BitCube: A Three-Dimensional Bitmap Indexing for XML Documents

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

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

ABSTRACT

XML is a new standard for exchanging and representing information on the Internet. Documents can be hierarchically represented by XML-elements. In this paper, we propose that an XML document collection be represented and indexed using a bitmap indexing technique. We define the similarity and popularity operations suitable for bitmap indexes. We also define statistical measurements in the BitCube: center, and radius. Based on these measurements, we describe a new bitmap indexing based technique to cluster XML documents. The techniques for clustering are motivated by the fact that the bitmap indexes are expected to be very sparse.

Furthermore, a 2-dimensional bitmap index is extended to a 3-dimensional bitmap index, called the BitCube. Sophisticated querying of XML document collections can be performed using primitive operations such as slice, project, and dice. Experiments show that the BitCube can be created efficiently and the primitive operations can be performed more efficiently with the BitCube than with other alternatives.

Keywords

XML Document Retrieval, Document Clustering, Bitmap Indexing, Bit-wise Operations.

1. Introduction

The 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) (Papadimitriou, et al. 1998) are not satisfactory. Information retrieval on the Web is not satisfactory due partly to poor usage of structure and content information available in XML documents (Kobayashi and Takeda, 2000).

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 (Papadimitriou, et al. 1998). 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 (Berchtold, et al. 1996). Inverted indexes have been studied extensively (Salton, 1983). Fast insertion algorithms on inverted indexes have been proposed (Tomasic, et al. 1994).

Numerous document clustering algorithms appear in the literature (Willet, 1988). Agglomerative Hierarchical Clustering algorithms are probably the most commonly used. Linear time clustering algorithms, e.g., K-Means algorithm (Hill, 1968), are also used for on-line clustering. An ordered sequence of words is used to cluster documents available on the Internet (Zamir and Etzioni, 1998). 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 (Chan and Ioannidis, 1998). A bitmap index has been used to optimize queries (Chan and Ioannidis, 1998 ; O'Neil and Quass, 1997; Wu, 1999). 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.

d_1:
```
<e0>
  <e1> V1 </e1>
  <e2>
    <e3> V2 V3 V5 </e3
    <e4> V3 V8 </e4>
    <e5 />
  </e2>
</e0>
```

d_2:
```
<e0>
  <e1> V1 </e1>
  <e2>
    <e3> V3 V7 </e3>
    <e4> V9
      <e6> V4 </e6>
      <e7> V6 </e7>
      </e4>
    </e2>
    <e8> V6 V12 </e8>
  </e0>
```

d_3:
```
<e0>
  <e1> V11 </e1>
  <e2>
    <e3> V2 V7 </e3>
    <e4> V3 V9 </e4>
    <e5 />
  </e2>
  <e9> V5 </e9>
</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: $\mathsf{dist}(d_i, d_j) = |\mathsf{xOR}(d_i, d_j)|$, 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 $|\mathsf{xOR}(d_1, d_2)| = 4$. 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 $|\mathsf{xOR}(d_i, d_j)|=0$.

3. Bit-wise Operations

3.1 Similarity of Bitmap

In practice, on the Internet, documents can be represented in the form of a 2-dimensional bitmap index, and can be retrieved by similarity matching, unless a user specifically queries for a particular document. 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, $\mathsf{sim}(d_i, d_j) = 1 - |\mathsf{xOR}(d_i, d_j)|/\mathsf{MAX}([d_i],[d_j])$. Two documents, d_i and d_j are ξ-similar if $\mathsf{sim}(d_i, d_j) \geq \xi$, 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 $\mathsf{pop}(p_i) = |p_i|/[p_i]$. A bit column p_i is n-popular if $\mathsf{pop}(p_i) \geq n$, where $0 \leq n \leq 1$ for a given n, *popularity threshold*. A bit column p_i is n-unpopular if $\mathsf{pop}(p_i) \leq n$, $(0 \leq n \leq 1)$. □

For example, in Figure 3, p_3 is 67 % popular because $pop(p_3)$ = .67, while p_4 is 33 % popular. Given a bitmap index, using this notion, we can determine whether an ePath is popular or unpopular. It is likely that popular bit columns are more often involved in similarity matching than unpopular bit columns. The popularity of an ePath changes when a new document is added or deleted to the document set.

We can classify bit columns into three cases. Now, consider a bitmap index (for convenience, call it "the new bitmap index") is included the new "input bitmap index" to result the target bitmap index. (1) If $pop(p_i) \geq n$ in the new bitmap index, then such p_is of the input bitmap are called "*popular bit columns*"; (2) If $pop(p_i) \leq m$, where m is the *strengthening unpopularity threshold*, and $m \leq n$ then p_i of the input bitmap is called "*weakening unpopular bit column*"; (3) If $m < pop(p_i) < n$, 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 that the popularity threshold n=0.67 and strengthening unpopularity threshold m = 0.33. 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

Definition 3.4 (*Radius*) The radius of a cluster of XML documents c is defined as $radius(c) = MAX\{dist(d_C, d_j)\}$, 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.

	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

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 when popularity threshold is 0.6 and the strengthening unpopularity threshold is defined to be 0.4. The radius is $\mathsf{dist}(d_c, d_5) = 2$, if d_5=11110110.

3.4 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 columns: Identify the popular bit columns by checking $\mathsf{pop}(p_i)$; Shift all the popular bit columns to left. For example, consider the bitmap index in Figure 4. Assume the thresholds for popularity, and sim are 0.6 and strengthening unpopularity is 0.4. 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 $\mathsf{sim}(d_i, d_j)$ 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 $\mathsf{sim}(d_i, d_j)$ 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.

(4) Eliminate bit columns, p_i, if the $\mathrm{pop}(p_i) = 0$, 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 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) …, (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 being the number of documents that are indexed, and paths in the chosen documents. Some of the main characteristics of the BitCube are listed below.

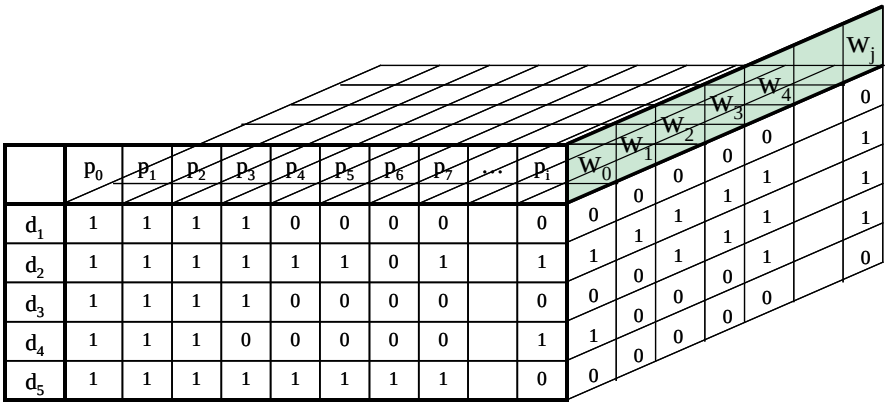

Figure 8: A BitCube: Example

- Basic word organization. All words used in the given XML documents are shown in a BitCube.

- Keyword organization. Only highly meaningful words 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 2-dimensional 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 2-dimensional 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 2-dimensional 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 2-dimensional 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.

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

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 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 called XMLGenerator (http://www.alphaworks.ibm.com/tech/xmlgenerator). Sample documents generated with the tool are illustrated in Figure 2. We measured the execution time (in milliseconds) of all the BitCube operations with increasing number of documents. The BitCube's performance is evaluated for various primitive operations with the results obtained by performing the same operations on two other existing XML indexing tools. We have selected XQEngine (http://www.fatdog.com) and XYZFind (http://www.XYZFind.com) for the performance comparisons. These are two of the prominent existing XML indexing tools. The experiments are conducted for several words, ePaths and documents chosen randomly from the document set. It is made sure that same words/ePaths/documents are queried from the document set across the different tools that are compared against each other. For each BitCube operation, the results are averaged and depicted in various graphs below for comparison. For example, for the comparison of path slice times (Figure 11), we compute the average computing time over two randomly selected ePaths for each combination of document collection size and XML indexing tool. For all the results reported, no clustering of the BitCube is performed. The experimental environment is 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 shows the construction time of the BitCube for the various cases of ePaths per document. We found that the construction time increases linearly with the increase in the number of ePaths per document. As the number of ePaths/document increases, the BitCube creation time increases at a higher rate.

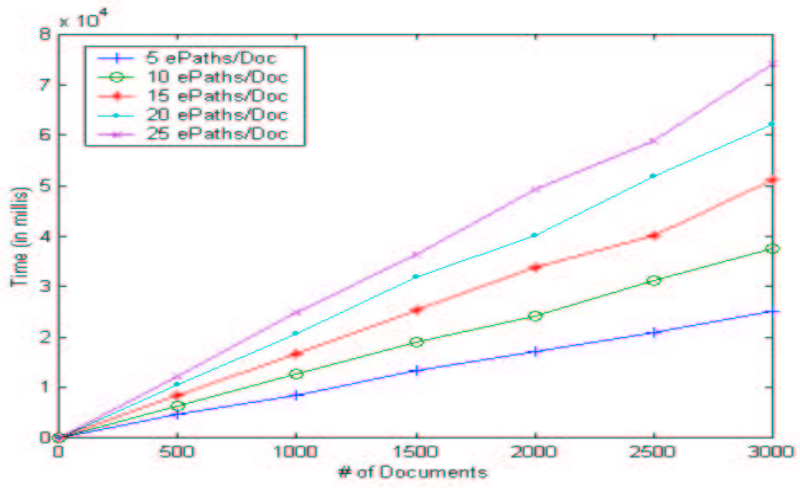

Figure 9: The effects of BitCube Creation with respect to ePath/document

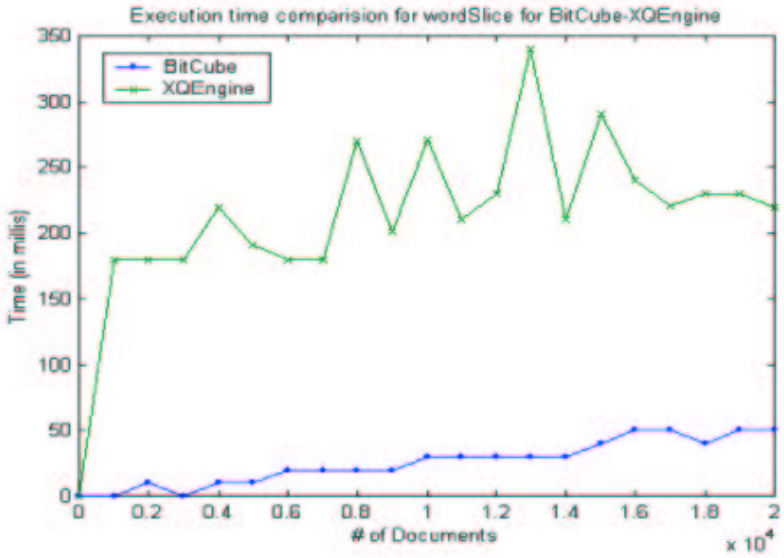

Figure 10: The comparison of word slice times for various tools

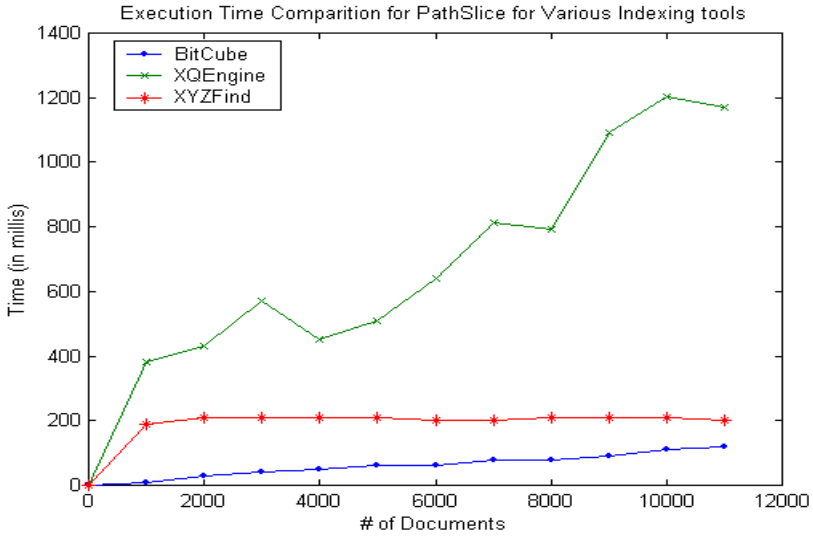

Figure 11: The comparison of path slice times for various tools

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 the three tools. The experimental results in terms of average execution times are plotted in the graphs. The experimental results for word-Slice for the BitCube and the XQEngine are depicted in figure-10. It can be easily read from the graph that the performance of BitCube is better than that of the XQEngine. We were unable to obtain the corresponding result for the XYZFind. The execution times of path-Slice are 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 times of the document-Project operation is shown in Figure-12 for the selected tools with the increasing document set size. The XQEngine is found to achieve the best performance for the document projection operation. But the performance of the BitCube is quite comparable. Furthermore, for all three operations, the execution times are nearly constant as the document set size increases.

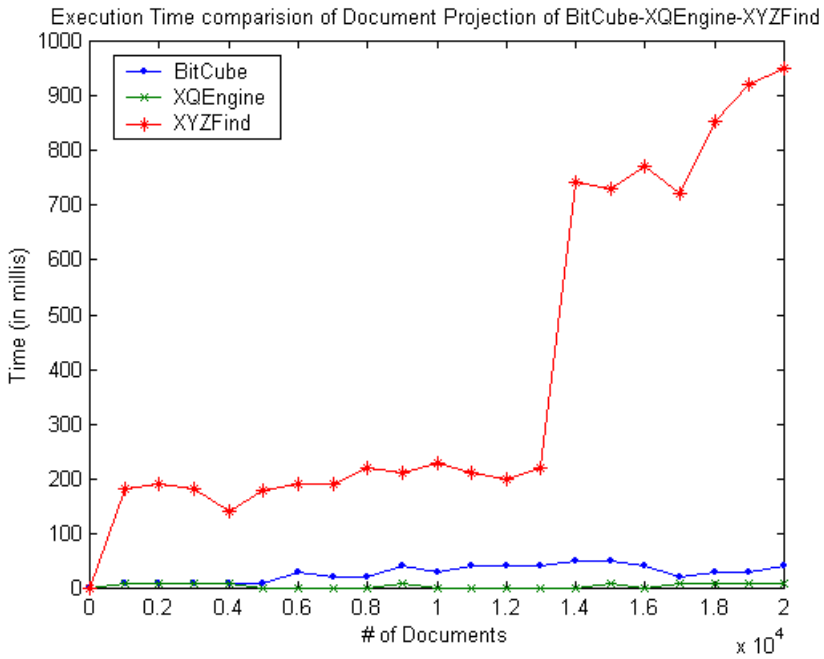

Figure 12: The comparison of doc-projection times for various tools

6. Conclusions

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 specification of BitCube-based schemes to partition XML documents into clusters in order to efficiently perform BitCube operations, and to reduce the storage constraints of the BitCube, (3) conduct of experiments where execution times for various primitive BitCube operations are compared with those of two other indexing tools and the BitCube is found to perform better than the others in most of the cases; where it doesn't, its times are quite close to the best execution times (4) conduct of experiments showing that even for fairly large XML document collections, the index creation is done in a reasonable amount of time and it scales linearly with the size of the document set.

In our future work, we plan to further improve the BitCube performance by applying the proposed clustering schemes. Clustering schemes will result in the original sparse BitCube being decomposed into several smaller, dense BitCubes. We also plan to compare the performance of the alternative index structures with respect to certain complex queries, including those that obtain retrieval output in which documents are rank ordered relative to the given query.

Acknowledgements

This research was supported in part by the U.S. Department of Energy grant DE-FG02-97ER1220 and La LEQSF research grant No LEQSF(2000-03)-RD-A-42. We thank B. Kim for his comments. This research was supported in part by the E-Center for E-business and the Virginia Center for Innovative Technology.

References

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

[2] Chan, C., and Ioannidis, Y. (1998) "Bitmap Index Design and Evaluation," in {*Proc. of Int'l ACM SIGMOD Conference*}, pp. 355-366.

[3] Gupta, A., and Mumick, IS (Eds.) (2000). {*Materialized Views*}, MIT Press, Cambridge, MA.

[4] Hill, D. (1968). "A Vector Clustering Technique", {*Mechanized Information Storage, Retrieval and Dissemination*}, North-Holland, Amsterdam.

[5] Kobayashi, M., and Takeda, K. (2000). "Information Retrieval on the Web", {*ACM Computing Surveys*} Vol. 32(2), pp. 144-173.

[6] O'Neil, P., and Quass, D. (1997) "Improved Query Performance with Variant Indexes," in {*Proc. of Int'l ACM SIGMOD Conference*}, pp. 38-49.

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

[8] Salton, G., and McGill, M. (1983). {*Introduction to Modern Information Retrieval*}, McGraw-Hill, NY.

[9] Tomasic, A., Garcia-Molina, H., and Shoens, K. (1994) "Incremental Updates of Inverted Lists for Text Retrieval," in {*Proc. ACM SIGMOD Conf. On Management of Data*}, Minneapolis, U.S.A., pp. 289-300.

[10] Willet, P. (1988). "Recent trends in hierarchical document clustering: a critical review", {*Information Processing and Management*} 24:577-97.

[11] Wu, M. (1999) "Query Optimization for Selections using Bitmaps," in {*Proc. Int'l ACM SIGMOD Conference*}, pp. 227-238.

[12] Yoon, J., and Kim, S. (1998) " A Three-Level User Interface to Multimedia Digital Libraries with Relaxation and Restriction," in {*IEEE Conf. on Advanced Digital Libraries*}, Santa Barbara, U.S.A., pp. 206-215.

[13] Zamir, O., and Etzioni, O. (1998) "Web Document Clustering: A Feasibility Demonstration," in {*Proc. of ACM SIGIR Conf. on Research and Development in Information Retrieval*}, pp. 46-54.