6. A. Doucet and H. A. Myka. Naive clustering of a large XML document collection. In *Proc. 1st Annual Workshop of the Initiative for the Evaluation of XML retrieval (INEX '02)*, Schloss Dagstuhl, Germany, 2002.
7. S. Flesca, G. Manco, E. Masciari, L. Pontieri, and A. Pugliese. Detecting structural similarities between XML documents. In *Proc. 5th Int. Workshop on the Web and Databases (WebDB '02)*, Madison, Wisconsin, 2002.
8. D. Guillaume and F. Murtagh. Clustering of XML documents. *Computer Physics Communications*, 127:215–227, 1989.
9. Carsten Isert. The editing distance between trees. Technical report, Ferienakademie, for course 2: Bume: Algorithmik Und Kombinatorik, Italy, 1999.
10. A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall advanced reference series. Prentice-Hall, 1988.
11. M. Pelillo, K. Siddiqi, and S. W. Zucker. Matching hierarchical structures using association graphs. In *Proc. 5th European Conf. on Computer Vision*, Freiburg, 1998.
12. M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques. In *Proc. ACM SIGKDD Workshop on Text Mining*, 2000.
13. K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262, 1989.