# XML Document Clustering Using Structure-preserving Flat Representation of XML Content and Structure

Fedja Hadzic[1], Michael Hecker[1], and Andrea Tagarelli[2]

[1] Digital Ecosystems and Business Intelligence Institute, Curtin University, Australia
Email: {fedja.hadzic, michael.hecker}@curtin.edu.au
[2] Dept. of Electronics, Computer and Systems Sciences, University of Calabria, Italy
Email: tagarelli@deis.unical.it

**Abstract.** With the increasing use of XML in many domains, XML document clustering has been a central research topic in semistructured data management and mining. Due to the semistructured nature of XML data, the clustering problem becomes particularly challenging, mainly because structural similarity measures specifically designed to deal with tree/graph-shaped data can be quite expensive. Specialized clustering techniques are being developed to account for this difficulty, however most of them still assume that XML documents are represented using a semistructured data model. In this paper we take a simpler approach whereby XML structural aspects are extracted from the documents to generate a flat data format to which well-established clustering methods can be directly applied. Hence, the expensive process of tree/graph data mining is avoided, while the structural properties are still preserved. Our experimental evaluation using a number of real world datasets and comparing with existing structural clustering methods, has demonstrated the significance of our approach.

## 1 Introduction

XML has become extremely popular in management and mining of semistructured/hierarchical text data due to its abilities in representing information in a well-defined, extensible and machine readable format. This is evidenced by the existence of many domain-specific XML based markup languages [17].

Clustering of XML documents has important applications in many domains that need management and processing of real-life complex objects that are represented as semistructured data. It is a more challenging task than the standard data clustering problem since the structural aspects in the data need to be taken into account. Defining a distance/similarity function over semistructured data that can be effectively and efficiently utilized for the XML clustering problem is known to be a difficult research problem. Initially proposed techniques were based on tree edit distance [13, 2]. Since calculating tree edit distance is known to be a computationally expensive problem [2], different approaches for measuring structural similarity have been proposed. For example, in [8] structural information of XML documents is represented in form of paths contained in the underlying tree model on which specialized similarity measures are defined. The S-GRACE approach [12] converts an XML document into a structure graph and the similarity is based on the number of common element-subelement relationships. In [3], the XRep method is developed focusing on a notion of structural XML cluster representative. This representative is built to capture the most relevant structural features of the documents within a cluster, and it is used to assign XML documents to structurally homogeneous groups in a cluster hierarchy. Another tree-based, summary-aware framework for clustering XML documents by structure is described in [4]. XML documents represented as rooted ordered labeled trees are individually summarized to reduce nesting and repetition of elements which are compared using a tree edit distance based on a variant of the Chawathe's algorithm [2]. The clustering scheme is based on a traditional graph-theoretic divisive approach. Given a collection of XML documents, a fully connected, weighted graph is built over the structural summaries of the input documents, where the edge weights correspond to the structural distance between summaries. A minimum spanning tree (MST) of the graph is computed, then the MST edges with a weight above a user-specified threshold are deleted. The connected components of the remaining graph are the single-link clusters.

More recently, one can witness an increasing number of XML document clustering approaches that take into account both the structure and content of the documents [18, 5, 10, 16, 11]. For instance, XProj [1] utilizes frequent substructures in a segment of data to measure the similarity, whereas HCX [10] determines the structural similarity based on frequent subtrees to extract (constrained) content and represent it in a Vector Space Model. Recently, the use of a Tensor Space Model to capture the content and structural information has been proposed in [11]. The structure features for the model are generated based on the length-constrained closed induced subtrees and these are used to constrain the content included in the model. In [16], subtrees that are cohesive according to the semantics of the original XML document trees are modeled into a transactional domain in which each item embeds a distinct combination of semantically-enriched structure and content XML features.

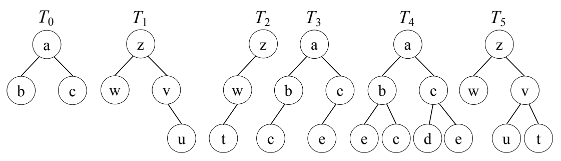
However, regardless of the techniques developed to use XML structural information at different refinement levels (i.e., node labels, edges, paths, twigs), the large majority of existing methods for structure-based XML clustering rely on tree/graph-shaped representation models. Consequently, the similarity/distance functions that drive the proximity search and detection during the clustering, are required to be suitable for the comparison of semistructured data, which is known to be a computationally difficult problem.

In this work we propose an alternative approach for XML document clustering, based on our recently proposed [6] structure preserving flat data representation for tree-structured data (henceforth referred to as FDT). The conversion process is based on the extraction of a document structure model within which each document instance can be matched to generate the flat data representation that captures the structural properties. Given such a representation, well-established clustering methods originally designed for vectorial data can be directly applied. CLUTO, a well-known toolkit for efficiently clustering large, high-dimensional document collections [9, 20] is taken as the case in point and is applied directly on the tree-structured data converted in the flat data format. The complexity of incorporating structural information in a distance/similarity measure is avoided, while the results can still be accompanied by the structural information captured by the document structure model. The implications of the approach are that the exact position of nodes within a general structure encompassing structural properties of all documents is taken into account during clustering. This is somewhat different from XML clustering methods based on subtree mining and matching, like XProj, which extract substructures that can occur anywhere within the documents structure of a data segment. In our approach, the user can specify a minimum frequency threshold during the document structure model generation, so that the general structure extracted only encompasses the frequently occurring structures among the documents being clustered. Furthermore, given that many more clustering techniques exist for flat data representation, the conversion approach in itself can potentially enable a wider range of techniques to be applied for the XML clustering problem.

XML content can be quite broad encompassing elements, values, attributes and attribute values. Although XML documents can be naturally modeled as trees, it is not clear how this information should be organized, and it may well be that it is dependent on the particular mining task as well as the application at hand. For example, in frequent subtree mining the values associated with elements are often assigned to a single node. This is both more efficient as we avoid building large trees, and practical because from the frequent subtrees one can discover interesting associations among the element values of an XML document from a single domain. On the other hand in XML document clustering, assigning the element name and value to a single node may not be desired, as one would like to detect similarity among element names, even when the values are different. This is even more so when the aim is to form clusters from heterogeneous collections of XML documents. In this work we also empirically study the impact of including/excluding partial content from XML documents for the clustering task. Our experimental evaluation was performed using several sets of XML documents from a variety of domains. The comparisons with existing XML clustering methods demonstrate the significance of our approach, in terms of both efficiency in performing the clustering task and of quality of the output clustering solutions.

## 2 Problem Background

The XML document clustering problem considered in this work can be defined in the homogeneous and heterogeneous context, in which the goal is: given a set of XML document instances from one domain, to group similar instances together (homogeneous context), and given a set of XML documents from different domains, to group the documents arising from the same domain together (heterogeneous context). Note however that even in the heterogeneous context



| | |
|---|---|
| 0 | a b -1 c -1 |
| 1 | z w -1 v u -1 -1 |
| 2 | z w t -1 -1 |
| 3 | a b c -1 -1 c e -1 -1 |
| 4 | a b e -1 c -1 -1 c d -1 e -1 -1 |
| 5 | z w -1 v u -1 t -1 -1 |

**Fig. 1.** Example of a tree-structured database consisting of 6 transactions

one can subdivide the clusters so as to further cluster the instances of each independent domain.

XML documents are conveniently modeled using a rooted ordered labeled tree (e.g., [1, 13]), which can be denoted as $T = (v_0, V, L, E)$, where $v_0 \in V$ is the root vertex; $V$ is the set of vertices or nodes; $L$ is a labeling function that assigns a label $L(v)$ to every vertex $v \in V$; $E = \{(v_1, v_2) | v_1, v_2 \in V \wedge v1 \neq v2\}$ is the set of edges in the tree, and for each internal node the children are ordered from left to right. Note that in relation to the particular mining task being considered, each vertex $v \in V$ can be chosen to correspond to a different aspect of XML (e.g., in frequent subtree mining [7, 19], the common choice is that $L(v)$ corresponds to an element name or a combination of element name and value). We will consider variation of XML content inclusion, and in Section 4 we explain how these will be captured in the tree representation of XML documents.

In order to represent trees in our approach, we used the pre-order (depth-first) string encoding ($\varphi$) [19]. This pre-order string encoding can be generated by adding vertex labels in the pre-order traversal of a tree $T = (v_0, V, L, E)$ and appending a backtrack symbol (e.g., '-1', '-1' $\notin L$) whenever we backtrack from a child node to its parent node. Figure 1 shows a tree-database ($T_{db}$) consisting of 6 tree instances (transactions). On the right of Fig. 1, the string encoding format representation commonly used in frequent subtree mining [7, 19] is shown for $T_{db}$. The first column stores transaction (tree) identifiers while the second column contains the string encoding $\varphi$ of each tree. Note that the backtrack symbols can be omitted after the last node in the string encoding (e.g., $\varphi(T_3)$ ='a b c -1 -1 c e').

## 3 Method Description

### 3.1 Conversion of Tree-structured Database to Flat Representation

The first row of a (relational) table consists of attribute names, which in a tree database $T_{db}$ are scattered through independent tree instances (transactions). One way to approach this problem is to first assume a structure/model according to which all the instances are organized. Each of the instances in a $T_{db}$ should be a valid subtree of this document structure model, denoted as DSM.

The process of extracting a DSM from a tree database consists of traversing the tree database and expanding the current DSM as necessary so that every tree instance can be matched against DSM. This process was formally described

**Algorithm 1** Database Structure Model (DSM) extraction from a tree database

```
Input: a tree database $T_{db}$
Output: the string encoding $\varphi(DSM)$ of the extracted DSM
 1: $inputNodeLevel = 0$                       {current level of $\varphi(tid_i)_k$}
 2: $DSMNodeLevel = 0$                         {current level of $\varphi(T(h_{\max}, d_{\max}))_k$}
 3: $\varphi(DSM) = \varphi(tid_0)$            {set default DSM (use x instead of labels)}
 4: for $i = 1 \to n-1$                        {$n = |T_{db}|$}
 5:   for all $\varphi(tid_i)_k \in \varphi(tid_i)$
 6:     for $p = 0 \to (|\varphi(DSM)| - 1)$
 7:       if $\varphi(tid_i)_k = -1$ then $dec(inputNodeLevel)$
 8:       else $inc(inputNodeLevel)$
 9:       if $\varphi(DSM)_p =$ ‘b’ then $dec(DSMNodeLevel)$
10:       else $inc(DSMNodeLevel)$
11:       if $inputNodeLevel \neq DSMNodeLevel$
12:         if $\varphi(tid_i)_k = -1$
13:           while $inputNodeLevel \neq DSMNodeLevel$ do
14:             $inc(p)$
15:             if $\varphi(DSM)_p = -1$ then $dec(DSMNodeLevel)$
16:             else $inc(DSMNodeLevel)$
17:         else
18:           while $inputNodeLevel \neq DSMNodeLevel$ do
19:             if $\varphi(tid_i)_k \neq -1$ then $\varphi(DSM)_{p+1} =$ ‘x’
20:             else $\varphi(DSM)_{p+1} =$ ‘b’
21:             $inc(k), inc(p)$
22:             if $\varphi(tid_i)_k = -1$ then $dec(inputNodeLevel)$
23:             else $inc(inputNodeLevel)$
24: return $\varphi(DSM)$
```

in [6], and we replicate it here for clarity purposes. Let the tree database consisting of $n$ transactions be denoted as $T_{db} = \{tid_0, tid_1, \ldots, tid_{n-1}\}$, and let the string encoding of the tree instance at transaction $tid_i$ be denoted as $\varphi(tid_i)$. Further, let $|\varphi(tid_i)|$ denote the number of elements in $\varphi(tid_i)$, and $\varphi(tid_i)_k$ ($k = \{0, 1, \ldots, |\varphi(tid_i)| - 1\}$) denote the $k$-th element (a label or a backtrack) of $\varphi(tid_i)$. The same notation for the string encoding of the (current) DSM is used, i.e., $\varphi(DSM)$. However, rather than storing the actual labels in $\varphi(DSM)$, **x** is always stored to represent a node in general, and **b** to represent a backtrack. The process of extracting the DSM from $T_{db}$ is depicted in Algorithm 1.

Note that we also store the number of times that a tree instance has matched a node or backtrack in the progressively built DSM. Occurrence of each node attribute (except the root node) within the DSM implies the existence of a specific backtrack attribute to ensure structural validity of the pre-order string encoding. Hence, after the whole DSM is extracted, it is safe to simply remove any nodes and backtracks that do not satisfy the minimum support set by the user. This will result in DSM reflecting the structure that was exhibited by as many instances as the user specified minimum support threshold. This relates to the next stage of populating the table, since the existence/non-existence of

**Table 1.** Flat representation of $T_{db}$ in Fig. 1 (optional columns in grey)

| $\mathbf{x}_0$ | $\mathbf{x}_1$ | $\mathbf{x}_2$ | $\mathbf{b}_0$ | $\mathbf{x}_3$ | $\mathbf{b}_1$ | $\mathbf{b}_2$ | $\mathbf{x}_4$ | $\mathbf{x}_5$ | $\mathbf{b}_3$ | $\mathbf{x}_6$ | $\mathbf{b}_4$ | $\mathbf{b}_5$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | b | 0 | 0 | 0 | 0 | 1 | c | 0 | 0 | 0 | 0 | 1 |
| z | w | 0 | 0 | 0 | 0 | 1 | v | u | 1 | 0 | 0 | 1 |
| z | w | t | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| a | b | c | 1 | 0 | 0 | 1 | c | e | 1 | 0 | 0 | 1 |
| a | b | e | 1 | c | 1 | 1 | c | d | 1 | e | 1 | 1 |
| z | w | 0 | 0 | 0 | 0 | 1 | v | u | 1 | t | 1 | 1 |

backtracks does not need to be stored in the generated table. These are stored in the DSM and can be used for mapping structural information onto detected cluster constraints.

To illustrate the conversion process using DSM please refer back to Fig. 1. In this example the DSM is reflected in the structure of $T_4$ in Fig. 1 and it becomes the first row in Table 1. Since the order of the nodes (and backtracks) is important, the nodes and backtracks are labeled sequentially according to their occurrence in DSM. For nodes (labels in the pre-order string encoding of DSM), $\mathbf{x}_i$ is used as the attribute name, where $i$ corresponds to the pre-order position of the node in DSM, while for backtracks, $\mathbf{b}_j$ is used as the attribute name, where $j$ corresponds to the backtrack number in DSM. Hence, from our example in Fig. 1, $DSM = \mathbf{x}_0\ \mathbf{x}_1\ \mathbf{x}_2\ \mathbf{b}_0\ \mathbf{x}_3\ \mathbf{b}_1\ \mathbf{b}_2\ \mathbf{x}_4\ \mathbf{x}_5\ \mathbf{b}_3\ \mathbf{x}_6\ \mathbf{b}_4\ \mathbf{b}_5$. To fill in the remaining rows, every transaction from $T_{db}$ is scanned and when a node or a backtrack is encountered, a '1' is placed to the matching column (i.e., under the matching node ($\mathbf{x}_i$) or backtrack ($\mathbf{b}_j$) in DSM), while the remaining entries are assigned values of '0' (non-existence). Table 1 shows the resulting FDT (structure preserving Flat Data representation for Tree-structured data).

During the FDT generation even if parts of a tree instance cannot all be matched to the DSM because of low occurrence, we still need to capture the partial information from the tree instance that structurally conforms to DSM. Each instance is matched against the DSM, and the current levels of the tree instances and DSM are tracked. Whenever the levels match, the label or '1' is stored in the corresponding column of the table, and when levels do not match, either the DSM or the tree instance encoding is traversed until the levels match. As an example, if minimum support threshold was set to 3 and the $T_{db}$ in Fig. 1 was converted, the resulting DSM would be $DSM = \mathbf{x}_0\ \mathbf{x}_1\ \mathbf{x}_2\ \mathbf{b}_0\ \mathbf{b}_1\ \mathbf{x}_3\ \mathbf{x}_4\ \mathbf{b}_2\ \mathbf{b}_3$, while the resulting flat representation would be equivalent to the one in Table 1 when columns with attributes $\mathbf{x}_3, \mathbf{b}_1, \mathbf{x}_6, \mathbf{b}_4$ are removed, and the remaining attributes renumbered sequentially to reflect the new DSM. Note that all the grey-coloured columns from the tables can be safely removed, as any results obtained can be mapped back to the extracted DSM, and the cluster constraints enriched with structural information. Hence, no information is lost, while the complex process of incorporating structural information during clustering is avoided. The minimum support threshold should be set to reflect the percentage of documents (instances) that are expected to form the smallest group/cluster. For example, in the experiments presented in Section 4, since the number of instances from each class was known, the support was set not far below the percentage of instances of minority class. The choice is thus dependent on some apriori expectation of the number of documents to be considered as part of one group/cluster. This comprises the tree-structured to flat data conversion step in our method, which enables a clustering approach to be directly applied without using specialized distance measures for tree structures.

### 3.2 Characteristics/Implications of the Proposed Approach

The DSM governs what a valid instance of a particular tree characteristic is, and the exact positions of the node within the DSM are taken into account. As an example, consider the subtree with encoding 'a b c' from the $T_{db}$ in Fig. 1. Within the current frequent subtree mining framework (upon which XProj [1] and HCX [10] XML clustering approaches are based) this subtree would be considered to occur in two transactions $T_3$ and $T_4$. Using the proposed approach the occurrence of 'a b c' in $T_3$ would be considered different than in $T_4$, because node 'c' occurs at different (pre-order) position, and hence the instance is represented differently in the table. This can be seen from Table 1, since the occurrence of node 'c' in $T_3$ is matched against attribute node $\mathbf{x}_2$, while it is matched against attribute node $\mathbf{x}_3$ for its occurrence in $T_4$. This illustrates the key difference in our approach, and it implies that two document instances will be considered similar only if their substructures occur at the same/similar positions. Different occurrences of a substructure in the DSM may indicate that it is used in a different context, and hence a different domain, and in these cases such property is useful. This is especially the case if all the instances in a $T_{db}$ follow the same structure and node layout as the general document model. This property could be less useful in scenarios, where the instances of a $T_{db}$ may not follow the same order or not have all the elements of the general document structure (e.g., XML schema) available. In spite of this difference the proposed method is an alternative approach to clustering, and can arrive at high quality clusters as will be demonstrated in the experimental results given in the next section.

## 4 Experimental Evaluation

### 4.1 Experimental Methodology

Our approach is composed of 5 steps, covering data preparation, document structure model extraction, conversion to a flat representation, document clustering, and clustering evaluation.

**Step 1.** Our approach starts by creating a single XML document that contains all the instances of all XML documents considered. They are organized in such a way that all instances are siblings on the root level of the document for further processing. The second part of the data preparation phase is the creation of multiple variations of the input document. We have chosen the following variations and show it on the following example (with labels applied):

```
<[N]dataset [A]subject=[W]‘‘astronomy’’>[V]‘‘Stars’’</[N]dataset>
```

(1) tag names only, which are labeled as N (dataset in our example), (2) tag names and #PCDATA elements (values), which are labeled as N and V (`dataset` and `‘‘Stars’’` in our example), (3) tag names and attribute names, which are labeled as N and A (`dataset` and `subject` in our example), (4) tag names, #PCDATA elements (values) and attribute names, which are labeled as N, V and A (`dataset`, `‘‘Stars’’`, and `subject` in our example), and (5), tag names, #PCDATA elements (values), attribute names and attribute values, which are labeled as N, V, A and W (`dataset`, `‘‘Stars’’`, `subject`, and `‘‘astronomy’’` in our example). For each of these variations, we output a string representation file and a mapping file that contains the mappings between the original document and the string representation. In each of the variations, we consider each item as a separate node for further processing. For instance, variation NVAW of our example would look as follows:

```
<[N]dataset><[A]subject><[W]astronomy/></[A]subject><[V]Stars/></[N]dataset>
```

Each variation is stored in a commonly used string representation format [7, 19], example of which was given in Fig. 1.

**Step 2.** This step performs the document structure model (DSM) extraction. The string encoding representation of each of the 5 variations from the previous step (N, NV, NA, NVA, NVAW) is traversed to generate the output that contains the string encoding of DSM (i.e., $\varphi(DSM)$) for each variation. As explained in Section 3, the user can specify the minimum support threshold so that the DSM only captures the structural characteristics if they have occurred in specified percentage of document instances. In our experiments, we choose the support value to be not far below the percentage of instances from the minority class.

**Step 3.** In the third step, the $\varphi(DSM)$ from the previous step and the respective output from step 1 (string encoding format representation of the tree database) are used to create the flat representation. This is done for two variations, one with the backtracks (-1 or $\mathbf{b}_i$) as attributes and second without, and each instance is labeled with a class value (if available). The output of each of these variations is the generated flat representation, example of which was given in Table 1.

**Step 4.** The fourth step converts these files into CLUTO dense matrix format, where we perform a number of CLUTO runs. Three runs are performed for each of the following similarity measures: *correlation coefficient* (*corr*), *Euclidean* (*eucl*) and *extended Jaccard* (*jacc*). The number of clusters is set with respect to the amount of different classes from the original input.

**Step 5.** Once CLUTO has finished generating all output files, clustering validation is performed to evaluate the clustering solution in terms of both internal and external clustering validity criteria. The internal validity criteria are based on the average of internal and external cluster similarity, respectively denoted as $ISim$ and $ESim$ as produced by CLUTO output, which essentially assess the cluster compactness and cluster separation, respectively, of a given clustering solution. For the exact formula of how such measures are computed, the reader is referred to [20]. The external criteria aim to evaluate how well a clustering solution fits a known organization of the data into predefined classes; we hereinafter refer to this organization as reference classification. In this work we resorted to commonly used external criteria for document clustering, namely Entropy ($E$), Purity ($Pty$), micro-averaged F-measure ($F^m$), and macro-averaged F-measure ($F^M$) (with relating overall precision, $P$, and recall, $R$), definitions of which can be found in [15, 20, 16].

## 4.2 Data

The proposed method is evaluated on three sets of data. The first dataset (henceforth *Real*) was previously used in [3], and it contains collections of documents from 5 different domains, namely 217 astronomy documents (http://adc.gsfc.nasa.gov), 264 documents representing messages from a Web forum (http://userland.com), 64 press news documents (http://www.prweb.com/rss.php), 51 documents containing issues of SIGMOD record (http://www.dia.uniroma3.it/Araneus/Sigmod/) and 53 documents representing wrapper programs for Web sites. Thus, it is a representative of the problem in heterogeneous context (cf. Section 2).

The second dataset (henceforth *DEBII_N*) was generated by taking Apache2 (v2.2.3) web server log files from the DEBI Institute website (http://debii.curtin.edu.au) for a 4-month period (i.e., homogeneous context). The order of web pages accessed is organized as a tree where the navigational pattern is reflected through the pre-order traversal of the tree. This way of representing web logs is based on the LOGML representation [14], which allows for a more detailed and informative representation of web logs using an XML template. The difference is that we are only storing the web pages accessed as nodes in a tree, and ignore other information (e.g., time stamps). Hence, only option N (elements only) in tree generation is used by all the approaches compared. The instances were labeled according to 3 classes, namely, external access, within-university access, and within-DEBI-institute access. Hence, there is some overlap between the classes, as it is well possible that similar usage patterns occur within each class.

The third dataset (henceforth *Process*) was generated from publicly available process log datasets (http://prom.win.tue.nl/tools/prom/). These datasets describe examples of business process logs in MXML, a business process management XML based template. Several sets of process logs were taken, each of which was assigned a unique class (8 classes in total) depending which category/system/format it came from. The majority of process logs were quite similar in structure, as the same MXML format was used. This data is therefore a good example of content-based XML document clustering application, where it is important to consider more extensive content information in addition to element names. While this dataset is rather homogeneous by nature, due to different categories and/or source systems used for data generation, it can be seen as a representative of the problem in mixed heterogeneous/homogeneous context.

As previously mentioned, in our data preparation phase all of the documents are merged into a single tree database, and varied content is included, variations of which we will denote as *dataset*_N, *dataset*_NA, *dataset*_NV, *dataset*_NVA

**Table 2.** Dataset characteristics

| | $\lvert Tr\rvert$ | $\lvert L\rvert$ | $Avg\lvert T\rvert$ | $Avg\lvert D\rvert$ | $Avg\lvert F\rvert$ | $Max\lvert T\rvert$ | $Max\lvert D\rvert$ | $Max\lvert F\rvert$ |
|---|---|---|---|---|---|---|---|---|
| *Real*_N | 649 | 127 | 94.85 | 4.43 | 3.18 | 1440 | 9 | 330 |
| *Real*_NA | 649 | 158 | 11.94 | 4.59 | 3.19 | 1474 | 10 | 330 |
| *Real*_NV | 649 | 15572 | 150.92 | 5.35 | 1.67 | 2307 | 9 | 330 |
| *Real*_NVA | 649 | 15603 | 173.006 | 5.43 | 1.79 | 2341 | 10 | 330 |
| *Real*_NVAW | 649 | 18022 | 194.426 | 5.59 | 1.67 | 2375 | 11 | 330 |
| *DEBII*_N | 10996 | 16946 | 6.93 | 4.13 | 1.52 | 30 | 29 | 26 |
| *Proc*_N | 20968 | 8 | 80.33 | 2.2 | 3.63 | 1256 | 4 | 258 |
| *Proc*_NA | 20968 | 12 | 89.56 | 2.8 | 3.41 | 1975 | 4 | 258 |
| *Proc*_NV | 20968 | 22185 | 138.67 | 3.2 | 1.69 | 2260 | 4 | 256 |
| *Proc*_NVA | 20968 | 22189 | 147.92 | 3.2 | 1.79 | 3019 | 4 | 258 |
| *Proc*_NVAW | 20968 | 23883 | 156.36 | 3.8 | 1.73 | 3833 | 5 | 258 |

and *dataset*_NVAW. Table 2 summarizes the structural characteristics of each dataset and its content variation, and the following notation is used: $|Tr|$ - # of transactions (independent tree instances), $|L|$ - # of unique node labels; $|T|$ - # of nodes (size) in a transaction; $|D|$ - depth; $|F|$ - fan-out or degree.

## 4.3 Results

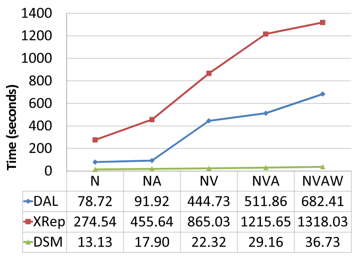
We assessed the clustering performance of our approach in terms of accuracy as well as efficiency, and also compared the results with those obtained by two existing XML clustering methods, namely the approach presented in [4] (hereinafter denoted as DAL) and XRep [3].

Unlike XRep (which is a stand-alone XML clustering method), DAL can enable the direct application of the CLUTO software, by providing it with a similarity matrix that stores the (normalized) structural distance values converted into similarity values (each distance value is subtracted to 1 and the difference is stored into the matrix). Hence, both our DSM and DAL are compared using CLUTO to cluster the data uniquely prepared by the different approaches. Note that all of the approaches have been enhanced in their preprocessing modules to deal with the variations of XML document as explained earlier (i.e., N, NA, NV, NVA, NVAW). Each algorithm name is appended with the symbol indicating the variation of content considered. In addition, our approach is also optionally appended with the similarity function it used within the CLUTO toolkit (e.g., DSM_corr_NA, DSM_eucl_NA, DSM_jacc_NA). All of our experiments were carried out on the same machine sequentially. The server in question is a quad socket quad core Xeon E7330 (2.4 GHz) machine, which is dedicated for running our experiments in a Windows Server 2008 64 bit environment. This machine has 128GB of RAM and we have allocated 50GB of memory to every experiment.

In Fig. 2(a) we summarize the clustering results for *Real* data, with input number of clusters set to 5. Note that for XRep we could not obtain the information necessary to determine the average internal and external similarity of the formed clusters, represented in column 8 and 9, respectively. The support used for DSM extraction phase was set to 5%. As can be seen, when *corr* similarity measure is used within our approach, better results were generally obtained in

| | $Pty$ | $E$ | $P$ | $R$ | $F^M$ | $F^m$ | $ISim$ | $ESim$ |
|---|---|---|---|---|---|---|---|---|
| DAL_N | 0.851 | 0.502 | 0.721 | 0.783 | 0.751 | 0.733 | 0.609 | 0.14 |
| DAL_NA | 0.901 | 0.505 | 0.8 | 0.94 | 0.864 | 0.875 | 0.597 | 0.09 |
| DAL_NV | 1 | 0 | 1 | 1 | 1 | 1 | 0.569 | 0.081 |
| DAL_NVA | 1 | 0 | 1 | 1 | 1 | 1 | 0.578 | 0.089 |
| DAL_NVAW | 1 | 0 | 1 | 1 | 1 | 1 | 0.561 | 0.081 |
| XRep_N | 1 | 0 | 1 | 1 | 1 | 1 | - | - |
| XRep_NA | 1 | 0 | 1 | 0.996 | 0.998 | 0.999 | - | - |
| XRep_NV | 1 | 0 | 1 | 1 | 1 | 1 | - | - |
| XRep_NVA | 1 | 0 | 1 | 0.996 | 0.998 | 0.999 | - | - |
| XRep_NVAW | 1 | 0 | 1 | 1 | 1 | 1 | - | - |
| DSM_corr_N | 1 | 0 | 1 | 1 | 1 | 1 | 0.777 | 0.014 |
| DSM_corr_NA | 0.998 | 0.408 | 0.999 | 0.996 | 0.998 | 0.998 | 0.782 | -0.02 |
| DSM_corr_NV | 0.998 | 0.408 | 0.999 | 0.996 | 0.998 | 0.998 | 0.757 | 0.067 |
| DSM_corr_NVA | 0.998 | 0.408 | 0.999 | 0.996 | 0.998 | 0.998 | 0.767 | 0.004 |
| DSM_corr_NVAW | 0.998 | 0.408 | 0.999 | 0.996 | 0.998 | 0.998 | 0.764 | 0.004 |
| DSM_eucl_N | 0.966 | 0.136 | 1 | 0.593 | 0.745 | 0.626 | 0.605 | 0.003 |
| DSM_eucl_NA | 0.965 | 0.162 | 1 | 0.587 | 0.740 | 0.571 | 0.512 | 0.007 |
| DSM_eucl_NV | 0.995 | 0.066 | 1 | 0.627 | 0.771 | 0.624 | 0.389 | 0.002 |
| DSM_eucl_NVA | 0.994 | 0.043 | 1 | 0.641 | 0.781 | 0.642 | 0.427 | 0.002 |
| DSM_eucl_NVAW | 0.994 | 0.059 | 1 | 0.641 | 0.781 | 0.679 | 0.357 | 0.001 |
| DSM_jacc_N | 0.968 | 0.142 | 1 | 0.598 | 0.748 | 0.598 | 0.458 | 0.005 |
| DSM_jacc_NA | 0.965 | 0.162 | 1 | 0.571 | 0.727 | 0.592 | 0.494 | 0.005 |
| DSM_jacc_NV | 0.998 | 0.005 | 1 | 0.725 | 0.840 | 0.681 | 0.406 | 0.003 |
| DSM_jacc_NVA | 0.997 | 0.006 | 1 | 0.708 | 0.829 | 0.691 | 0.424 | 0.003 |
| DSM_jacc_NVAW | 0.997 | 0.006 | 1 | 0.714 | 0.833 | 0.683 | 0.422 | 0.002 |
| DSM_corr_N(s=10) | 0.919 | 0.075 | 0.799 | 0.913 | 0.852 | 0.858 | 0.735 | 0.008 |
| DSM_corr_N(s=20) | 0.919 | 0.075 | 0.799 | 0.941 | 0.864 | 0.892 | 0.794 | -0.029 |

(a)



| | N | NA | NV | NVA | NVAW |
|---|---|---|---|---|---|
| DAL | 78.72 | 91.92 | 444.73 | 511.86 | 682.41 |
| XRep | 274.54 | 455.64 | 865.03 | 1215.65 | 1318.03 |
| DSM | 13.13 | 17.90 | 22.32 | 29.16 | 36.73 |

(b)

**Fig. 2.** *Real* data: (a) cluster quality comparison and (b) total runtime taken for different *Real* dataset variations

terms of $Pty$, $F^M$, and $F^m$; moreover, in case of $F^M$, recall scores were consistently higher than *eucl* and *jacc* over the variations of XML document (from about +0.42 to +0.22). Focusing on our *corr* based approach, we can see that the external-criteria-based quality of clusters was recognized as optimal for variation N (element names only); as far as internal evaluation, $ISim$ for variation N was slightly worse than variation NA only, nevertheless DSM $ISim$ values were always higher than the corresponding $ISim$ values obtained by the competing clustering methods. In general, DSM behaved as comparable to the competing methods with some minor differences. For example, our approach performed best for option N while the DAL approach did not achieve optimal results for this option. We also observed that, for this data, there was no difference in the results obtained by DSM when backtrack attributes were included. The last two columns of the table of Fig. 2(a) show how the different support thresholds (i.e., 10% and 20%) affect the results for DSM_corr_N option. Note that the percentage of instances from the minority class in this dataset, cover less than 10% of the database, and hence the structural characteristics unique to those instances will not be considered in DSM. Hence, one can expect less optimal values for all of the values except for internal and external cluster similarity, which can be confirmed by comparing these cases with the results of row 12. At $s = 20\%$, more optimal values for internal and external cluster similarity were achieved, but this could be mainly due to the fact that the unique structural characteristics of the instances of minority class were not captured by any formed cluster.

The time performance comparison between the different approaches, for the different variations of the data, is shown in Fig. 2(b). This is the total time

**Table 3.** Cluster quality comparison for *DEBII* data

| | $Pty$ | $E$ | $P$ | $R$ | $F^M$ | $F^m$ | $ISim$ | $ESim$ |
|---|---|---|---|---|---|---|---|---|
| DAL_N | 0.801 | 0.860 | 0.349 | 0.513 | 0.415 | 0.655 | 0.277 | 0.187 |
| XRep_N | 0.897 | 0.188 | 1 | 0.001 | 0.002 | 0.273 | - | - |
| DSM_corr_N | 0.801 | 0.806 | 0.384 | 0.504 | 0.436 | 0.504 | 0.470 | 0.191 |
| DSM_corr_N_B | 0.801 | 0.797 | 0.387 | 0.511 | 0.441 | 0.5 | 0.505 | 0.237 |
| DSM_eucl_N | 0.802 | 0.622 | 0.639 | 0.146 | 0.237 | 0.332 | 0.537 | 0 |
| DSM_eucl_N_B | 0.802 | 0.622 | 0.639 | 0.146 | 0.237 | 0.332 | 0.537 | 0 |
| DSM_jacc_N | 0.801 | 0.601 | 0.546 | 0.092 | 0.157 | 0.316 | 0.475 | 0 |
| DSM_jacc_N_B | 0.801 | 0.647 | 0.541 | 0.162 | 0.249 | 0.459 | 0.475 | 0 |
| DSM_corr_N($s = 10$) | 0.801 | 0.546 | 0.358 | 0.446 | 0.397 | 0.501 | 0.468 | 0.102 |
| DSM_corr_N($s = 20$) | 0.801 | 0.513 | 0.389 | 0.446 | 0.416 | 0.57 | 0.507 | 0.003 |

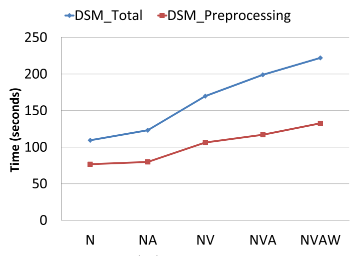
taken including preprocessing and clustering of the data. As more content from XML data was considered, the task became more complex (especially when element values were considered), but such a higher complexity negatively affected mainly the performance of the competing methods; precisely, total runtimes (in seconds) were 2744.14 (DAL), 115885.16 (XRep), and 91.30 (DSM), where in all cases the preprocessing stage took about from 88% (DAL) to 99% (DSM) of the total runtime. Overall, the proposed approach dramatically outperformed the competing methods in terms of runtime (from 2 up to 4 orders of magnitude) and the improvements were particularly evident for the content-oriented variations.

Table 3 shows the clustering results for *DEBII* data, with input number of clusters set to 3. The support used for DSM extraction phase was set to 3%. Note that here we have also included the results when the backtrack attributes were considered, but as one can see there was only a slight difference for *corr* based similarity measure. Again, DSM performed as good as or better than DAL in terms of all criteria except $F^m$, whereas XRep was not able to detect the 3-class organization of the data as it ended to identify many small clusters. The total runtimes (seconds) were as follows: 91.3 (90.4 for preprocessing) for DSM, 2744.14 (2421.65 for preprocessing) for DAL, and 115885.16 for XRep. The last two columns of Table 3 show how the different support thresholds (i.e., 10% and 20%) affect the results for DSM_corr_N option. Comparing with the results of row 4, one can see that increasing the support has no effect on purity, better results for entropy and external cluster similarity, while there is a small difference in remaining criteria. In contrast to the previous experiment, in this case increasing the support can positively affect entropy, which is because in *DEBII* data there is a higher possibility of overlapping classes.

When we ran the different approaches on *Process* data (input number of clusters set to 8), the XRep approach exhausted the available memory and the DAL approach took nearly 3 days to complete. The support used for DSM extraction phase was set to 0.4%. In Fig. 3(a) we show clustering results for our approach (no backtracks considered) and DAL approach. In contrast to the other two data, higher external-criteria-based quality was achieved by our approach when more content variations were included, leading to improvements up to about 0.22 ($F^m$) on NVAW variation. Higher intra- and extra-cluster similarity were instead obtained on N variation. However, this was not the case for the DAL approach, as $F^m$ only increased by 0.004 for NVAW option and $ISim$ and $ESim$ were higher for NVA and NVAW options. Overall, one can see DSM performed better than DAL in terms of all criteria except $R$ and $ESim$; however, these two criteria should be considered w.r.t. $P$ and $ISim$, respectively, and hence the results were still comparable according to all four criteria as a whole. The DAL approach took approximately 21.5h to complete the clustering for NVAW variation, 15.5h for NV, 18.5h for NVA, 3h for NA and 2.5h for N. Due to these large runtimes, in Fig. 3(b) we show the total and preprocessing time taken by our approach for the variations of *Process* data. As expected, the presence of element values impacted more on the preprocessing as well as the clustering step than non-V variations.

| | $Pty$ | $E$ | $P$ | $R$ | $F^M$ | $F^m$ | $ISim$ | $ESim$ |
|---|---|---|---|---|---|---|---|---|
| DAL_N | 0.396 | 0.999 | 0.125 | 0.998 | 0.222 | 0.363 | 0.488 | 0.246 |
| DAL_NA | 0.396 | 0.999 | 0.228 | 0.874 | 0.361 | 0.362 | 0.404 | 0.224 |
| DAL_NV | 0.397 | 0.999 | 0.35 | 0.751 | 0.478 | 0.363 | 0.492 | 0.273 |
| DAL_NVA | 0.397 | 0.999 | 0.35 | 0.752 | 0.478 | 0.363 | 0.516 | 0.253 |
| DAL_NVAW | 0.4 | 0.996 | 0.248 | 0.886 | 0.387 | 0.367 | 0.493 | 0.189 |
| DSM_corr_N | 0.583 | 0.73 | 0.341 | 0.443 | 0.386 | 0.4 | 0.838 | 0.464 |
| DSM_corr_NA | 0.601 | 0.667 | 0.461 | 0.501 | 0.480 | 0.431 | 0.829 | 0.431 |
| DSM_corr_NV | 0.752 | 0.446 | 0.43 | 0.549 | 0.483 | 0.548 | 0.753 | 0.243 |
| DSM_corr_NVA | 0.756 | 0.423 | 0.445 | 0.583 | 0.505 | 0.543 | 0.746 | 0.245 |
| DSM_corr_NVAW | 0.787 | 0.385 | 0.556 | 0.604 | 0.579 | 0.618 | 0.749 | 0.206 |

(a)



(b)

**Fig. 3.** *Process* data: (a) cluster quality comparison and (b) preprocessing and total runtime taken by DSM for different variations of the dataset

## 5 Conclusions

In this paper we have presented an alternative approach to XML document clustering. The structural aspects from XML data are first extracted from the document to generate a structure-preserving flat data format, to which well-established traditional clustering approaches, such as the partitional one, can be directly applied. Within this view, we have taken the CLUTO clustering toolkit as a case in point and have compared the results with existing structural clustering methods. There is strong empirical evidence that the complexity associated with XML document clustering due to the structural aspects that need to be considered, can be significantly reduced with the proposed approach. At the same time high quality clustering results can be obtained that are comparable or better than those obtained by XML clustering methods that deal with complex structural similarity determination. Furthermore, given that many more clustering techniques exist for flat data representation, the conversion approach in itself can potentially enable a wider range of techniques to be applied for XML document clustering problem.

**Acknowledgments.** The authors wish to acknowledge Theodore Dalamagas for providing his XML clustering tool [4].

## References

1. C. C. Aggarwal, N. Ta, J. Wang, J. Feng, and M. Zaki. XProj: a framework for projected structural clustering of XML documents. In *Proc. ACM KDD Conf.*, pages 46–55, 2007.
2. P. Bille. A survey on tree edit distance and related problems. *Theoretical Computer Science*, 337(1–3):217–239, 2005.
3. G. Costa, G. Manco, R. Ortale, and A. Tagarelli. A Tree-Based Approach to Clustering XML Documents by Structure. In *Proc. PKDD Conf.*, pages 137–148, 2004.
4. T. Dalamagas, T. Cheng, K.-J. Winkel, and T. K. Sellis. A methodology for clustering XML documents by structure. *Information Systems*, 31(3), 2006.
5. A. Doucet and M. Lehtonen. Unsupervised Classification of Text-Centric XML Document Collections. In *Proc. INEX Workshop*, pages 497–509, 2006.
6. F. Hadzic. A Structure Preserving Flat Data Format Representation for Tree-Structured Data. In *Proc. PAKDD Workshops (QIME'11)*. Springer, 2011.
7. F. Hadzic, H. Tan, and T. S. Dillon. *Mining of Data with Complex Structures*. Studies in Computational Intelligence. Springer, 1st edition, 2010.
8. S. Joshi, N. Agrawal, Krishnapuram R., and S. Negi. A bag of paths model for measuring structural similarity in Web documents. In *Proc. ACM KDD Conf.*, pages 577–582, 2003.
9. G. Karypis. CLUTO - Software for Clustering High-Dimensional Datasets. http://glaros.dtc.umn.edu/gkhome/cluto/cluto/download, 2002/2007.
10. S. Kutty, R. Nayak, and Y. Li. HCX: an efficient hybrid clustering approach for XML documents. In *Proc. ACM Symposium on Document Engineering*, pages 94–97, 2009.
11. S. Kutty, R. Nayak, and Y. Li. XML Documents Clustering Using a Tensor Space Model. In *Proc. PAKDD Conf.*, pages 488–499, 2011.
12. W. Lian, D. W.-L. Cheung, N. Mamoulis, and S.-M. Yiu. An Efficient and Scalable Algorithm for Clustering XML Documents by Structure. *IEEE Transactions on Knowledge Data Engineering*, 16(1):82–96, 2004.
13. A. Nierman and H. V. Jagadish. Evaluating Structural Similarity in XML Documents. In *Proc. WebDB Workshop*, pages 61–66, 2002.
14. J. R. Punin, M. S. Krishnamoorthy, and M. J. Zaki. LOGML: Log Markup Language for Web Usage Mining. In *Proc. WEBKDD Workshop*, pages 88–112, 2001.
15. M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques. In *Proc. KDD Workshop on Text Mining*, 2000.
16. A. Tagarelli and S. Greco. Semantic clustering of XML documents. *ACM Transactions on Information Systems*, 28(1), 2010.
17. J. T. Yao, A. Varde, E. Rundensteiner, and S. Fahrenholz. XML Based Markup Languages for Specific Domains. In *Web-based Support Systems*, Advanced Information and Knowledge Processing, pages 215–238. Springer London, 2010.
18. J. P. Yoon, V. Raghavan, V. Chakilam, and L. Kerschberg. BitCube: A Three-Dimensional Bitmap Indexing for XML Documents. *Journal of Intelligent Information Systems*, 17(2–3):241–254, 2001.
19. M. J. Zaki. Efficiently Mining Frequent Trees in a Forest: Algorithms and Applications. *IEEE Transactions on Knowledge and Data Engineering*, 17(8):1021–1035, 2005.
20. Y. Zhao and G. Karypis. Empirical and Theoretical Comparison of Selected Criterion Functions for Document Clustering. *Machine Learning*, 55(3):311–331, 2004.