BitCube: A Three-Dimensional Bitmap Indexing for XML Documents

Jong P. Yoon, Vijay Raghavan, Venu Chakilam
Ctr. for Advanced Computer Studies
University of Louisiana, Lafayette, LA 70504-4330
{jyoon, raghavan, vmc0583}@cacs.louisiana.edu

Larry Kerschberg
Department of Information and Software Engineering
George Mason University, Fairfax, VA 22030-4444
kersch@gmu.edu

ABSTRACT

In this paper, we describe a new bitmap indexing based technique to cluster XML documents. XML is a new standard for exchanging and representing information on the Internet. Documents can be hierarchically represented by XML-elements. XML documents are represented and indexed using a bitmap indexing technique. We define the similarity and popularity operations available in bitmap indexes and propose a method for partitioning a XML document set.

Furthermore, a 2-dimensional bitmap index is extended to a 3-dimensional bitmap index, called BitCube. We define statistical measurements in the BitCube: mean, mode, standard derivation, and correlation coefficient. Based on these measurements, we also define the slice, project, and dice operations on a BitCube. BitCube can be manipulated efficiently and improves the performance of document retrieval.

Keywords

XML Document Retrieval, Bitmap indexing, Bit-wise Operations.

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 terms (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 Image 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. To increase performance, bitmap indexing of XML documents will be used.

EXAMPLE 2: Suppose that a query Q2 is posed to find all documents that describe Image 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 XML documents in a database may not have a regular structure in terms of which XML elements are used and how. To resolve this irregularity, bitmap indexes generated in the previous example, EXAMPLE 1, will be clustered. 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 [13]. 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. Section 4 introduces a BitCube that represents a set of triplets (document *d*, XML-element *p*, terms or contents *w*). The operations of a BitCube are developed: horizontal and vertical slice and dice.

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.

Definition 2.1 (*Element Content*) An XML-element contains (1) simple content, (2) element content, (3) empty content, or (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 originated from …
 (3)       <section> It is a new standard … </section>

 (4)       <section> An application to multimedia data is SVG as shown in
 (5)         <figure> http://www.a.b.c/syntax.xml </figure>
 (6)         <caption> XML Syntax </caption>
 (7)         <verticalskip />
 (8)       </section>
 (9)     <section> SGML was invented … </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. A bitmap index is 2-dimensional. 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 4. 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.

Now, we are ready to construct a bitmap index. 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 or 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 (*Distance*) The distance between two documents can be defined: $dist(d_i, d_j) = |xOR(d_i, d_j)|$, 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 $|xOR(d_1, d_2)| = 4$. Notice that in a bitmap index, if a bit represents a word, then the document distance in terms of word can be obtained. Documents d_i and d_j are the same if $|xOR(d_i, d_j)|=0$.

3. Bit-wise Operations

3.1 Similarity of Bitmap

In practice, on the Internet, documents can be represented in the form of a bitmap index, and can be retrieved by similarity matching, unless a user wants the exact copy of the original. To retrieve similar documents, the similarity of XML documents is defined.

Definition 3.1 (*Similarity*) The similarity of two XML documents (or bitmap rows), d_i and d_j, $sim(d_i, d_j) = 1- |xOR(d_i, d_j)|/MAX([d_i],[d_j])$. Two documents, d_i and d_j are ξ-similar if $sim(d_i, d_j) \geq \xi$, where $0 \leq \xi \leq 1$, and ξ is given. □

For example, in Figure 3, the similarity of d_1 and d_2 is 1-4/8 = 1/2, while the similarity of d_1 and d_3 is 1-1/8=7/8. That is, d_1 is closer to d_3 than d_2 in terms of ePath. This similarity check can be applied to a bit vectors (ePath-wise in this case). Again, if a bitmap takes element contents or words into account, the similarity in terms of words will be obtained.

3.2 Mode: 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 3.2 (*Popularity*) The popularity of a bit column is $pop(p_i) = |p_i|/[p_i]$. A bit column p_i is n-popular if $pop(p_i) \geq n$, where $0 \leq n \leq 1$ for a given n. A bit column p_i is m-unpopular if $pop(p_i) \leq m$, $(0 \leq m \leq 1)$. □

For example, in Figure 3, p_3 is 67 % popular because $popl(p_3) = .67$, while p_4 is 33 % popular. Given a bitmap index, using this notion, we can determine whether a ePath is popular or unpopular.

It is likely that popular bit columns are more often involved in similarity matching than unpopular bit columns.

Given a threshold n ($0.5 \leq n \leq 1$) for popularity, and a bitmap index corresponding to a cluster of XML documents, a bitmap (row) for an XML document to be added. 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 $\mathsf{pop}(p_i) \geq n$ in the new bitmap index, then such p_is of the input bitmap are called "*popular bit columns*"; (2) If $\mathsf{pop}(p_i) \leq 1-n$, then p_i of the input bitmap is a so called "*weakening unpopular bit column*"; (3) If $1-n < \mathsf{pop}(p_i) < n$, then p_i is called "*strengthening unpopular bit column*."

For example, consider a new input bitmap to be added to the bitmap index in Figure 3. d_5=11110110. Suppose the threshold n=0.67. The bit columns, p_0- p_2, are popular bit columns, p_3 is a strengthening unpopular bit column, and p_4- p_8 are weakening popular bit columns.

3.3 Variance and Mean: Radius and Center

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

Definition 3.3 (*Center*) In a cluster of XML documents, the center is a bit vector where bits corresponding to all popular bit columns and strengthening unpopular bit columns are to be set to 1, and all weakening unpopular bit column indexes that are to be set to 0. □

Notice that only popular and strengthening unpopular bits are considered for determining the "center" because the clustering should occur based on the popularity of the document-features.

	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

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

Figure 5: A Bitmap Index after Shifting Popular Bits to Left

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

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

Figure 6: Two Partitioned Bitmap Indexes before Eliminating 0 Bits

	p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_{15}	p_{16}
d_2	1	1	1	1	1	1	0	1	1	1	1
d_5	1	1	1	1	1	1	1	1	0	0	0

	p_0	p_1	p_2	p_3	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}	p_{15}	p_{17}	p_{18}
d_1	1	1	1	1	1	0	0	0	1	1	1	0	0	1
d_3	1	1	1	1	0	1	1	1	0	1	0	1	1	1
d_4	1	1	1	0	1	0	1	1	1	1	1	0	0	1

Figure 7: Bitmap Indexes partitioned from Figure 4

Definition 3.4 (*Radius*) The radius of a cluster c is defined as $\mathsf{radius}(c) = \mathsf{MAX}\{\mathsf{dist}(d_c, d_j)\}$, where d_c is the center of the bitmap index for the cluster and d_j is a bitmap for j_{th} document in the cluster c. ☐

For example, in Figure 3, the center of the bitmaps d_1, d_2, and d_3, in the bitmap index, d_c, is 11110000. The radius is $\mathsf{dist}(d_c, d_5) = 2$, if $d_5 = 11110110$.

3.4 Standard Deviation and Correlation Coefficient

The standard deviation (S) is a measure of the difference from the mean. Large value for S means the data is spread widely around the mean. Units are the same as the data itself.

$$S_i = \sqrt{\frac{\sum_i dist\,(d_i,\,center\,(d_i))^2}{[d_i]-1}}$$

The statistical definition of "relatedness" of two documents, d_i and d_j, is called correlation. We can calculate a "correlation coefficient" F as follows:

3.5 Document Partitioning

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 bitmap indexes representing documents and ePath.

The steps of document partitioning are as follows:

(1) Rearrange rows and columns: Identifying popular bit columns by checking $\mathsf{pop}(p_i)$; Shifting all similar bit columns together to left by checking $\mathsf{sim}(p_i,\, p_j)$. For example, a bitmap index in

Figure 4 is considered. Assume the thresholds for pop and sim are 0.6. Bit columns, p0-p3, p8, p13, and p18 are popular bit columns. The result after shifting them to left is shown in Figure 5.

(2) Identify similar bitmaps by checking $sim(d_i, d_j)$ for popular bits. Back to the previous example, two document pairs, (d1, d4) and (d2, d5), are similar.

(3) If still more bitmaps to be clustered from the above step, identify similar bitmaps by checking $sim(d_i, d_j)$ for unpopular bit columns. As the same example again, d3 is similar to the partition of (d1,d4). Hence, the two partitions are (d1,d3,d4) and (d2,d5) as shown in Figure 6. Notice that documents in a cluster are higher correlation coefficient than those in different clusters.

(4) Eliminate bit columns, p_i, if the $pop(p_i)$ = 0, meaning 0 bit columns. Back to the previous example again, two partitions are obtained and they are shown in Figure 7.

So far, the sim operation is based on pair-wise computation. We can modify this operation to some extent. Instead of using two bitmaps, we can use the centers of those bitmap indexes. In summary, there are three approaches of identifying similarity:

- Clustering by computing sim for pairs of documents at a time.
- Clustering by computing sim of a bitmap with the center of a target bitmap index. We already discussed about the center of bitmap indexes.

- Clustering by computing radius of a bitmap from the center of a target bitmap index. All bitmaps within a given radius are in the same partition.

4. Three-Dimensional Indexing

In this section, we describe a 3-dimensional bitmap index, called "BitCube." A BitCube is

$$F(d_i, d_j) = \sum_{i,j} dist(d_i, center(c_i)) \cdot dist(d_i, center(c_i)) \cdot MAX([d_i],[d_j]) \cdot S_i \cdot S_j$$

extended from a bitmap index we discussed in Section 3 by considering one more dimension. A BitCube represents a set of documents together with both (1) a set of ePaths (or XML-elements) and (2) a set of words (or element contents) for each ePath. For a BitCube, we propose two types of slice (slice of ePath and slice of Content) and project (of documents).
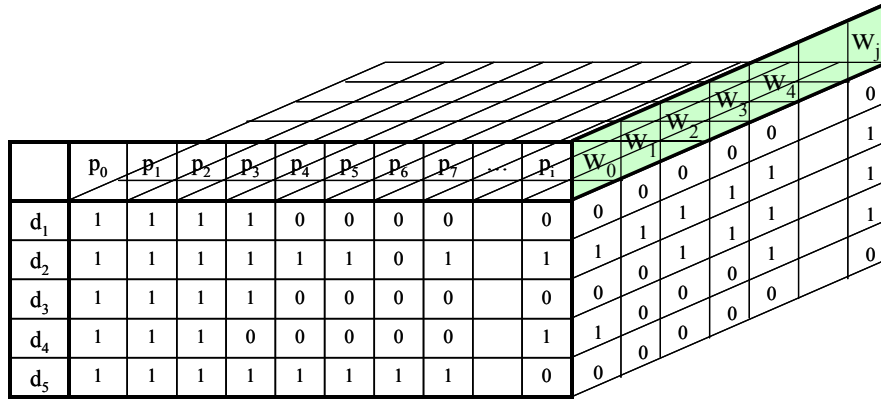
4.1 BitCube

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. 5 XML documents are represented in Figure 8. A BitCube for a set of documents: $\{d_1, d_2, d_3, d_4, d_5\}$. Each documents $d_1=\{(p_0, v_1), (p_1, v_2), (p_1, v_3), (p_1, v_5), (p_2, v_3), (p_2, v_8) \}, .., d_3=\{(p_0, v_{11}), (p_1, v_2), (p_1, v_7), (p_2, v_3), (p_2, v_9) \dots, (p_i, v_{i2}), (p_i, v_{i3}), (p_i, v_{i4}), \dots, (p_i, v_{ij})\}$, and so on.

The approximate size of the BitCube is (docs*words*paths)/8 bytes, where docs being the number of documents that are indexed, and paths in the chosen documents.

	p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	...	p_i
d_1	1	1	1	1	0	0	0	0		0
d_2	1	1	1	1	1	1	0	1		1
d_3	1	1	1	1	0	0	0	0		0
d_4	1	1	1	0	0	0	0	0		1
d_5	1	1	1	1	1	1	1	1		0

Figure 8: A BitCube: Example

Bit columns for ePath are initially organized in the same order as the order in which the documents are processed. Later, when a BitCube is partitioned, ePath bits can be shifted as shown in Section 3.

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

- Novice 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.

4.2 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 bitmap index. Furthermore, these operations will be extended to "dicing" and "querying."

4.2.1 ePath Slice
Each bit for a particular ePath is sliced. This operation takes a Path as input and returns a set of documents with words associated with it.

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

The outcome of this slicing is a bitmap index that represents a set of documents with a set of words. Typical web searches may not possible for ePath.

4.2.2 Word Slice
Each bit for a particular word can be sliced. This 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 bitmap index that represents a set of documents with a set of ePath with which the word is associated. Bitmap indexes illustrated in Section 3 are this type of bitmap index. Typical web searches are based on this word slice operation if they search XML documents.

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.

4.2.3 Document Project

Each row of a BitCube can be projected. This 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.

4.2.4 Dice: A Mixed Operation

The operations "Slice" and "Project" can be applied multiple times. Multiple operations can be treated as an operation that specifies a range. If we mix those three operations, each of which specifies a range, the outcome is again a BitCube that is smaller or equal to the original BitCube.

$$\text{Dice}([d_s..d_e],[p_s..p_e],[w_s..w_e]) = \{(doc, ePath, word) \mid doc \in [d_s..d_e], ePath \in [p_s..p_e], \text{ and } word \in [w_s..w_e]\}, \text{ where } x \in [x_s..x_e] \text{ implies that } x_s \leq x \leq x_e.$$

This operation is useful to generate document views [3]. An example of using views is to explore queries in EXAMPLE 2 in Section 1.1. An example of the reediting by the "Dice" operation is a customized publication.

5. Experimental Results

In order to evaluate the construction and manipulation of bitmap indexes and BitCubes, we generated document collections with the help of a tool. Sample documents generated with the tool are illustrated in Figure 2. We measured the execution time (in milliseconds) of all the BitCube operations with the increasing number of documents. The BitCube's performance is compared with the results obtained by performing the similar operations on other existing XML indexing tools. We have selected XQEngine [14] and XYZFind [15] for the performance comparisons. These are two of the prominent existing XML indexing tools. All the primitive query operations are performed with these two tools and the results are depicted in graphs. The same operations are performed on two other indexing tools. The experimental environment is on Windows 2000 with 256M Byte Memory.

5.1 Scalability of BitCube Construction

We measured the construction time for BitCubes for various number of ePaths associated with a document. Figure 9 show the construction time for the various scenarios of ePaths per document. We found that the construction time increases reasonably with the increase in the document set size. But once the indexing of XML is done, and then all the benefits of a BitCube can reaped from it.
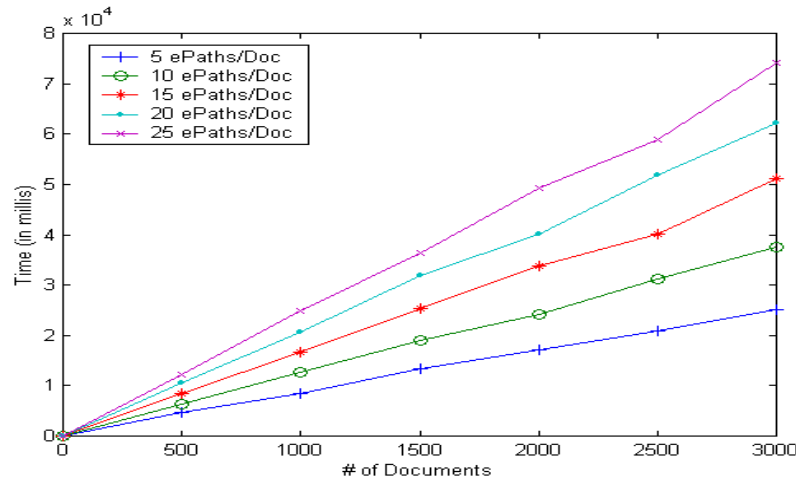

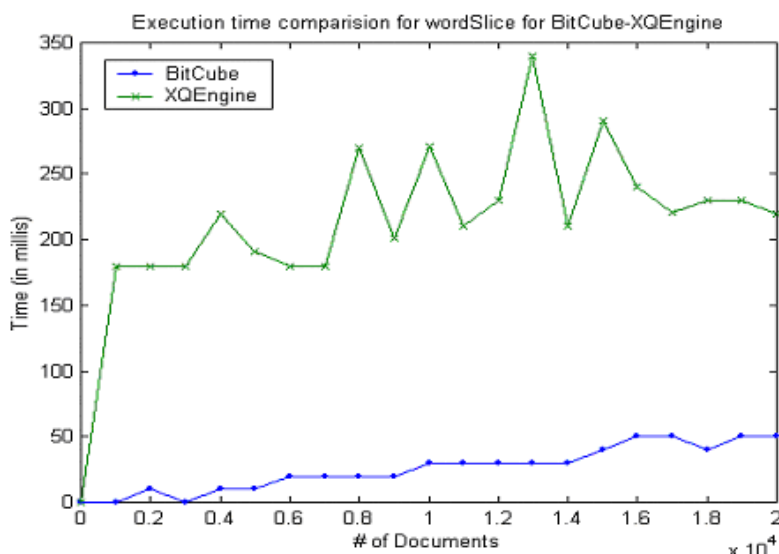**Figure 9: The effects of BitCube Creation with respect to ePath/document**

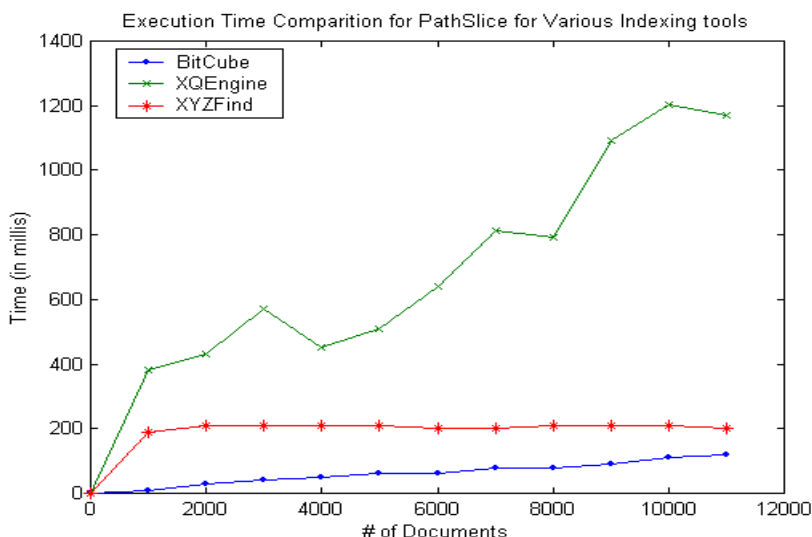**Figure 10: The comparison of word slice times for various tools**

Figure 11: The comparison of path slice times for various tools

5.2 Execution Time of BitCube Operations

We measured the execution time for the three operations: word-Slice, path-Slice, and document-Project with the increasing document set size for various tools. The experimental results are then compared with the help of graphs drawn. The experimental results for word-Slice for the BitCube and the XQEngine are comparatively depicted in figure-10. It can be easily read from the graph that the BitCube word-Slice operation is scalable and the performance of BitCube is better than the XQEngine. The execution time of path-Slice is depicted in Figure-11 for the tools BitCube, XQEngine and XYZFind. The comparative graph in figure-11 conveys that BitCube outperforms the other tools in path-Slice operation in terms of execution time. Similarly, the execution time of the document-Project operation is shown in Figures-12 for the selected tools with the increasing document set size. BitCube's performance in document-Projection is also good.
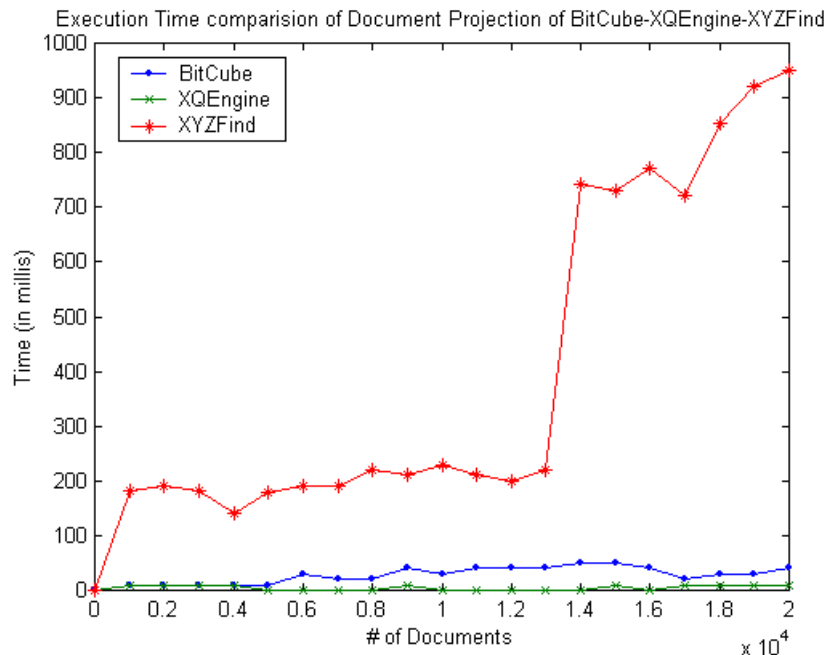
Figure 12: The comparison of doc-projection times for various tools

We found that the execution time of all the BitCube operators is very less when compared to the other tools and is scalable. This is partly because they are executed using bit-wise operations that require less execution time once the BitCube is constructed.

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. Experiments to show that our bitmap approach improves document clustering and performance of document retrieval on the Internet over alternative approaches are in progress. (4) Even for big XML document collections, the indexing is done in considerable amount of time. The time taken for various BitCube operations remained constant.

Acknowledgement

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.

[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, Matrialized 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] 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.

[14] http://www.fatdog.com, "XML Query Engine", XQEngine-1.0.

[15] http://www.xyzfind.com, "XML Database – Repository, search, and query – XYZFind", xyzfind-2.0.