Bitmap Indexing-based Clustering and Retrieval of XML Documents

Jong P. Yoon
Ctr. for Advanced Computer Studies
University of Louisiana
Lafayette, LA 70504-4330
jyoon@cacs.louisiana.edu

Vijay Raghavan
Ctr. for Advanced Computer Studies
University of Louisiana
Lafayette, LA 70504-4330
raghavan@cacs.louisiana.edu

Venu Chakilam
Ctr. for Advanced Computer Studies
University of Louisiana
Lafayette, LA 70504-4330
vmc0583@cacs.louisiana.edu

ABSTRACT

This paper describes a bitmap indexing based technique to cluster XML documents. XML documents can be hierarchically represented by elements. To improve performance of information retrieval, documents can be indexed using bitmap techniques. Such a bitmap index is sparse, meaning it contains unnecessarily many zero bits, especially for the word dimension. To remove zero bits and improve the performance of information retrieval, we propose to generate several small bitmap indexes that are not sparse. Using the similarity and popularity operations available in bitmap indexes, three clustering techniques are discussed: top-down clustering, bottom-up clustering, and mixed clustering. Experimental results are also shown in this paper.

Keywords

Clustering, Partitioning, BitCube, XML Indexing.

1. INTRODUCTION

EXtensible Markup Language (XML) is a standard for representing and exchanging information on the Internet. As such, documents can be represented in XML and therefore content-based retrieval is possible. However, because the size of XML documents is very large and the types vary, typical information retrieval techniques such as LSI (Latent Semantic Index) [7] are not satisfactory. Information retrieval on the Web is not satisfactory due to partly poor usage of structure and content information available in XML documents[5].

We consider a document database (*D*). Each document (*d*) is represented in XML. So, *d* contains XML-elements (*p*), where *p* has zero or more words (*w*) bound to it. Typical indexing requires a frequency table that is a two-dimensional matrix indicating the number of occurrence of the terms used in documents. By generalizing this idea, we use a three-dimensional matrix that consists of (*d*, *p*, *w*). We also treat a pair (*p*, *w*) as a query. Given a pair (*p*, *w*), we want to find *d* from a document database that is a triplet (*d*, *p*, *w*). In many cases on the Internet, this query answering is often too slow. A simple way to speed up query answering is to speed up the distance calculations from well-organized document clusters. In this paper, we propose a bitmap indexing technique, which we call the "BitCube," that represents (*d*, *p*, *w*), and operations that can cluster such documents efficiently. Before going further, consider the following examples.

1.1 Motivating Examples

EXAMPLE 1: Suppose that a query Q1 is posed to find all documents that describe "Clustering" in any figure caption(s) of subsections. This type of queries cannot easily be processed in relational document databases or object-oriented document databases due to inflexible modeling of irregularity of documents and unacceptable performance. However, in XML, irregularity of elements can be flexibly represented as shown in Figure 1.

EXAMPLE 2: Suppose that a query Q2 is posed to find all documents that describe "Indexing" in more than one sub-subsection. Notice that this type of queries asks for a specific document structure, that is, not for section, nor for subsection, but for sub-subsections. Searching an entire XML database is costly because a word pattern for search is rarely used if we search against a large document database. That is, a word list for a document is sparse as compared to the list of words available in the database. Search for a sparse list of words is not efficient. To resolve this problem, this paper proposes a way of clustering XML documents (based on word). In this way, searching can be restricted within only a cluster, instead of all documents in order to improve the performance.

1.2 Related Work

The conventional techniques used for document retrieval systems include stop lists, word stems, and frequency tables. The words that are deemed "irrelevant" to any query are eliminated from searching. The words that share a common word stem are replaced by the stem word. A frequency table is a matrix that indicates the occurrences of words in documents. The occurrence here

can be simply the frequency of a word or the ratio of word frequency with respect to the size of a document.

However, the size of frequency table increases dramatically as the size of the document database increases. To reduce frequency tables, the latent semantic indexing (LSI) technique has been developed [7]. LSI retains only "most significant" of the frequency table. Although the SVD trick reduces the size of the original frequency table, finding such a singular matrix is not trivial. Instead, this paper considers a more complex frequency table that represents terms (or values) according to an XML element ePath used in an XML document. We describe a novel approach to decompose a frequency table, if the table is a sparse matrix.

In addition, a new data structure, called X-tree, has been introduced for storing very high dimensional data [1]. Inverted indexes have been studied extensively [8]. Fast insertion algorithms on inverted indexes have been proposed [9].

Numerous document-clustering algorithms appear in the literature [10]. Agglomerative Hierarchical Clustering algorithms are probably the most commonly used. Linear time clustering algorithms, e.g., K-Means algorithm [4], are also used for on-line clustering. An ordered sequence of words is used to cluster documents available on the Internet [14]. On the Internet, there are some attempts, e.g., Alta Vista, to handle the large number of documents returned by query refinement features.

The collection of bitmaps in a bitmap index forms a 2-dimensional bit matrix [2]. A bitmap index has been used to optimize queries [2,6,11]. In this paper, we propose a 3-dimensional bit matrix. Bit-wise operations developed in the earlier work will also be generalized to the 3-dimensional bit matrix context.

1.3 Organization

The remainder of this paper is as follows. Section 2 describes preliminaries such as element paths in XML documents, and bit-wise operations in bitmap indexes. Section 3 describes the similarity of XML documents, the popularity of XML-elements, and partitioning techniques. BitCube, a set of triplets (document *d*, XML-element *p*, terms or contents *w*), is also introduced. Section 4 describes various clustering techniques, and their application to the BitCube indexing for content querying. Section 5 describes the experimental results. Interestingly enough, we find that once a BitCube is constructed, bit-wise operations on XML documents are executed in constant time. Section 6 concludes our work by summarizing our contributions and providing directions for future work.

2. PRELIMINARIES

This section defines technical terms borrowed from [13].

2.1 XML Document

Definition 2.1 (*Element Content*) An XML-element contains (1) simple content, (2) element content, (3) empty content, and (4) reference content. □

As an example, consider an XML document as shown in Figure 1. The element <section> in line (9) has a simple content. The element <section> in line (1) has element content, meaning that it contains two subsections as shown in lines (2) and (9). Of course, two content types can be mixed, e.g., the element <section> in line (2) contains a simple content in line (2) and also elements in lines (3)-(8). The element <verticalskip> contains empty content. The content <figure> has reference content that hyperlinks to a site.

```
(1)  <section>
(2)    <section> XML is represented in a bitmap indexing …
(3)      <section> It is a new standard … </section>
(4)      <section> An application is as shown in
(5)        <figure> http://www.a.b.c/clustering.algs </figure>
(6)        <caption> Clustering Algorithm </caption>
(7)      </section>
(8)    </section>
(9)    <section> Bitmap indexing technique … </section>
(10) </section>
```

Figure 1: XML Document

Definition 2.2 (*ePath*) Element Path, called "ePath," is a sequence of nested elements where the most nested element is simple content element. □

For example, in Figure 1, section.section.section.figure is an ePath, but section itself is not an ePath due to the top element <section> does not have simple content.

An XML document is defined as a sequence of ePaths with associated element contents. An XML document database contains a set of XML documents. In this paper, we propose a bitmap index for an XML document database. In a document-ePath bitmap index, a bit column represents an ePath, and a row represents an XML document. Of course, element contents, that is, values or words, need to be taken into account. In doing so, we need to consider 3-dimensional bitmap index, which will be discussed in detail in Section 3. In this

section, we consider only a 2-dimensional bitmap index. As an example of a bitmap index, assume those XML documents in Figure 2.

```
d1:                  d2:                  d3:
<e0>                 <e0>                 <e0>
  <e1> V1 </e1>        <e1> V1 </e1>        <e1> V11 </e1>
  <e2>                 <e2>                 <e2>
    <e3> V2 V3 V5 </e3   <e3> V3 V7 </e3>   <e3> V2 V7 </e3>
    <e4> V3 V8 </e4>     <e4> V9            <e4> V3 V9 </e4>
    <e5 />               <e6> V4 </e6>      <e5 />
  </e2>                  <e7> V6 </e7>      </e2>
</e0>                    </e4>              <e9> V5 </e9>
                       </e2>                </e0>
                       <e8> V6 V12 </e8>
                       </e0>
```

Figure 2: Example of XML Documents

Figure 2 is a set of simple XML documents. First, we need to define ePaths as follows:

p0=e0.e1, p1=e0.e2.e3, p2=e0.e2.e4, p3=e0.e5, p4=e0.e2.e4.e6, p5=e0.e2.e4.e7, p6=e0.e8, p7=e0.e9, Vi is a (key) word that is chosen from simple content to be used for search.

2.2 Bitmap Indexing

If a document has ePath, then set the corresponding bit to 1. Otherwise, all bits are set to 0. For each ePath, documents can be represented as shown in Figure 3.

Definition 2.3 (*Size of Bitmap*) $|b_i|$ denotes the size of a bitmap b_i, which is the number of 1's in a bitmap b_i, and $\|b_i\|$ denotes the cardinality of a bitmap b_i, which is the number of 1's plus 0's.

	p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7
d_1	1	1	1	1	0	0	0	0
d_2	1	1	1	0	1	1	1	0
d_3	1	1	1	1	0	0	0	1

Figure 3: A Bitmap Index for Figure 2

Definition 2.4 (*Hamming Distance*) The distance between two documents can be defined: $\text{dist}(d_i, d_j) = |\text{xOR}(d_i, d_j)|$, where xOR is a bit-wise exclusive OR operator. □

For example, the distance of two documents d_1 and d_2 in Figure 3 is $|\text{xOR}(d_1, d_2)| = 4$. Notice that in a bitmap index, if a bit represents a word, then the document distance in terms of word can be obtained.

2.3 Popularity of Bit Column

A bit column in a bitmap index can be described by its popularity. It is popular if used frequently enough. The index for the most popular bit column is mode in a bitmap index.

Definition 2.5 (*Popularity*) The popularity of a bit column is $\text{pop}(p_i) = |p_i|/\|p_i\|$. A bit column p_i is n-popular if $\text{pop}(p_i) \geq n$, where $0 \leq n \leq 1$ for a given n. A bit column p_i is m-unpopular if $\text{pop}(p_i) \leq m$, $(0 \leq m \leq 1)$. □

For example, in Figure 3, p_3 is 67 % popular because $\text{popl}(p_3) = .67$, while p_4 is 33 % popular. Given a bitmap index, using this notion, we can determine whether an ePath is popular or unpopular. Popularity of an ePath changes when a new document is added or deleted.

We can classify bit columns into three cases. Now, consider a bitmap index (for convenience, call it "the new bitmap index") after including the new "input bitmap" in the target bitmap index. (1) If $\text{pop}(p_i) \geq n$ in the new bitmap index, then such p_is of the input bitmap are called "*popular bit columns*"; (2) If $\text{pop}(p_i) \leq 1\text{-}n$, then p_i of the input bitmap is a so called "*weakening unpopular bit column*"; (3) If $1\text{-}n < \text{pop}(p_i) < n$, then p_i is called "*strengthening unpopular bit column*."

2.4 Radius and Center

This section describes two features of bitmap indexes: Radius and Center. Radius is a variance while center is a mean as in statistics.

Definition 2.6 (*Center*) In a cluster of XML documents, the center is a vector where each element of the vector is the mean value of the corresponding bits of the documents. □

For example, assuming that all documents in Figure 2 are in one cluster, the center of that cluster is {1,1,1,.67,.33,.33,.33,.33}.

Notice that a center can be computed by the mode or min (max) value rather than the mean value in other application domain.

Definition 2.7 (*Radius*) The radius of a cluster c is defined as $\text{radius}(c) = \text{MAX}\{\text{dist}(d_C, d_j)\}$, where d_C is the

center of the bitmap index for the cluster *c* and d_j is a bitmap for j_{th} document in the cluster *c*.

□

Notice that dist used in this case is the generalized version defined in Definition 4.1. For example, in Figure 3, the center of the bitmaps d_1, d_2, and d_3, in the bitmap index, d_c, is {1,1,1,.67,.33,.33,.33}. The radius is $dist(d_c, d_5) = .2$.

	p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}	p_{15}	p_{16}	p_{17}	p_{18}
d_1	1	1	1	1	0	0	0	0	1	0	0	0	1	1	1	0	0	0	1
d_2	1	1	1	1	1	1	0	1	1	0	0	0	0	0	0	1	1	0	0
d_3	1	1	1	1	0	0	0	0	0	1	1	1	0	1	0	1	0	1	1
d_4	1	1	1	0	0	0	0	0	1	0	1	1	1	1	1	0	0	0	1
d_5	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

Figure 4: A Bitmap Index

3. BITCUBE

In this section, we describe a 3-dimensional bitmap index, called "BitCube" [13]. This technique was originally introduced for the purpose of extending two-dimensional bitmap indexes to three-dimensional indexes.

3.1 BitCube

We revisit the representation of documents. XML document is defined as a set of (p, v) pairs, where (1) p denotes an element path (or ePath) described from the root element, and (2) v denotes a word or a content for an ePath. Typical methods of handling text-based documents use a frequency table or inverted (or signature) file that represents words for documents. However, since XML documents are represented by XML elements (or XML tags), the typical methods are not sufficient. We propose in this section a 3-dimensional bitmap representation, called BitCube.

A BitCube for XML documents is defined as BitCube = (*d, p, v, b*), where *d* denotes XML document, *p* denotes ePath, *v* denotes word or content for ePath, and *b* denotes 0 or 1, the value for a bit in BitCube (if ePath contains a word, the bit is set to 1, and 0 otherwise).

For example, consider XML documents similar to those documents shown in Figure 2. Five XML documents are represented in Figure 5. A BitCube for a set of documents: {d_1, d_2, d_3, d_4, d_5}. Each documents d_1={(p_0, v_1), (p_1, v_2), (p_1, v_3), (p_1, v_5), (p_2, v_3), (p_2, v_8) }, .., d_3={(p_0,v_{11}), (p_1, v_2), (p_1, v_7), (p_2, v_3), (p_2, v_9) …, (p_i,v_{i2}), (p_i,v_{i3}), (p_i,v_{i4}), …, (p_i,v_{ij})}, and so on.

The approximate size of the BitCube is (docs*words*paths)/8 bytes, where docs is the number of documents, and paths represents the number of different ePaths represented in the set of documents.

Bit columns for ePaths are initially organized in the same order as the order in which the documents are processed as in figure-4. Later, when a BitCube is partitioned, ePath bits can be shifted.

Bit columns for words may be organized in many ways that are well known.

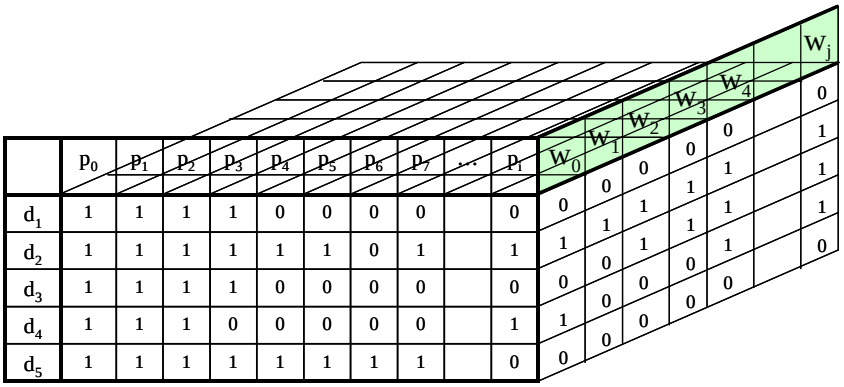

Figure 5: A BitCube: Example

- Simple word organization. All words used in the given XML documents are shown in a BitCube.
- Keyword organization. Only words importantly meaningful in the given XML documents are shown in a BitCube. The size of word list in this way is smaller than the previous organization.
- Signature word organization. This is similar to keyword organization, but those meaningful words are shown in the order of significance.

3.2 BitCube Operations

Three operations are described in this section: (1) ePath slice, (2) word slice, and (3) document project. The outcome of these operations, if applied against a BitCube, is a 2-dimensional bitmap index. Furthermore, these operations will be extended to "dicing" and "querying" which results in a bitmap index.

The ePath slice operation takes a ePath as input and returns a set of documents with words associated with it.

P_Slice(ePath) = {(doc, word) | ePath is used in doc, and the word is associated with the ePath}.

The outcome of this slicing is a 2-dimensional bitmap index that represents a set of documents with a set of words. The word slice operation takes a (search key) word as input and returns a set of documents.

W_Slice(word) = {(doc, ePath) | word is associated with the ePath which is in turn used in doc}.

The outcome is a 2-dimensional bitmap index that represents a set of documents with a set of ePath with which the word is associated. Multiple word slices can be combined together. The outcome of multiple word slices is a combination of the outcomes of each word slices. The way of combination depends on the way the words are requested. For example, if they are conjunctive, the outcomes need to be combined by conjunction.

The document project operation takes a document as input and returns a set of ePaths with words associated with those ePaths.

Project(doc) = {(ePath, word) | entire content and ePath pairs appeared in doc}.

The outcome is a bitmap index that represents a set of ePaths with their content (or words). A typical method for this project operation is a web browsing.

4. DOCUMENT CLUSTERING

Since low frequency words are represented for all the documents in the BitCube, the BitCube may become very sparse. It can be observed that very large number of distinct words can possibly occur in the given set of XML documents. These two factors make the Bitcube large and sparse. A sparse BitCube is not efficient in terms of space and access time. The bigger the BitCube, the more is time taken for accessing it.

In order to overcome the problem of sparse BitCubes, several smaller BitCubes are constructed or the BitCube is divided into several smaller BitCubes, there by reducing the size of each BitCube, and the content access time. There are two approaches proposed: partitioning in a top-down approach, and clustering in a bottom-up approach. The querying is faster in a smaller BitCube. The best way of constructing a smaller BitCubes is clustering of the indexing data.

4.1 Partitioning: Top-down Approach

partitioning (cluster c)

Consider a bitmap index c(i,w_j), where i denotes (document, ePath) pairs, and j denotes words.

Let n be n-popularity threshold and m be m-nonpopularity threshold.

1. Compute pop(w_j) for all j.
2. Split rows in c into three types of sub-clusters, T_1 for pop(w_j) ≥ n; T_2 for 1- m < pop(w_j)< n, and T_3 for pop(w_j) ≤ m.
3. Eliminate columns if the value of w_j is 0.
4. If any T_i is not empty and $\| T_i \| < \|c\|$ then partitioning(c_i) for all i

else stop.

Figure 6: Bitmap-based Partitioning Algorithm

In this approach all the documents are indexed for ePaths and considered as one cluster in the beginning. This cluster is recursively divided into smaller clusters such that all the ePath-similar clusters are put together into one cluster. Identifying the set of popular ePaths is the basis for defining similarity in the TopDown approach. There are three types of clusters possible in this approach.

- Type-1 cluster would be collection of all the documents that contain all the popular ePaths. That is it's the collection of all the rows in the original cluster that have the bits corresponding to all the popular ePaths set to 1.
- Type-2 cluster is the collection of all the documents that contain at least one of the popular ePaths but not all.
- The collection of all the documents that doesn't contain any of the popular ePaths is identified to be of Type-3.

A set of XML documents can be partitioned into n clusters. The number of partitions depends on the characteristics of documents; that is ePath and words used in the documents. In this section, for simplicity, we consider 2-dimensional bitmap indexes, representing documents and ePath. The algorithm of document partitioning is in Figure 6.

Given a collection D of documents, let t_i and r_i be the center and the radius of a clusters c_i, respectively. Let γ be the radius threshold ($\gamma \geq r_i \geq 0$).

Consider two clusters, c_i and c_j, and assume $r_i \geq r_j$. Compute distance l = dist (t_i, t_j)

1. if $l + r_j \leq r_i$

   then the cluster c_j is merged into c_i.

2. else if $\frac{l + r_i + r_j}{2} < \gamma$ then two clusters c_i and c_j are clustered to c_{i+j}.

3. else they are not clustered.

Figure 7: Bitmap-based Clustering Algorithm

4.2 Clustering: Bottom-Up Approach

In this approach, each document is compared with the existing clusters, and put into one if it potentially belongs to any, otherwise a new cluster is formed with the current document as centroid. The decision of whether a document belongs to a cluster is made by comparing the distance of the document from the centroid of that cluster with a pre-set radius threshold. In order to improve the performance of computing the center and distance defined earlier, we redefine the distance as follows:

Definition 4.1 (*Generalized Distance*)

$$dist(c_i, c_j) = \frac{1}{n} \sum_{k=0}^{n-1} |center_k(c_i) - center_k(cj)|$$

where center(c_i) denotes the average of the k^{th} bit in the cluster c_i for k>0. □

If the distance is less than the radius threshold, then the document is considered to belong to the cluster. If there are more than one such clusters to which the document can potentially belong to, then the closest cluster is chosen for the document to be put in. Clusters generated from this approach contain the documents that are nearer to each other in the two-dimensional space. The algorithm is in Figure 7. Notice that a cluster c_i contains one or more documents.

As per the earlier discussion, a BitCube is resulted from the indexing of the given XML data set, but it is sparse. In order to reduce the sparseness, here we propose two types of clustering of the indices.

4.3 Early Clustering

The clustering is done before the indexing of the documents. The given set of XML documents is divided into number of smaller clusters. Each cluster is indexed separately to form a separate BitCube. It can very well be observed that the size of each of the resulting BitCubes will always be smaller than that of the original BitCube, which is formed from the normal indexing method before clustering.

Initially all the documents are indexed on ePaths and then clustered using one of the clustering approaches discussed earlier. This results in the smaller document sets, which generate comparably smaller BitCubes. As the number of distinct words in each cluster be less than or equal to the actual number of distinct words in the given original data set, the over all space the BitCube occupies is reduced.

When a query is posted, then first we have to find out to which BitCube the queried contents belong to, and then steer the query to that BitCube. There are two levels of indexing, one, the indexing of the BitCubes and the second being the BitCube itself. This is the main disadvantage of this approach. For some kinds of queries we might have to access more than one BitCube.

4.4 Delayed Clustering

The given set of XML documents is indexed to construct the corresponding BitCube followed by clustering. The number of words is the main measure of sparseness of a BitCube. So if the number of words in the BitCube is reduced, then the density of the BitCube can be improved.

The BitCube is then expanded over the word dimension for all the documents and ePaths, resulting in a bitmap index with columns being all the distinct words of the data set, and the rows being all the possible combinations of the documents and ePaths of the given data set. It's thus a BitCube represented in two-dimensional space.

The bitmap index thus obtained is divided into smaller clusters. The all clusters have similar column index; that is, all the distinct words of the input data set. The all-zero columns are then removed from the clusters in order to reduce the number of words in each cluster. Thus the smaller clusters are made denser by removing the redundant words. As in early clustering, these smaller clusters are indexed so that the queries can be directed to the appropriate clusters.

5. EXPERIMENTAL RESULTS

The two approaches of clustering are evaluated for the clustering time, and the number of clusters generated in different test data scenarios. The test data collections are generated by running a program. Sample documents generated by the program are illustrated in Figure 2. The experimental environment is on Windows 2000 with 256M Byte Memory. The measurement of the efficiency and quickness of data retrieval is depicted in various graphs. The evolution of a measuring technique and the measurement of the effectiveness and the precision of the data retrieval are in progress.

5.1 Comparison of the Clustering Techniques

The two clustering techniques described earlier have their own advantages. The TopDown clustering technique, as by definition clustered based on the similarity of the words in different document-ePath combinations.

And so there can be many possible all-zero columns that can be removed. Thus the TopDown approach gets to denser indexing by discarding the many of the unnecessary zeros from the simple Bitcube indexing. But the TopDown clustering takes more time because this

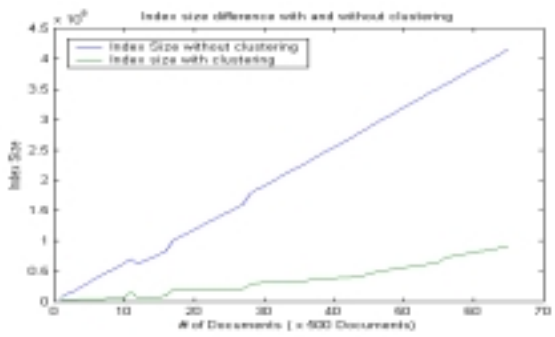

Figure 8: Indexing size with and without clustering

technique clusters all the possible because this technique clusters all the possible combinations of documents and ePaths compared to only the documents in case of BottomUp approach. The BottomUp approach is more like dividing the data set into smaller sets based on the similarity of the structure of the documents and not the contents. Then each set is indexed to generate a smaller BitCube, and each such set contains less number of distinct words than in the original data set, there by reducing the over all size of the BitCubes. The querying can be done easier in this case compared to TopDown clustering. The main disadvantage of BottomUp technique is that there is an extra indexing on ePaths of all the documents, to facilitate the clustering of the documents based on their structure.

There are various methods evaluating the cluster quality. We consider the two types of metrics for evaluating the clustering quality: entropy, and F-measure. We assume that a class *c* is predefined.

- *Entropy*. The entropy of clustering c_j, $E_j = -\sum_{i=1}^{|c|} p_{ij} \log(p_{ij})$, where $p_{ij} = \frac{n_{ij}}{n_j}$, n_{ij} is the number of documents in cluster c_j that belong to the class c, and n_j is the number of documents c_j.
- *F-measure*. The F-measure of cluster c_j and class c, $F(i) = \frac{2 \cdot \mathrm{Re}call(i) \cdot \mathrm{Pr}ecision(i)}{\mathrm{Re}call(i) + \mathrm{Pr}ecision(i)}$

5.2 Clustering and Index Size

In the simple Bitcube indexing technique, the size of the indexing is more, as all the ePaths, documents are indexed against all the words. So even the less frequently occurring words also indexed for all the ePaths and documents, which make the indexing big and sparse. The clustering of the documents is instrumental in making the indexing denser and compact. The clustering saves a considerable amount of indexing space. As depicted in Figure 8, the experimental results for the increasing indexing sizes of the document sets, indexed with and without clustering techniques do reveal that the amount of saved indexing space gets high for bigger document sets. Figure 8, recommends using clustering techniques while indexing because the saved indexing space makes it possible to index bigger sets of documents.

5.3 Execution Time of BitCube Operations

We measured the execution time for the BitCube operations: P-slice, W-slice and Document project with and with out the clustering technique incorporated. The execution times in both cases are scrutinized to find that the time for slicing doesn't change when the clustering is used. As depicted in Figure 9, execution time for W-slice is not deteriorated with the increasing number of documents when the clustering is also incorporated. Figure 10 describes that the P-Slice execution time is not affected by the clustering used. When the clustering is used, the execution time of the BitCube operations are remained more or less same, while reducing the sparcity of the indexing.

6. CONCLUSION

The main contributions of this paper are (1) the application of bitmap indexing to represent XML document collection as a 3-Dimensional data structure: XML document, XML-element path, and terms or words, (2) the definition of BitCube index based schemes to partition documents into clusters in order to efficiently perform BitCube operations, and (3) a document retrieval technique based on application of BitCube operations to subcubes resulting from the clustering phase. (4) Even for big XML document collections, the indexing is done in reasonable amount of time. The time taken for various BitCube operations remained constant.

REFERENCES

[1] S. Berchtold, D. A. Keim, and H. P. Kriegel, The X-tree: An Index Structure for High-Dimensional Data, Proc. Intl. Conf. On Very Large Data Bases, Bombay, India, 1996, 28-39.

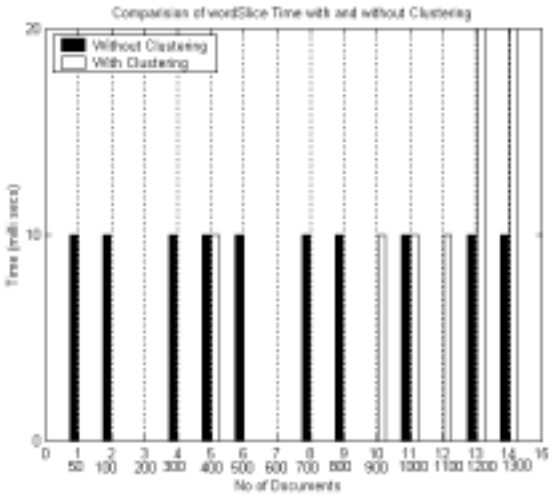

Figure 9: WordSlice Time Comparision with and without clustering.

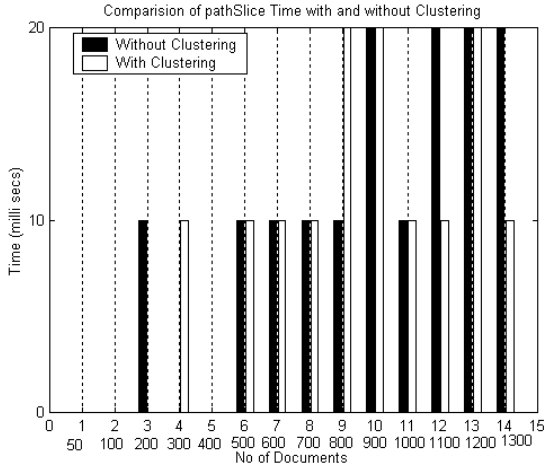

Figure 10: PathSlice Time Comparision with and without clustering.

[2] C. Chan and Y. Ioannidis, Bitmap Index Design and Evaluation, Proc. of Int'l ACM SIGMOD Conference, 1998, 355-366

[3] A. Gupta and I. Mumick, eds, Materialized Views, Cambridge, MA: MIT Press, 2000.

[4] D. Hill, A Vector Clustering Technique, Mechanised Information Storage, Retrieval and Dissemination, North-Holland, Amsterdam, 1968.

[5] M. Kobayashi and K. Takeda, Information Retrieval on the Web, ACM Computing Surveys, 32(2):144-173, 2000.

[6] P. O'Neil and D. Quass, Improved Query Performance with Variant Indexes, Proc. of Int'l ACM SIGMOD Conference, 1997, 38-49.

[7] C. Papadimitriou, H. Tamaki, P. Raghavan, and S. Vempala, Latent Semantic Indexing: a Probabilistic Analysis, Proc. of the 17th ACM Symposium on Principles of Database Systems, 1998, 159-168.

[8] G. Salton and M. McGill, Introduction to Modern Information Retrieval, New York, McGraw-Hill, 1983.

[9] A. Tomasic, H. Garcia-Molina, and K. Shoens, Incremental Updates of Inverted Lists for Text Retrieval, Proc. ACM SIGMOD Conf. On Management of Data, Minneapolis, 1994, 289-300.

[10] P. Willet, Recent Trends in Hierarchical Document Clustering: a Critical Review, Information Processing and Management, 24:577-97, 1988.

[11] M. Wu, Query Optimization for Selections using Bitmaps, Proc. Int'l ACM SIGMOD Conference, 1999, 227-238.

[12] J. Yoon and S. Kim, A Three-Level User Interface to Multimedia Digital Libraries with Relaxation and Restriction, IEEE Conf. on Advanced Digital Libraries, Santa Barbara, 1998, 206-215.

[13] J. Yoon, V. Raghavan and Venu Chakilam, BitCube: A Three Dimensional Bitmap Indexing for XML Documents, Thirteenth International Conference on Scientific and Statistical Database Management, FairFax, VA, 2001.

[14] O. Zamir and O. Etzioni, Web Document Clustering: A Feasibility Demostration, Proc. of ACM SIGIR Conf. on Research and Development in Information Retrieval, 1998, 46-54.