A Tree-based Approach to Clustering XML Documents by Structure

Gianni Costa[1], Giuseppe Manco[1], Riccardo Ortale[2], and Andrea Tagarelli[2]

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

Abstract. We propose a novel methodology for clustering XML documents on the basis of their structural similarities. The idea is to equip each cluster with an *XML cluster representative*, i.e. an XML document subsuming the most typical structural specifics of a set of XML documents. Clustering is essentially accomplished by comparing cluster representatives, and updating the representatives as soon as new clusters are detected. We present an algorithm for the computation of an XML representative based on suitable techniques for identifying significant node matchings and for reliably merging and pruning XML trees. Experimental evaluation performed on both synthetic and real data shows the effectiveness of our approach.

1 Introduction

As the heterogeneity of XML sources increases, the need for organizing XML documents according to their structural features has become challenging. In such a context, we address the problem of inferring structural similarities among XML documents, with the adoption of clustering techniques. This problem has several interesting applications related to the management of Web data. For example, structural analysis of Web sites can benefit from the identification of similar documents, conforming to a particular schema, which can serve as the input for wrappers working on structurally similar Web pages. Also, query processing in semistructured data can take advantage from the re-organization of documents on the basis of their structure. 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 comparing semistructured documents has been recently investigated from different perspectives [5, 14, 4, 3, 8]. Recent studies have also proposed techniques for clustering XML documents. [7] describes a partitioning method that clusters documents, represented in a vector-space model, according to both textual contents and structural relations among tags. The approach in [13] proposes to measure structural similarity by means of an XML-aware

edit distance, and applies a standard hierarchical clustering algorithm to evaluate how closely cluster documents correspond to their respective DTDs.

In our opinion, the main drawback of the above approaches is the lack of a notion of *cluster prototype*, i.e. a summarization of the relevant features of the documents belonging to a cluster. The notion of cluster prototype is crucial in most significant application domains, such as wrapper induction, similarity search, and query optimization. Indeed, in the context of wrapper induction, the efficiency and effectiveness of the extraction techniques strongly rely on the capability of rapidly detecting homogeneous subparts of the documents under consideration. Similarity search can substantially benefit from narrowing the search space. In particular, this can be achieved by discarding clusters whose prototypes exhibit features which do not comply with the target properties specified by a user-supplied query.

To the best of our knowledge, the only approach devising a notion of cluster prototype is [11]. Indeed, the authors propose to compare documents according to a structure graph, *s-graph*, summarizing the relations between elements within documents. Since the notion of s-graph can be easily generalized to sets of documents, the comparison of a document with respect to a cluster can be easily accomplished by means of their corresponding s-graphs. However, a main problem with the above approach relies on the loose-grained similarity which occurs. Indeed, two documents can share the same prototype s-graph, and still have significant structural differences, such as in the hierarchical relationship between elements. It is clear that the approach fails in dealing with application domains, such as wrapper generation, requiring finer structural dissimilarities.

In this paper we propose a novel methodology for clustering XML documents by structure, which is based on the notion of *XML cluster representative*. A cluster representative is a prototype XML document subsuming the most relevant structural features of the documents within a cluster. The intuition at the core of our approach is that a suitable cluster prototype can be obtained as the outcome of a proper overlapping among all the documents within a given cluster. Actually, the resulting tree has the main advantage of retaining the specifics of the enclosed documents, while guaranteeing a compact representation. This eventually makes the proposed notion of cluster representative extremely profitable in the envisaged applications: in particular, as a summary for the cluster, a representative highlights common subparts in the enclosed documents, and can avoid expensive comparisons with the individual documents in the cluster.

The proposed notion of cluster representative relies on the notions of XML tree *matching* and *merging*. Specifically, given a set of XML documents, our approach initially builds an *optimal matching tree*, i.e. an XML tree that is built from the structural resemblances that characterize the original documents. Then, in order to capture all such peculiarities within a cluster, a further tree, called a *merge tree*, is built to include those document substructures that are not recurring across the cluster documents. Both trees are exploited for suitably computing a cluster representative as will be later detailed. Finally, a hierarchical clustering algorithm exploits the devised notion of representative to partition

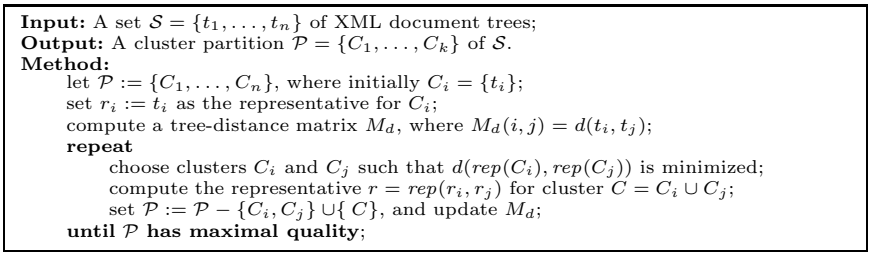


Fig. 1. The *XRep* algorithm for clustering XML documents.

XML documents into structurally homogeneous groups. Experimental evaluation performed on both synthetic and real data states the effectiveness of our approach in identifying document partitions characterized by a high degree of homogeneity.

2 Problem Statement

Clustering is the task of organizing a collection of objects (whose classification is unknown) into meaningful or useful groups, namely *clusters*, based on the interesting relationships discovered in the data. The goal is grouping highly-similar objects into individual partitions, with the requirement that objects within distinct clusters are dissimilar from one another.

Several clustering algorithms [10] can be suitably adapted for clustering semistructured data. We concentrate on hierarchical approaches, which are widely known as providing clusters with a better quality, and can be exploited to generate cluster hierarchies. Fig.1 shows *XRep*, an adaptation of the agglomerative hierarchical algorithm to our problem. Each XML tree (derived by parsing the corresponding XML document) is initially placed in its own cluster, and a pairwise tree distance matrix is computed. The algorithm then walks into an iterative step in which the least dissimilar clusters 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. In this paper, we follow the approach devised in [9], and address the problem of clustering XML documents in a parametric way. More precisely, the general scheme of the *XRep* algorithm is parametric to the notions of *distance measure* and *cluster representative*.

Concerning the distance measure, we choose to adapt the *Jaccard coefficient* [10] to the context of XML trees. A first measure can be straightforwardly defined by considering the feature space representing the set of labels (i.e. tag names) associated with the nodes in a tree: if we denote with $tag(t)$ the set of tag names for a tree t, then we define as $d_J^{(1)}(t_1, t_2) = 1 - \frac{|tag(t_1) \cap tag(t_2)|}{|tag(t_1) \cup tag(t_2)|}$ the Jaccard distance between two trees t_1 and t_2. An alternative (and more refined)

definition is given by taking into account the paths in the trees rather than only the node labels. More precisely, $d_J^{(2)}(t_1, t_2) = 1 - \frac{|path(t_1) \cap path(t_2)|}{\max\{|path(t_1)|, |path(t_2)|\}}$, where $path(t_i)$ denotes the set of paths in t_i, and $path(t_1) \cap path(t_2)$ is the set of common paths between t_1 and t_2.

Intuitively, the representative of a cluster of XML documents is a document which effectively synthesizes the most relevant structural features of the documents in the cluster. The notion of representative can be formalized as follows.

Definition 1. *Given a set $\mathcal{U}$, 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 $\mathcal{S}$ (denoted by $rep(\mathcal{S})$) is the tree t^* that minimizes the sum of the distances:*

$$t^* = rep(\mathcal{S}) \in \mathcal{U} \iff t^* = argmin_{t \in \mathcal{U}} f(t)$$

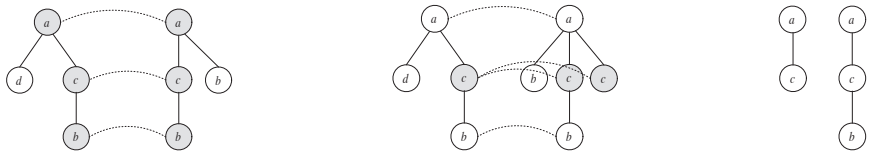
where $f(t) = \sum_{i=1}^{n} d(t_i, t)$. □

The computation of the representative of a set turns out to be a hard problem if the above distance measures are adopted. Therefore we exploit a suitable heuristic for addressing the above minimization problem. Viewed in this respect, our goal is to find a *lower-bound-tree* and an *upper-bound-tree* for the optimal representative. The lower-bound-tree (resp. upper-bound-tree) is a tree on which any node deletion (resp. node insertion) leads to a worsening in function f. Thus, a representative can be heuristically computed by traversing the search space delimited by the above trees. Two alternative greedy strategies can be devised: either a growing approach, which iteratively adds nodes to the lower-bound, or a pruning approach, which iteratively removes nodes from the upper-bound. In the following, we will denote the lower-bound-tree and the upper-bound-tree as *optimal matching tree* and *merge tree*, respectively. Notice that the optimal matching tree represents a stopping condition for the pruning approach, whereas the merge tree is always a sub-optimal solution since it contains the optimal representative. Dually, the merge tree defines a stopping condition for the growing approach, whereas the optimal matching tree is a sub-optimal solution since it is contained in the optimal representative.

We develop a pruning approach in which the computation of an XML cluster representative consists of the following three main stages: the construction of an optimal matching tree, the computation of a merge tree, and the pruning of the merge tree. Fig.3 sketches an algorithm which has been developed according to the above three stages.

3 Mining Representatives from XML Trees

We give now some definitions which are at the basis of our approach. A tree t is a tuple $t = (r_t, V_t, E_t, \lambda_t)$ 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 $\lambda_t : V_t \mapsto \Sigma$ is a node labelling function where Σ is an alphabet of node labels. In particular, we say that an *XML tree* is a tree where Σ is an alphabet of *element tags*. Moreover, let $depth_t(v)$ denote



(a) (b) (c)

Fig. 2. (a) Strong and (b) multiple matching nodes, and (c) their trees.

the depth level of node v in t, with $depth_t(r_t) = 0$, and let $path_t(v) = \langle v_{i_1} = r_t, v_{i_2}, \ldots, v_{i_p} = v \rangle$ denote the list of p nodes that lead up to the node v from the root r_t.

Definition 2 (strong matching). *Given two trees t_1 and t_2, and two nodes $v \in V_{t_1}$, $w \in V_{t_2}$, a* strong matching *$match(v, w)$ between v and w exists if $\lambda_{t_1}(v_i) = \lambda_{t_2}(w_i)$ and $depth_{t_1}(v_i) = depth_{t_2}(w_i)$, for each pair of nodes (v_i, w_i) such that $v_i \in path_{t_1}(v)$ and $w_i \in path_{t_2}(w)$.* □

The above definition states that two nodes, v and w, have a strong matching if v and w together with their respective ancestors share both the same label (i.e. tag name) and depth level. Fig.2(a) displays an example of strong matching among the colored nodes.

The detection of matching nodes between two trees allows the construction of a new tree, called a *matching tree*, which resembles the *intersection* of the original trees.

Definition 3 (matching tree). *Given two trees t_1 and t_2, a tree $t = (r_m, V_m, E_m, \lambda_m)$ is a* matching tree*, denoted by $t = match(t_1, t_2)$, if the following conditions hold:*

1. *there exist two mappings $f_1 : t \mapsto t_1$ and $f_2 : t \mapsto t_2$ associating nodes and edges in t with a subtree in t_1 and t_2;*
2. *for each $u \in V_m$, there exists a strong matching between $v = f_1(u)$ and $w = f_2(u)$ (i.e. $match(v, w)$ holds); moreover, $\lambda_m(u) = \lambda_{t_1}(v) = \lambda_{t_2}(w)$;*
3. *$f_1(r_m) = r_{t_1}$, and $f_2(r_m) = r_{t_2}$; moreover, for each $e = (u, v) \in E_m$, $f_1(e) = (f_1(u), f_1(v))$ and $f_2(e) = (f_2(u), f_2(v))$.* □

Notice that, in general, multiple matchings may occur when a node in a tree has a matching with more than one node in a different tree. More formally, given two trees t_1 and t_2, a node $v \in V_{t_1}$ has a *multiple matching* if $\exists w', w'' \in V_{t_2}$ such that both $match(v, w')$ and $match(v, w'')$ hold. An example of multiple matching between nodes in two trees is shown in Fig.2(b). Multiple matchings trigger ambiguities in defining matching trees: Fig.2(c) represents two alternative matching trees for the documents in Fig.2(b).

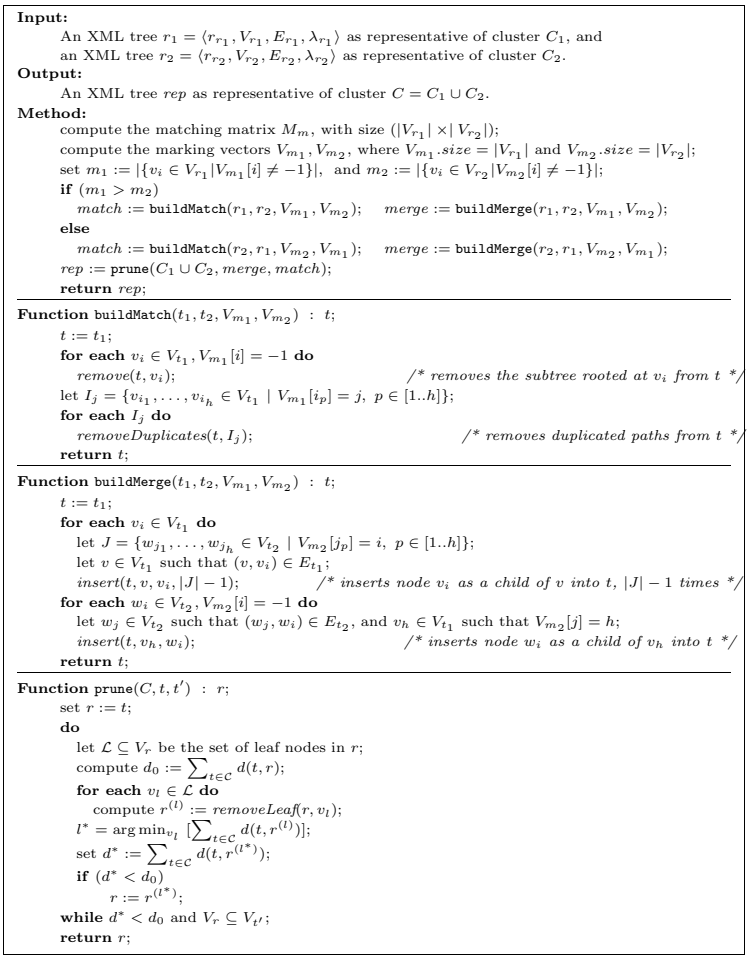
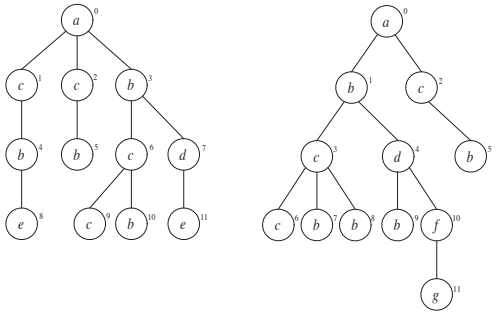


Fig. 3. The algorithm for the computation of an XML cluster representative.

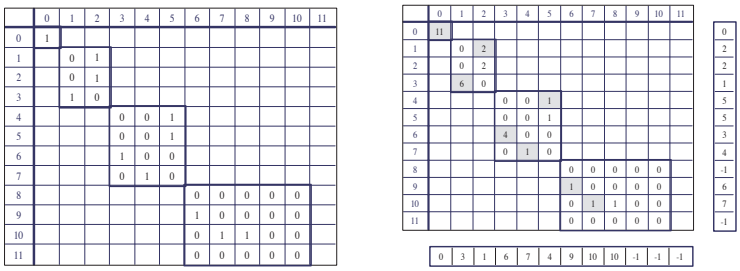
3.1 XML tree matching

In order to capture as many structural affinities as possible, we are interested in finding matching trees with maximal size. Formally, a matching tree $t_m = match(t_1, t_2)$ is an *optimal matching tree* for two trees t_1 and t_2 if there not exists another matching tree $t'_m = match(t_1, t_2) \neq t_m$ such that $|V_{t_m}| \geq |V_{t'_m}|$. We describe a dynamic-programming technique for building an optimal matching tree from two XML trees. The technique consists of three steps: *i*) detection of matching nodes, *ii*) selection of best matchings, and *iii*) optimal matching tree construction.

Matching detection. Given two trees t_1 and t_2, the detection of matching nodes is performed building a $(|V_{t_1}| \times |V_{t_2}|)$ matching matrix M_m. In this matrix,



(a) Examples XML trees t_1 and t_2



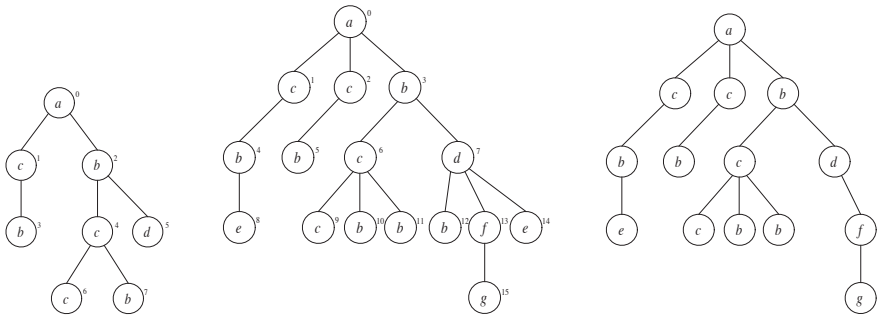
(b) Matching matrix (c) Matching selection

Fig. 4. Data structures for the construction of an optimal matching tree.

the generic (i, j)-th element corresponds to nodes $v_i \in V_{t_1}$ and $w_j \in V_{t_2}$, and contains a weight $\omega_m(v_i, w_j)$ to be associated with the matching between v_i and w_j. Initially, the weight is 1 if $match(v_i, w_j)$ holds, and 0 otherwise. In order to ease the construction of the matching matrix, nodes are enumerated by level, thus guaranteeing a particular block structure for M_m. Indeed, for each level k, a sub-matrix $M_m(k)$ collects the matchings among the nodes in t_1 and t_2 with depth equal to k. Fig.4(a) displays two example XML trees with numbered nodes. The corresponding matching matrix is shown in Fig.4(b).

Selection of best matchings. The problem of multiple matchings can be addressed by discarding those matchings which are less relevant according to the weighting function ω_m. The weight $\omega_m(v, w)$, associated to two matching nodes $v \in V_{t_1}$ and $w \in V_{t_2}$, is computed by taking into account the matches between the children nodes of both v and w. Formally, $\omega_m(v, w) = 1 + \sum_{i,j} \omega_m(v_i, w_j)$, where nodes v_i, w_j are such that $(v, v_i) \in E_{t_1}$ and $(w, w_j) \in E_{t_2}$. Fig.4(c) shows the weights associated with each possible node pair.

Multiple matchings relative to any node of t_1 (resp. t_2) can be detected by checking multiple entries with non-zero values within the corresponding row



(a) (b) (c)

Fig. 5. (a) Lower-bound (optimal matching tree), (b) upper-bound (merge tree), and (c) optimal representative tree relative to the trees of Fig.4(a).

(resp. column) of M_m. We now describe the process for detecting multiple matchings. In the following we focus on the identification of nodes within t_1 that have multiple matchings with those in t_2: the dual situation (i.e. identification of nodes in t_2 having multiple matching with nodes in t_1) has a similar treatment.

Let $v_i \in V_{t_1}$ denote the node corresponding to the i-th row in M_m, and let $J_{v_i} = \{j_1, \ldots, j_h\}$ be the set of column indexes, corresponding to the nodes $w_{j_1}, \ldots, w_{j_h}$ of t_2, such that $M_m(i, j_k) > 0$ (i.e. such that $\omega_m(v_i, w_{j_k}) > 0$), $k = [1..h]$. Thus, v_i exhibits multiple matchings if $|J_{v_i}| > 1$. For each node $v_i \in V_{t_1}$, the *best matching* node corresponds to the column index $j^*_{v_i} = \arg\max_{j_1,\ldots,j_h} \{M_m(i, j_1), \ldots, M_m(i, j_h)\}$. If the maximum in $\{M_m(i, j_1), \ldots, M_m(i, j_h)\}$ is not unique we choose $j^*_{v_i}$ to be the minimum index. The overall best matchings for nodes of t_1 can be easily tracked by using a *marking vector* $V_{m_1} = \{j^*_{v_1}, \ldots, j^*_{v_n}\}$, whose generic i-th entry indicates the node of t_2 with which $v_i \in V_{t_1}$ has the best matching. We set $V_{m_1}[i] = -1$ if the node $v_i \in V_{t_1}$ has no matching. Fig.4(c) shows the marking vectors V_{m_1} and V_{m_2} associated with t_1 and t_2, respectively.
Optimal matching tree construction. An optimal matching tree is effectively built by exploiting the above marking vectors: it suffices that all nodes with no matching are discarded. Fig.5(a) shows the optimal matching tree computed for t_1 and t_2 of Fig.4(a). As we can see in the figure, the optimal matching tree is obtained from t_1 by removing nodes $2, 5, 8, 11$.

3.2 Building a merge tree

The optimal matching tree of two documents represents an optimal intersection between the documents. The notion of *merge tree* resembles an optimized *union* of the original trees. Notice that, firstly an optimal matching tree has to be detected, in order to avoid redundant nodes to be added. Indeed, a trivial merge tree could be simply built as the union of the trees under investigation. Function `buildMerge` in Fig.3 details the construction of a merge tree, which takes

into account nodes discarded while building the optimal matching tree. To this purpose, given two trees t_1 and t_2, we first consider nodes in t_1 having duplicate nodes, and insert such duplicates into the merge tree. Next, nodes in t_2 which do not match with any node in t_1 are added.

Fig.5(b) shows the merge tree associated to the trees of Fig.4(a). Nodes $8, 11$ from t_1 and $9, 10, 11$ from t_2 have no matching, whereas nodes $2, 5$ from t_1 and 8 from t_2 exhibit multiple matchings.

3.3 Turning a merge tree into a cluster representative

An effective cluster representative can be obtained by removing nodes from a merge tree in such a way to minimize the distance between the refined merge tree and the original trees in the cluster. Procedure `prune`, shown in Fig.3, iteratively tries to remove leaf nodes until the distance between the refined merge tree and the original trees cannot be further decreased. It is worth noticing that, on the basis of the definition of procedure `prune`, the representative of a cluster is always bounded by the optimal matching tree built from the documents in that cluster. The correctness of the pruning procedure is established by the following result.

Theorem 1. *Let t_1, t_2 be two XML trees. Moreover, let $t_M = merge(t_1, t_2)$, $t_m = match(t_1, t_2)$ and $t^* = rep(\{t_1, t_2\})$. Then, $t_m \subseteq t^* \subseteq t_M$.* □

Let us consider again the trees t_1 and t_2 of Fig.4(a) and their associated merge tree $merge(t_1, t_2)$ of Fig.5(b). Suppose that t_1 and t_2 belong to the same cluster C. In order to compute the representative tree for C, the pruning procedure is initially applied to the set of leaves $\mathcal{L} = \{5, 8, ..., 12, 14, 15\}$. If we choose to adopt the $d_J^{(2)}$ distance, the procedure computes an initial intra-cluster distance $d_0^C = 5/8$. This distance is reduced to $4/7$ as leaf node 14 is removed. Yet, d_0^C can be decreased by removing node 12. Since at this point no further node contributes to the minimization of d_0^C, the pruning process ends. Fig.5(c) shows the cluster representative resulting from pruning the merge tree in Fig.5(b), with the adoption of the $d_J^{(2)}$ distance.

4 Evaluation

We evaluated the effectiveness of *XRep* by performing experiments on both synthetic and real data. In the former case, we mainly aimed at assessing the effectiveness of our clustering scheme with respect to some prior knowledge about the structural similarities among the XML documents taken into account. Specifically, we exploited a synthetic data set that comprises seven distinct classes of XML documents, where each such class is a structurally homogeneous group of documents randomly generated from a previously chosen DTD. Tests were performed in order to investigate the ability of *XRep* in catching such groups.

To the purpose of assembling a valuable data set, we developed an automatic generator of synthetic XML documents, that allows the control of the degree

of structural resemblance among the document classes under investigation. The generation process works as follows. Given a seed DTD $\texttt{DTD}_0$, a similarity threshold τ, and a number k of classes, the generator randomly yields a set S_τ^k of k different DTDs, hereinafter called class DTDs, that individually retain at most τ percent of the element definitions within $\texttt{DTD}_0$. The k class DTDs are eventually leveraged to generate as many collections of conforming XML documents, on the basis of suitable statistical models ruling the occurrences of the document elements [8].

The seed DTD was manually developed and exhibits a quite complex structure. For the sake of brevity, we only focus on its major features. $\texttt{DTD}_0$ contains 30 distinct element declarations that adopt neither attributes nor recursion. Non empty elements contain at most 4 children. Yet, the occurrences of such elements are suitably defined by exploiting all kinds of operators, namely $+$,$*$,?, and $|$. Finally, the tree-based representation of any XML document conforming to $\texttt{DTD}_0$ has a depth that is equal to 6.

Each test on synthetic data was performed on a distinct set of seven class DTDs, sampled from $\texttt{DTD}_0$, at increasing values of the similarity threshold τ: we chose τ to be respectively equal to 0.3, 0.5, and 0.8.

Real XML documents were extracted from six different collections available on Internet:

- 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 that provides free online press release distribution.
- Sigmod, 51 documents concerning issues of SIGMOD Record. Such documents were obtained from the XML version of the ACM SIGMOD Web site produced within the *Araneus* project [6].
- Wrapper, 53 documents representing wrapper programs for Web sites, obtained by means of the *Lixto* system [2].
- Xyleme_Sample, a collection of 1000 documents chosen from the *Xyleme*'s repository, which is populated by a Web crawler using an efficient native XML storage system [12].

The distribution of tags within these documents is quite heterogeneous, due to the complexity of the DTDs associated with the classes, and to the semantic differences among the documents. In particular, wrapper programs may have substantially different forms, as a natural consequence of the structural differences existing among the various Web sites they have been built on: thus, the skewed nature of the documents in Wrapper should be taken into account. Also, documents sampled from Xyleme exhibit a more evident heterogeneity, since they have been crawled from very different Web sources.

Clustering results were evaluated by exploiting the standard *precision* and *recall* measures [1]. However, in the case of Xyleme_Sample, we had no knowl-

Table 1. Quality results

type	docs	avg size	classes	clusters	τ	precision	recall	F-measure	$\mathcal{IC}$
synth	1400	0.13KB	7	7	0.3	0.979	0.978	0.978	0.219
synth	1400	0.81KB	7	7	0.5	0.802	0.909	0.852	0.304
synth	1400	3.19KB	7	7	0.8	0.689	0.773	0.728	0.369
real	649	5.74KB	5	5	-	1	1	1	0.208
real	500	8.56KB	-	7	-	-	-	-	0.376
real	1000	9.42KB	-	9	-	-	-	-	0.43

edge of an a-priori classification. As a consequence, we resorted to an internal quality criterium that takes into account the compactness of the discovered clusters. More precisely, given a cluster partition $\mathcal{P} = \{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$, where $\mathcal{C}_i = \{x_1^i, \ldots, x_{n_i}^i\}$, we defined an *intra-cluster distance* measure as $\mathcal{IC}(\mathcal{P}) = \frac{1}{n}\sum_{\mathcal{C}_i \in \mathcal{P}} \frac{1}{n_i} \sum_{x \in \mathcal{C}_i} d(x, rep(\mathcal{C}_i))$.

Table 1 summarizes the quality values obtained testing *XRep* on both synthetic and real data. All the experiments have been carried out by adopting the Jaccard distance $d_J^{(2)}$ introduced in Section 2. Tests on synthetic data evaluated the performance of *XRep* on three collections of 1400 documents (200 documents for each class DTD). Experimental evidence highlights the overall accuracy of *XRep* in distinguishing among classes of XML documents characterized by different average sizes due to different choices for the threshold τ. As we can see, *XRep* exhibits an excellent behavior for $\tau = \{0.3, 0.5\}$, while the acceptable performance reported on row 3 (i.e. $\tau = 0.8$) is due to the intrinsic difficulty in catching minimal differences in the structure of the involved XML documents. Indeed, two clearly distinct class DTDs, say $\texttt{DTD}_i$ and $\texttt{DTD}_j$, may share a number of element definitions inducing similar paths within the conforming XML documents. If such definitions assign multiple occurrences to the elements of the common paths, the initial class separation between $\texttt{DTD}_i$ and $\texttt{DTD}_j$ may be potentially vanished by a strong degree of document similarity due to a large number of common paths in the corresponding XML trees.

Tests on real data considered separately the first five collections (649 XML documents with an average size that is equal to $5.74KB$), and the Xyleme_Sample collection. In the first case, *XRep* showed amazingly optimal accuracy in identifying even latent differences among the involved real documents. As far as Xyleme_Sample is concerned, we conducted two experiments (rows 5 and 6 in Table 1), where in the first one we considered only one and a half of the dataset. However, as we expected, in both cases intra-cluster distance provides fairly good values: this is mainly due to the high heterogeneity which characterizes documents in Xyleme_Sample.

5 Conclusions and Further Work

We presented a novel methodology for clustering XML documents, focusing on the notion of *XML cluster representative* which is capable of capturing the significant structural specifics within a collection of XML documents. By exploiting the

tree nature of XML documents, we provided suitable strategies for tree matching, merging, and pruning. Tree matching allows the identification of structural similarities to build an initial substructure that is common to all the XML document trees in a cluster, whereas the phase of tree merging leads to an XML tree that even contains uncommon substructures. Moreover, we devised a suitable pruning strategy for minimizing the distance between the documents in a cluster and the document built as the cluster representative. The clustering framework was validated both on synthetic and real data, revealing high effectiveness.

We conclude by mentioning some directions for future research. The approach described in the paper has to be considered an initial approach to clustering tree-structured XML data. Further notions of cluster representative can be investigated, e.g. by relaxing the requirement that a prototype corresponds to a single XML document. Indeed, there are many cases in which a collection of XML documents is better summarized by a forest of subtrees, where each subtree represents a given peculiarity shared by some documents in the collection. A typical case raises, for instance, when the collection has an empty matching tree, and still exhibits significant homogeneities.

References

1. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press Books. Addison Wesley, 1999.
2. R. Baumgartner, S. Flesca, and G. Gottlob. Visual web information extraction with Lixto. In *Proc. VLDB'01 Conf.*, pages 119–128, 2001.
3. 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), 2004.
4. S. Chawathe et al. Change detection in hierarchically structured information. In *Proc. SIGMOD'96 Conf*, pages 493–504, 1996.
5. G. Cobena, S. Abiteboul, and A. Marian. Detecting changes in XML documents. In *Proc. ICDE'02 Conf.*, pages 41–52, 2002.
6. V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *Proc. VLDB'01 Conf.*, pages 109–118, 2001.
7. A. Doucet and H. A. Myka. Naive clustering of a large XML document collection. In *Proc. INEX'02 Workshop*, 2002.
8. S. Flesca et al. Detecting structural similarities between XML documents. In *Proc. WebDB'02 Workshop*, 2002.
9. F. Giannotti, C. Gozzi, and G. Manco. Clustering transactional data. In *Proc. ECML-PKDD'02 Conf.*, pages 175–187, 2002.
10. A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, 1988.
11. W. Lian et al. An efficient and scalable algorithm for clustering XML documents by structure. *IEEE TKDE*, 16(1):82–96, 2004.
12. L. Mignet, D. Barbosa, and P. Veltri. The XML Web: a First Study. In *Proc. WWW'03 Conf.*, pages 500–510, 2003.
13. A. Nierman and H. V. Jagadish. Evaluating structural similarity in XML documents. In *Proc. WebDB'02 Workshop*, 2002.
14. Y. Wang, D.J. DeWitt, and J. Cai. X-Diff: A fast change detection algorithm for XML documents. In *Proc. ICDE'03 Conf.*, pages 519–530, 2003.