JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 26, 393-407 (2010)

A Hierarchical Bitmap Indexing Method for Similarity Search in High-Dimensional Multimedia Databases*

JONGHO NANG, JOOHYOUN PARK, JIHOON YANG AND SAEJOON KIM
Department of Computer Science and Engineering
Sogang University
Seoul 121-742, Korea

This paper proposes an efficient indexing mechanism for similarity search in high-dimensional multimedia database that quickly filter-outs the irrelevant objects using a novel indexing structure, called *HBI* (*Hierarchical Bitmap Index*). In this bitmap index, the feature (or attribute) value of object at each dimension is represented with a set of two bits each of which indicates whether it is relatively high ('11'), low ('00'), or neither ('01') compared to the feature values of other objects at a hierarchical organized interval. This approximation helps to reduce the CPU time of filtering process because many irrelevant objects could be simply excluded by just XORing the bitmaps of two objects. Upon experimental results, we find that there is an optimal number of bitmaps that keeps the filtering rate as high as possible while keeping the search time as short as possible. Furthermore, we also find that the similarity search using the proposed indexing mechanism is about 2-3 times faster than VA-File while guaranteeing the exact solutions.

Keywords: multimedia retrieval, similarity search, high dimensional indexing, hierarchical bitmap indexing, curse of dimensionality

1. INTRODUCTION

The similarity in CBMR (Content Based Multimedia Retrieval) is often measured by the L_p-distance between the feature vectors extracted from multimedia objects such as images or videos. Since they are usually represented in a very high dimensional space and the measuring L_p-distance in this space incurs a high cost for a large data set, it is essential to use an efficient indexing method to develop a practical CBMR system. The various indexing methods for similarity search in multimedia retrieval have been widely research- ed. Some conventional data partitioning approaches such as R-tree [1], R*-tree [2], X-tree [3], VP-tree [4] or M-tree [5] can be used for solving this problem. However, their performances are known to drastically degrade as the number of dimensions increases because of the problem called "curse of dimensionality [6]". Recently, some new researches (for example, the VA-file [7] and the LPC-file [8]) have been proposed to resolve this problem. They are called *filtering approach* because they first filter-out many irrelevant objects by scanning the compact approximations of objects and secondly compute the exact distances between the query and remaining relevant objects to find out the similar objects. Similarly, Signature File [11, 12] method has been often used in document retrieval area. This method makes a compact abstraction of documents to provide a quick test, which discards many of the non-qualifying documents. In the view of Signa-

Received June 19, 2008; revised September 18, 2008; accepted March 19, 2009.
Communicated by Suh-Ying Lee.

* This is an extended version of paper, titled "An Efficient Indexing Structure for Content Based Multimedia Retrieval with Relevance Feedback" in Proceedings of ACM Symposium on Applied Computing, Vol. 1, 2007, pp. 517-524.

ture File, VA-File and LPC-File suggest the way of hashing vector sets and calculating the approximation of the distance between them. Although this approach could eliminate the I/O time for reading the feature vectors of irrelevant objects in the second phase so that the total search time could be reduced, it still requires a lot of CPU time because it should calculate the L_p-distances (a costly computation) between the query and all objects in the database with their compact approximations in the filtering process. Some works [13, 14] introduced the improved data partitioning approach combined with the filtering approach. A-tree [13] is motivated by SR-tree and VA-File. The basic idea of A-tree is the introduction of virtual bounding rectangles (VBRs), which contain and approximate minimum bounding rectangles (MBRs) or data objects. xS-tree [14] is also based on R-tree but uses a lossy compression of bounding regions to guarantee a reasonable minimum fan-out within the allocated storage space for each node. These indexing methods show good performance compared SR-tree and R-tree. However, it can not be free from "curse of dimensionality" problem because they are basically based on the data partitioning approach.

This paper proposes a new indexing scheme, called *HBI* (*Hierarchical Bitmap Index*), which significantly improves the speed of filtering process. In the proposed indexing mechanism, a high dimensional object is represented as a bitmap of size $2 \cdot d \cdot l$ bits, where d is the number of dimensions of object's feature vector and l is the number of bitmaps. In the proposed bitmap, the attribute (or feature) value of an object at each dimension is approximated with two bits that indicate whether it is relatively high, low, or neither compared to the feature values of other objects, and represented as '11', '00', and '01', respectively. If the approximated values of two objects at the same dimension are '00' (low) and '11' (high), the feature values of two objects at that dimension might be very different. This dimension is called *a discriminative dimension* between two objects, and could be identified by simply XORing their bitmaps. Furthermore, the minimum distances between two objects at the discriminative dimension is easily computed by measuring the difference between two thresholds that divide the feature values into relatively high and low. This mechanism (*i.e.,* XORing two bitmaps and counting the number of '11' in the resulting bitmap) is used to filter-out the irrelevant objects in the proposed indexing mechanism, rather than the filtering based on the direct L_p-distances as in VA-File. Since the XORing is much simpler than L_p-distance calculation, the filtering process itself could be sped-up dramatically. Of course, since the two-bits approximation is too simple to represent the feature values of object at some dimensions, the filtering power of the indexing mechanism would be poor if only one bitmap is used for indexing. This problem is resolved by introducing the multiple bitmaps each of which hierarchically represents the relative positions of the feature values at a certain range. That is, the adjacent high and neither regions, or low and neither regions of parent bitmap are merged and further hierarchically divided into another three regions and indexed with one or two additional child bitmaps. If l bitmaps are used for approximating the feature vector of the object, only l XOR operations and l summation operations are required to check whether two objects are irrelevant or not. To show the superiority of the proposed method, we implemented this algorithm and compared it with the VA-file and a native sequential search. Upon experimental results with three real image sets and three synthetic data sets with different statistical distributions, we found that there was an optimal number of bitmaps that minimized the total elapsed search time, and the proposed HBI was about 2~3 times faster than VA-file while guaranteeing to find the exact solutions.

2. HIERARCHICAL BITMAP INDEXING METHOD

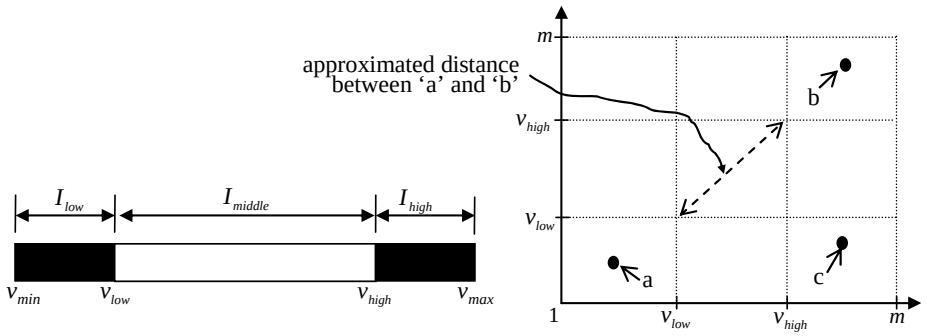
In order to speed-up the filtering process, the indexing structure should be as compact and simple as possible so that the distance calculation using approximated values could be processed efficiently, while keeping the filtering rate as high as possible. These two requirements for indexing structure, a simple representation and a high filtering rate, are usually contrary to each other, and it is very important to find an indexing structure that satisfies both of these requirements as much as possible. In the proposed indexing structure, since each attribute value of object is represented with only a set of two bits and the distance calculation between two objects is a simple XORing operation, it is efficient enough. A hierarchical structure is also introduced to improve the filtering power of the proposed indexing structure. Let us present how to represent a high dimensional object with a set of bitmaps, and how to calculate the approximate distance between two objects using their bitmaps.

2.1 Hierarchical Interval Partitioning Tree

Consider a multimedia database Λ ($\Lambda = \{o_i | 1 \le i \le n\}$, where o_i is the ith object.) consisting of n multimedia objects. For simplicity, let the space of high dimensional objects be a d-dimensional hypercube space Ω^d under the L_p-norm, where $\Omega = [0, m]$ and m is the maximum value at that space. That is, ith object, o_i, in Ω^d is defined as, $<o_i^1, o_i^2, \ldots, o_i^j, \ldots, o_i^d>$, where o_i^j is the attribute value of o_i at jth dimension ($o_i^j \in \Omega$), and its length $\|o_i\|$ in Ω^d is $\left(\sum_{j=1}^{d} \left|o_i^j\right|^p\right)^{1/p}$. Furthermore, let $L_p(a, b)$ be the L_p-distance between the objects a and b in Ω^d and computed with $L_p(a, b) = \left(\sum_{j=1}^{d} \left|a^j - b^j\right|^p\right)^{1/p}$, where a^j and b^j are the feature values of objects a and b at jth dimension, respectively.

The main idea of proposed indexing mechanism is inspired from the characteristics of L_p-distance that the dominant components of L_p-distance are the feature values at the dimensions on which the points are the farthest apart. That is, if the difference of feature values of two objects at jth dimension, $|o_p^j - o_q^j|^p$, is large enough, the L_p-distance between o_p and o_q, $L_p(o_p, o_q)$, is heavily dependent on the distance at that dimension. It leads us to divide the space Ω into three intervals, $[v_{\min}, v_{low}]$, (v_{low}, v_{high}), and $[v_{high}, v_{\max}]$, which are denoted by I_{low}, I_{middle}, and I_{high}, respectively, as shown in Fig. 1 (a), where $v_{\min} \le v_{low} < v_{high} \le v_{\max} \in \Omega$. This partition of the space is applied to all dimensions of Ω^d with the same threshold values v_{low} and v_{high} as shown in Fig. 1 (b) that is an example of space partition when $d = 2$. These intervals are used to approximate the distance between two objects. That is, if $o_p^j \in I_{low}$ and $o_q^j \in I_{high}$, the dimension j is called *a discriminative dimension* between o_p and o_q, and the contribution of that dimension to L_p-distance between these two objects is approximated as $|v_{high} - v_{low}|^p$. By counting the number of discriminative dimensions between two objects (say C), we can estimate the L_p-distance between these two objects as $(C \cdot \left|v_{high} - v_{low}\right|^p)^{1/p}$. For example, the estimated value of L_2-distance between the objects 'a' and 'c' in Fig. 1 (b) is $(1 \cdot \left|v_{high} - v_{low}\right|^2)^{1/2}$, whereas the one between the objects 'a' and 'b' is $(2 \cdot \left|v_{high} - v_{low}\right|^2)^{1/2}$ as depicted with a dotted line.

The problem of the proposed simple bitmap indexing scheme is that if there are few dimensions that satisfy the above condition (*i.e.*, $C \approx 0$), the approximated distance would



(a) Partitioning of the sapace Ω^1 into three intervals. (b) Partitioning of the space Ω^2 into nine regions.

Fig. 1. Partitioning the hypercube space Ω^d into 3^d subspaces.

be zero and could not be used to filter-out the irrelevant objects. In order to overcome this drawback, the adjacent two intervals are merged and further hierarchically partitioned into another three intervals and encoded with a separated bitmap. Let us explain this partitioning scheme in more detail. Let o_p^j and o_q^j be the attribute values of two objects at jth dimension that we want to compute the distance, I^1 be the initial interval ($I^1 = \Omega$). Then, there are six cases that o_p^j and o_q^j could be placed in I^1 as follow;

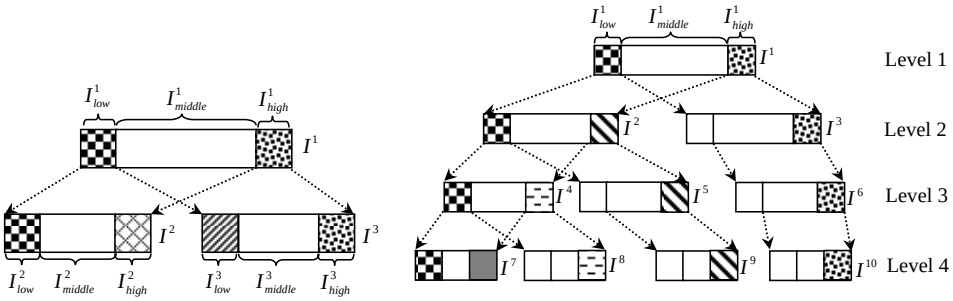
Case 1: $o_p^j \in I_{low}^1$ and $o_q^j \in I_{low}^1$ Case 2: $o_p^j \in I_{low}^1$ and $o_q^j \in I_{middle}^1$

Case 3: $o_p^j \in I_{low}^1$ and $o_q^j \in I_{high}^1$ Case 4: $o_p^j \in I_{middle}^1$ and $o_q^j \in I_{middle}^1$

Case 5: $o_p^j \in I_{middle}^1$ and $o_q^j \in I_{high}^1$ Case 6: $o_p^j \in I_{high}^1$ and $o_q^j \in I_{high}^1$

Among these cases, if o_p^j and o_q^j are in Case 3, the estimated distance at jth dimension could be computed with I^1 easily because the jth dimension is a discriminative one in I^1. In other cases, we could not compute the estimated distance with I^1 because they do not satisfy the basic condition of bitmap indexing that one is relative high and the other is relative low. In order to handle Case 1, 2, and 4, $I_{low}^1 \vee I_{middle}^1$ is hierarchically form a new interval I^2 and it is partitioned into three intervals, I_{low}^2, I_{middle}^2, and I_{high}^2. The interval $I_{middle}^1 \vee I_{high}^1$ is also partitioned into three intervals, I_{low}^3, I_{middle}^3, and I_{high}^3 to handle the Cases 5 and 6[1]. This basic interval partitioning scheme is shown in Fig. 2 (a). In this partitioning scheme, by restricting $I_{low}^1 = I_{low}^2$ and $I_{high}^1 = I_{high}^3$, the interval I^3 does not need to have the left child interval for handling $I_{low}^3 \vee I_{middle}^3$ (*i.e.*, in the case of o_p^j, $o_q^j \in I_{low}^3 \vee I_{middle}^3$) because $(I_{low}^3 \vee I_{middle}^3) \subset I^2$ so that it would be handled by I^2 or its descendants. It is true for all intervals that are the right child of an interval. In Case 4, since $I_{middle}^1 \subset I^2$ and $I_{middle}^1 \subset I^3$, one may think that $|o_p^j - o_q^j|^p$ is computed and accumulated twice by I^2 and I^3 or their descendants when o_p^j, $o_q^j \in I_{middle}^1$. However, it is always not true because o_p^j or o_q^j could not be belonging to I_{high}^3 by the restriction that $I_{high}^1 = I_{high}^3$. It means that they could not contribute to the total distance by I^3 or its descendants. On the other hand, they could be belonging to the high or low intervals in the right descendants of I^2 (*i.e.*, I^5 or I^9) so that they could eventually contribute to the total distance. An example of hierarchical interval partitioning tree more than one level with this restriction is shown in Fig. 2 (b), in which the intervals that have the same ranges are filled with the same patterns. For ex-

[1] Note that $I^1 \vee I^2$ means the interval that merges two adjacent intervals I^1 and I^2. For example, if I^1 is [1, 5] and I^2 is [5, 10], then $I^1 \vee I^2$ is [1, 10].



(a) A basic interval partitioning scheme. (b) A hierarchical interval partitioning tree.

Fig. 2. Hierarchical interval partitioning scheme.

ample, the ranges and lengths of I^1_{low}, I^2_{low}, I^4_{low} and I^7_{low} are the same.

Let us formalize this interval partitioning scheme in more detail. Let I^k be the kth interval and be divided into three intervals, I^k_{low}, I^k_{middle}, and I^k_{high}, and $level(k)$ be the level of I^k in the interval partitioning binary tree. Then, I^k has one or two child intervals as follows:

1. If $k = 1$, I^k has I^{k+1} as the left child, I^{k+2} as the right child, and $level(k) = 1$.
2. If I^k is the left child of an interval, it has $I^{k+level(k)}$ as the left child and $I^{k+level(k)+1}$ as the right child.
3. Otherwise, it has $I^{k+level(k)+1}$ as the right child (without the left child),

where $I^{k+level(k)} = I^k_{low} \vee I^k_{middle}$ and $I^{k+level(k)+1} = I^k_{middle} \vee I^k_{high}$ with the restrictions that $I^{k+level(k)}_{low} = I^k_{low}$ and $I^{k+level(k)+1}_{high} = I^k_{high}$.

The proposed interval partitioning scheme is used to approximate the distance between the attribute values of two objects. That is, if $o^j_p \in I^k_{low}$ and $o^j_q \in I^k_{high}$, then the estimated distance of the objects o_p and o_q at jth dimension in I^k is $|v^k_{high} - v^k_{low}|^p$, where v^k_{low} and v^k_{high} are the thresholds that partition the interval I^k into I^k_{low}, I^k_{middle}, and I^k_{high}. In order to be $|v^k_{high} - v^k_{low}|^p \approx |o^j_p - o^j_q|^p$, these thresholds should be selected based on the distribution of the attribute values of all objects at jth dimension. However, if we use a different set of thresholds for each dimension, we should compute the estimated distance for each dimension separately. It will require a lot of computations to estimate the L_p-distance. It forces us to use a set of thresholds that is determined by the distributions of attribute values of all objects at all dimensions, and use it to partition the intervals at all dimensions. These thresholds are selected to maximize $\sum_{o^j_p, o^j_q \in I^k} \left| v^k_{high} - v^k_{low} \right|^p$ that is the expected (estimated) distance between two attribute values in I^k of all objects at all dimensions. However it can lead to a problem when the distribution of objects is biased or when the number of objects is too small to reflect the characteristics of a feature. The proposed interval partitioning algorithm could divide partitions that over-fit the given set of objects and could not be used in dynamic database which is prone to deletion and insertion. However, we can avoid this over-fitting problem if the amount of objects is sufficient to

represent the characteristics of a feature. According to [15], the L_p-distances shows a specific distribution irrespective of the number of dimensions. It implies that we can get the characteristics of the feature through the analysis of fewer vectors in high dimensional case. Actually, we could get reasonable intervals even if a sub-group of full set is used to divide intervals in our experiments. That is why HBI shows a good perfor-mance in dynamic environments where insertion and deletion operations are frequent.

2.2 Hierarchical Bitmap Encoding Algorithm

In order to easily count the number of discriminative dimensions, the attribute value of object is basically encoded with two bits and an XOR operation is used in the proposed indexing mechanism. That is, if o_i^j belongs to I_{low}, I_{middle}, and I_{high}, it is encoded as '00', '01', and '11', respectively. By encoding the attribute values at all dimensions with this encoding scheme and concatenating them into a bit string, we can build *a bitmap of the object* that simply indicates the intervals to which the attribute values at each dimension are belonging. Then we can get the number of the discriminative dimensions from counting "11"s in the XORing result between two bitmaps. '11' in the XORing result implies that the values of two objects at the same dimension are '00' and '11'. That is, the difference of the feature values at that dimension is at least more than the size of I_{middle}. Now, let us formally present a bitmap encoding method using the hierarchical interval partitioning tree proposed in the previous section. In the proposed HBI, the bitmap of object $o_i(1 \le i \le n)$ at kth bitmap is denoted by $B^k(o_i)$ and generated as follows, where I^k_{high}, I^k_{low}, I^k_{middle} are three intervals in I^k.

$$B^k(o_i) = b^k(o_i^1)\, b^k(o_i^2) \ldots b^k(o_i^d),$$

$$\text{where } b^k(o_i^j) = \begin{cases} 00, \text{ if } o_i^j \in I^k_{low} \\ 11, \text{ if } o_i^j \in I^k_{high} \\ 01, \text{ otherwise } (o_i^j \in I^k_{middle} \text{ or } o_i^j \notin I^k) \end{cases}.$$

This bitmap index is generated for all objects $o_i(1 \le i \le n)$ in database in advance, and used to filter-out the irrelevant objects compared to the query object.

Let us show a simple example to build the bitmap index. Consider two 4-dimensional vectors, p = <1, 8, 3, 9> and q = <1, 7, 9, 3>, where Ω = [1, 10] and d = 4. Assume that I^1_{low}, I^k_{middle}, and I^1_{high} are [1, 3], (3, 9), [9, 10], respectively. Then, since the 1st attribute value of p is in I^1_{low} it is encoded as '00'. In the same way, the bit representations of 2nd, 3rd, and 4th attribute values of p are '01', '00', and '11', respectively. By concatenating these bits, we finally get the bitmap index of p at I^1, $B^1(p)$, as "00 01 00 11". Assume that l = 3, and I^2_{low} = [1, 3], I^2_{middle} = (3, 7), I^2_{high} = [7, 9), I^3_{low} = (3, 6], I^3_{middle} = (6, 9), and I^3_{high} = [9, 10]. Then, the bitmap indexes for p and q are shown in Table 1. Note that since the 1st attribute values of p and q when k = 3 are not in I^3 (*i.e.* p^1, $q^1 \notin I^3$), it is encoded as '01' in $B^3(p)$ and $B^3(q)$. It means that these feature values are meaningless in $B^3(p)$ and $B^3(q)$ and would be handled in other bitmaps.

Table 1. An example of generating bitmap index.

k	I^k_{low}	I^k_{middle}	I^k_{high}	$B^k(p)$	$B^k(q)$
1	[1, 3]	(3, 9)	[9, 10]	00 01 00 11	00 01 11 00
2	[1, 3]	(3, 7)	[7, 9)	00 11 00 01	00 11 01 00
3	(3, 6]	(6, 9)	[9, 10]	01 01 01 11	01 01 11 01

Table 2. XORing results and relationships between actual and approximated distances.

	$b^k(o^j_p)$	$b^k(o^j_q)$	$b^k(o^j_p)$ XOR $b^k(o^j_q)$	Relationships between $L_p(o_p, o_q)$ and $D_p(o_p, o_q)$
Case 1	00	11	11	$\left\|o^j_p - o^j_q\right\| \geq \left\|I^k_{middle}\right\|$
Case 2	11	00		
Case 3	00	00	00	$\left\|o^j_p - o^j_q\right\| \leq \max\left(\left\|I^k_{high}\right\|, \left\|I^k_{middle}\right\|, \left\|I^k_{low}\right\|\right)$
Case 4	01	01		
Case 5	11	11		
Case 6	00	01	01	$\left\|o^j_p - o^j_q\right\| \leq$ $\max\left(\left\|I^k_{high}\right\| + \left\|I^k_{middle}\right\|, \left\|I^k_{low}\right\| + \left\|I^k_{middle}\right\|\right)$
Case 7	01	00		
Case 8	11	01		
Case 9	01	11		

2.3 Calculating Approximated Distance with Bitmaps

In this section, we show how to approximate the L_p-distance with the proposed bitmap representation. Let $L_p(o_p, o_q)$ be the L_p-distance between two objects o_p and o_q, and $D_p(o_p, o_q)$ be the approximated distance computed with the proposed bitmap index. By XORing the bit representations of o_p and o_q at jth dimension in kth bitmap, $b^k(o^j_p)$ and $b^k(o^j_q)$, we can get the results shown in Table 2, where $\|A\|$ denotes the length of interval A. Among the cases in Table 2, since the lower bound of difference between $L_p(o_p, o_q)$ and $D_p(o_p, o_q)$ at jth dimension in the kth bitmap could be computed in Cases 1 and 2, only these two cases are considered in the proposed distance calculation scheme. Furthermore, because the same interval I^k is used to encode the attribute values of objects at all dimensions (*i.e.*, the lengths of I^k_{middle}, $\|I^k_{middle}\|$ are the same in all dimensions) at the kth bitmap, we can approximate the distance between two objects o_p and o_q, $L_p(o_p, o_q)$, as follows;

$$D_p(o_p, o_q) = \left[\sum_{k=1}^{l} C_k \times \left\|I^k_{middle}\right\|^p\right]^{1/p}, \tag{3}$$

where l is the number of bitmaps and C_k is the number of discriminative dimensions between objects o_p and o_q at the kth bitmap.

3. SIMILARITY SEARCH USING HIERARCHICAL BITMAP INDEX

The similarity search problems in the multimedia retrievals could be classified into

two classes; one is *r*-range search that finds the multimedia objects whose distance to the query object is less than *r*, and the other is *k*-NN (Nearest Neighbor) search that finds *k* objects with the smallest distance to the query object. Both of these search problems could be solved with the proposed bitmap indexing structure efficiently by constructing the bitmaps of multimedia objects in database in advance. These indexes are used to filter-out the irrelevant objects compared to the query object in both of two search problems. For the *r*-range search, the bitmap for the query object (Q) is first generated and it is used to filter out the irrelevant objects whose D_p-distance to query object is larger than *r*. They are collected to form a candidate set. Then, the objects among the candidate set whose L_p-distance are actually less than *r* are selected as the final results of the *r*-range search.

In the case of *k*-NN search problem, the candidate set could not be completely generated in the filtering process because it should select the objects whose distances to query object are *relatively* small. It forces us to keep a set of potential nearest objects, and the L_p-distance of an object is calculated only when its D_p-distance is less than the largest L_p-distance among the distances of objects in this set. If its L_p-distance is actually less than the currently largest one, it is inserted into the set and the object whose L_p-distance is the largest among the objects in the set is deleted. Of course, if the number of objects in the set is currently less than m(the size of the final results), it is inserted regardless of its D_p- or L_p-distances. The procedure to find the object with the largest distance could be implemented easily if the potential nearest objects set is kept as an ordered list. Note that as $D_p(q, o_i) \le L_p(q, o_i)$, $\forall i(1 \le i \le n)$, the objects whose L_p-distances are less than *r* or less than the largest one among the set are never filtered-out by these filtering processes. It means that these two search algorithms based on the proposed bitmap indexing guarantee to find the exact solutions. On the other hand, the L_p-distances of the objects whose D_p-distance are larger than *r* or larger than the largest one among the set are never computed in the filtering process, and it is the main source of the efficiency of HBI as shown in the experimental results in section 5.

4. ANALYSIS

4.1 Storage Overheads for Indexing Structure

The total storages requirements for storing the multimedia objects themselves are $n \cdot d \cdot v$ bytes where n is the number of multimedia objects in database, d is the number of dimensions of the object's feature vector, v is the size of storages for storing one feature value. In the case of VA-File, the storage requirements for indexing structure are $n \cdot \left\lceil \frac{d \cdot b}{8} \right\rceil$ bytes where b is the average number of bits used to divide each dimension into several regions. In order for indexing this database using HBI, $n \cdot \left\lceil \frac{d \cdot 2}{8} \right\rceil \cdot l$ bytes are required where l is the number of bitmaps used for the indexing. For example, if there are 100,000 multimedia objects each of which is represented with MPEG-7 Color Structure descriptor [9] whose number of dimensions is 256, and each bin value of histogram is represented with 4-bytes integer, the total storages for this multimedia database itself are $100{,}000 \cdot 256 \cdot 4 = 100$ Megabytes. In the case of indexing with VA-File, $100{,}000 \cdot \left\lceil \frac{256 \cdot 4}{8} \right\rceil = 12.8$ Mega bytes are required to index the database when $b = 4$. On the other hand, in the case of indexing with HBI, $100{,}000 \cdot \left\lceil \frac{256 \cdot 2}{8} \right\rceil \cdot 3 = 19.2$ Mega bytes are required when $l = 3$. As

shown in this example, although the storage requirement of indexing with HBI is larger than that with VA-File because more bits are usually required for each dimension in HBI (*i.e.*, usually, $b \leq 2 \cdot l$) in order to get a comparable filtering rate as VA-File, it helps to reduce the CPU time of the filter process very much as shown in the experiments in section 5.

4.2 Analysis of Total Search Time

Let us now quantitatively compare and analyze the total time of similarity search in multimedia database using the sequential search (or BFS (Brute-Force Search)), the search with VA-File, and the search with the proposed HBI. The total elapsed time for similarity search with BFS, T_{BFS}, is estimated as follows because it should read the feature vectors of all objects and compute their L_p-distances to query object;

$$T_{BFS} \approx n \cdot (R_{vec} + C_L)$$

where R_{vec} is the time to read the feature vector of an object, and C_L is the time to compute the L_p-distance to the query object. On the other hand, the total time for similarity searches with VA-File, T_{VA}, and with HBI, T_{HBI}, would be estimated as follows because they first filter-out the irrelevant objects and compute the L_p-distances of the remaining objects only;

$$T_{VA} \approx n \cdot (R_{VA} + C_L) + (1 - \mu_{VA})\, n \cdot (R_{vec} + C_L)$$
$$T_{HBI} \approx n \cdot (R_{HBI} + C_D) + (1 - \mu_{HBI}) \cdot n \cdot (R_{vec} + C_L)$$

where R_{VA} and R_{HBI} are the time to read the index for an object in VA-File and HBI, respectively, C_D is the time to compute the D_p-distance of the object to the query object, and μ_{VA} and μ_{HBI} are the filtering rates of the VA-File and HBI, respectively. If we assume that the filtering rates of VA-File and HBI are similar to each other and they are greater than 0.95 (*i.e.*, $\mu_{VA} \approx \mu_{HBI} \geq 0.95$), the dominant component of the search time is the filtering time. If we further assume that the time for distance calculations with indexes are large enough than the time for index readings (*i.e.*, C_L, $C_D >> R_{VA}$, R_{HBI}), the time for similarity search with VA-File and HBI would be $n \cdot C_L$ and $n \cdot C_D$, respectively. It means that the time for similarity search with HBI could be $\frac{C_L}{C_D}$ times faster than the one with VA-File. Actually, under our experimental environment (Microsoft Windows XP on an Intel Pentium 4 (3.0GHz) with 1GB main memory), because C_L and C_D are approximately 0.009 milli-seconds and 0.003 milli-seconds, respectively, when the type of feature value is the floating-point number and the number of bitmaps used in HBI is 10 (*i.e.*, $l = 10$), the expected speed-up would be about $\frac{C_L}{C_D} = \frac{0.009}{0.003} \approx 3.0$. This quantitative analysis has been verified by several experiments as shown in the following section.

5. EXPERIMENTS

The main idea of the proposed HBI is to encode the feature values of the object with a set of two bits that hierarchically indicates their relative positions at each dimension. It

Table 3. Multimedia data sets used in the experiments[2].

Data Set	Feature Vector	Data Distribution	Number of Dimensions (l)	Feature Value Type & Range (Ω)	Number of Objects (n)
$R1$	MPEG-7 Color Structure Descriptor [9]	Extracted from Real Image Sets	256	Integer, [0, 255]	25,160
$R2$	MPEG-7 Edge Histogram Descriptor [9]		80	Integer, [0, 7]	25,160
$R3$	HSV Color Histogram		32	Float, [0, 1]	68,040
$S1$	Synthetically Generated [10]	Uniformly	256	Float, [0, 255]	100,000
$S2$		Skewed			
$S3$		Clustered			

causes that its performance would be highly dependent on the number of bitmaps used for indexing and the distributions of the feature vectors in the search space. In order to show the effects of the data distributions to the search performances, we have experimented with three real data sets ($R1$, $R2$, $R3$) and three synthetic data sets ($S1$, $S2$, $S3$) whose parameters are summarized in Table 3. We performed the r-range similarity search under Microsoft Windows XP on an Intel Pentium 4 (3.0GHz) with 1GB main memory.

5.1 Number of Bitmaps and Filtering Rates

In the filtering approach for similarity search in multimedia database, the irrelevant objects are first filtered-out using some compact approximations of the feature vectors of multimedia objects. The advantage of this filtering approach is shown in Fig. 3 in which the filtering-rates of the R-Tree, VA-File, and HBI for a uniformly distributed data set are presented as a function of the number of dimensions of feature vectors. As shown in this example, the filtering rate of the R-Tree indexing scheme is dramatically decreased when the number of dimensions is larger than 10, whereas the filtering rates of VA-File and HBI are not decreased although the number of dimensions is being increased. This experiment shows that the filtering approach including VA-File and HBI could overcome the "curse of dimensionality".

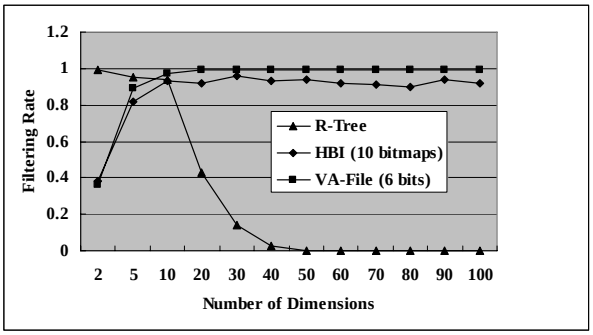
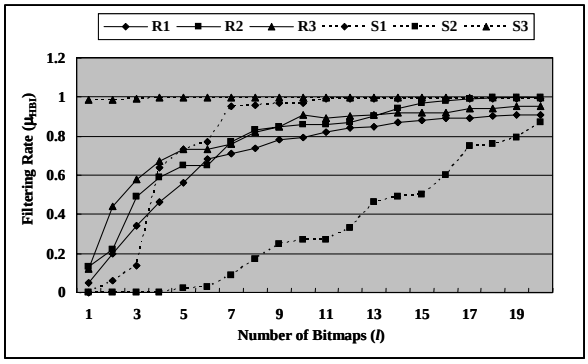
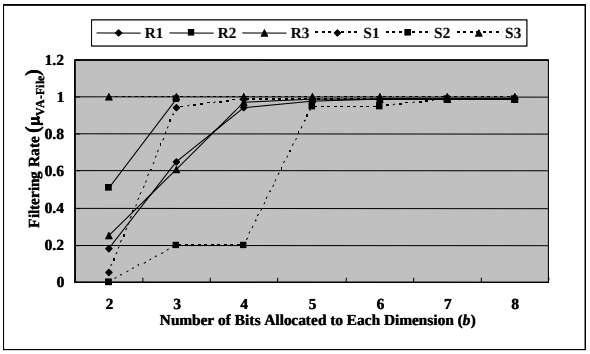


Fig. 3. Examples of filtering rates of R-tree, VA-file, and HBI.

[2] $R1$ and $R2$ are extracted from http://clib.cs.berkeley.edu/photos/use.html using MPEG-7 XM. $R3$ is downloaded at (http://kdd.ics.uci.edu/databases/CorelFeatures/CorelFeatures.data.html). $S1$~$S3$ are synthetic data sets generated as in [10].



(a) Filtering rates of HBI.



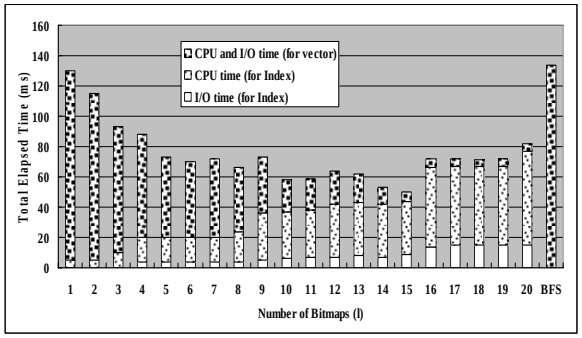
(b) Filtering rates of VA-file.

Fig. 4. Comparisons of filtering rates of VA-file and HBI.

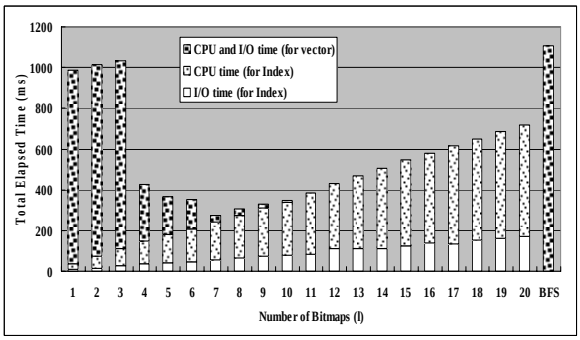
Now, let us show some experiments on the filtering rates of VA-File and HBI with the data sets shown in Table 3. As shown in Fig. 4 (a), the filtering rates of HBI are increased as the number of bitmaps used for indexing is being increased for the real image data sets (*R*1, *R*2, *R*3) and *S*1. It is the same in VA-File when the number of bits allocated to each dimension is increased as shown in Fig. 4 (b). In the case of *S*2, in which the feature values are skewed to some ranges and these ranges are different to each other at each dimension, the filtering rate of HBI is not so good although more than 10 bitmaps are used because it is very hard to determine a good set of thresholds that could partition the intervals at all dimensions effectively. On the other hand, if the feature values are clustered within small ranges as in *S*3, a high filtering rate could be achieved with HBI although a small number of bitmaps is used for indexing.

5.2 Experimental Analyses and Comparison of Search Time

The total search time with HBI (and also VA-File) consists of the I/O time for index reading, the CPU time for approximated distance calculations with index, and I/O and CPU time for actual distance calculations with real feature vectors of relevant objects that are not filtered-out. In order to minimize I/O and CPU time for actual distance calculations, the number of relevant objects should be minimized by a higher filtering rate. How-



(a) Total search times for *R*3.



(b) Total search tomes for *SI*.

Fig. 5. Experimental search time of HBI with respect to the number of bitmaps.

ever, to get a higher filtering rate, more bitmaps should be used and it causes to increase the I/O and CPU time for index. It means that there is a trade-off between time for indexing and time for L_p-distance calculations in the search with HBI indexing.

Fig. 5 shows the elapsed total search time of HBI for *R*3 and *S*1 while increasing the number of bitmaps for indexing. In the case of *R*3 (Fig. 5 (a)), the total search time is minimized when 15 bitmaps are used. On the other hand, in the case of *S*1 (Fig. 5 (b)), the filtering rate is almost 1.0 when 11 bitmaps are used because the feature vectors are clustered, and the total search time is minimized when 7 bitmaps are used. For these data sets, the search times are 2.5~4.0 times faster than BFS when an optimal number of bitmaps are used as shown in Fig. 5. From these experiments, we find that there is an optimal number of bitmaps that minimizes the total search time. Although the optimal number of bitmaps would be dependent on the distributions of feature vectors in the search space, we find that, from several experiments with the data sets with different feature vector distributions, a good performance would be usually achieved when about 10 bitmaps are used for indexing.

Now, let us experimentally compare the search time of HBI, VA-File, and BFS. Fig. 6 shows the time for indexing and the time for L_p-distance calculations with the feature vectors of relevant objects when HBI, VA-File, and BFS are used for similarity searches for data sets in Table 3[3]. As shown in this experiment, the I/O and CPU time for L_p-distance calculations for feature vectors of relevant objects with HBI are larger thanthe ones

[3] Note that the numbers of bitmaps used in the experiments are 6 for *R*1, 15 for *R*2, 15 for *R*3, 7 for *S*1, 20 for *S*2, and 1 for *S*3. In the experiments with VA-File, 6-bits are allocated to each dimension, and it produces the filtering rates of more than 0.95 for the all data sets. They are selected to produce the best performances.

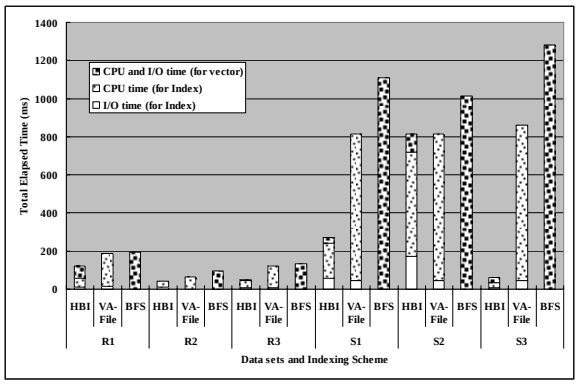


Fig. 6. Experimental comparisons of the search time with BFS, VA-file and HBI.

with VA-File because of its lower filtering rate. However, since the time for filtering (especially, the CPU time for D_p-distance calculations) with HBI is much less than the ones with VA-File, the total search time with HBI is 2.0~3.0 times faster than the ones with VA-File, and 2.5~4.0 times faster than the ones with BFS, in the case of experiments with *R*1, *R*2, *R*3, and *S*1. This result verifies the quantitative speed-up analysis of HBI presented in section 4.2. In the experiment with *S*2, since a lot of bitmaps are required to get a filtering rate of 0.9 (actually, 20 bitmaps are used in this experiment) and it results a lot of I/O and CPU time for indexing, the total search time with HBI is similar to the one with VA-File. On the other hand, in the experiment with *S*3, since only one bitmap is enough to get a filtering rate of 0.99, the total search time with HBI is about 25 times faster than the one with VA-File. From these experiments, we find out that the similarity search with HBI is averagely 2.0~3.0 times faster than the one with VA-File, and could produce the best performance in the clustered data set as *S*3, and the comparable performance in the skewed data set as *S*2.

6. CONCLUDING REMARKS

This paper proposed an efficient indexing scheme, called HBI (Hierarchical Bitmap Indexing), in which the feature values of an object are represented with a set of two bits that hierarchically indicates whether it is relatively high, low, or neither compared to the feature values of other objects. In the view of storage efficiency, it requires only about 6-14 bits to represent a feature value of objects on average. In the view of computational efficiency, many irrelevant objects could be quickly filtered-out by the approximated distance calculated by a simple XOR operation. Upon experimental results, we found out that the total search time based on HBI was averagely 2~3 times faster than that of VA-File, and its performance was best when the distribution of feature vectors was highly clustered, and was comparable to the VA-File when the feature vectors are highly skewed at each dimension differently. Furthermore, we also show an experiment that the proposed HBI scheme could be used for indexing of multimedia database with a lot of multimedia objects. The proposed indexing mechanism could be used to build an efficient CBMR system that guarantees a quick response time.

REFERENCES

1. A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1984, pp. 47-57.
2. N. Beckmann and H. Kriegel, "The R*-tree: An efficient and robust access method for points and rectangles," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1990, pp. 322-331.
3. S. Berchtold, D. Keim, and H. Kriegel, "The X-tree: An index structure for high dimensional data," in *Proceedings of International Conference on Very Large Data Bases*, 1996, pp. 28-39.
4. P. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," in *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1993, pp. 311-321.
5. P. Ciaccia, M. Patella, and P. Zezula, "M-tree: An efficient access method for similarity search in metric space," in *Proceedings of International Conference on Very Large Data Bases*, 1997, pp. 426-435.
6. S. Berchtold, C. Bohm, and H. Kriegel, "The pyramid technique: Towards breaking the course of dimensionality," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1998, pp. 142-153.
7. R. Weber, H. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high dimensional spaces," in *Proceedings of International Conference on Very Large Data Bases*, 1998, pp. 194-205.
8. G. Cha, X. Zhu, D. Petkovic, and C. Chung, "An efficient indexing methods for nearest neighbor search in high dimensional image databases," *IEEE Transactions on Multimedia*, Vol. 4, 2002, pp. 76-87.
9. ISO/IEC JTC1/SC29/WG11, *Information Technology Multimedia Content Description Interface-Part3: Visual*, 2001.
10. T. Bozkaya and M. Ozsoyoglu, "Distance based indexing for high dimensional metric spaces," in *Proceedings of ACM SIGMOD Conference on Management of Data*, 1997, pp. 357-368.
11. C. Faloutsos, "Signature based text retrieval methods," *Data Engineering*, Vol. 13, 1990, pp. 25-32.
12. J. S. Yoo, Y. J. Lee, J. W. Chang, and M. H. Kim, "The HS file: A new dynamic signature file method for efficient information retrieval," *Lecture Notes in Computer Science*, Vol. 856, 1994, pp. 571-580.
13. Y. Sakurai, M. Yoshikawa, S. Uemura, and H. Kojima, "The A-tree: An index structure for high-dimensional spaces using relative approximation," in *Proceedings of International Conference on Very Large Data Bases*, 2000, pp. 516-526.
14. C. Wang and X. Wang, "Indexing very high dimensional sparse and quasi-sparse vectors for similarity searches," *Very Large Database Journal*, Vol. 9, 2001, pp. 344-361.
15. G. Burghouts, A. Smeulders, and J. Geusebroek, "The distribution family of similarity distances," in *Proceedings of Neural Information Processing Systems*, 2007, pp. 201-208.

Jongho Nang (浪鍾鎬) received his Ph.D. and M.S. degrees in Computer Science from Korea Advanced Institute of Science and Technology (KAIST), Daejon, Korea, in 1992 and 1988, respectively, and his B.S. degree in Computer Science from Sogang University, Seoul, Korea, in 1986. He has been a Professor of Department of Computer Science and Engineering, Sogang University, Seoul, Korea, since 1993. His research interests include multimedia system, parallel processing, and internet technology.

Joohyoun Park (朴柱炫) received his B.S. and M.S. degrees in Computer Science and engineering from Sogang University, Seoul, Korea, in 1999 and 2002 respectively. He has joined the Software Lab of Visual Display Division at Samsung Electronics as a Senior Engineer after completing Ph.D. in Sogang University, Seoul, Korea, in 2007. His research interests include multimedia contents management and retrieval.

Jihoon Yang (楊枝勳) is an Associate Professor of Department of Computer Science and Engineering at Sogang University. His research interests include machine learning, data mining and knowledge discovery, artificial intelligence, pattern recognition, and bioinformatics. Dr. Yang holds a B.S. in Computer Science from Sogang University, and M.S. and Ph.D. degrees in Computer Science from Iowa State University.

Saejoon Kim (金世埈) received the B.S. degree from Columbia University, New York, in 1994, and the M.S. and Ph.D. degrees from Cornell University, Ithaca, in 1996 and 1998, respectively. He is presently an Assistant Professor in the Department of Computer Science and Engineering at Sogang University, Seoul, Korea. In 2003, he authored the book “Fundamentals of Codes, Graphs, and Iterative Decoding,” Springer, Boston, M.A. (w/S.B. Wicker). His research interests include coding theory and machine learning.