

QUT Digital Repository:
<http://eprints.qut.edu.au/>



Kutty, Sangeetha and Nayak, Richi and Li, Yuefeng Y. (2009) *XCFS - a novel approach for clustering XML documents using both the structure and the content*. In: Association for Computing Machinery , 2-6 November 2009, Asia World-Expo, Hong Kong.

© Copyright 2009 [please consult the authors]

XCFS – A Novel Approach for Clustering XML documents using both the Structure and the Content

ABSTRACT

An XML clustering algorithm should process both structural and content information of XML documents in order to improve the accuracy and meaning of the clustering solution. However, the inclusion of both kinds of information in the clustering process results in a huge overhead for the underlying clustering algorithm because of the high dimensionality of the data. This paper introduces a novel approach that first determines structural similarity in the form of frequent subtrees and then uses these frequent subtrees to represent the constrained content of the XML documents in order to determine the content similarity. The proposed method reduces the high dimensionality of input data by using only the structure-constrained content. The empirical analysis reveals that the proposed method can effectively cluster even very large XML datasets and outperform other existing methods.

Categories and Subject Descriptors

I.7.2 [Document and Text Processing]: Document Preparation – Markup Languages; H.2.8 [Database Management]: Database Applications – Data mining; H.3 [Information Systems]: Information Storage and Search and Retrieval.

General Terms

Algorithms, Management, Design Experimentation.

Keywords

XML documents, Clustering, Frequent mining, Subtree mining, Structure and content.

1. INTRODUCTION

Due to the inherent flexibility in document representation, XML (*eXtensible Markup Language*) has become a ubiquitous standard [5] for information representation and exchange on the Internet and in intranets. Its increased popularity has raised many issues regarding methods of data management and retrieval in large repositories. Clustering of similar XML documents has been seen as potentially one of the more effective solutions to improve document handling by facilitating better information retrieval, data indexing, data integration and query processing [14].

Unlike the clustering of text documents, XML document clustering is an intricate process. XML documents contain structural features (that is, element tags and their nesting therein) as well as their text data (called content features in this paper). These structural features could show the relationship between the content features of the XML document enclosed at various levels. Most of the existing clustering methods for XML documents use either the structural features [1,6,8,10] or the content features [12] for grouping similar documents, because of the sheer size and complexity of using the data elements of both features, in matrix/tree/graph representation, and then processing those elements to identify similarity. However, as the content of an XML document is stored using its structure, it is essential to

combine both structural and content features to derive meaningful clustering results.

XML clustering techniques that use only the content features to determine the similarity between XML documents result in poor performance for situations in which the hierarchical structure within the documents plays an important role in determining the groups [3]. On the other hand, grouping XML documents based only on their structural similarity fails to provide accurate results in the following two situations: (1) when the grouping exists according to the theme/content; or (2) when the documents belong to the same schema [8]. By including both the structural and the content features within the clustering process, not only are meaningful clustering solutions derived, but also some of the problems inherent in XML document handling, such as polysemy and hyponymy, can be alleviated.

This paper proposes a novel way to represent the common substructures of XML documents using frequent subtrees to represent the structural similarity among them. It also aims to utilize these frequent subtrees when representing content features. By eliminating the content corresponding to infrequent substructures, the dimensionality of the input (content) data is reduced. Our proposed method could group the documents based on the content and structural similarity by containing only the relevant content constrained by the common substructure of the documents.

There are two popular forms of frequent subtrees: induced and embedded [2]. The induced subtrees maintain the parent-child relationship among its nodes and the embedded subtrees preserve the ancestor-descendant relationship among its nodes. In this research, we propose to utilize the embedded subtrees to facilitate the clustering process as this type of subtree presents a less stringent relationship among its nodes in comparison to induced subtrees and hence increases the possibility to discover some of the hidden similarities that may exist between trees [2, 13,17]. These similarity values improve the quality of clustering solutions. In spite of its benefits, the number of embedded subtrees generated could result in information explosion [4]. In order to control the number of embedded subtrees, we propose to generate only the concise but “lossless” representation of frequent subtrees, called Closed Frequent Embedded (*CFE*) subtrees, using the pattern-growth technique in a computationally efficient manner.

We utilize the generated *CFE* subtrees to represent the content of XML documents in a Vector Space Model (*VSM*). A *Pre-Cluster Form (PCF)* and an *Intermediate Cluster Form (ICF)* are proposed for efficient representation of XML data for clustering. *PCF* is a vector of the sum of occurrences of the unique terms of the constrained content corresponding to a set of *CFE* subtrees in a given XML document. *ICF* is a collection of *PCFs* for all the documents in the collection. The constrained content in *ICF*, representing the *CFE* subtrees, preserves the structural similarity among the XML documents. The computation of content similarity on the *ICF* matrix includes the structural relationship and similarity that exist in the set of documents implicitly. It

should be noted that the empirical evaluation indicates that the inclusion of structural relationship and similarity in the content improves the accuracy of the clustering solution.

In this study, we explore the effectiveness of using the *CFE* subtrees and the constrained content in the clustering process. In particular, the significant contributions are: (1) a novel representation called *CFE* subtrees which encodes the structural relationships among XML documents; (2) a computationally effective frequent mining approach to identify *CFE* subtrees among a large collection of XML documents; and (3) a novel way to combine both the semantic and structural information of XML documents using the proposed *CFE* subtrees, *PCF* and *ICF* for efficient clustering.

The remainder of the paper has 7 sections. Section 2 discusses the related works. Section 3 covers the overview of the XCFS method. Section 4 details Phase-I of XCFS for generating the frequent subtrees. Section 5 covers the details on using these frequent subtrees to represent the content for clustering in Phase-II of XCFS. In Section 6, the empirical analysis is presented, and the conclusion is presented in Section 7.

2. RELATED WORK

XML document clustering has been an active area of research. However, with some exceptions [14,15,16], clustering using both structural and content features has received little attention due to the complexity of the process. Clustering approaches using both these features fail to scale for even small-sized datasets [15] or have reported poor accuracy [14,16]. Hence, most of the works on XML clustering focus on utilizing either the structure or the content of XML documents.

Several XML clustering methods determining structural similarity have been based on paths shared between documents [1, 8,10]. However, the substructure paths do not include the sibling relation between the nodes in a tree; this type of approach may result in structural information loss. There have also been several clustering methods developed that use only the content features of XML documents [12]. XML is primarily used to represent the text in a hierarchical structure. By excluding the structural information in the input representation, these methods may fail in situations where the hierarchical structure of the documents plays a dominant role in determining the grouping of documents. Recently, Yao and Zarida [16] utilized the paths to cluster XML documents using both structural and content features. Their experimental results showed that the clustering performance (F1-Score) degrades by including the structure into content on the Initiative for the Evaluation of XML Retrieval (INEX) 2007 Wikipedia dataset. In comparison to using paths, the use of frequent subtrees in the proposed XCFS approach represents the common substructures and preserves the hierarchical and sibling relationships among the nodes.

Various algorithms [9,13,17,18] have been proposed to extract either induced or embedded frequent subtrees efficiently from tree datasets such as the XML documents. There are two popular approaches for mining embedded subtrees, namely *generate-and-test* and *prefix-based pattern-growth*. The representatives for generate-and-test algorithms, TreeMinerV, TreeMinerH [17], X3-Miner and IMB3-Miner [13] utilize either the join enumeration strategy or the schema-guided TMG (Tree Model Guided) approach. In recent years, *prefix-based pattern-growth* algorithms

have gained attention due to their improved performance over the *generate-and-test* techniques. One such approach is PrefixTreeESpan [18], which avoids unnecessary generation of candidates inherent in *generate-and-test* technique.

However, both types of embedded subtree mining algorithms face an immense disadvantage. There are often situations in which the number of frequent embedded subtrees increases exponentially with the size of the tree dataset. For very large datasets, often the embedded subtree mining algorithms fail to provide a complete output. In order to provide a feasible solution as well as to improve the performance, we focus on generating a concise but “lossless” representation of frequent subtrees, called Closed Frequent Embedded (*CFE*) subtrees. To the best of our knowledge, there exists no subtree mining algorithm to generate *CFE* subtrees.

3. THE XCFS APPROACH: OVERVIEW AND DEFINITIONS

The proposed XCFS approach involves two phases, as shown in Figure 1.

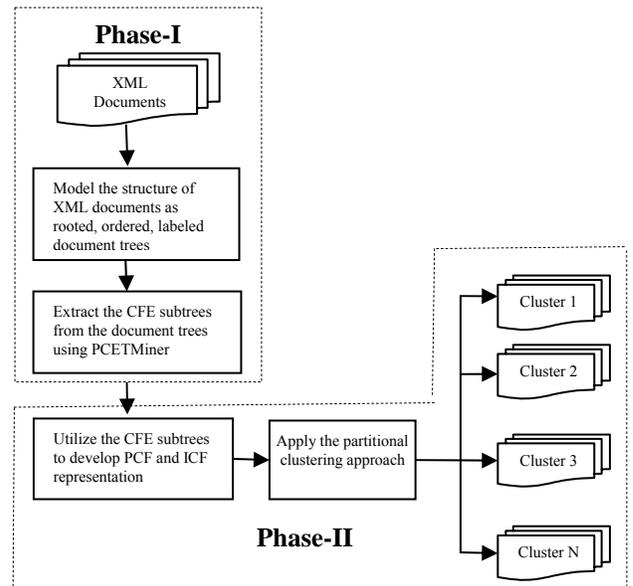


Figure 1. Overview of XCFS

Phase-I of XCFS begins with the modeling of XML documents as rooted, ordered, and labeled trees using their structure. We do not distinguish between attributes and element tag names of an XML document, since both are mapped to the node label set. Let there be an XML document D_i containing tags (or node labels) and data (or node values) enclosed within those tags. The structure of the XML document D_i can be defined as a list of tags showing the hierarchical relationships between nodes in the document. The structure of an XML document is modeled as a rooted, ordered, and labeled document tree. A document tree DT_i representing the structure of an XML document D_i is denoted as $DT_i = (N, n0, E, f)$, where (1) N is the set of nodes or vertices that exist in the document tree DT_i ; (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 $f: E \rightarrow N \times N$.

The next task is to identify the common or frequent embedded subtrees in the document tree dataset DT , based on the user-defined minimum support (min_supp). Assume there is a collection of XML documents $D = \{D_1, D_2, D_3, \dots, D_n\}$ modeled as a corresponding collection of document trees $DT = \{DT_1, DT_2, DT_3, \dots, DT_n\}$. For a document tree DT_i with node set N and edge set E , a tree dt_i with node set n' and edge set e' is an embedded subtree of DT_i iff (1) $n' \subseteq N$; (2) $e' \subseteq E$; (3) the labeling of nodes of n' in dt_i is preserved in DT_i ; (4) $(n1, n2) \in e'$ where $n1$ is the ancestor node of $n2$ in dt_i iff $n1$ is the ancestor node of $n2$ in DT_i ; and (5) for $n1, n2 \in n'$, $preorder(n1) < preorder(n2)$ in dt_i iff $preorder(n1) < preorder(n2)$ in DT_i . In other words, an embedded subtree dt_i preserves the ancestor–descendant relationship (including parent–child relationship) among the nodes of the document tree, DT_i . Figure 2(b) shows an embedded subtree generated from the document tree in Figure 2(a). By imposing a less strict relationship among the nodes, embedded subtrees could identify hidden common structural information and also avoid information loss.

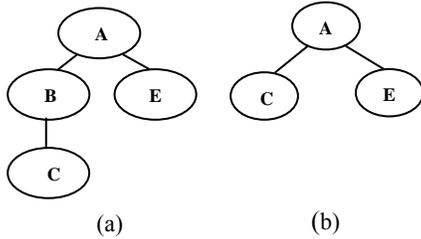


Figure 2 (a). A Document tree (b). an Embedded subtree

Mining for the frequent embedded subtrees from a large collection of trees usually results in a huge number of frequent subtrees generated at a lower support threshold. Consequently, the subtree mining algorithms become intractable [2]. It is essential to control the number of subtrees generated. In order to reduce the number of subtrees, two concise representations of frequent subtrees, namely maximal and closed, were proposed [2]. The maximal frequent subtree representation suffers from information loss [9] and hence, in this paper, we chose to generate Closed Frequent Embedded (CFE) subtrees.

Definition 1: Closed Frequent Embedded (CFE) Subtree

In DT , there exist two frequent embedded subtrees dt_i and dt_j . The frequent embedded subtree dt_i is closed of dt_j iff (1) dt_i is an embedded subtree of dt_j , $supp(dt_i) = supp(dt_j)$; (2) there exists no supertree for dt_i having the same support as that of dt_i . This property is called embedded closure and dt_i is the closed frequent embedded subtree of dt_j .

The $Support(dt_i)$ (or $frequency(dt_i)$) of an embedded subtree dt_i is the number of document trees in DT for which dt_i is a subtree. dt_i is a frequent subtree among the document trees DT such that $(frequency(dt_i)/|DT|) \geq min_supp$, where min_supp is the user-given minimum support.

We develop the CFETreeMiner algorithm to generate the CFE subtrees utilizing backward scan and extension closure checking techniques to incorporate closure checking in the prefix-based pattern-growth algorithm. The backward scan attempts to reduce the search space by avoiding projections for frequent subtrees which have the same ancestor node. The extension closure

technique checks for closure of frequent subtrees by counting the support of the subtrees efficiently.

The generated CFE subtrees are then utilized to represent the content of XML documents. An XML document can now be represented by the number of CFEs that it contains. The content features of the XML document D_i , $C(D_i)$, are a collection of node values for all the node labels in the document, D_i . Node value for a node (or tag), $C(N_i)$, in an XML document is a vector of terms, $\{t_1, \dots, t_k\}$ that the node contains. The term t is obtained after stop-word removal and stemming [11].

Definition 2: Structure-Constrained content features of an XML document according to the CFEs that represent it

The structure-constrained content features of a given CFE $_i$, $C(D_i, CFE_i)$ of an XML document D_i , are a collection of node values corresponding to the node labels in CFE $_i$ of D_i . The structure-constrained content features of a set of CFE subtrees, (CFE), $C(D_i, CFE)$, that represent a XML document D_i , are a collection of node values corresponding to node labels in the set of CFE subtrees (CFE) of D_i .

Compared with the content features of an XML document, the structure-constrained content features include the node values corresponding to the node labels of the set of CFE subtrees only. Figure 3 illustrates a high-level description of the XCFS algorithm.

XCFS
Input: Document Dataset: D , Document Tree Dataset: DT , Minimum Support: min_supp , NumCluster: c
Output: Clusters: $\{Clust_1, \dots, Clust_c\}$
Begin

1. Compute the CFE subtrees $CFE = \{CFE_1, \dots, CFE_p\}$ for DT using proposed backward scan and extension closure techniques with CFETreeMiner.
2. For every document $D_i \in D$
 - a. Identify the coverage of the document tree DT_i , $\delta(DT_i) = \{CFE_1, \dots, CFE_k\}$.
 - b. For every CFE $_i$ in $\delta(DT_i)$ extract the structure-constrained content in D_i , $C(D_i, CFE_i) = \{C(N_1), \dots, C(N_m)\}$. The set $C(N_i) = \{t_1, \dots, t_k\} \in T$, where T is the term list for the document D_i .
 - c. Represent the PCF of D_i as a vector of the sum of occurrences of the unique terms in $C(D_i, \delta(DT_i))$.
3. Combine the PCFs of all D_i in D to form ICF
4. Divide the ICF into two clusters $Clust_x$ and $Clust_y$.
5. If the similarity criterion of the content collection clusters $Clust_x$ and $Clust_y$ is greater than a threshold then bisect $Clust_x$ and $Clust_y$ until the number of clusters obtained is c .

End

Figure 3. High level definition of the XCFS approach

Phase-I of XCFS, which involves generating CFE subtrees from the dataset using CFETreeMiner, is presented in step 1 in this figure. Phase-II of XCFS that includes the generation of two data structures: PCF and ICF and clustering them, is presented in steps 2 to 5. The PCF is a vector of the sum of occurrences of the distinct terms in the structure-constrained content features of a document D_i . ICF is a collection of PCFs for the document collection D . A partitioning clustering approach is then applied to the ICF representation to group the XML documents in the required number of clusters, based on their structural and semantic similarity.

4. PHASE-I OF XCFS: GENERATING CFE SUBTREES USING CFETREEMINER

XCFS conducts frequent subtree mining using the prefix-based pattern-growth approach, considering the patterns as subtrees. The prefix-based pattern-growth is an efficient approach in comparison to the *generate-and-test* approach as it avoids the expensive task of candidate subtree generation that is inherent in *generate-and-test* approaches [18].

4.1 Frequent Embedded Subtree mining

Consider a document tree dataset DT in which the document trees are represented in a pre-order string format as $\langle X^a Y^b Z^c \dots K^n -1 -1 \rangle$ where X, Y, Z and K are node labels and superscripts a, b, c, \dots, n are the increasing positions of nodes in the pre-order traversal of the tree. ‘-1’ is used to indicate the end flag for the corresponding node. We will be using the example document trees dataset DT shown in Table 1 to explain the concepts of the proposed CFETreeMiner algorithm.

Table 1. Document tree dataset Example (DT)

Tree Id	Tree Pre-order string
1	$\langle A^1 B^2 C^3 -1 -1 E^4 -1 -1 \rangle$
2	$\langle A^1 B^2 C^3 -1 -1 E^4 F^5 -1 -1 -1 \rangle$
3	$\langle A^1 E^2 F^3 -1 G^4 -1 -1 -1 \rangle$

The *prefix-based pattern-growth* for mining frequent subtrees involves three phases:

1. The 1-*Length* frequent subtree generation
2. Projecting the dataset using the prefix-trees
3. Mining the prefix-tree projected dataset

4.1.1 The 1-*Length* frequent subtree generation.

The prefix-based subtree growth technique starts with a scan of the DT dataset to determine the 1-*Length* frequent subtrees with a minimum support. A 1-*Length* frequent subtree containing a single node is represented using the following subtree pre-order string format as $Sub_X = (\langle X^a -1 \rangle : Supp)$. The subtree pre-order representation includes an element called *Supp* that indicates the support value of the subtree.

In the running example, the 1-*Length* frequent subtrees from DT , setting $min_supp=2$, are $(\langle A^1 -1 \rangle : 3)$, $(\langle B^1 -1 \rangle : 2)$, $(\langle C^1 -1 \rangle : 2)$, $(\langle E^1 -1 \rangle : 3)$ $(\langle F^1 -1 \rangle : 2)$. Using these generated 1-*Length* frequent subtrees as prefix-trees, DT is projected. Let there be a document tree DT_p with m nodes and T_p be a tree with n nodes where $n \leq m$. The pre-order scanning of the document tree DT_p from its root until its n -th node results in a tree T_j . If T_j is isomorphic to T_p then T_p is called the prefix tree of document tree DT_p [25].

The prefix-trees of the document tree (DT_1), with a tree-id of 1 with respect to 1-*Length* frequent subtree $(\langle A^1 -1 \rangle : 3)$, is given by $\langle A^1 B^2 -1 -1 \rangle$, $\langle A^1 B^2 C^3 -1 -1 -1 \rangle$, $\langle A^1 B^2 C^3 -1 -1 E^4 -1 -1 \rangle$ in our running example.

4.1.2 Projecting the Dataset using the Prefix-trees.

The next step in this process involves projecting the dataset using the 1-*Length* frequent subtrees as prefix-trees. To build the T_p prefix-projected dataset, every document tree in DT is checked whether it contains the prefix-tree T_p . If a document tree DT_i contains T_p then its projected instance for T_p is constructed.

Consider a prefix-tree T_p with n nodes. If there exists a document tree $DT_i \in DT$ with m nodes containing the prefix-tree T_p , then the T_p -prefix projected instance of DT_i is the pre-order scanning of DT_i from $n+1$ node to m . A T_p -prefix-tree projected dataset is obtained by constructing the T_p -prefix-tree projected instances for all the document trees in DT . For the document tree (DT_1) in the running example, the projected instances for the prefix-tree $\langle A^1 -1 \rangle$ are $\langle B^2 C^3 -1 -1 \rangle$ and $\langle E^4 -1 -1 \rangle$.

4.1.3 Mining the Prefix-tree Projected Dataset.

Each of the prefix-tree projected dataset is now mined to identify the Growth Node (GN).

Definition 3: Growth Node (GN)

Given two prefix-trees T_p and T_p' with m and $m+1$ nodes respectively, where T_p is the prefix of T_p' . If there occurs a node n in T_p' , but, not in T_p then the node n is the Growth Node (GN) of T_p with respect to T_p' .

If a GN is frequent then the prefix-tree with the GN forms the frequent embedded subtree. For each of the frequent GN the corresponding projection is constructed and mined recursively until there are no more frequent GNs to be projected.

In the running example, several frequent GNs namely ‘B’, ‘C’, ‘E’ and ‘F’ (support $\geq min_supp$) are found in order to mine the prefix-tree projected dataset containing the prefix-tree dataset of $\langle A^1 -1 \rangle$. These frequent GNs are combined with the prefix-tree $\langle A^1 -1 \rangle$ to form their corresponding new prefix-trees $\langle A^1 B^2 -1 -1 \rangle$, $\langle A^1 C^2 -1 -1 \rangle$, $\langle A^1 E^2 -1 -1 \rangle$, $\langle A^1 F^2 -1 -1 \rangle$. These prefix-trees are used to construct their corresponding projections to mine the dataset recursively.

4.2 Closed Frequent Embedded Subtree Mining

So far, we have discussed the basic process of generating frequent embedded subtrees using the prefix-pattern growth approach. Now we propose the method to apply the closure property to provide concise and lossless frequent subtrees. The following example will demonstrate the potential of closed frequent subtrees over frequent subtrees.

Considering the running example dataset DT in Table 1, a set of 15 frequent embedded subtrees, $freqT(DT)$, can be extracted. It is $\{(\langle A^1 -1 \rangle : 3)$, $(\langle A^1 B^2 -1 -1 \rangle : 2)$, $(\langle B^1 -1 \rangle : 2)$, $(\langle B^1 C^2 -1 -1 \rangle : 2)$, $(\langle A^1 B^2 C^3 -1 -1 -1 \rangle : 2)$, $(\langle A^1 B^2 C^3 -1 -1 E^4 -1 -1 \rangle : 2)$, $(\langle A^1 B^2 E^3 -1 -1 -1 \rangle : 2)$, $(\langle C^1 -1 \rangle : 2)$, $(\langle A^1 C^2 -1 -1 \rangle : 2)$, $(\langle A^1 C^2 -1 E^3 -1 -1 \rangle : 2)$, $(\langle E^1 -1 \rangle : 3)$, $(\langle A^1 E^2 -1 -1 \rangle : 3)$, $(\langle A^1 E^2 F^3 -1 -1 -1 \rangle : 2)$, $(\langle F^1 -1 \rangle : 2)$, $(\langle A^1 F^2 -1 -1 \rangle : 2)\}$. It can be noted that $(\langle A^1 B^2 C^3 -1 -1 E^4 -1 -1 \rangle : 2)$ is the superset of the 8 subtrees having the same support value of 2 in this set and can replace them. Similarly, $(\langle A^1 E^2 -1 -1 \rangle : 3)$ and $(\langle A^1 F^2 -1 -1 \rangle : 2)$ can also replace their respective subset subtrees that have the same support. The proposed CFETMiner algorithm method would produce a set of only three closed frequent embedded (CFE) subtrees, $freqT(DT) : \{(\langle A^1 E^2 -1 -1 \rangle : 3)$, $(\langle A^1 F^2 -1 -1 \rangle : 2)$, $(\langle A^1 B^2 C^3 -1 -1 E^4 -1 -1 \rangle : 2)\}$.

The challenge is to impose closure on the frequent embedded subtrees using prefix-based subtree mining. A naïve approach to impose closure is to first generate all the frequent embedded subtrees and then eliminate the subtrees based on their support by checking the closure property. It is an expensive task when there are a large number of frequent subtrees generated. Additionally,

there is extra processing involved. A method should generate the closed frequent result set such that it reduces the overall time by benefiting from the reduced number of projections. There are a number of approaches proposed in the frequent itemset and sequential mining [15,18]. Unlike the itemset or sequential mining, trees have branches and these closure mechanisms cannot be directly applied. In CFETreeMiner uses two mechanisms to apply closure efficiently:

1. Search Space reduction using the backward scan
2. Node Extension Closure checking

4.2.1. Search Space Reduction using a Backward Scan

This technique does a backward scan to reduce the search space using the following lemma:

Lemma 1:

Let there be two 1-Length prefix-trees T_p and T_p' having the node labeled k and k' respectively for a given document tree dataset DT . If k' is the ancestor node of k in all trees in DT then the projection of T_p is stopped as the projection of the subtree T_p' based on k' includes all the subtrees generated using the prefix-tree T_p .

This technique is applied after the generation of 1-Length frequent subtrees obtained by scanning the document tree dataset. The root nodes in the trees are exempted from the backward scan. In our running example, among the 1-Length frequent embedded subtrees ($\langle A^1-1 \rangle:3$), ($\langle B^1-1 \rangle:2$), ($\langle C^1-1 \rangle:2$), ($\langle E^1-1 \rangle:3$) and ($\langle F^1-1 \rangle:2$), the subtree having the node labeled as 'A' is a root node in all the trees in the DT , so it is not checked for its ancestors. But the subtrees containing the internal nodes 'B', 'C', 'E' and 'F' are checked for their ancestor node in all the document trees in DT . If it reveals that the ancestor node is 'A' in all the trees then there is no need to project the subtrees ($\langle B^1-1 \rangle:2$), ($\langle C^1-1 \rangle:2$), ($\langle E^1-1 \rangle:3$) and ($\langle F^1-1 \rangle:2$) as the projection of ($\langle A^1-1 \rangle:3$) includes the projections of all the other subtrees. By doing so, the number of subtrees and the number of projections required are reduced. Due to the reduced search space, the efficiency of the algorithm is improved.

4.2.2 Node Extension check for closure

With repeated projections and mining of the projected dataset, there occurs a possibility that some of the subtrees generated are not closed. In order to check the closure for generated frequent subtrees, the node extension closure checking is performed.

From Definition 1 on *CFE* subtrees in Section 3, a prefix-tree T_p is non-closed if there exists at least one prefix-tree T_p' that has the same support as that of T_p . With the use of the prefix-based subtree growth technique to generate frequent subtrees, the prefix-tree T_p' can occur in two possible ways:

1. In the same prefix-projected dataset of T_p
2. In a different prefix-projected dataset of T_p

Checking the closure of T_p with respect to T_p' varies according to the way T_p' occurs. The *Growth Node extension closure checking lemma* is used when the prefix-tree T_p' occurs in the same prefix-projected dataset of T_p . The *ancestor node closure checking lemma* is used when the prefix-tree T_p' occurs in a different prefix-projected dataset of T_p .

Considering the running example, the prefix-tree $T_p' = \langle A^1 B^2-1-1 \rangle$ occurs in the same prefix-projected dataset as that of $T_p = \langle A^1-1 \rangle$. The closure of T_p with respect to T_p' can be checked by using the *Growth Node extension check for closure lemma*. On the contrary, to check the closure for $T_p = \langle A^1 C^2-1-1 \rangle$ and prefix-tree $T_p' = \langle A^1 B^2 C^3-1-1-1 \rangle$, it can be noted that T_p' is generated from the prefix-tree $\langle A^1 B^2-1-1 \rangle$. Hence T_p' occurs in a different projected dataset from $T_p = \langle A^1 C^2-1-1 \rangle$. As the ancestor of node 'C' in $T_p = \langle A^1 C^2-1-1 \rangle$ has been extended, to check the closure we need to apply the *ancestor node extension check for closure lemma*.

Lemma 2: Growth Node extension check for closure

If a prefix-tree T_p can be extended to T_p' in the same prefix-projected dataset using its *GNs* and the prefix-projected instance has the same support as that of T_p , then T_p is not closed.

The growth node extension for closure checking is not a computationally expensive step as it involves checking only the support of the *GN* in the projected dataset. This technique can be used to reduce the number of frequent embedded subtrees to generate *CFE* subtrees. From our running example, we can note that for the $\langle A^1-1 \rangle$ prefix-projection dataset, the *GN* 'E' has the same support of 3 as that of its prefix-tree and hence there is no need to store the prefix-tree $\langle A^1-1 \rangle$.

In order to check for the extension of T_p in a different prefix-projected dataset from T_p , the closure checking is applied using the following lemma:

Lemma 3: Ancestor Node extension check for closure

If there exists a prefix-tree T_p with m nodes and a prefix-tree T_p' with the common m nodes and an additional node b having the same support as that of T_p in a different prefix-projected instance from T_p then T_p is not closed and b is the ancestor node extension of T_p .

In order to efficiently check for closure for ancestor node extensions, a technique called *maintain-and-test* is deployed to check for closure. A rudimentary approach to check for closure is to check for all the ancestor node extension events having the same support. However, it is an expensive operation. To reduce the number of checks, a parameter, which is the sum of the tree ids, is included to check for closure. To apply this technique, we first maintain the previously generated prefix-trees and then test the recently generated prefix-tree T_p for closure. If there exists an ancestor node extension of a node b for the prefix-tree T_p among the maintained prefix-trees T_p' that has the same support and sum of tree ids as T_p , then this node is checked for closure. The *maintain-and-test* approach reduces the number of checks as it avoids checking all the prefix-trees with the same support, hence reducing the computational overhead.

From our running example, we can see that if we apply the *maintain-and-test* technique then there will be 15 candidates for closure checking. However, when we apply the search space reduction using a backward scan and the growth node extension check for closure, we could reduce the search space to a set of 6 prefix-trees. They are $\{(\langle A^1 B^2 C^3-1-1-1 \rangle:2), (\langle A^1 C^2-1 E^3-1-1 \rangle:2), (\langle A^1 E^2-1-1 \rangle:3), (\langle A^1 E^2 F^3-1-1-1 \rangle:2), (\langle A^1 F^2-1-1 \rangle:2), (\langle A^1 B^2 C^3-1-1 E^4-1-1 \rangle:2)\}$, among which there are only three for which testing is required for prefix-trees having the same support and sum of tree-id.

Figures 4(a) and (b) outline the CFETreeMiner in Phase-I of XCFS and the *Fre* subroutine to mine for *CFE* subtrees respectively.

Algorithm CFETreeMiner

Input: Document Tree Dataset: *DT*, Minimum Support: *min_supp*.
Output: All *CFE* subtrees, *CFE*.
Begin

1. Scan *DT* and find all 1-length frequent subtrees $f = \{f_1, f_2, \dots, f_n\}$
2. for the node *b* in each of the frequent subtree f_i in *f*
 - a. If there exists the same predecessor node *c* for f_i in all the document trees and *c* is a node in $f_j \in f$ then
 - i. Do not construct the prefix-tree projected dataset for f_i .
 - b. else,
 - i. Find all occurrences of f_i in *DT*, and construct $\langle b-1 \rangle$ -projected dataset (i.e. *ProDB*(*DT*, $\langle b-1 \rangle$)) through collecting all corresponding Project-Instances in *DT*.
 - ii. Call **Fre** ($\langle f_i - 1 \rangle$, 1, *ProDB*(*DT*, $\langle f_i - 1 \rangle$), *min_supp*, *supp*(f_i)) to mine the projected dataset and obtain *CFE* subtrees until no more *GN*s that can be found.

End

Figure 4(a). CFETreeMiner

Function Fre (T_p , *n*, *ProDB*(*DT*, T_p), *min_supp*, *prepat_supp*)

Input: Prefix-tree: T_p , Length of T_p : *n*, $\langle T_p \rangle$ -projected dataset : *ProDB*(*DT*, T_p), *min_supp*, Support of T_p : *prepat_supp*
Output: *CFE* subtrees

Method:

1. Scan *ProDS*(*DT*, T_p) once to find all the 1-length frequent *GN*s, $\{GN_{0, \dots, k}\}$ according to Lemma 1.
2. Set output=true.
3. Count the support of all *GN*s.
4. If $\text{supp}(GN_0 || GN_1, \dots || GN_k) == \text{supp}(T_p)$ then the subtree is not a *CFE* subtree, output = false.
5. For each GN_i in *GN*
 - a. if GN_i is frequent then
 - i. Extend T_p with GN_i to form the prefix-tree T_p' .
 - ii. if (output) then Insert T_p' into *CFE*.
 - b. else
 - i. Check T_p' for occurrence of any of its subtree with the same support and sum of tree ids in the output *CFE*. If there exists any subtree for T_p' then remove the subtree of T_p' and insert T_p' into *CFE*.
6. Find all occurrences of GE_i in *ProDB*(*DT*, T_p), construct the $\langle T_p \rangle$ -projected database (i.e. *ProDB*(*DT*, T_p')) through collecting all corresponding Project-Instances in *ProDB*(*DT*, T_p).
7. Call **Fre** (T_p' , *n* + 1, *ProDB*(*DT*, T_p'), *min_supp*, *prepat_supp*)

Figure 4(b). Subroutine Fre

CFETreeMiner in Phase-I of XCFS begins with the scanning of *DT* to identify all 1-Length frequent subtrees $f = \{f_1, f_2, \dots, f_n\}$. The backward scan property is used to check the ancestor node for the node in each subtree in *f* for all trees in *DT*. If the node in f_k is the ancestor node of the node in f_i in all the document trees then f_i is pruned, since f_k includes the *CFE* subtrees generated from f_i . Otherwise, the subroutine *Fre* outlined in Figure 4(b) recursively identifies all the occurrences of f_i in *DT* to construct $\langle f_i - 1 \rangle$ prefix-tree projected dataset by collecting all the corresponding

prefix-tree projected instances. The subroutine *Fre* applies the closure checks for the *GN* extension and ancestor-node extension against the corresponding projected dataset. This subroutine is recursively called until there are no more frequent *GN*s to form the prefix-tree projected dataset.

5. PHASE-II OF XCFS: CLUSTERING WITH THE ICF

The next task is to combine the structural commonalities that are identified in the form of subtrees with the content features of XML documents in order to utilize both these features in clustering the XML documents. We propose to combine the structural and content features of XML documents non-linearly in VSM representation. The set of *CFE* subtrees obtained using Phase-I of XCFS are now utilized to constrain the content of XML documents. The process begins by using the *CFE* subtrees to identify the coverage of the document trees. The concept of coverage of the document trees for a given *CFE* is defined below:

Definition 4: A document tree DT_i is said to be covered by a *CFE* subtree, $CFE_j \in CFE$ if DT_i preserves the same parent-child relationship as that of CFE_j where *CFE* is the set of *CFE* subtrees representing the entire collection of documents. The coverage (δ) of a document tree DT_i is the set of *CFE* subtrees, $\{CFE_1, \dots, CFE_k\} \in CFE$, that covers the DT_i . The coverage of a document tree DT_i , δ equals to $\{CFE_1, \dots, CFE_k\}$ if DT_i preserves the same ancestor-descendant relationship among its nodes as that of $\{CFE_1, \dots, CFE_k\}$.

Now we apply closure property on every document tree DT_i in *DT* to identify and remove the overlapping *CFE* subtrees in the coverage(δ) of a DT_i . The overlapping *CFE* subtrees for a given DT_i are identified by checking whether there exist any *CFE* subtrees which have an embedded subtree relationship with other *CFE* subtrees in the coverage. Only *CFE* subtrees that have no *CFE* super subtrees are retained. This process further reduces the information overload resulting from the overlapping subtrees.

The next step is to represent the document content according to its coverage. As defined in section 3, the structure-constrained content contained within *CFE* subtrees according to the coverage of a DT_i is retrieved from the XML document D_i . The sum of the occurrences of the terms in the structure-constrained content features is computed according to the coverage of the document. Given the coverage of the document DT_i , $\delta(DT_i) = \{CFE_1, \dots, CFE_k\}$ the structure-constrained content of $\delta(DT_i)$ is a collection of the node values corresponding to $\delta(DT_i)$ given by $C(D_i, \delta(DT_i)) = \{C(D_i, CFE_1), \dots, C(D_i, CFE_k)\}$. The structure-constrained content of CFE_i in D_i is a collection of *m* node values (*N*), $C(D_i, CFE_i) = \{C(N_1), \dots, C(N_m)\}$. For a given term t_i in $C(D_i, CFE_i) = \{t_1, \dots, t_k\}$, the sum of the occurrence of the term t_i in all *CFE*s of $\delta(DT_i)$ is computed using

$$\zeta(t_i) = \sum_{i=1}^{k'} t_i(CFE_k) \quad (1)$$

The resulting vector containing the sum of occurrences of all the terms for $\delta(DT_i)$, called the *Pre-Cluster Form (PCF)*, is generated for each document D_i in the collection *D*. All the PCFs are combined in a matrix representation called the *Intermediate Cluster Form (ICF)*. *ICF* is a matrix of the form *D* X *T* where *D* represents the documents, *T* represents the terms that are present

in $\delta (DT_i)$, and the value of the matrix cell is represented by ζ , the sum of the occurrences of a term in the *PCF*.

The repeated bisection partitioning clustering method [7] is applied on the *ICF* to generate the required number of clusters. This method divides the *ICF* into two groups and then selects one of the groups according to a clustering criterion function and bisects further. This process is repeated until the desired number of clusters is achieved. The similarity between each pair of *PCFs* in *ICF* is computed. Let there be two *PCFs* d_x and d_y in the given *ICF* matrix for two documents D_x and D_y respectively. The similarity between d_x and d_y is computed using the cosine similarity function,

$$\cos \theta = \frac{d_x \cdot d_y}{|d_x| |d_y|} \quad (2)$$

6. EXPERIMENTS AND DISCUSSIONS

Experiments are conducted to evaluate the accuracy and scalability performance of XCFS on one synthetic dataset, as well as two real-life datasets.

6.1 Datasets

We have used both the synthetic dataset and two large-size real-life datasets for experiments. The synthetic dataset, TIM, is generated for the evaluation of the CFETreeMiner in Phase-I of XCFS on various factors such as branching factor, scalability, total number of patterns generated and the response time. The TIM dataset is generated using the commonly used Zaki's tree generator [17] for parameters $f=10$, $d=10$, $n=100$, $m=10000$, $t=1000000$ where "f" represents the fan-out factor, "d" the depth of the tree, "n" the number of unique labels for the trees, "m" the total number of nodes in a parent tree and "t" the number of trees. This dataset was chosen to measure the performance of the frequent mining algorithms on a very large number of trees which are shallow and have a lower branching factor. Unfortunately, this dataset cannot be utilized for clustering using structure and content information due to the absence of content information.

Two real-life XML datasets, ACM SIGMOD and INEX 2007 Wikipedia [4], were used after a careful analysis of a number of datasets. These datasets were chosen in order to understand how XCFS and the existing approaches perform on XML datasets which have extreme characteristics, as shown in Table 2.

Table 2. Details of the real-life datasets

Dataset Names Attributes	ACM SIGMOD Dataset	INEX 2007 Wikipedia Dataset
No. of Docs	140	48305
No. of internal nodes	4907	4487819
Max Depth of a document	6	48
No. of distinct terms	7135	535351
Total No. of words	38141	16,682,466
Size of the collection	≈ 1 MB	≈ 360 MB
Presence of formatting tags	No	Yes
Presence of Schema	Yes	No

The ACM SIGMOD dataset 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 [1]. In contrast, the INEX 2007 Wikipedia dataset contains a very large number of documents with deeper structure and a high branching factor. It does not contain any schema definitions such as XSD or DTD. Also, this dataset contains both semantic tags and formatting tags. To our knowledge, most of the existing XML clustering algorithms have been tested on datasets with a maximum of few hundred small-sized documents.

6.2 Experimental Design

Experiments were conducted on 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 response time, the number of frequent subtrees generated, the accuracy of the clustering results and the scalability of XCFS over other clustering techniques and representation. Previous researchers [1] have used the ACM SIGMOD dataset to cluster the documents into two groups according to their structural similarity. To compare XCFS with theirs, we repeated experiments with two cluster classes. Since the documents come with two different schema definitions in this dataset, it is rather easier and straightforward to group this dataset according to structural similarity. In order to add complexity, we have designed a second set of experiments by including the clustering classes according to both the structure and content-based similarity in documents. This experiment design utilizes expert knowledge and is based on 5 groups that consider both structural and content features of XML documents. The first category is based on the document structure and the remaining four categories are based on the document content, namely General, Mobile computing, Database Management Systems (DBMS) and Others. We have made this dataset available.¹

The INEX 2007 document mining track used the same Wikipedia dataset [4] to perform the clustering task. CRP and 4RP [16] approaches, which includes both the structural and the content information of the dataset, performed best in the clustering task. We compare the results of CRP and 4RP [16] approach with those of XCFS. We also compare the results of XCFS against the structure-only results using the Self-Organizing Map (SOM) [6] on this dataset. Moreover, since the main focus of the proposed method is to combine the structural and the content features of XML documents for effective clustering, clustering performance is also evaluated by considering (1) only the structural features (SO) and (2) only the content (CO) features. The same partitioning clustering method is applied to the resulting SO and CO input representation matrices for finding the clusters.

Structure Only (SO) Representation. The coverage (δ) of a document tree DT_i is used to generate this representation. An input matrix $D \times CFE$ is generated where D represents the documents and CFE represents the list of closed embedded frequent document subtrees. Each document is represented by the CFE subtrees that cover it.

¹ <http://sky.fit.qut.edu.au/~nayak/datasets>

Content Only (CO) Representation. The content of XML documents is represented in a matrix form denoted by $D \times C$ where D represents the documents and C represents the list of terms in the dataset. This term set is obtained after pre-processing using techniques such as stop-word removal, stemming and integer removal. Each cell in the matrix contains the $tf \times idf$ weighting of the C terms in D where tf denotes the term frequency and idf is the inverse document frequency.

6.3 Evaluation Measures

Two standard measures, Micro F1 (intra-cluster purity) and Macro F1 (inter-cluster purity), are used to evaluate the accuracy of the clustering solutions. These two measures are based on F1-score which is defined as

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \quad (3)$$

$$Precision = \frac{TP}{TP + FP} \quad (4)$$

$$Recall = \frac{TP}{TP + FN} \quad (5)$$

where TP denotes True Positives, FP denotes False Positives (FP), and FN denotes False Negatives. Micro-average F1 (Micro-avg F1) is calculated by summing up the TP, FP and FN values from all the categories individually. Macro-average F1 (Macro-avg F1), on the other hand, is derived from averaging the F1 values over all the categories. The best clustering solution for an input data set is the one where micro- and macro-average F1 measures are close to 1.

6.4 Empirical Analysis

A comparison of the CFETreeMiner in Phase-I of XCFS against PrefixTreeESpan and TreeMinerV[17] on T1M, Wikipedia and ACM SIGMOD dataset is presented in Figures 5, 6 and 7 respectively. Figures 5(a) and 6(a) show that there is a significant improvement in the response time of CFETreeMiner over other algorithms. A comparison on the T1M dataset shows that the algorithm is scalable to a very large number of shallow document trees and reports a considerable improvement over other methods.

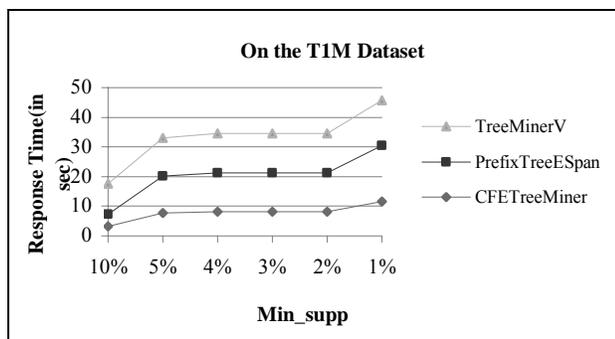


Figure 5(a). Comparison of algorithms on the response time on the T1M dataset

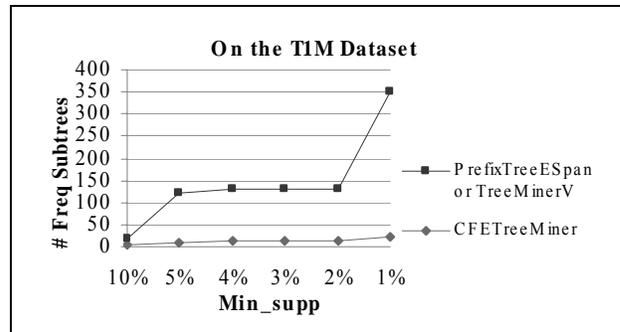


Figure 5(b). Comparison of algorithms on the number of frequent subtrees generated on the T1M dataset

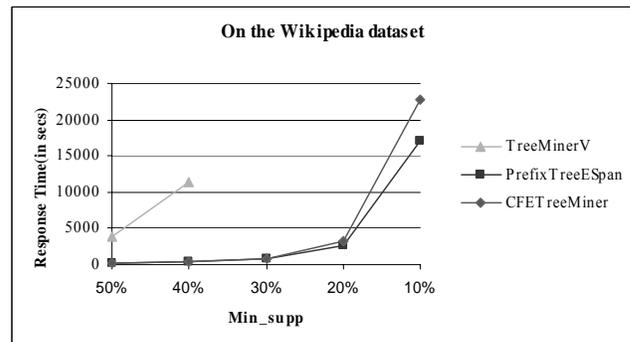


Figure 6(a). Comparison of algorithms on the response time on the Wikipedia dataset

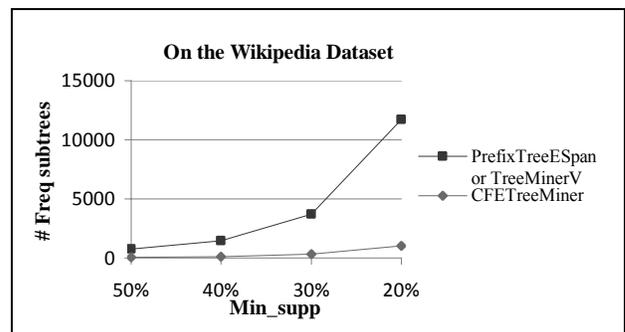


Figure 6(b). Comparison of algorithms on the number of frequent subtrees generated on the Wikipedia dataset

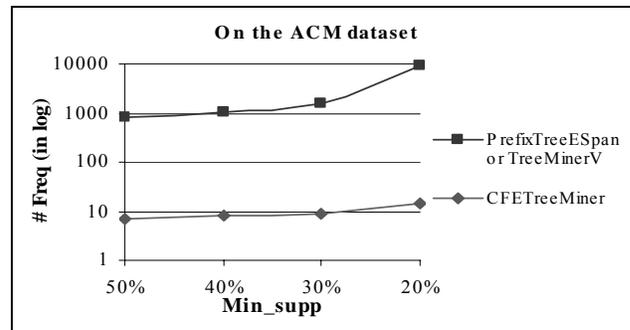


Figure 7. Comparison of algorithms on the number of frequent subtrees generated on the ACM SIGMOD dataset

On the Wikipedia dataset, Phase-I of XCFS reports a prominent improvement in response time especially under lower support thresholds when there are a very large number of frequent subtrees generated. Also, it should be noted that from Figure 6(a) that the TreeMinerV fails to respond to lower or equal to 30% support threshold. The results clearly indicate that incorporating the closure property within the frequent mining algorithm instead in post-processing improves the efficiency of the algorithm as it reduces the search space.

On the other hand, on the very small ACM SIGMOD dataset, the CFETreeMiner in Phase-I of XCFS, PrefixTreeESpan and TreeMinerV could generate frequent subtrees in less than 7 seconds even for the 20% support threshold, XCFS results in a very significant reduction in the number of *CFE* subtrees generated as shown in Figure 7. Results shown in Figures 5, 6 and 7 clearly indicate that incorporating the closure property within the frequent mining algorithm rather than in post-processing improves the efficiency of the algorithm as it reduces the search space.

Now we evaluate Phase-II of XCFS using clustering accuracy and reduction in the size of input matrix representation (the number of unique terms and the total number of terms). Table 3 summarizes the performance of XCFS on the ACM SIGMOD dataset using the structure features based clustering and structure-and-content features-based clustering designs. It also includes the clustering results of XCFS on the Wikipedia dataset collection and other clustering approaches submitted for the INEX 2007 document mining challenge.

Table 3. Performance of XCFS and other methods on the ACM SIGMOD dataset and the Wikipedia dataset

Dataset	#Clusters	Method	Micro-avg F1	Macro-avg F1
ACM SIGMOD	2	XCFS	1	1
		SO	1	1
	5	XCFS	0.82	0.49
		SO	0.74	0.42
WIKIPEDIA DATASET	21	XCFS	0.58	0.64
		CRP [16]	0.44	0.49
		4RP [16]	0.42	0.49
		SO	0.28	0.31
		SOM [6]	0.27	0.27
		LSK [14]	0.37	0.40

Results reveal that XCFS demonstrates an improvement in micro and macro F1 values over other clustering methods on both real-life datasets. It can be noted for the ACM SIGMOD dataset that both the XCFS and SO methods achieve Micro-average and Macro-average F1 scores of 1 when considering classes according to structural similarity, which is similar to the results reported in a previous study [1]. However, the method proposed in [1] can distinguish only the structural variations in documents and may fail in situations where there is a grouping based on the structure and the content. XCFS is able to achieve higher accuracy in terms of recognizing classes based on both the structural and the content

features in comparison to considering these features independently as shown by the results of SO and CO methods due to its novel way of combining these two features efficiently.

On the other hand, it was infeasible to conduct clustering using CO methods on Wikipedia dataset due to the very high dimensionality of the dataset. The significant reduction in the number of unique terms by about 65% for the Wikipedia dataset in XCFS has facilitated it to be used for clustering this large dataset. It can be noted from Table 4 that even for a small dataset such as ACM SIGMOD there is about 25% reduction in the number of unique terms and 30% on the total number of Non-zeroes.

Table 4. Dimensionality of the input matrix for the ACM SIGMOD Dataset and the Wikipedia dataset

DATASETS	Method	# Unique terms	Total # of Non-Zeroes
ACM SIGMOD	XCFS	3971	8855
	CO	5334	12741
WIKIPEDIA	XCFS	189,681	4,226,178
	CO	535,351	6,674,842

The CO method represents the content features of XML documents by capturing only the text terms and ignoring their hierarchical relationship. The SO method represents the structural features of XML documents by capturing only the hierarchical relationship amongst them and ignoring their content information. In contrast, the XCFS method utilizes the *CFE* subtrees to represent the content and preserves the structural relationships among the content. We also conducted experiments to compare the effect of utilising closed and Non-Closed Frequent Embedded(*Non-CFE*) subtrees in clustering and the dimensionality reduction. The frequent embedded subtrees are extracted using PrefixTreeESpan and the clustering results are compared with the proposed XCFS method on ACM SIGMOD dataset using structure-and-content based clustering design (with 5 clusters) as shown in Table 5.

Table 5. Comparison of XCFS for the closed and non-closed frequent subtrees on ACM SIGMOD dataset

Methods	Micro F1	Macro F1	Total # Of Non-Zeroes	Total # Of Terms
<i>CFE</i>	0.82	0.49	8855	11070
<i>Non-CFE</i>	0.69	0.40	9076	4,646,912

XCFS using *CFE* subtrees reports a significant drop in the total number of terms as it uses only 7 *CFE* subtrees to extract the content in comparison to the 805 *Non-CFE* subtrees at the same minimum support threshold. By utilizing the content corresponding to *CFE* subtrees only, dimensionality of the dataset is significantly reduced without any information loss. Due to having less or no redundant content information, accuracy of clustering solution is improved. It should be noted that the classes in the Wikipedia dataset are based on thematic categories and previous researchers [4,16] have shown that the content-only method performs best. It is remarkable to see the accuracy improvement by using frequent embedded subtrees to capture the structural similarity and then using the structure-constrained content for finding clusters.

6.5 Scalability Analysis

Scalability is an important issue in evaluating XML clustering algorithms as they fail to scale when both the features are considered due to the high dimensionality as reported in previous research [15]. Hence we conducted a scalability analysis by recording the response time of XCFS for the varying size of the Wikipedia dataset at a support threshold of 10% as shown in Figure 8. The response time is the time taken to mine for *CFE* and using them for clustering. The analysis shows a linear scalability of XCFS as the frequent mining and clustering algorithms have a linear scalability.

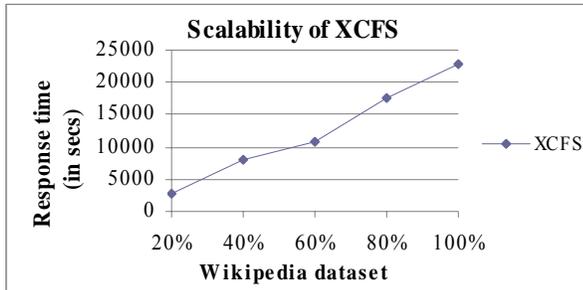


Figure 8. Scalability analysis of XCFS on Wikipedia dataset

Although XCFS includes extra processing steps of mining for frequent subtrees and utilizing these frequent subtrees to represent the content, it should be noted that there is an improvement in accuracy in both the datasets with varied characteristics. The extraneous data in the CO representation makes the grouping of XML documents less accurate and adds redundancy in VSM representation. On the other hand, in XCFS, the structural similarity for the documents is extracted and implicitly represented while modeling the VSM. Hence, including only the term vector containing good quality features in the XCFS method has higher accuracy than including all the feature vectors.

7. CONCLUSION AND FUTURE DIRECTIONS

In this paper, we have proposed a frequent mining algorithm and a clustering method for effectively combining both the structural and the content features in XML documents. By utilizing the frequent subtrees in clustering the content, we developed a scalable clustering method that can efficiently work for very large datasets. The experimental results clearly ascertain that XCFS outperforms other existing approaches in improving accuracy, while using a significantly reduced number of content features. Our future work will focus on extending this approach to representing the frequent mining results in a bitcube form and then clustering them.

8. REFERENCES

- [1] Aggarwal, C. C., N. Ta, et al. 2007. Xproj: a framework for projected structural clustering of xml documents. Proceedings of the 13th ACM SIGKDD International Conference on Knowledge discovery and data mining (San Jose, USA). 46-55.
- [2] Chi, Y., S. Nijssen, et al. 2005. Frequent Subtree Mining- An Overview. *Fundamenta Informaticae*. 66(1): 161-198.
- [3] Denoyer, L. and P. Gallinari. 2007. Report on the XML mining track at INEX 2005 and INEX 2006: categorization and clustering of XML documents. *ACM SIGIR Forum*. 41(1): 79-90.
- [4] Denoyer, L. and P. Gallinari. 2008. Report on the XML mining track at INEX 2007 categorization and clustering of XML documents. *ACM SIGIR Forum*. 42(1): 22-28.
- [5] Erik, W. and J.G. Robert. 2008. XML Fever. *Commun. ACM* 51(7): 40-46.
- [6] Hagenbuchner, M., A. C. Tsoi, et al. 2008. Efficient clustering of structured documents using Graph Self-Organizing Maps. *Focused Access to XML Documents*. 4862/2008: 207-221.
- [7] Karypis, G. 2002. CLUTO - A Clustering Toolkit. Technical Report. University of Minnesota
- [8] Kutty, S., R. Nayak, et al. 2007. Clustering XML documents using closed frequent subtrees - A Structural Similarity Approach. *Focused Access to XML Documents*, 4862/2008: 183-194.
- [9] Kutty, S., R. Nayak, et al. 2007. PCITMiner - Prefix-based Closed Induced Tree Miner for finding closed induced frequent subtrees. *Proceedings of the Sixth Australasian Data Mining Conference (Gold Coast, Australia)*. 151-160.
- [10] Leung, H.-p., F.-l. Chung, et al. 2005. XML Document Clustering Using Common XPath. *Proceedings of the International Workshop on Challenges in Web Information Retrieval and Integration (Tokyo, Japan)*. 91-96.
- [11] Salton, G. and M. J. McGill. 1986. *Introduction to Modern Information Retrieval*. New York, NY, USA, McGraw-Hill, Inc.
- [12] Shen, Y. and B. Wang. 2003. Clustering Schemaless XML Documents. *On The Move to Meaningful Internet Systems 2003*: 767-784.
- [13] Tan, H., T. Dillon, et al. 2005. X3-Miner: Mining Patterns from XML Database. *Proceedings of the 6th International Conference on Data Mining, Text Mining and their Business Applications (Skiathos, Greece)*. 287-298.
- [14] Tran, T. and R. Nayak. 2008. Document Clustering using Incremental and Pairwise Approaches. *Focused Access to XML Documents*. 4862/2008: 222-232.
- [15] Vercoustre, A.-M., M. Fegas, et al. 2006. A Flexible Structured-Based Representation for XML Document Mining. *Advances in XML Information Retrieval and Evaluation*. 3977/2006:443-457.
- [16] Yao, J. and N. Zarida. 2007. Rare Patterns to improve Path-based Clustering of Wikipedia articles. *Pre-proceedings of the Sixth Workshop of Initiative for the Evaluation of XML Retrieval (Dagstuhl, Germany)*. 224-231.
- [17] Zaki, M. J. 2002. Efficiently mining frequent trees in a forest. *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining (Edmonton, Canada)*. 71-80.
- [18] Zou, L., Y. Lu, et al. 2006. PrefixTreeESpan: A Pattern Growth Algorithm for Mining Embedded Subtrees. *Proceedings of the Web Information Systems (Wuhan, China)*. 499-505.