XML Documents Clustering using a Tensor Space Model

Sangeetha Kutty, Richi Nayak, and Yuefeng Li

Faculty of Science and Technology
Queensland University of Technology
GPO Box 2434, Brisbane Qld 4001, Australia
{s.kutty, r.nayak, y2.li}@qut.edu.au

Abstract. The traditional Vector Space Model (VSM) is not able to represent both the structure and the content of XML documents. This paper introduces a novel method of representing XML documents in a Tensor Space Model (TSM) and then utilizing it for clustering. Empirical analysis shows that the proposed method is scalable for large-sized datasets; as well, the factorized matrices produced from the proposed method help to improve the quality of clusters through the enriched document representation of both structure and content information.

1 Introduction

Rapid growth of web technologies has witnessed a sudden surge in the number of XML (eXtensible Markup Language) documents. For instance, English Wikipedia contains 3.1 million web documents in XML format; the ClueWeb dataset, used in Text Retrieval Conference (TREC) tracks, contains 503.9 million XML documents collected from the web in January and February 2009. The majority of existing XML document clustering methods utilize either the structure features [2] or the content features present in the documents. Clustering methods utilizing only the content features of the documents consider the documents as a "bag of words" or a Vector Space Model (VSM) and ignore the structure features [2]; clustering methods utilizing only the structure features of the documents represent each document as a set of paths (sequences) or trees.

However, these methods, with their single-feature focus, tend to falsely group documents that are similar in for documents that are similar in both features. To correctly identify similarity among documents, the clustering process should use both their structure and their content information. Approaches on clustering both the structure and the content features of the XML documents are limited. Approaches using the VSM often fail to scale for even small collections of a few hundred documents, and in some situations have resulted in poor accuracy [14]. VSM cannot model both structure and content features of XML documents effectively as the mapping between the structure and its corresponding content is lost. The content and structure features inherent in an XML document should be modeled in a way that the mapping between the content of the path or tree

can be preserved and used in further analysis. In this paper we propose a novel method that represents the XML documents in a Tensor Space Model (TSM) and uses the TSM for clustering. In the TSM, storing the content corresponding to its structure helps to analyze the relationship between structure and content.

Unlike the VSM, which uses a vector to model, TSM is based on the multilinear algebraic character level high-order tensors (generalization of matrices) [13]. Decomposition algorithms are used to analyze the relationships between various tensor orders (ways or modes). However, existing decomposition algorithms could be used to analyze small size and sparse TSMs. TSMs that are large and dense cannot be loaded into memory. Consequently, large datasets with tensor representation cannot be analyzed using these decomposition techniques. In this paper, we propose a randomized tensor decomposition technique that could upload the large size tensors into memory and decompose them with significant speedups. Experiments on a real-life dataset containing more than 50K documents show that the proposed method helps to improve the cluster quality through the enriched document representation of both structure and content information. The contributions of this paper can be summarized as: (1) a clustering method, XML document Clustering with TSM (XCT), that utilizes the tensor model to efficiently combine the content and structure features of XML documents; and (2) a new tensor decomposition algorithm, Progressive Tensor Creation and Decomposition (PTCD), for large sized tensors.

2 RELATED WORK

Tensor Space Modeling (TSM) has been successfully used in representing and analyzing multi-way data in signal processing, web mining and many other fields [13]. Tensor clustering is a multi-way data analysis task which is currently gaining importance in the data mining community. The simplest tensor clustering scenario, co-clustering or bi-clustering, in which two orders are simultaneously clustered, is well established [6]. Another recently proposed approximation based Combination Tensor Clustering algorithm [7] clusters along each of the orders and then represents the cluster centers in the tensor. These co-clustering techniques capture only the 2-way relationships among the features and ignore the dependence of multiple orders in clustering: this may result in loss of information while grouping the objects.

Several decomposition algorithms, such as Higher Order SVD (HOSVD); CP, a higher-order analogue of Singular Value Decomposition (SVD) or Principal Component Analysis (PCA); Tucker and Multi-Slice Projection, have been reviewed in detail in [5]. Incremental Tensor Analysis (ITA) methods [8] have been proposed recently to detail with large datasets for efficiently decomposing sparse tensors ($density \leq 0.001\%$). However, real-life XML documents represented in TSM are dense with about 127M non-entries with over 1M terms and these decomposition algorithms fail to scale. MET [9], a memory-efficient implementation of Tucker proposed to avoid the intermediate blow-up in tensor factorization, is shown in our results shows not to scale to our medium-sized and

large-sized datasets. In MACH [13], a recently proposed random decomposition algorithm suitable for large dense datasets, the number of entries in the tensor is randomly reduced using Achlioptas-McSherry's technique [1] to decrease the density of the dataset. However, as discussed in section 5, MACH often ignores smaller length documents and tends to group most of the smaller length documents in a single cluster in spite of differences in their structure and content. To remove this lack of decomposition algorithms suitable for very large-sized datasets, in this paper we propose a new decomposition algorithm, the Progressive Tensor Creation and Decomposition (PTCD) algorithm, that progressively unfolds a tensor into a matrix and applies SVD on this generated matrix.

3 The Proposed XCT Method

3.1 Problem Definition and preliminaries

Let there be a collection of XML documents $D = \{D_1, D_2, \ldots, D_n\}$, where D_i is an XML document containing tags and data enclosed within those tags. The structure of D_i can be defined as a list of tags showing the hierarchical relationships between them. The structure of D_i is modeled as a rooted, ordered and node-labeled document tree, $DT_i = (N, n0, E, f)$, where (1) N is the set of nodes that correspond to tags in D_i, with the node labels corresponding to tag names; (2) $n0$ is the root node which does not have any edges entering in it; (3) E is the set of edges in DT_i; and (4) f is a mapping function $f : E \rightarrow N \times N$. Previous research has shown that, in a dataset, only the content constrained within the concise common or frequent subtrees (Closed Frequent Induced - CFI) can be used to group the documents, rather than the entire content of the XML documents [10]. Therefore the proposed XCT method generates these CFI subtrees to represent the common subtrees in the dataset and uses these CFI subtrees to extract the content of the documents corresponding to them. The process begins by identifying the subtrees that belongs to a document tree. A subtree $CFI_j \in CFI$ is present in document tree DT_i, if CFI_j preserves the same parent-child relationship as that of DT_i. The document content(or *structure-constrained content*) contained within the CFI_j subtree in DT_i, noted as $C(D_i, CFI_j)$, is retrieved from the XML document D_i, a collection of node values or terms. The node value of a node (or tag) of a CFI_j, $C(N_i)$ in D_i is a vector of terms, $\{t_1, \ldots, t_k\}$ that the node contains. The term t is obtained after stop-word removal and stemming.

The next step involves modeling the derived structure and content features of a tensor model. Firstly,the tensor notations and conventions used in this paper are akin to the notations used by previous works [5, 8, 13]. Let $\mathcal{T} \in \mathbb{R}^{M_1 \times M_2 \times M_3 \times \ldots \times M_n}$ be a tensor of n orders where M_i is an order. In this work, we focus on the third-order tensor, $\mathcal{T} \in \mathbb{R}^{M_1 \times M_2 \times M_3}$. Entries of a tensor are shown using a_{ijk} and the subscript (i, j, k) range from I, J, K in each order. Each element (or entry) of a tensor needs n indices to represent or reference its precise position in a tensor. For example, the element a_{ijk} is an entry value at the i, j and k orders. Given the documents set D, its corresponding set of

CFI subtrees and the set of terms for each CFI subtree, the collection of XML documents is now represented as a third-order tensor $\mathcal{T} \in \mathbb{R}^{D \times CFI \times Terms}$. The tensor is populated with the number of occurrences of the structure-constrained term $Terms_i$ that corresponds to the CFI_j for document D_k. Two optimization techniques are applied on the two orders, CFI and $Terms$, to reduce the size of the tensor. Fig. 1 provides an overview of the XCT method. It begins with mining the CFI subtrees using the PCITMinerConst algorithm and then identifying the constrained content within those CFI subtrees for a given document. Once the structure and content features are obtained for each document, the documents are represented in the TSM along with their structure and content features. The next task is to decompose the created TSM to obtain factorized matrices. Lastly, the K-means algorithm is applied to one of the factorized matrices representing the left singular matrix for the "Document" order U_D and the clusters of documents are obtained.

Input: *Document Dataset;* ***D****, Document Tree Dataset:****DT****, Minimum Support:****min_supp****, Length Constraint:* ***len****, NumCluster:* ***c****, RI Vectors Length:* γ
Output: *Clusters:* $\{Clust_1 \ldots Clust_c\}$
Method:
1. Compute $CFI = \{CFI_1, \ldots, CFI_p\}$ for DT using the PCITMinerConst algorithm for the given *min_supp* and *len*.
2. Form clusters of similar CFI subtrees, $CFISC = \{(CFI_1, \ldots, CFI_q), \ldots, (CFI_t, \ldots, CFI_u)\}$, where $CFISC = \{CFISC_1, \ldots, CFISC_h\}$, $k \ll p$ using large itemset algorithm.
3. For every document $D_i \in D$
   a.Identify the $CFISC$ existing in DT_i, $\delta(DT_i) = \{CFISC_l, \ldots, CFISC_h\}$
   b. For every $CFISC_j$ in $\delta(DT_i)$ retrieve the structure-constrained content in D_i, $C(D_i, CFISC_j) = C(N_1), \ldots, C(N_m)$. The set $C(N_m) = t_1, \ldots, t_k \in Terms$, where *Terms* is the term list in D.
4. Apply random indexing using the γ length random vectors on the terms collection to reduce the term space to $Terms'$
5. Form a tensor $\mathcal{T} \in \mathbb{R}^{D \times CFISC \times Terms'}$, where each tensor element is the number of times a term t_k occurs in $CFISC_j$ for a given document D_i.
6. Apply the proposed tensor decomposition algorithm, PTCD to the tensor $\mathcal{T}$ and get the resulting left singular matrices U_D, U_{CFISC} and $U_{Terms'}$.
7. Apply K-means clustering to U_D to generate the c number of clusters.

Fig. 1. High-level definition of XCT

3.2 Generation of Structure Features for TSM

The Prefix-based Closed Induced Tree Miner (PCITMiner) algorithm [10] is modified to generate the length-constrained CFI subtrees from the document tree dataset DT. The length constrained CFI subtrees are used in this method for the following reasons: (1) Extracting all the CFI subtrees is computationally expensive for datasets with a high branching factor; (2) All CFI subtrees are not required while utilizing them in retrieving the content. In fact the long sized CFI subtrees become more specific and result in retrieving distinct terms associated only with this tree. This may result in a higher number of clusters with uneven sizes. We call the modified algorithm the PCITMinerConst algorithm.

Fig. 1 illustrates the computationally expensive operation of checking whether the mined CFI exists in a given document tree due to the graph isomorphism problem. This step can be optimized by grouping similar subtrees based on their similarity and then retrieving the content corresponding only to the group of similar CFI. A large itemset algorithm for clustering transactional data has been modified to include subtrees, rather than items, to conduct the grouping of the CFI trees based on the similarity of the subtrees. The clusters of CFI subtrees, called Closed Frequent Induced Subtree Cluster ($CFISC$), become a tensor order for representing and analyzing XML documents. Let $CFISC$ be a set of CFI subtrees given by $\{(CFI_1, \ldots, CFI_q)(CFI_r, \ldots, CFI_s)(CFI_t, \ldots, CFI_u)\}$

3.3 Generation of Content Features for TSM

$CFISC$ is used to retrieve the structure-constrained content from the XML documents. We now define the coverage of a $CFISC_j$ and its constrained content for the given document D_i. Compared with the content features of an XML document, the structure-constrained content features include the node values corresponding only to the node labels of the set of CFI subtrees in $CFISC_j$.
Definition 1: Structure-Constrained content features. These features of a given $CFISC_j, C(D_i, CFISC_j)$ of an XML document D_i, are a collection of node values corresponding to the node labels in the $CFISC_j$ where $CFISC_j$ is a cluster of CFI subtrees corresponding to DT_i. The node value of a node (or tag) of a $CFISC_j \in CFISC, C(N_i)$, in D_i is a vector of terms, $\{t_1, \ldots, t_k\}$ that the node contains. The term t is obtained after using pre-processing techniques such as stop-word removal and stemming. Firstly, the CFI subtrees corresponding to the $CFISC_j = \{CFI_r, \ldots, CFI_s\}$ for a given document D_i are flattened into their nodes $\{N_1, \ldots, N_m\} \in N$, where N is the list of nodes in DT. Then the node values of $\{N_1, \ldots, N_m\}$ are accumulated and their occurrences for a document D_i are recorded.

In large datasets, the number of terms in the structure-constrained content is very large with more than 1M terms and 127M tensor entries for INEX (Initiative for Evaluation of XML retrieval) 2009 Wikipedia even after pre-processing. To reduce this very large term space, we apply a Random Indexing (RI) technique which has been favored by many researchers due to its simplicity and low computationally complexity [12]. In RI, each term in the original space is given a randomly generated index vector as shown in Fig. 2. These index vectors are sparse in nature and have ternary values (0 , -1 and 1). Sparsity of the index vectors is controlled via a seed length that specifies the number of randomly selected non-zero features.

$$r_{ij} = \sqrt{3} \begin{cases} +1 \; with \; probability \; 1/6 \\ 0 \; with \; probability \; 2/3 \\ -1 \; with \; probability \; 1/6 \end{cases} \tag{1}$$

We utilize Achlioptas's proposed equation 1 [1] to generate distribution for creating the random index vector for every term in the structure-constrained content of $CFISC$. For a given document D_i, the index vectors of length l for

all the terms corresponding to the given $CFISC_j$ are added. We illustrate this concept of RI on tensor using the Fig. 2, in which we consider a tensor= $\mathbb{R}^{3\times2\times4}$ (in Fig. 2(a)) with 3 documents, 2 $CFISC$, 4 terms and 7 non-zero entries. The entries in the tensor correspond to the occurrences of a given term in the given $CFISC$ for the document. Using that equation 1, the random index vectors of length 6 for the 4 terms are generated (see in Fig. 2(b). Let us consider document D_1 with three tensor entries $a_{121} = 1$, $a_{123} = 1$ and $a_{124} = 1$ corresponding to $CFISC_1$ and three terms $Term_1$, $Term_3$ and $Term_4$. The random vectors (from Fig. 2(b)) are added to these three terms in D_1. The sparse representation of the resulting vector $(a_{12:})$ for D_1 (given in Fig. 2(c)) contains two non-zero tensor entries $a_{123} = 1$ and $a_{124} = -1$. Fig. 2(d) shows the final reduced tensor $\mathcal{T}_r$ in sparse representation containing 6 non-zero entries.This example demonstrates how our technique can reduce the term space for such a small dataset.

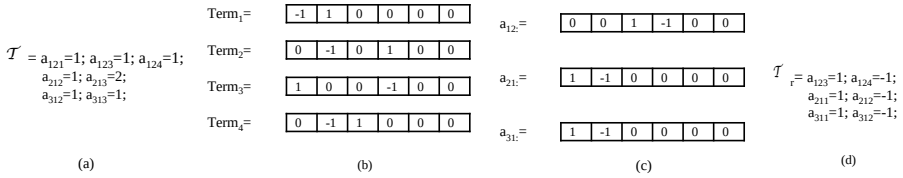


Fig. 2. Illustration of Random Indexing on a 3-order tensor resulting in a randomly reduced tensor $\mathcal{T}_r$

It can be seen that the number of entries in $\mathcal{T}_r$, randomly reduced $\mathcal{T}$, is less in number than its original and maintains the shape of $\mathcal{T}$ as it retains the same similarity between D_2 and D_3. The index vectors in RI are sparse; hence the vectors use less memory store and they are added faster. The randomly-reduced structure-constrained content of $CFISC$ becomes another tensor mode for representing and analyzing XML documents.

3.4 The TSM Representation, Decomposition and Clustering

Given the tensor $\mathcal{T}$, the next task is to find the hidden relationships between the tensor orders. The tensor decomposition algorithms enable an overview of the relationships that can be further used in clustering. However, as already mentioned, most of these decomposition algorithms cannot be applied on very large or dense tensor as the tensors cannot be loaded into memory [13]. To alleviate this problem, the tensors need to be built and unfolded or matricized incrementally. Fig. 3 shows the process of matricization or unfolding along the mode-1 of $\mathcal{T}$ which results in a matrix $\mathrm{T}_{(1)}$. This means that the mode-1 fibers (higher order analogue of rows and columns) are aligned to form a matrix. Essentially this means that the mode-1 fibers of $\mathcal{T}$ are mapped to the rows of matrix $\mathrm{T}_{(1)}$ and the modes-2 and -3 are mapped to the columns of this matrix. We apply the proposed PTCD as shown in Fig. 4 to progressively build and then decompose the

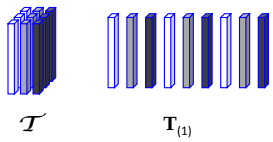


Fig. 3. Mode-1 matricization of a 3-order tensor

tensor using SVD. The motivation for this new tensor decomposition algorithm is that the computations by other decompositions store the fully formed matrices, which are dense and hence cannot scale to very large sized tensors. But PTCD stores the sparse matrices generated progressively and enables further processing to be performed on the tensor. PTCD builds the tensor progressively by unfolding the tensor entries for the user-defined block size b to a sparse matrix $\mathrm{T}'_{(m)}$ where $m \in \{1, 2, \ldots, M\}$ and M is the number of modes. Then this unfolded matrix, $\mathrm{T}'_{(m)}$ is used to update the final sparse matrix $\mathrm{T}_{(m)}$. After updating, all the tensor entries to the final matrix, $\mathrm{T}_{(m)}$ is then decomposed to the user-defined number of required dimensions η using SVD.

Input: Data File: TF, block size: b, number of modes(orders) : M
where $m \in \{1, 2, \ldots, M\}$ and Number of required dimensions: η
Output: *Matricized Tensor*: $\mathbf{T}$
and left singular matrix with η dimensions for 1-mode : $\mathbf{U}_\eta$
1. For every $\mathbf{T}_{(m)} \in \{\mathbf{T}_{(1)}, \mathbf{T}_{(2)}, \ldots, \mathbf{T}_{(M)}\}$
   a. Initialize $\mathbf{T}_{(m)} = \phi$
2. Divide TF into blocks of size b
3. For every block b do
   a. Create tensor $\mathcal{T}_b$
   b. For $m = 1$ to M do
      $\mathbf{T}'_{(m)}$ = Unfold $\mathcal{T}_b$ along its mth mode //Matricize the tensor
      $\mathbf{T}_{(m)} = \mathbf{T}_{(m)} + \mathbf{T}'_{(m)}$ //Update the Mode-m matricized tensor
4. Compute SVD on $\mathbf{T}$ with η dimensions, $\mathbf{T}^\eta = \mathbf{U}_\eta \Sigma_\eta \mathbf{V}_\eta^T$

Fig. 4. Progressive Tensor Creation and Decomposition algorithm (PTCD)

Huang et al. [6] have theoretically proved that HOSVD on a tensor simultaneously reduces the subspace and groups the values in each order. For the 3-order tensor $\mathcal{T}$, the left singular matrix on the document order, $\mathbf{U}_{\eta(D)}$ provides the clustering results on the data index direction; hence they are the cluster indicators for grouping the documents. Consequently, we apply the K-means clustering algorithm on the $\mathbf{U}_{\eta(D)}$ matrix to generate the required number of clusters of the documents.

4 EXPERIMENTS AND DISCUSSION

Experiments are conducted to evaluate the accuracy and scalability performance of XCT on the real-life datasets.

4.1 Datasets

Three real-life XML datasets which have extreme characteristics, INEX 2009 Wikipedia documents collection (Wikipedia)[1], INEX 2006 IEEE [4](IEEE) and ACM SIGMOD (ACM)[2, 10], were used after a careful analysis of a number of datasets. The INEX 2009 document mining track used the Wikipedia dataset

Table 1. Details of the real life datasets

Dataset Names / Attributes	Wikipedia	IEEE	ACM
No. of Docs	54,575	6054	140
No. of tags	34,686	165	38
No. of internal nodes	15,128,407	472,351	2070
Max length of a document	10347	691	45
No. of distinct terms	1,900,072	114,976	7135
Total No. of words	21,480,198	3,695,550	38141
Size of the collection	2.94GB	272MB	1 MB
Presence of formatting tags	Yes	Yes	No
Presence of Schema	Yes	Yes	Yes
Number of Categories	12,803	18	5

with semantically annotated tags to perform the clustering task. This dataset contains a very large number of documents with deeper structure and a high branching factor. It also supports multi-label categories in which one document can have more than one category. On the other hand, IEEE has single-labeled categories and contains more formatting tags and fewer semantic tags. Finally, the ACM is a small dataset that contains 140 XML documents corresponding to two DTDs, IndexTermsPage.dtd and OrdinaryIssuePage.dtd (with about 70 XML documents for each DTD), similar to the setup in XProj [2]. This dataset has been chosen in order to evaluate our method against other representations and decomposition algorithms which could work only on this kind of small datasets.

4.2 Experimental design

Experiments were conducted on the High Performance Computing system, with a RedHat Linux operating system, 16GB of RAM and a 3.4GHz 64bit Intel Xeon processor core. Experiments were conducted to evaluate the accuracy of clustering results of XCT over other clustering techniques, decomposition techniques and representation. Previous research for XML documents clustering [2] has used the ACM to cluster the documents into two groups according to their structural similarity. To compare our work with this earlier research, we conducted our experiments not only with two cluster categories according to structural similarity

[1] http://www.inex.otago.ac.nz/tracks/wiki-mine/wiki-mine.asp

but also on 5 categories using expert knowledge considering both the structure and the content features of XML documents. Due to the small number of terms in this dataset, the random indexing option for XCT has been disabled.

Following are the representation and the other existing algorithms used for comparing the outputs of the proposed XCT method.

Structure Only (SO) Representation: An input matrix $D \times CFI$ is generated similar to XProj [2].

Content Only (CO) Representation: The content of XML documents is represented in a matrix $D \times Terms$ with each matrix entry containing term frequency of terms in D.

Structure and Content Representation (S+C) using VSM: The structure and the content features for the documents are represented in a matrix by concatenating the CO and SO representations side by side.

Clustering using CP and Tucker: The left singular matrix resulting from applying CP or Tucker decomposition on the tensor is used as an input for k-means clustering.

Clustering using MACH: The MACH decomposition technique has been applied on the original tensor with random indexing. MACH randomly projects the original tensor to a reduced tensor with smaller percentage of entries (10% from the original tensor as specified in [13]) and then uses Tucker decomposition to decompose the reduced tensor. To compare with XCT, we apply k-means clustering on the left singular matrix to group the documents.

Moreover, since the INEX dataset has been used by other researchers: we provide the results cited by other researchers [4, 11] as well in our analysis.

4.3 Evaluation measures

The standard criterion of purity is used to determine the quality of clusters by measuring the extent to which each cluster contains documents primarily from one class. The macro and micro purity of the entire clustering solution is obtained as a weighted sum of the individual cluster purity. In general, the larger the value of purity,the better the clustering solution is.

$$Purity = \frac{\#\ Documents\ with\ the\ majority\ label\ in\ cluster\ k}{\#\ Documents\ in\ cluster\ k} \tag{2}$$

$$Micro - Purity = \frac{\sum_{k=0}^{n} Purity(k) * \#\ Documents\ Found\ By\ Class(k)}{\sum_{k=0}^{n} \#\ Documents\ Found\ By\ Class(k)} \tag{3}$$

$$Macro - Purity = \frac{\sum_{k=0}^{n} Purity(k))}{Total\ Number\ of\ Categories} \tag{4}$$

4.4 Empirical Analysis

Accuracy of Clustering: Tables 2 and 3 provide the purity results of clustering on the datasets using XCT, other representations and other decomposition algorithms. As can be seen from these three tables, the proposed XCT method not only outperforms our benchmarks but also other INEX submissions in terms

Table 2. Clustering results on Wikipedia and IEEE

Methods	#50 clusters		#100 clusters	
	Micro-purity	Macro-purity	Micro-purity	Macro-purity
XCT	0.13	0.14	0.14	0.13
S+C using VSM	0.13	0.14	0.13	0.14
Clustering using MACH	0.087	0.089	0.089	0.088
CO	0.10	0.12	0.12	0.13
SO	0.09	0.11	0.11	0.10
BilWeb-CO[11]	NA	NA	0.10	0.13

Methods	Micro-purity	Macro-purity
XCT	0.23	0.23
S+C using VSM	0.18	0.14
Clustering using MACH	0.17	0.20
CO	0.10	0.12
SO	0.08	0.10
Nayak et. al [4]	NA	0.18
Doucet et. al [4]	NA	0.13

Table 3. Results of clustering on ACM

Methods	#2 clusters		#5 clusters	
	Micro-purity	Macro-purity	Micro-purity	Macro-purity
XCT	1	1	0.91	0.91
S+C using VSM	0.98	0.98	0.75	0.79
Clustering using MACH	0.97	0.93	0.70	0.75
Clustering using Tucker	0.56	0.48	0.59	0.87
Clustering using CP	0.89	0.93	0.84	0.93
CO	0.97	0.94	0.73	0.78
SO	1	1	0.64	0.72

of the accuracy of their clustering solution. It should be noted that algorithms such as CP, Tucker could not scale even to the medium-sized dataset, IEEE and hence their results were not reported but the proposed PTCD was able to decompose even large dataset as shown in Tables 2 and 3. As the categories in

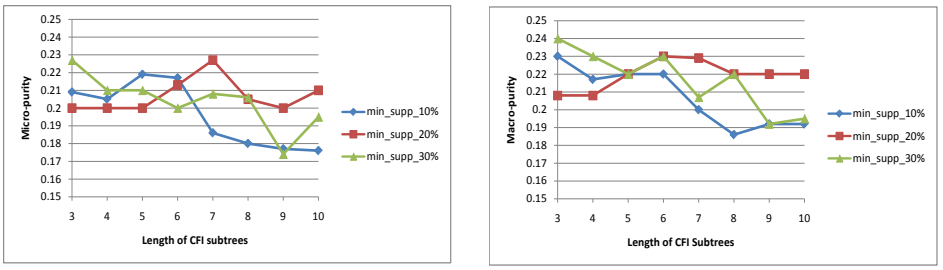
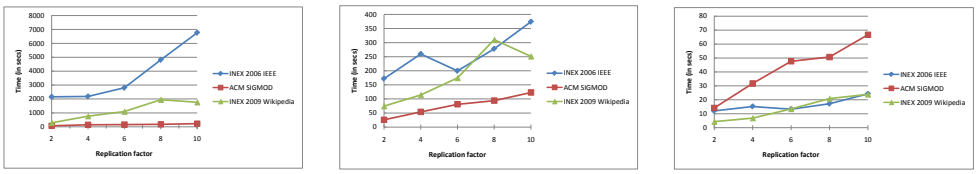


Fig. 5. Sensitivity of length constraint on the micro-purity and macro-purity values for IEEE

IEEE was based on both the structure and the content we utilized this dataset for analyzing the sensitivity of the length constraint and *min_supp* values on the purity. We conducted experiments by varying the length constraint(***len***) of the CFI subtrees from 3 to 10 for support thresholds from 10% to 30%. From Fig. 5 which indicates that with the increase in the length constraint the micro-purity and macro-purity values drops especially at 10% and 30% support threshold. Also, length constraint of over 7 shows a negative impact on the purity. With longer length patterns the content corresponding to the CFI subtrees becomes specific and hence results in less accuracy than the content corresponding to



(a) XCT (b) PTCD (c) Decomposition in PTCD

Fig. 6. Scalability analysis

shorter subtrees. This shows the suitability of constraining the CFI subtrees as in PCITMinerConst.

Time complexity analysis: The time complexity of XCT is composed of five major components, namely frequent mining for CFI subtrees, clustering of CFI subtrees, random indexing, matricization and decomposition in PTCD. It is given by $O(dsm) + O(drp) + O(tk\gamma) + O(dr\gamma) + O(dk'\gamma)$ where d represents the number of documents, s is the number of 1-Length CFI subtrees, m is the number of PCITMinerConst iterations, r is the number of structure-based clusters, p is the number of similarity computation iterations, γ is the size of the random index vector, k and k' are the non-zero entries per column in the tensor before random indexing and in the matricized tensor after random indexing respectively. The time complexity of PTCD is $O(dr\gamma) + O(dk'\gamma)$, which includes the cost of matricization along the *mode-n* and the sparse SVD.

Scalability Test: All the three datasets were used for this analysis with min_supp at 10%, ***len*** at 5, γ at 100 and the number of clusters chosen, 5, 18 and 100 respectively. Also we used 1000 documents each for IEEE and Wikipedia. The execution time of PCITMinerConst is less than a few 10 milliseconds and hence it has not been reported as it is not of much significance. We can see from Fig. 6 that both XCT and PTCD scale nearly linearly with the dataset size. The PTCD algorithm includes two main steps: (1)loading the tensor file into memory by matricization, and (2) decomposing the matrices using SVD. As it can be seen from Fig. 6 that minimal time is spent on decomposition and a large chunk of time on loading the tensor file into memory. Also, it is interesting to note the PTCD for ACM is greater in comparison to IEEE which indicates that random indexing option in XCT method helps to reduce the complexity of decomposition by reducing the term space.

An interesting problem is how to choose the value γ for the seed length in random indexing. The Johnson- Lindenstrauss result was used to get the bounds for γ as given by $\gamma = \lceil 4(\epsilon^2/2 - \epsilon^3/3)^{-1} ln\ n \rfloor$. However, we found that for Wikipedia dataset with ϵ=0.5, γ is 433 but in the experiments, $\gamma \approx 100$ was sufficient to obtain good accuracy similar to the results by [3].

5 Conclusion

In this paper, we have proposed a clustering method, XCT, for effectively combining both the structure and the content features in XML documents using TSM model. The experimental results clearly ascertain that XCT outperforms other existing approaches in improving accuracy. Also, our proposed decomposition algorithm PTCD has demonstrated that it has potential for decomposing tensors effectively and could scale for very large datasets. Our future work will focus on reducing the complexity of XCT and applying it on various types of other types of semi-structured documents.

References

1. D. Achlioptas and F. Mcsherry. Fast computation of low-rank matrix approximations. *J. ACM*, 54(2):9, 2007.
2. C. C. Aggarwal, N. Ta, J. Wang, J. Feng, and M. Zaki. Xproj: a framework for projected structural clustering of xml documents. In *KDD*, pages 46–55. ACM, USA, 2007.
3. E. Bingham and H. Mannila. Random projection in dimensionality reduction: applications to image and text data. In *KDD*, pages 245–250, USA, 2001. ACM.
4. L. Denoyer and P. Gallinari. Report on the xml mining track at inex 2005 and inex 2006: categorization and clustering of xml documents. *SIGIR Forum*, 41(1):79–90, 2007.
5. A. Evrim, B, and Y. lent. Unsupervised multiway data analysis: A literature survey. *IEEE TKDE*, 21(1):6–20, 2009.
6. H. Huang, C. Ding, D. Luo, and T. Li. Simultaneous tensor subspace selection and clustering: the equivalence of high order svd and k-means clustering. In *KDD*, pages 327–335, USA, 2008. ACM.
7. S. Jegelka, S. Sra, and A. Banerjee. Approximation algorithms for tensor clustering. In R. Gavald, G. Lugosi, T. Zeugmann, and S. Zilles, editors, *Algorithmic Learning Theory*, volume 5809 of *LNCS*, pages 368–383. Springer, 2009.
8. S. Jimeng, T. Dacheng, P. Spiros, S. Y. Philip, and F. Christos. Incremental tensor analysis: Theory and applications. *ACM TKDD*, 2(3):1–37, 2008.
9. T. G. Kolda and J. Sun. Scalable tensor decompositions for multi-aspect data mining. In *ICDM*, pages 363–372, 2008.
10. S. Kutty, R. Nayak, and Y. Li. HCX: an efficient hybrid clustering approach for XML documents. In *DocEng*, pages 94–97, USA, 2009. ACM.
11. R. Nayak, C. deVries, S. Kutty, and S. Geva. Report on the XML Mining Track Clustering Task at INEX 2009. In S. Geva, J. Kamps, and A. Trotman, editors, *Focused Retrieval and Evaluation*. Springer, 2010.
12. M. Sahlgren. An introduction to random indexing. In *Methods and Applications of Semantic Indexing Workshop, TKE*, 2005.
13. C. E. Tsourakakis. MACH: Fast Randomized Tensor Decompositions. In *SDM*, pages 689–700, USA, 2010.
14. A.-M. Vercoustre, M. Fegas, S. Gul, and Y. Lechevallier. A flexible structured-based representation for xml document mining. In N. Fuhr, M. Lalmas, S. Malik, and G. Kazai, editors, *Advances in XML Information Retrieval and Evaluation*, volume 3977 of *LNCS*, pages 443–457. Springer, 2006.