XProj: A Framework for Projected Structural Clustering of XML Documents

Charu C. Aggarwal
IBM T. J. Watson Research Center
19 Skyline Drive
Hawthorne, NY 10532
charu@us.ibm.com

Na Ta
Tsinghua University
Beijing, China
dan04@tsinghua.edu.cn

Jianyong Wang*
Tsinghua University
Beijing, China
jianyong@tsinghua.edu.cn

Jianhua Feng
Tsinghua University
Beijing, China
fengjh@tsinghua.edu.cn

Mohammed Zaki
Rensselear Polytechnic Institute
Troy, NY
zaki@cs.rpi.edu

ABSTRACT

XML has become a popular method of data representation both on the web and in databases in recent years. One of the reasons for the popularity of XML has been its ability to encode structural information about data records. However, this structural characteristic of data sets also makes it a challenging problem for a variety of data mining problems. One such problem is that of clustering, in which the structural aspects of the data result in a high implicit dimensionality of the data representation. As a result, it becomes more difficult to cluster the data in a meaningful way. In this paper, we propose an effective clustering algorithm for XML data which uses substructures of the documents in order to gain insights about the important underlying structures. We propose new ways of using multiple sub-structural information in XML documents to evaluate the quality of intermediate cluster solutions, and guide the algorithms to a final solution which reflects the true structural behavior in individual partitions. We test the algorithm on a variety of real and synthetic data sets.

*The research of the authors from Tsinghua University was partly supported by the Scientific Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry of China, National Basic Research Program of China under grant number 2006CB303103, Basic Research Foundation of Tsinghua National Laboratory for Information Science and Technology (TNList), Program for Selected Talents (i.e., "Gu Gan Ren Cai") at Tsinghua University, and National Natural Science Foundation of China under Grant No. 60573061 and 60573094.

KDD'07, August 12–15, 2007, San Jose, California, USA.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data Mining*

General Terms

Algorithms

Keywords

XML, clustering

1. INTRODUCTION

The clustering problem is defined as follows. For a database $\mathcal{D}$ of records, we would like to segment the data into groups of objects which are similar to one another. The similarity between the objects is defined with respect to some user-defined objective function. The clustering problem is widely known in the literature because of its use in a large number of applications. A number of interesting methods for clustering are discussed in [7, 11, 17]. The wide use of the web and the flexibility of the XML representation has popularized the use of XML documents. The semi-structured nature of XML document definitions allows the modelling of a wide variety of databases as XML documents. The textual nature of XML documents allows the use of standard Information Retrieval methods on XML representations. Other alternatives include the use of a flattened multi-dimensional representation of the data in order to perform the clustering. However, these methods ignore the structural information in the data, which often turns out to be crucial for the mining process [3, 15, 16].

The problem of structural clustering of XML documents is a challenging task because most known clustering algorithms cannot be generalized easily to take into account the structural behavior of XML documents. There are several reasons for this: (1) Many clustering algorithms typically require the computation of similarity between different objects as a subroutine. The problem of computing similarity between XML documents has itself been known to be a difficult research problem. (2) Even if the computation of

similarity can be performed effectively, the use of such functions is often not likely to lead to meaningful clusters. This is because much of the clustering information is encoded in substructures of the XML documents. This is somewhat analogous to the problem of projected clustering [1, 2] in which useful clustering information is encoded in subsets of dimensions. In the case of structural data, the problem is even more acute because a very high implicit dimensionality is encoded in the structural behavior of the documents. (3) Many clustering algorithms require the use of intermediate clustering representatives to perform the clustering task effectively. The structural representation of XML documents makes this task much more difficult.

This paper presents a structural method for clustering XML data. In this case, the similarity within a cluster is defined in terms of the containment of particular frequent substructures which occur frequently in that particular segment of the data. Thus, similarity is quantified in terms of *containment* of frequent substructures within a particular segment. These substructures are those analogous to the projected dimensions used in subspace clustering [1, 2]. As we shall see, such a substructure containment definition helps us define clusters in a very robust way, since the clusters are defined not by individual substructures, but by *sets of substructures*. We further note that while our methods have been developed for the case of XML data, they are applicable to a variety of data domains such as biological data which use structural information in the representation.

This paper is organized as follows. In Section 2, we will discuss the related work to XML data clustering. In Section 3, we present the basic algorithm for clustering of XML documents. Section 4 discusses some algorithm implementation issues and optimization techniques, mainly focusing on efficient approaches for mining the frequent substructures of the XML documents. This approach concentrates on a novel use of a sequential pattern representation of a traversal of the tree structure. The empirical results are presented in section 5. Section 6 contains the conclusions and summary.

2. RELATED WORK

One of the earliest work on clustering tree structured data is the XClust algorithm [8], which was designed to cluster XML schemas in order for efficient integration of large numbers of Document Type Definitions (DTDs) of XML sources. It adopts the agglomerative hierarchical clustering method which starts with clusters of single DTDs and gradually merges the two most similar clusters into one larger cluster. The similarity between two DTDs is based on their element similarity, which can be computed according to the semantics, structure, and context information of the elements in the corresponding DTDs.

One of the shortcomings of the XClust algorithm is that it does not make full use of the structure information of the DTDs, which is quite important in the context of clustering tree-like structures. Two recent approaches of clustering tree structured data are also based on the hierarchical clustering method [10, 5]. S-GRACE is a ROCK-like [6] hierarchical clustering algorithm [10]. In [10], an XML document is converted to a structure graph (or s-graph), and the distance between two XML documents is defined according to the number of the common element-subelement relationships, which can capture better structural similarity relationships than the tree edit distance in some cases [10]. In [5], the XML documents are modeled as rooted ordered labelled trees, and a framework for clustering XML documents by using structural summaries of trees is presented. The aim is to improve algorithmic efficiency without compromising cluster quality.

In contrast to the previous work, our XProj algorithm employs a projection based structural approach and uses a set of frequent substructures as the representative with respect to an intermediate cluster of XML documents. As a frequent substructure preserves more structural information than the simple element-subelement relationships [10], the use of multiple frequent substructures as the representative makes structural similarity (and self-similarity) proposed for XProj algorithm very robust and accurate. Furthermore, multiple substructural portions of the document collection are used to in order to associate a *representative set* with a given cluster, rather than a single representative. To make the structural similarity more comparable among different sets of representatives and the similarity calculation more efficient, the representatives only include the frequent substructures of size l. The idea behind this design is analogous to the projection-based subspace clustering [1, 2].

To speed up the frequent substructure representative mining, XProj adopts a set of high quality approximate structures, that is, sequences of tree edges. The selection of representative substructures is based on the sequential covering paradigm, which can be leveraged for rule based classification. In addition, recent advances in frequent sequence mining [9, 12, 14] can be leveraged in order to perform approximate substructure mining in an efficient way. In this paper, we will utilize these methodologies in order to improve the efficiency of the process.

3. THE XPROJ ALGORITHM

We model XML documents as ordered, labelled, rooted trees. We do not distinguish between attributes and elements of an XML document, since both are mapped to the label set. The basic XML clustering algorithm is implemented as a partition based algorithm which tries to construct partitions that maximize the structural commonalities among the documents within a partition. The construction of clusters of XML documents presents a number of unique challenges which are not encountered for the case of other kinds of multi-dimensional data sets. This is because most clustering algorithms require us to construct the following measures on multi-dimensional records:

- A measure of similarity among the documents of different clusters needs to be implemented. This is difficult to achieve in a complex structural environment which has an inherently high implicit dimensionality. In such cases, similarity needs to be measured in terms of the substructures in a group of documents.
- In many cases, clustering algorithms need to construct representatives of groups of documents. While this is easy to achieve in a multi-dimensional environment (by simple averaging), it is not quite as easy to do so robustly in the case of structural documents.
- A method needs to be designed for measuring the similarity of a set of structural documents within a given group. This is related to our earlier discussion on measuring similarity among different documents.

In order to achieve these goals, we design an XML clustering algorithm which works with *frequent substructures* of the underlying documents. These frequent substructures are utilized to measure the similarity between particular groups of documents. Thus, a group of documents is defined to be most similar, when it results in a large number of similar substructures at a specified support level. We note that the use of frequent substructures in order to define similarity among documents is analogous to the concept of projected clustering in multi-dimensional data. As in the case of projected clustering [1, 2], we are using (structural) projections of the space in order to define similarity.

In the projected XML clustering algorithm, instead of using individual XML documents as representatives for partitions, we use *sets* of substructures of the documents as possible representatives. A set $\mathcal{S}$ of substructures is said to be a representative of a given collection, if each structure in it appears as a frequent substructure in that collection. In general, we would like to construct clusters of documents which are similar enough so that the underlying frequent structures *cover* a significant fraction of the tree nodes in the collection. In order to understand this way of defining similarity, let us define the concept of coverage of XML documents. First we define the concept of substructures.

DEFINITION 1. *A substructure T, of a rooted, ordered, labelled tree, T', is an undirected, connected, labelled, acyclic graph, whose vertices and edges can be one-to-one mapped to a subset of vertices and edges of T' that preserves the vertex labels and ancestor-descendant relationships among the corresponding vertices.*

DEFINITION 2. *Let T be a substructure of the document R. A substructural alignment of T to R is defined to be a correspondence from each node in T to a node in R, which define the substructural relationship of T to R.*

We note that there can be more than one possible substructural alignment of T to the document R. Also, we note that even though a node in T corresponds to each node in R, the reverse may not be true.

DEFINITION 3. *A structure T is defined to be frequent in collection of XML documents $\mathcal{R}$ at a user defined minimum support, min_sup, if it occurs as a substructure of at least min_sup fraction of the documents in the collection $\mathcal{R}$.*

DEFINITION 4. *A node x in the document R is said to be* uncovered *by structure T, if a substructural alignment from T to R cannot be found so that node x aligns with some node in T.*

This definition can now be generalized to a set of structures as opposed to a single structure. This generalization is defined as follows.

DEFINITION 5. *A node x in the document R is said to be* uncovered *by the set of structures $\mathcal{T} = \{T_1 \ldots T_k\}$, if a substructural alignment from any structure T_i to R cannot be found such that node x aligns with some node in T_i.*

Clearly, if a node in a document remains uncovered by the frequent structures in a collection, this is not very good from a clustering point of view. This provides us a natural way to define the similarity of the documents in a collection to a set of structures. The similarity of a document to a set of structures in a collection is defined as the fraction of nodes in the document which are covered by any structure in the collection.

DEFINITION 6. *The structural similarity $\delta(R, \mathcal{T})$ of a document R to a structural collection $\mathcal{T} = \{T_1 \ldots T_k\}$ is defined to be the fraction of nodes in R which are covered by some structure in T.*

We note that we use the *fraction* of nodes which are covered by the structural collection, since it normalizes for the total number of nodes in the document. We can easily generalize this definition to similarity between *sets* of documents and sets of structures by averaging the structural similarity over the different documents. Therefore, we have:

DEFINITION 7. *The structural similarity $\Delta(\mathcal{R}, \mathcal{T})$ of the set of documents $\mathcal{R} = \{R_1 \ldots R_j\}$ to the set of frequent structures $\mathcal{T} = \{T_1 \ldots T_k\}$ is defined as the average structural similarity over the different documents in $\mathcal{R}$ to $\mathcal{T}$. Therefore, we have:*

$$\Delta(\mathcal{R}, \mathcal{T}) = \sum_{i=1}^{j} \delta(R_i, \mathcal{T})/j \tag{1}$$

This lays the ground for us to define the *frequent sub-structural self-similarity* of a document collection.

DEFINITION 8. *For a given level of user defined minimum support denoted by min_sup, the frequent sub-structural self-similarity of a document collection $\mathcal{R}$ at level l is defined as the structural similarity $\Delta(\mathcal{R}, \mathcal{F}_l)$, where $\mathcal{F}_l$ are the set of frequent substructures of $\mathcal{R}$ with l nodes.*

We note that the frequent sub-structural self-similarity of a document collection provides a good understanding of the level of homogeneity in the document collection from a sub-structural point of view. When a collection contains noisy and random documents, it will not be possible to mine frequent structures which cover a significant fraction of nodes in the collection. Consequently, the self-structural similarity index is also likely to be low. On the other hand, for a homogeneous collection, a very high percentage of nodes are likely to be covered by frequent structures. Therefore, the frequent substructural self-similarity can be used as a surrogate for the self-similarity behavior of collection at the structural level. Therefore, our aim is to construct the partition of the document collection in such a way that the frequent substructural set from the collection covers as many nodes as possible. An interesting observation is that we have used only frequent structures of size l in order to measure the structural self-similarity. This is because structures of size l cannot be fairly compared to structures of size $(l+1)$ for the purpose of coverage. For example, if we allow substructures of any size to be present in the set of frequent structures, then the structures containing only 1 node would lead to a very high level of coverage. Thus, choosing the rank of the substructures is analogous to choosing the dimensionality of the projection in the case of projected clustering.

The pseudo-code for clustering of XML documents is illustrated in Figure 1. The primary approach is to use a sub-structural modification of a partition based approach in which the clusters of documents are built around groups

Algorithm XProj(Document Set: $\mathcal{D}$, Minimum Support: *min_sup*, Structural Size: l, NumClusters: k)
begin
Initialize representative sets $\mathcal{S}_1 \ldots \mathcal{S}_k$; /*See Sect. 4.3*/
while (*convergencecriterion* =false)
begin
Assign each document $D \in \mathcal{D}$ to one of the sets in $\{\mathcal{S}_1 \ldots \mathcal{S}_k\}$ using coverage based similarity criterion;
/* Let the corresponding document partitions be denoted by $\mathcal{M}_1 \ldots \mathcal{M}_k$; */
Compute the freq. substructures of size l from each set $\mathcal{M}_i$ using sequential transformation paradigm;
/*See Sect. 4.2*/
if $(|\mathcal{M}_i| \times min\_sup) \geq 1$
set $\mathcal{S}_i$ to frequent substructures of size l from $\mathcal{M}_i$;
/* If $(|\mathcal{M}_i| \times min\_sup) < 1$, $\mathcal{S}_i$ remains unchanged; */
end;
end

Figure 1: The Sub-structural Clustering Algorithm (High Level Description)

of representative sub-structures. Thus, instead of a single representative of a partition-based algorithm, we use a *substructural set representative* for the structural clustering algorithm. Initially, the document set $\mathcal{D}$ is randomly divided into k partitions with equal size, and the sets of sub-structure representatives are generated by mining frequent sub-structures of size l from these partitions. Similarly, in each iteration, the sub-structural representatives (of a particular size, and a particular support level) of a given partition are the frequent structures from that partition. These structural representatives are used to partition the document collection and vice-versa. We note that this can be a potentially expensive operation because of the determination of frequent substructures; in the next section, we will illustrate an interesting way to speed it up. In order to actually partition the document collection, we calculate the number of nodes in a document which are covered by each substructural set representative. A larger coverage corresponds to a greater level of similarity. The aim of this approach is that the algorithm will determine the most important *localized sub-structures* over time. This is analogous to the projected clustering approach which determines the most important localized projections over time. Once the partitions have been computed, we use them to re-compute the representative sets. These re-computed representative sets are defined as the frequent sub-structures of size l from each partition. Thus, the representative set S_i is defined as the substructural set from the partition $\mathcal{M}_i$ which has size l, and which has absolute support no less than $(|\mathcal{M}_i| \times min\_sup)$. Thus, the newly defined representative set S_i also corresponds to the local structures which are defined from the partition $\mathcal{M}_i$. Note that if the partition $\mathcal{M}_i$ contains too few documents such that $(|\mathcal{M}_i| \times min\_sup) < 1$, the representative set S_i remains unchanged.

Another interesting observation is that the similarity function between a document and a given representative set is defined by the number of nodes in the document which are covered by that set. This makes the similarity function more sensitive to the underlying projections in the document structures. This leads to more robust similarity calculations in most circumstances. In order to ensure termination, we need to design a convergence criterion. One useful criterion is based on the increase of the average sub-structural self-similarity over the k partitions of documents. Let the partitions of documents with respect to the current iteration be $\mathcal{M}_1 \ldots \mathcal{M}_k$, and their corresponding frequent sub-structures of size l be $\mathcal{S}_1 \ldots \mathcal{S}_k$ respectively. Then, the average sub-structural self-similarity at the end of the current iteration is $\Phi = \sum_{i=1}^{k} \Delta(\mathcal{M}_i, \mathcal{S}_i)/k$. Similarly, let the average sub-structural self-similarity at the end of the the previous iteration be Φ'. In the beginning of the next iteration, the algorithm computes the increase of the average sub-structural self-similarity, $\Phi - \Phi'$, and checks if it is smaller than a user-specified threshold ϵ. If not, the algorithm proceeds with another iteration. Otherwise, the algorithm terminates. In addition, an upper bound on the number of iterations is imposed. This is done in order to effectively handle situations in which the threshold ϵ is chosen to be too small.

4. EFFICIENT SUBSTRUCTURE MINING

A key issue is the frequent substructure mining in each iteration, which can make the procedure rather expensive. In this section, we will describe the process of efficiently finding frequent substructure representatives by using approximate mining techniques.

4.1 Approximate Substructure Mining

In the sub-structural clustering algorithm shown in Figure 1, there are two main time-consuming operations. One is the mining of the representative frequent substructures of size l from each XML document set. Another is the computation of the alignments of a set of representative substructures in a given XML document. This is partly due to the high computational complexity of the graph isomorphism problem. Because our goal is to cluster XML documents, we may not need the exact algorithm to mine frequent substructures or compute substructural alignments. One approach for improving the algorithm efficiency is to use approximate data representation in order to remove or simplify the graph isomorphism problem, while maintaining as much structural relationship among tree nodes as possible.

One simple way of doing this is to use the set of node labels to represent a sub-structure and use a set of frequent label sets to approximate the corresponding set of frequent substructures. Although this method is fast because of the presence of many efficient frequent itemset mining algorithms, it no longer preserves any structural information and cannot achieve good clustering quality. In order to alleviate this problem, the *sequence* of a pre-order depth-first traversal of the tree edges is adopted as the compromise between the complex tree structure and its corresponding simple node label set, where an edge is denoted by a pair of node labels. The advantage of the *edge sequence* representation is that it preserves both the parent-child relationship and the ordering among the sibling nodes. For example, the edge sequence representations of the three sub-structures in Figure 2 are $\langle AB, BC, CE, AB, BD\rangle$, $\langle AB, BC, BD, BD\rangle$, and $\langle AB, BC, CE, BD, AB, BD\rangle$, respectively. Furthermore, the use of the sequence representation also guarantees the ability to use efficient data mining algorithms.

The pre-order depth-first traversal of a tree structure assures that a parent node is always visited prior to its child nodes and a left sibling node is always visited prior to its

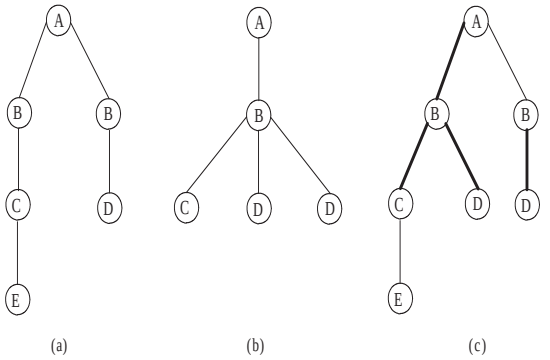

Figure 2: Tree Structures (Illustration 1).

right sibling nodes. Based on this edge traversal ordering, the following property holds.

PROPERTY 1. *(Subsequence relationship) If a tree T_1 is a subtree of another tree T_2, the edge sequence representation of T_1 must be a subsequence of the edge sequence representation of T_2.* □

We note that Property 1 connects the sequence relationship to the substructural relationship. The property indicates that if a substructure is frequent, its corresponding edge sequence must be frequent too. For example, the tree structure shown in Figure 2(a) is a substructure of the tree structure shown in Figure 2(c). Suppose the minimum support is 2, the tree shown in Figure 2(a) is frequent. And since $\langle AB, BC, CE, AB, BD\rangle \sqsubseteq \langle AB, BC, CE, BD, AB, BD\rangle$, edge sequence $\langle AB,\ BC,\ CE,\ AB,\ BD\rangle$ is also frequent. Note there may exist a *false mapping* between two tree structures and their corresponding edge sequences, where the *false mapping* means although one tree is not a fully connected substructure of another tree, their edge sequences have subsequence relationship. For example, the tree shown in Figure 2(b) is not a substructure of the tree shown in Figure 2(c). However, there is a subsequence relationship between their edge sequence representations. For example, $\langle AB, BC, BD, BD\rangle \sqsubseteq \langle \mathbf{AB},\ \mathbf{BC},\ CE,\ \mathbf{BD},\ AB,\ \mathbf{BD}\rangle$ holds. The dark edges in Figure 2(c) correspond to the edge sequence of $\langle AB, BC, BD, BD\rangle$. Due to the existence of potential *false mappings*, the support of an edge sequence may be higher than that of the corresponding tree structure.

One way to reduce the *false mapping* problem is to add some constraints during the process of mining frequent sequences. One such constraint is that all sibling nodes should have the same parent node. For example, the last three adjacent edges in sequence $\langle AB, BC, BD, BD\rangle$ indicate that the three child nodes with labels C, D, and D have the same parent node with a label B. However, in sequence $\langle \mathbf{AB},\ \mathbf{BC},\ CE,\ \mathbf{BD},\ AB,\ \mathbf{BD}\rangle$, the first BD edge does not share the same parent node with the second BD edge. Thus the alignment of $\langle AB, BC, BD, BD\rangle$ in $\langle \mathbf{AB}, \mathbf{BC}, CE, \mathbf{BD}, AB, \mathbf{BD}\rangle$ is not a valid one. A simpler solution for the *false mapping* problem is to allow it. Since our goal is to cluster XML documents, and the edge sequence partly preserves the structural information, the *false mapping* of a subsequence relationship still reflects a kind of similarity between the two corresponding structures. Therefore, the advantages of adding such constraints remains unclear from an effectiveness point of view. While we have mentioned a possible constraint-based solution to the false mapping problem for other data mining problems, we chose not to use it for the clustering problem. It is possible to adapt a number of efficient sequential pattern mining algorithms to mine frequent edge sequences of size $l-1$ (corresponding to l tree nodes). In our XProj algorithm, we revise one of the latest sequential pattern mining algorithms, BIDE [14], to mine frequent sequences of size $l-1$. BIDE is a projection-based algorithm, which mines frequent closed sequences with respect to a prefix by building and scanning its projected database. As we are only interested in the frequent sequences of size $l-1$, some enhancements can be made to the BIDE algorithm. For example, some short projected sequences in the sequence databases cannot be used to count frequent sequences of size $l-1$, and can therefore be safely pruned from the projected sequence database. Furthermore, once the size of the current prefix sequence reaches $l-1$, we do not grow it any more. Note that the revised BIDE algorithm stops growing the current prefix sequence once it reaches a size of $l-1$, and hence it will not be subsumed by its super-sequences even if it is non-closed. In addition, we turn off the sequence closure checking and search space pruning. Thus, the revised BIDE algorithm can mine the complete set of frequent sequences of size $l-1$. For more details of the BIDE algorithm, we refer to [14].

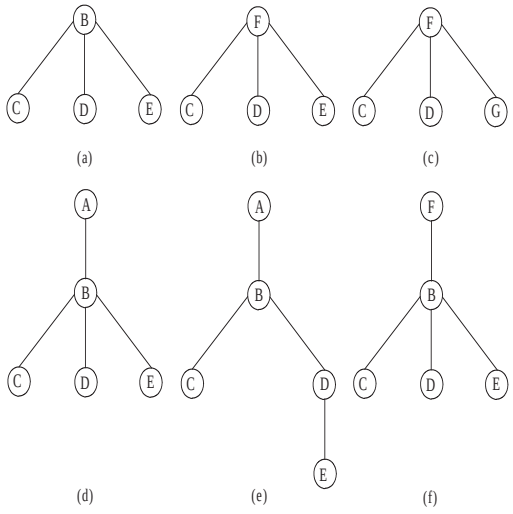

Figure 3: Tree Structures (Illustration 2)

There are several other potential forms of sequence representation of a tree structure. A simple variant of the edge sequence representation is the node label sequence. Because it does not maintain the parent-child relationship, it preserves too little structural information, and may not work well for XML clustering. A typical example is shown in Figure 3(a)-(c), in which we expect that the structural similarity between the two trees in Figure 3(b)-(c) is larger than that between the two trees of Figure 3(a)-(b). However, if we examine their node label sequences (i.e., $\langle B, C, D, E\rangle$, $\langle F, C, D, E\rangle$, and $\langle F, C, D, G\rangle$), it is difficult to differentiate the two similarities, because each pair of them has a common subsequence of size 3. Another example is shown in Figure 3(d)-(e). Figure 3(d) and Figure 3(e) are two different tree structures, but they have identical node label sequences, that is, $\langle A, B, C, D, E\rangle$. Another extreme rep-

resentation is the path sequence. In the pre-order depth-first traversal of a tree structure, each tree node is represented by the path from the root node to itself. For example, the path sequences of the two trees shown in Figure 3(d) and Figure 3(f), are $\langle A, AB, ABC, ABD, ABE\rangle$ and $\langle F, FB, FBC, FBD, FBE\rangle$, respectively. This example illustrates that the path sequence representation encodes more differentiating structural information.

4.2 Representative Substructure Selection for Similarity Computation

Given a document partition $\mathcal{M}_i$ $(1 \leq i \leq k)$, its corresponding seed set $\mathcal{S}_i$ may contain a large number of frequent sub-structures of size l. Although we have adopted the sequence to approximate a tree structure, the computation of the similarity function between a document and the seed set $\mathcal{S}_i$ is still quite costly. It will be desirable to select a small number of high quality substructures (or sequences), and only use this set of representative sub-structures to compute the similarity function and the average sub-structure self-similarity over the k partitions of the documents. We note that the support of a frequent sub-structure with respect to a given partition $\mathcal{M}_i$ usually indicates the degree of its locality to $\mathcal{M}_i$. Thus, a good representative sub-structure should have a support as high as possible. One way of achieving this goal is to simply select the top-K most frequent sub-structures of size l as the corresponding seed set $\mathcal{S}_i$.

One problem with the set of the top-K most frequent sub-structures of size l is that they may overlap a lot with each other, and thus may not cover all the documents in the corresponding partition. A solution to this problem is to use the sequential covering paradigm to select a number of frequent sub-structures which cover the documents in the partition. Starting from the sub-structure with the highest support, the documents covered by this sub-structure are removed from the partition and this sub-structure is added to the seed set. Then, the next sub-structure is retrieved, if it covers some remaining documents in the partition. The documents covered by this new sub-structure are removed from the partition, and the corresponding sub-structure is treated as a new representative sub-structure of the seed set. This procedure is repeated until all the documents in the partition are covered or the number of selected sub-structures reaches a user-specified threshold.

4.3 Representative Substructure Selection for Initialization

The XProj algorithm described in Section 3 initializes the representative sets by randomly dividing the XML database into k partitions and mining frequent substructures of size l for each partition, where k is the number of clusters. We refer to this kind of initialization method as *Randomized Initialization*. One problem with this method is that it is very hard to find some highly frequent substructures from the randomly generated partitions. Thus, it tends to produce low quality clusters in the following iterations or needs too many iterations to converge.

In order to generate more robust sets of representative substructures, one method is to directly compute k high-quality frequent sequences of size l-1 from the original database, and use each one of them as a representative substructure for one cluster. This results in a robust initialization. The problem is in choosing the k representative frequent sequences of size l-1. Our criteria is that the selected k representative sequences should be frequent and distinctive enough from each other in order to cover as many XML documents as possible. To achieve this goal, we adopt a variant of the sequential covering paradigm. We first use the revised BIDE algorithm to find the complete set of frequent sequences of size l-1 from the original database. We then choose one frequent sequence which covers the greatest number of input XML documents and remove the input XML documents covered by this selected sequence from the database. Subsequently, we choose another frequent sequence which can cover the remaining input XML documents the most and remove the corresponding covered input XML documents from the database. This procedure continues until k representative frequent sequences have been selected, which will be used to create the set of initialized seeds for the algorithm. One advantage of this approach is that it tends to create highly non-overlapping clusters of documents, and represents the entire input space fairly well for an initialization approach. We refer to this method as *Coverage-based Initialization*.

Note that each initial representative set contains only one frequent sequence, and all the initial representative sequences have the same length. Given an XML document, R, it is possible that R supports several initial representative sequences at the same time, and the similarity between R and any initial representative supported by R will equal the similarity between R and another initial seed supported by R. As a result, we need a tie-breaking rule in order to determine which partition R should be assigned to. In XProj, we simply choose the initial seed with the lowest support among all the R-supported seed sequences. As our experiments will demonstrate, this heuristic works well in practice.

4.4 Other Issues

Outlier Documents. We note that the tree approximation method discussed in Section 4.1 converts an XML document to a sequence of tree edges. A set of frequent sequences of size l-1 is used as a representative for a given partition of XML documents. However, in some cases, the length of a sequence representation of an XML document may be shorter than l-1. If this happens, it is hard to determine the partition to which the corresponding XML document should be assigned. In XProj algorithm, we will temporarily treat it as an outlier, which will not participate in the iterations of partitioning. After XProj finds a semi-final set of k clusters, we can compute the similarity between the outlier and each cluster of XML documents and assign it to the most similar cluster. If desirable, some documents with very low similarity to all clusters can remain as outliers. In the following, we will define the similarity between an outlier and a cluster of XML documents.

Consider an outlier XML document, denoted by R_o, and a cluster of XML documents, $\mathcal{R} = \{R_1 \ldots R_m\}$. Let $|R_i|$ denote the number of edges in the sequence representation of XML document R_i, while $|R_o \bigcap R_i|$ is the number of common edges of the sequence representations of R_o and R_i. The similarity between R_o and $\mathcal{R}$, $\Lambda(R_o, \mathcal{R})$, is defined as follows.

$$\Lambda(R_o, \mathcal{R}) = \sum_{i=1}^{m} \frac{|R_o \bigcap R_i|}{(|R_o| + |R_i|) \times m} \tag{2}$$

Highly Frequent sequences. If the XML documents to be clustered are homogenous, their corresponding sequences are very similar to each other even when they come from different classes. In this case, the frequent sequence mining algorithm will generate many highly frequent subsequences. From the clustering point of view, these subsequences almost appear in each XML document, and are not differentiable in terms of their cluster membership. As a result, the highly frequent subsequences are not useful for the clustering task and can be removed from the clustering process. In XProj, a user can specify a maximum support threshold, *max_sup*, in order to not generate the sequences with very high support.

5. EXPERIMENTAL RESULTS

We compared XProj with some recently developed XML clustering algorithms. We evaluated various aspects of the algorithm design, analyzed the algorithm sensitivity with several important parameters, and tested scalability. The results will show that the XProj algorithm is an extremely effective algorithm which retains a high level of scalability.

5.1 Test Environment and Data Sets

We implemented the Xproj algorithm using Microsoft Visual C++ 6.0 and performed a thorough experimental study on a Windows machine with AMD Athlon 2000+ and 768MB memory installed. We used both real and synthetic data sets to test the algorithm. These data sets are described below.

Synthetic Data Sets. We used the same sets of synthetic XML documents which were generated by the XML generator provided by the author of [5]. The first two data sets, denoted by DB1000DTD10MR3 and DB1000DTD10MR6, both contain 1000 XML documents and were generated from 10 different real DTDs as shown in Figure 6, each of which was used to generate 100 documents. The parameter MaxRepeats, which determines the maximum number of times a node will appear as a child of its parent node, was set for DB1000DTD10MR3 and DB1000DTD10MR6 at 3 and 6 respectively. The actual number of repeats generated is a random value between 0 and MaxRepeats. The parameter NumLevels that determines the maximum number of tree levels was set to 7 as in [5].

The third data set, DB300DTD3MR6, contained 300 synthetic XML documents generated from three similar DTDs as shown in Figure 7. Similarly, each DTD was used to generate 100 documents. The parameter MaxRepeats was set at 6 for this dataset. Note the three DTDs in Figure 7 are quite similar to each other, and this makes the clustering process for this collection a quite challenging one.

Real Data Set. The real data set we used is SIGMOD Record, which can be downloaded from *http://www.acm.org/sigmod/record/xml*. It contains 140 XML documents corresponding to two DTDs, IndexTermsPage.dtd and OrdinaryIssuePage.dtd(70 XML documents for each DTD). It is denoted by SIGMOD140DTD2.

As in [5], we used two popular information retrieval metrics, precision PR and recall R, to evaluate the clustering quality. Given a cluster C_i, let its dominant DTD be D_i (i.e., the majority of its documents have a DTD D_i), a_i be the number of documents in C_i which have a DTD D_i, b_i be the number of documents in C_i which do not have a DTD D_i, c_i be the number of documents which are not in C_i but have a DTD D_i. The precision PR and recall R are defined as follows.

$$PR = \frac{\sum_i a_i}{\sum_i a_i + \sum_i b_i}, R = \frac{\sum_i a_i}{\sum_i a_i + \sum_i c_i} \quad (3)$$

Clustering metric	Chawathe	Structure	XProj
Precision	0.83	1	1
Recall	0.96	0.98	1
# clusters	12	11	10

Table 1: *DB1000DTD10MR6* data set

5.2 Algorithm Evaluation

In this section, we will present the effectiveness, scalability, and sensitivity analysis of the XProj algorithm.

5.2.1 Comparison with Other Algorithms

We compared XProj with two XML clustering algorithms. One is the *Chawathe* algorithm, which is based on Chawathe's tree edit distance [4] to compute the similarity. Another point of comparison is the latest XML clustering algorithm, which is based on structure summaries and an enhanced tree edit distance algorithm [5]. We denote it by the *Structure* algorithm. Both algorithms are single-link hierarchial clustering algorithms and the results about them are from [5]. We compared the *Precision* and *Recall* of the three algorithms on data sets with both heterogeneous and homogeneous DTDs.

We first compared XProj with the other two algorithms using data sets DB1000DTD10MR6 and DB1000DTD10MR3, which were generated according to 10 heterogeneous DTDs. In order for XProj to generate 10 clusters, we set k=10. The minimum support *min_sup* was set to 0.01, l was set to 4, and the maximum support *max_sup* was set to 0.8, and we used the coverage based initialization to initialize the seed sets. Also, for each partition, we adopted the sequential coverage paradigm to choose the set of representative frequent sequences of size l. Table 1 depicts the comparison results for data set DB1000DTD10MR6. We can see that both the XProj and *Structure* algorithms work very well for this data set, and have higher precision and recall than Chawathe's tree edit distance based algorithm. The *Structure* algorithm generates one more cluster than XProj, and has a recall of 0.98, which means that it treats 20 XML documents as outliers, while XProj can perfectly cluster all the 1000 documents into exact 10 clusters. Thus, it has better clustering quality than the *Structure* algorithm. Compared to DB1000DTD10MR6, DB1000DTD10MR3 contains some smaller XML documents. XProj shows similar comparative results for this data set as well. Both the Chawathe and *Structure* algorithms generate more than 10 clusters, but have worse clustering quality than XProj for data set DB1000DTD10MR3.

The 10 DTDs used to generate DB1000DTD10MR6 and DB1000DTD10MR3, are quite different from each other, and most clustering algorithms can achieve reasonable quality with them. However, the three DTDs used to generate data set DB300DTD3MR6 are quite similar to each other, which makes the clustering quite challenging. The *Structure* algorithm was unable to identify groups of XML documents without using tree summaries and the calculated PR values were lower than 0.3 [5]. We were very interested in the per-

Clustering metric	Structure	XProj
Precision	0.78	1
Recall	0.78	1
# clusters	3	3

Table 2: *DB300DTD3MR6* data set

formance of XProj on DB300DTD3MR6 data set with homogeneous DTDs. For this data set, we chose $l = 4$, $k = 3$, *min_sup* $= 0.01$, and *max_sup* $= 0.8$. XProj can cluster DB300DTD3MR6 with perfect quality. As shown in Table 2, even with structure summaries turned on, the *Structure* algorithm can only achieve a precision of 0.78 and a recall of 0.78. These are much lower than those of XProj. This demonstrates that XProj also works well for difficult data sets with homogeneous DTDs. The reason for this is that the XProj algorithm is able to differentially find substructures which can discriminate between the different DTDs. Note the perfect results for XProj in the above experiments can only be achieved by using the *Coverage based Initialization* method with some tuned input parameters. First, we compare the performance between *Coverage based Initialization* and *Randomized Initialization*. Then, we present some sensitivity analysis to illustrate that XProj can generate good clustering results over a wide range of parameters.

5.2.2 Randomized Initialization vs. Coverage based Initialization

In Section 4.3 we proposed *Coverage based Initialization* in order to generate more robust seed sets of representative substructures than *Randomized Initialization*. We compared the two methods on data sets DB1000DTD10MR6, DB300DTD3MR6, and SIGMOD140DTD2. In the experiments, we set *min_sup* at 0.01, *max_sup* at 0.8, sequence length parameter l at 4, and the number of clusters at the number of DTDs. Table 3 shows the comparison results in terms of the algorithm precision. We can see that *Coverage based Initialization* provides more than 20% higher precision than *Randomized Initialization* for both DB1000DTD10MR6 and DB300DTD3MR6 data sets. This indicates that *Coverage based Initialization* is very helpful in improving the clustering quality of the algorithm. The SIGMOD140DTD2 data set is simple, and both methods achieve the same precision when the parameter l is not too small. However, *Coverage based Initialization* still outperforms *Randomized Initialization* with a small l for this data set. For example, when l equals 3, XProj with *Coverage based Initialization* has a precision 100%, while the precision of XProj with *Randomized Initialization* is only 89.8% for this data set. All our results over a variety of parameter values show similar trends as suggested in Table 3. This suggests that *Coverage based Initialization* is able to avoid the local optima in which the randomized method may get trapped. This can improve the clustering quality significantly.

5.2.3 Sensitivity Analysis

We used DB300DTD3MR6 and SIGMOD140DTD2 to perform the sensitivity analysis with some important input parameters. Here, we are interested in how the sequence length parameter l, the minimum support threshold *min_sup*, and the maximum support threshold *max_sup* affect the algorithm's clustering quality.

Data sets	Rand. Init.	Cover. Init.
DB1000DTD10MR6	0.734	1
DB300DTD3MR6	0.737	1
SIGMOD140DTD2	1	1

Table 3: Comparison: *Randomized* and *Coverage based Initialization* (Precision).

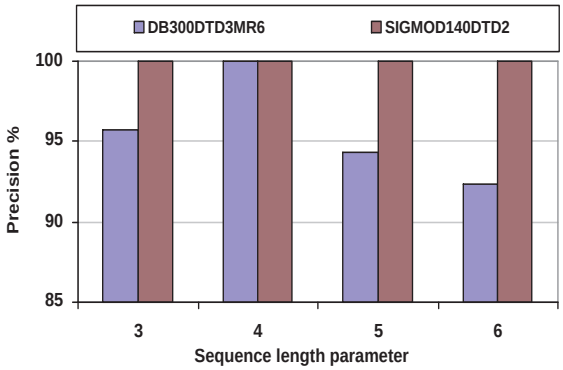

Figure 4: Sensitivity analysis (Varying l, SIGMOD140DTD2 and DB300DTD3MR6).

We first tested the impact of the sequence length parameter l on the clustering quality. Figure 4 shows the results with respect to precision. In the experiment, we set the number of clusters at 3 and 2 for data sets DB300DTD3MR6 and SIGMOD140DTD2, and fixed *min_sup* and *max_sup* at 0.01 and 0.8 respectively. We can see that when we change the sequence length l from 3 to 6 the precision is always higher than 92.0% for data set DB300DTD3MR6. This is much better that the best precision that can be achieved by the *Structure* algorithm. For data set SIGMOD140DTD2 the precision is always 100.0%. This illustrates that the XProj algorithm is able to perform effectively within practical sequence length parameter limits.

We then evaluated the impact of the minimum support *min_sup* and the maximum support *max_sup* to the clustering quality of $XProj$. We fixed the length parameter l at 4 and used the difficult data set DB300DTD3MR6. From Table 4 we see that different combinations of the minimum and maximum support thresholds have different precisions. However, for a variety of minimum and maximum support

max_sup	*min_sup*	Precision
0.6	0.01	0.951
0.7	0.01	0.965
0.8	0.01	1.0
0.6	0.02	0.933
0.7	0.02	0.935
0.8	0.02	0.942
0.6	0.03	0.927
0.7	0.03	0.927
0.8	0.03	0.930

Table 4: Sensitivity analysis (Varying *min_sup* and *max_sup*, DB300DTD3MR6).

thresholds, XProj algorithm always achieves much higher precision than the *Structure* algorithm.

5.2.4 Scalability Test

We used all the three synthetic data sets and the real data set SIGMOD140DTD2 to test clustering scalability by replicating them from 2 to 16 times. For all these data sets, we fixed the minimum support, *min_sup*, at 0.1, the maximum support, *max_sup*, at 0.8, sequence length l at 4, and the number of clusters, k, at the number of DTDs used to generate the corresponding data sets. As we can see from Figure 5, XProj shows linear scalability against the number of XML documents. This is because both the sequential pattern mining and cluster assignment procedures scale linearly with data set size. This is a useful property, since it means that the algorithm can easily be scaled up to very large data sets.

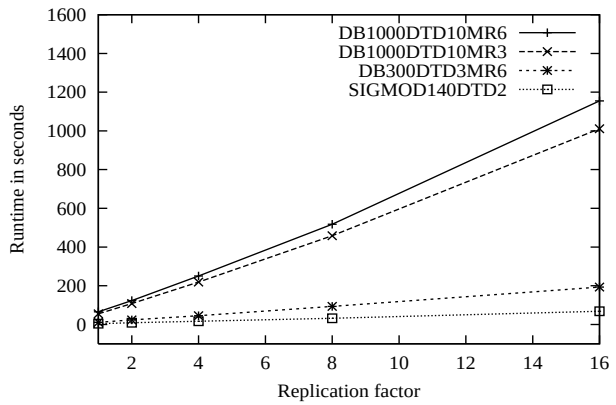

Figure 5: Scalability test.

6. CONCLUSIONS AND SUMMARY

In this paper, we presented a projected clustering algorithm for XML documents. The algorithm works with the use of subspace projections for finding multiple substructures which represent the seed sets for individual clusters. The use of multiple substructures to represent seed sets results in a robust clustering approach for the algorithm. At the same time, we discuss how to use a sequential pattern based approach for finding the substructures of interest from the documents. Since sequential pattern mining is a well studied problem, known algorithms can be leveraged for finding the relevant frequent substructures. This results in an efficient approach which can be used over very large data sets. We also show the qualitative advantages of the method over the best known techniques for XML document clustering.

7. REFERENCES

[1] C. C. Aggarwal, C. Procopiuc, J. Wolf, P.S. Yu, J.-S. Park. Fast Algorithms for Projected Clustering. *Proceedings of the ACM SIGMOD Conference*, 1999.

[2] C. C. Aggarwal. A Human-Computer Interactive Method for Projected Clustering. *IEEE Transactions on Knowledge and Data Engineering*, 16(4), 448–460, 2004.

[3] T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Satamoto, S. Arikawa. Efficient substructure discovery from large semi-structured data. *ACM SIAM International Conference on Data Mining*, 2002.

[4] S.S. Chawathe. Comparing Hierachical data in external memory. *Very Large Data Bases Conference*, 1999.

[5] T. Dalamagas, T. Cheng, K. Winkel, T. Sellis. Clustering XML Documents Using Structural Summaries. *Information Systems*, Elsevier, January 2005. Also appeared in *EDBT 2004 Workshops on Current Trends in Database Technology*, 2004.

[6] S. Guha, R. Rastogi, K. Shim. ROCK: a Robust Clustering Algorithm for Categorical Attributes, *International Conference on Data Engineering*, 1999.

[7] A. Jain and R. Dubes. Algorithms for Clustering Data. Prentice Hall, Englewood Cliffs NJ, USA, 1988.

[8] M. Lee, W. Hsu, L. Yang, X. Yang. XClust: Clustering XML Schemas for Effective Integration. *ACM Conference on Information and Knowledge Management*, 2002.

[9] W. Li, J. Han, J. Pei. CMAR: Accurate and Efficient Classification Based on Multiple Class-Association Rules. *International Conference on Data Mining*, 2001.

[10] W. Lian, D.W. Cheung, N. Mamoulis, S. Yiu. An Efficient and Scalable Algorithm for Clustering XML Documents by Structure. *IEEE Transactions on Knowledge and Data Engineering, Vol 16, No. 1*, 2004.

[11] R. Ng, J. Han. Efficient and Effective Clustering Methods for Spatial Data Mining. *VLDB Conference*, 1994.

[12] J. Pei, J. Han, B.-M. Asl, H. Pinto, Q. Chen, U. Dayal, M.-C. Hsu. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. *International Conference on Data Engineering*, 2001.

[13] A. Termier, M-C. Rousset, M. Sebag. TreeFinder: a First Step towards XML Data Mining. *International Conference on Data Mining*, 2002.

[14] J. Wang, J Han. BIDE: Efficient Mining of Frequent Closed Sequences. *International Conference on Data Engineering*, 2004.

[15] K. Wang, H.Q. Liu. Discovering Typical Structures of Documents: A Road Map Approach. *ACM SIGIR Conference*, 1998.

[16] M. J. Zaki, C. C. Aggarwal. XRules: An Effective Structural Classifier for XML Data. *ACM KDD Conference*, 2003.

[17] T. Zhang, R. Ramakrishnan, M. Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. *ACM SIGMOD Conference*, 1996.

[18] M. J. Zaki. Efficiently Mining Frequent Trees in a Forest. *ACM KDD Conference*, 2002.

customer.dtd

```
<!ELEMENT customer (name,address)>
<!ELEMENT name (firstname,lastname)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT address (street+, city, state,
  zip)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT zip (#PCDATA)>
```

fruitbasket.dtd

```
<!ELEMENT FruitBasket (#PCDATA|Apple
  Orange)*>
<!ELEMENT CitrusBasket (Orange*)>
<!ELEMENT Apple EMPTY>
<!ELEMENT Orange(#PCDATA)>
<!ELEMENT Apple kind CDATA "McIntosh"
  rotten (true|false) #REQUIRED>
<!ELEMENT Orange sizeCm NMTOKEN "15">
```

personal.dtd

```
<!ELEMENT personnel (person)+>
<!ELEMENT person (name,email*,url*,link?)>
<!ATTLIST person id ID #REQUIRED>
<!ELEMENT family (#PCDATA)>
<!ELEMENT given (#PCDATA)>
<!ELEMENT name (#PCDATA|family|given)*>
<!ELEMENT email (#PCDATA)>
<!ELEMENT url EMPTY>
<!ATTLIST url href CDATA #REQUIRED>
<!ELEMENT link EMPTY>
```

bookstore.dtd

```
<!ELEMENT bib (book*)>
<!ELEMENT book (title, (author+ | editor+),
  publisher, price)>
<!ATTLIST book year CDATA #REQUIRED>
<!ELEMENT author (last, first)>
<!ELEMENT editor (last, first, affliation)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT last (#PCDATA)>
<!ELEMENT first (#PCDATA)>
<!ELEMENT affliation (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

memo.dtd

```
<!ELEMENT memo (to, from, date,
  subject?,body)>
<!ATTLIST memo security (public |
  cern) 'public'>
<!ATTLIST memo lang CDATA #IMPLIED>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT body (para+)>
<!ELEMENT para (#PCDATA | emph)*>
<!ELEMENT emph (#PCDATA)>
```

population.dtd

```
<!ELEMENT population (continent*)>
<!ELEMENT continent (name, country*)>
<!ELEMENT country (name, province*)>
<!ELEMENT province (name, city*)>
<!ELEMENT city (name, pop)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT pop (#PCDATA)>
```

newspaper.dtd

```
<!ELEMENT NEWSPAPER (ARTICLE+)>
<!ELEMENT ARTICLE (HEADLINE,
  BYLINE, LEAD, BODY,NOTES)>
<!ELEMENT HEADLINE (#PCDATA)>
<!ELEMENT BYLINE (#PCDATA)>
<!ELEMENT LEAD (#PCDATA)>
<!ELEMENT BODY (#PCDATA)>
<!ELEMENT NOTES (#PCDATA)>
<!ATTLIST ARTICLE AUTHOR CDATA
  #REQUIRED>
<!ATTLIST ARTICLE EDITOR CDATA
  #IMPLIED>
<!ATTLIST ARTICLE DATE CDATA
  #IMPLIED>
<!ATTLIST ARTICLE EDITION CDATA
  #IMPLIED>
```

recipes.dtd

```
<!ELEMENT collection (description, recipe*)>
<!ELEMENT description ANY>
<!ELEMENT recipe (title, ingredient*,
  preparation, comment?,nutrition)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT ingredient (ingredient*,
  preparation)?>
<!ATTLIST ingredient name CDATA
  #REQUIRED amount CDATA #IMPLIED
  unit CDATA #IMPLIED>
<!ELEMENT preparation (step*)>
<!ELEMENT step (#PCDATA)>
<!ELEMENT comment (#PCDATA)>
<!ELEMENT nutrition EMPTY>
<!ATTLIST nutrition protein CDATA
  #REQUIRED carbohydrates CDATA
  #REQUIRED fat CDATA #REQUIRED
  calories CDATA #REQUIRED alcohol
  CDATA #IMPLIED>
```

tvschedule.dtd

```
<!ELEMENT TVSCHDULE (CHANNEL+)>
<!ELEMENT CHANNEL (BANNER,DAY+)>
<!ELEMENT BANNER (#PCDATA)>
<!ELEMENT DAY (DATE, (HOLIDAY|
  PROGRAMSLOT+))+>
<!ELEMENT HOLIDAY (#PCDATA)>
<!ELEMENT DATE (#PCDATA)>
<!ELEMENT PROGRAMSLOT (TIME, TITLE,
  DESCRIPTION?)>
<!ELEMENT TIME (#PCDATA)>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT DESCRIPTION (#PCDATA)>
<!ATTLIST TVSCHDULE NAME CDATA
  #REQUIRED>
<!ATTLIST PROGRAMSLOT VTR CDATA
  #IMPLIED>
<!ATTLIST TITLE RATING CDATA #IMPLIED>
<!ATTLIST TITLE LANGUAGE CDATA
  #IMPLIED>
```

catalog.dtd

```
<!ELEMENT CATALOG (PORODUCT+)>
<!ELEMENT PORODUCT (SPECIFICATIONS+,OPTIONS?,PRICE+,NOTES?)>
<!ATTLIST PORODUCT NAME CDATA #IMPLIED CATEGORY (HandTool|Table|Shop-Professional) "HandTool" PARTNUM CDATA
  #IMPLIED PLANT (Pittsburgh|Milwaukee|Chicago) "Chicago" INVENTORY (InStock|Backordered|Disconinued) "InStock">
<!ELEMENT SPECIFICATIONS (#PCDATA)>
<!ATTLIST SPECIFICATION WEIGHT CDATA #IMPLIED POWER CDATA #IMPLIED>
<!ELEMENT OPTIONS (#PCDATA)>
<!ATTLIST OPTIONS FINISH (Metal|Polished|Matte) "Matte" ADAPTER (Included|Optional|NotApplicable) "Included" CASE (HardShell
  |Soft|NotApplicable) "HardShell">
<!ELEMENT PRICE (#PCDATA)>
<!ATTLIST PRICE MSRP CDATA #IMPLIED WHOLESALE CDATA #IMPLIED STREET CDATA #IMPLIED SHIPPING CDATA #IMPLIED>
<!ELEMENT NOTES (#PCDATA)>
```

Figure 6: Heterogeneous DTDs for synthetic data.

bookstore1.dtd

```
<!ELEMENT entry (book*)>
<!ELEMENT book (title,author+,
  publisher, price )>
<!ATTLIST book year CDATA
  #REQUIRED)>
<!ELEMENT author (last, first)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT last (#PCDATA)>
<!ELEMENT first (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

bookstore2.dtd

```
<!ELEMENT bib (book*)>
<!ELEMENT book (title, (author+|editor+)
  publisher, price )>
<!ATTLIST book year CDATA
  #REQUIRED)>
<!ELEMENT author (last, first)>
<!ELEMENT editor (last, first, affiliation )>
<!ELEMENT title (#PCDATA)>
<!ELEMENT last (#PCDATA)>
<!ELEMENT first (#PCDATA)>
<!ELEMENT affiliation (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

bookstore3.dtd

```
<!ELEMENT bib (book*)>
<!ELEMENT book (title,author+,
  publisher, cost )>
<!ATTLIST book year CDATA
  #REQUIRED)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT cost (#PCDATA)>
```

Figure 7: Homogeneous DTDs for synthetic data.