An Efficient Indexing Structure for Content Based Multimedia Retrieval with Relevance Feedback*

Jongho Nang and Joohyoun Park
Dept. of Computer Science and Engineering
Sogang University, 1 Shinsu-Dong, Mapo-Ku, Seoul 121-742, Korea
(jhnang, parkjh)@sogang.ac.kr

ABSTRACT

This paper proposes an efficient indexing structure for CBMR (Content-Based Multimedia Retrieval), called *HBI (Hierarchical Bitmap Index*), in which each 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 this bitmap representation, 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. Using these compact representations of feature vectors, a lot of irrelevant objects could be quickly filtered-out by a couple of simple XOR operations, and it helps to reduce the filtering process of similarity search in high-dimensional data space. It also presents an optimization algorithm, called *FQD (Filtering by Query Difference)*, for the similarity search with relevance feedback that reuses the previously calculated distances between the original query and all objects in the database when filtering the irrelevant objects in the successive search with modified query. It helps to further reduce the search time of CBMR with relevance feedback. Experimental results show that the similarity search using HBI is about $2 \sim 3$ times faster than VA-File while guaranteeing the exact solutions, and FQD for relevance feedback helps to further reduce the elapsed time of successive similarity search compared to the one for the first search.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing-Indexing methods

General Terms

Algorithms, Performance, Experimentation

*This work was supported by the Brain Korea 21 Project in 2006.

SAC'07 March 11-15, 2007, Seoul, Korea

Keywords

High Dimensional Indexing, Content Based Multimedia Retrieval, Hierarchical Bitmap Indexing, Relevance Feedback

1. INTRODUCTION

The similarity in CBMR 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 studied. 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. Although this approach could eliminate the I/O times 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. Furthermore, in the case of multimedia search with relevance feedback in which the user modify and re-submit the query using the previous search results, this process should be repeated iteratively until to get the desired results.

This paper proposes a new indexing scheme, called HBI, 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',

or '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 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. This paper also presents an optimized filtering process, called FQD (Filtering by Query Difference), for CBMR with relevance feedback. In this optimized filtering process, the lower bounds of the new distances between the objects and the revised query are calculated with by simply subtracting the distance between original and revised query from the distances calculated in the previous search. This paper shows how to reuse the distances calculated in the filtering process of the previous search to filter-out the irrelevant objects in the successive multimedia object searches.

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 times, and the proposed HBI was about $2 \sim 3$ times faster than VA-file while guaranteeing to find the exact solutions. We also experimentally show that, using the optimized search algorithm for the similarity search with relevance feedback, the 2^{nd} and later search could be $2 \sim 2.5$ times faster than the 1^{st} search.

2. HIERARCHICAL BITMAP INDEXING

Let us first 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 $\Lambda(\Lambda = \{o_i | 1 \leq i \leq n\}$, where o_i is the i^{th}object.) consisting of n multimedia objects. For simplicity, let the space of high dimensional objects be a d-dimensional discrete hypercube space Ω^d under the L_p-norm, where $\Omega = [1, m]$ and m is the maximum value at that space. That is, i^{th} object, o_i, in Ω^d is defined as $< o_i^1, o_i^2, ..., o_i^j, ..., o_i^d >$, where o_i^j is the attribute value of o_i at j^{th} dimension ($o_i^j \in \Omega$), and its length $\|o_i\|$ in Ω^d is $(\sum_{j=1}^{d} |o_i^j|^p)^{1/p}$. Furthermore, let $L_p(a, b)$ be the L_p-distance between the objects a and b in Ω^d and compute with $L_p(a, b) = (\sum_{j=1}^{d} |a^j - b^j|^p)^{1/p}$, where a^j and b^j are the feature values of objects a and b at j^{th} 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 j^{th} 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

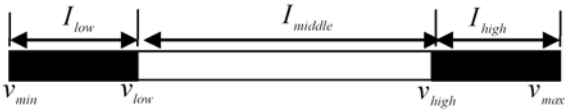

Figure 1: Partitioning of the Space Ω^1into Three Intervals

Table 1: Six Cases that Two Objects can be placed in Interval

Cases	Conditions
Case1	$o_p^j \in I_{low}^1$ and $o_q^j \in I_{low}^1$
Case2	$o_p^j \in I_{low}^1$ and $o_q^j \in I_{middle}^1$
Case3	$o_p^j \in I_{low}^1$ and $o_q^j \in I_{high}^1$
Case4	$o_p^j \in I_{middle}^1$ and $o_q^j \in I_{middle}^1$
Case5	$o_p^j \in I_{middle}^1$ and $o_q^j \in I_{high}^1$
Case6	$o_p^j \in I_{high}^1$ and $o_q^j \in I_{high}^1$

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 **Figure 1**, where $v_{min} \leq v_{low} < v_{high} \leq v_{max} \in \Omega$. 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 **Figure 1** 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 distance at that dimension 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 $(C \cdot |v_{high} - v_{low}|^p)^{1/p}$.

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 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 j^{th} 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 places in I^1 as shown in **Table 1**;

Among these cases, if o_p^j and o_q^j are in **Case3**, the estimated distance at dimension j could be computed with I^1 easily because the j^{th} 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 **Case1**, **Case2**, and **Case4**, $I_{low}^1 \vee I_{middle}^1$ 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 **Case 5** and **Case 6**[1]. This basic interval partitioning scheme is shown in **Figure 2**. In this partitioning scheme, by restricting $I_{low}^1 = I_{low}^2$

[1]Note that $I^1 \vee I^2$ means he 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 $[1, 10]$

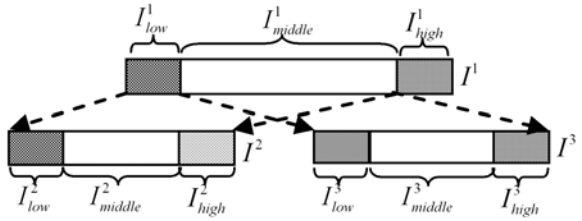

Figure 2: Hierarchical Interval Partitioning Scheme

and $I^1_{high} = I^3_{high}$, the interval I^3 does not need to have the left child interval for handling $I^3_{low} \vee I^3_{middle}$ (*i.e.*, in the case of $o^j_p, o^j_q \in I^3_{low} \vee I^3_{middle}$) because $(I^3_{low} \vee I^3_{middle}) \subset I^2$ 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 $I^1_{middle} \subset I^2$ and $I^1_{middle} \subset I^3$, one may think that $|o^j_p - o^j_q|^p$ is computed and accumulated twice by I^2 and I^3 or their descendants when $o^j_p, o^j_q \in I^1_{middle}$. However, it is always not true because o^j_p and o^j_q could not be belonging to I^3_{high} by the restriction that $I^1_{high} = I^3_{high}$. 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.

Let us formalize this interval partitioning scheme in more detail. Let I^k_{low} be the k^{th} 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 $I^{k+level(k)+1}_{high} = I^k_{high}$.

This 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 j^{th} dimension in I^k is $|v^k_{high} - v^k_{low}|^p$, where v^k_{high} and v^k_{low} 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 j^{th} 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 $\sum_{o^j_p, o^j_q \in I^k} |v^k_{high} - v^k_{low}|^p$ that is the expected (estimated) distance between two attribute values in I^k of all objects at all dimensions. **Figure 3** shows an algorithm to create a hierarchical interval partitioning tree that is used to encode the feature values of all objects at all dimensions, in which **NumOfOccurrences(I)** is a function that returns the number of occurrences of feature values of all objects at all dimensions that are belonging to the interval I. It recursively finds two set of intervals for left child and right child intervals that maximize $\sum_{o^j_p, o^j_q \in I^k} |v^k_{high} - v^k_{low}|^p$ if I^k is the left child of an interval, and find a set of intervals for right child interval if I^k is the right child of an interval.

```
//k:the interval identifier
//l:the number of bitmaps to be generated
//I^k_low, I^k_middle, and I^k_high:three intervals in I^k (1 ≤ k ≤ l)
//ChildFlag:a flag that indicates whether it is the left
child or right child of an interval
//level(k):the level of k^th interval

Procedure CIT(I^k,v,ChildFlag)//CreateIntervalTree
Begin Procedure
E_max = 0;
if k > l then
  return E_max;
end if
if ChildFlag == LeftChild then
  v^k_low = v;
  for all v^k_high such that v^k_low < v^k_high ≤ v^k_max do
    E = |v^k_high − v^k_low|^p ·NumofOccurrence([v^k_min, v^k_low])
    ·NumofOccurrence([v^k_high, v^k_max]);
    I^{k+level(k)} = [v^k_min, v^k_high);
    E=E+CIT(I^{k+level(k)}, v^k_low, LeftChild);
    I^{k+level(k)+1} = (v^k_low, v^k_max];
    E=E+CIT(I^{k+level(k)+1}, v^k_high, RightChild);
    if E > E_max then
      E_max = E; I^k_low = [v^k_min, v^k_low];
      I^k_middle = (v^k_low, v^k_high); I^k_high = [v^k_high, v^k_max];
    end if
  end for
else
  v^k_high = v;//inherits the high interval of parent
  for all v^k_low such that v^k_min < v^k_low ≤ v^k_high do
    E = |v^k_high − v^k_low|^p ·NumofOccurrence([v^k_min, v^k_low])
    ·NumofOccurrence([v^k_high, v^k_max]);
    I^{k+level(k)+1} = (v^k_low, v^k_max];
    E=E+CIT(I^{k+level(k)+1}, v^k_high, RightChild);
    if E > E_max then
      E_max = E; I^k_low = [v^k_min, v^k_low];
      I^k_middle = (v^k_low, v^k_high); I^k_high = [v^k_high, v^k_max];
    end if
  end for
end if
return E_max; //E_max is the maximum expected distance
End Procedure
```

Figure 3: Algorithm to Build the Hierarchical Interval Partitioning Tree

2.2 Hierarchical Bitmap Encoding Algorithm

In order to easily count the number of discriminative dimensions, the attribute value of object is basically encoded

Table 2: XORing Results and Relationships between Actual and Approximated Distances.

	$b^k(o_p^j)$	$b^k(o_q^j)$	XOR	Relationships
C1	00	11	11	$\|o_p^j - o_q^j\| \geq \|I_{middle}^k\|$
C2	11	00		
C3	00	00	00	$\|o_p^j - o_q^j\| \leq \max\{\|I_{high}^k\|, \|I_{middle}^k\|, \|I_{low}^k\|\}$
C4	01	01		
C5	11	11		
C6	00	01	01 or 10	$\|o_p^j - o_q^j\| \leq \max\{\|I_{high}^k\| + \|I_{middle}^k\|, \|I_{low}^k\| + \|I_{middle}^k\|\}$
C7	01	00		
C8	11	01		
C9	01	11		

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. 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 $o_i (1 \leq i \leq n)$ at k^{th} bitmap is denoted by $B^k(o_i)$ and generated as follows, where I_{low}^k, I_{middle}^k, and I_{high}^k are three intervals in I^k.

$$B^k(o_i) = b^k(o_i^1)...b^k(o_i^j)...b^k(o_i^d)$$

$$\text{,where } b^k(o_i^j) = \begin{cases} 00 & \text{if } o_i^j \in I_{low}^k \\ 11 & \text{if } o_i^j \in I_{high}^k \\ 01 & \text{otherwise } o_i^j \in I_{middle}^k \text{ or } o_i^j \notin I^k \end{cases} \quad (1)$$

This bitmap index is generated for all objects $o_i (1 \leq i \leq n)$ in database in advance, and used to filter-out the irrelevant objects compared to the query object. Note that the total size of indexes encoded with HBI for database with n objects is $n \cdot 2 \cdot d \cdot l$ bits, where d is the number of dimensions of object's feature vector and l is the number of bitmaps. **Figure 4** shows an algorithm to make the bitmap indexes for all objects in the multimedia database. It first constructs a hierarchical interval partitioning tree with the algorithm in **Figure 3**, and then makes the bitmaps for all objects in database. Note that the same intervals in I^k (*i.e.*, $I_{low}^k, I_{middle}^k, I_{high}^k$) are used to build the k^{th} bitmaps of all objects.

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 $B_p(o_p, o_q)$ be the approximated distance computed with the proposed bitmap index. By XORing the bit representations of o_p and o_q at j^{th} dimension in k^{th} bitmap, $b^k(o_p^j)$ and $b^k(o_q^j)$, we can get the 9 cases as shown in **Table 2**, where $\|A\|$denotes the length of interval A;

Among these cases, since the lower bound of difference between $L_p(o_p, o_q)$ and $B_p(o_p, o_q)$ and at j^{th} dimension in the k^{th} bitmap could be computed in **C1** and **C2**, 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_{middle}^k, $\|I_{middle}^k\|$, are the same in all dimensions) at the k^{th} bitmap, we can approximate the distance between two objects o_p and o_q , $B_p(o_p, o_q)$, as follows;

$$B_p(o_p, o_q) = [\sum_{k=1}^{l} C_k \times \|I_{middle}^k\|^p]^{1/p} \quad (2)$$

Note that l is the number of bitmaps and C_k is the number of discriminative dimensions between objects o_p and o_q at the k^{th} bitmap.

THEOREM 1. *The approximated distance between o_p and o_q, $B_p(o_p, o_q)$, is always less than or equal to $L_p(o_p, o_q)$. That is $B_p(o_p, o_q) \leq L_p(o_p, o_q)$.*

PROOF. It is trivial from **Table 2** because only **C1** and **C2** are considered in the distance approximation of o_p and o_q at each dimension in all bitmaps. □

This theorem implies that if we filter-out the irrelevant objects with the distance calculated with **Equation 2**, the objects that are not filtered-out by L_p-distance would not be filtered-out by B_p-distance also. This property guarantees that the recall of similarity search with the proposed HBI would be 1.

```
//o_i (1 ≤ i ≤ n):a multimedia object
//k:the bitmap identifier
//I^k_low, I^k_middle, and I^k_high:three intervals in I^k (1 ≤ k ≤ l)
//B^k(o_i):the k^th bitmap of o_i

Procedure MakeBitmap(o_i, I^k_low, I^k_middle, I^k_high)
Begin Procedure
for all j such that 1 ≤ j ≤ d do
  if o_i^j ∈ I^k_low then
    b^k(o_i^j) =' 00';
  else if o_i^j ∈ I^k_high then
    b^k(o_i^j) =' 11';
  else
    b^k(o_i^j) =' 01';
  end if
end for
tmpBM = b^k(o_i^1)b^k(o_i^2)...b^k(o_i^d);
return tmpBM;
End Procedure

Procedure CreateHBI()
Begin Procedure
CIT(Ω,α,LeftChild);
// Ω = I^1 and α is its initial left interval
for all o_i such that 1 ≤ i ≤ n do
  for all I^k such that 1 ≤ k ≤ l do
    B^k(o_i)=MakeBitmap(o_i,I^k_low,I^k_middle,I^k_high);
  end for
end for
End Procedure
```

Figure 4: An Algorithm to Build the Bitmap Index

3. SIMILARITY SEARCH USING HBI

Let us show a simple similarity search algorithm with HBI structure, and extends it to the one with relevance feedback.

3.1 A Simple Multimedia Search with HBI

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. For the r-range search as shown in **Figure 5**, the bitmap for the query object (Q) is first generated and it is used to filter out the irrelevant objects whose B_p-distance to query object is larger than r (**Filtering_r-Search()**). 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 (**Exact_r-Search()**). Note that the function **CountNumOf11(A)** that count the number of bit patters '11' in A could be implemented efficiently by a table look-up indexed by A. The k-NN search problem could be solved efficiently with HBI also by filter-out the objects whose approximated distances are larger than the objects with the largest distance.

3.2 An Optimized Filtering Algorithm for Similarity Search with Relevance Feedback

The relevance feedback in CBMR is a mechanism to allow the user to check the search results as relevant or irrelevant, and a new query (q_{i+1})is generated automatically by combining the feature vectors of original query (q_i) and relevant/irrelevant results using the following equation[9];

$$q_{i+1} = \alpha \cdot q_i + \frac{\beta}{n_1} \cdot \sum_{k=1}^{n_1} a_k - \frac{\gamma}{n_2} \cdot \sum_{k=1}^{n_2} b_k \qquad (3)$$

,where a_k and b_k are the search results that the user checked as the relevant and the irrelevant ones, respectively, and n_1 and n_2 are their numbers of checked results, and α, β, and γ are the parameters to control their contributions to the query modification. This process is usually repeated until the user satisfies with the search results. It means that the search algorithm presented in **Figure 5** is executed for each new query, and the total search time would be $I \cdot T$ where I is the number of required iterations and T is the average search time for the query $q_i (1 \le i \le I)$. However, since the new query is a modified version of the previous query and the distances between the previous query and all objects are calculated in the previous searching process, the search with the new query could be executed efficiently if we reused the previously calculated distances. It is called *FQD* because the filtering of irrelevant objects in 2nd and later search is based on the approximated distance calculated with the distance between two successive queries. Let us explain this optimized search algorithm in more detail.

Let q_i and q_{i+1} be the queries at i^{th} and $(i+1)^{\text{th}}$ iteration, respectively, and o_k be an object in Ω^d as shown in **Figure 6**. Then, the following equation is always true by triangle inequality;

$$\begin{aligned} L_p(q_i, o_k) - L_p(q_i, q_{i+1}) &\le L_p(q_{i+1}, o_k) \\ &\le L_p(q_i, o_k) + L_p(q_i, q_{i+1}) \end{aligned} \qquad (4)$$

```
//q,q^j:the query object and its attribute value at the j^th dimension
//Q,Q^k:the set of bitmaps for the query objects and its the k^th bitmap
//r:the range of r-range search
//CountNumOf11(A):a function that counts the number of '11' in the bit string A
//C_{r-search}:the set of candidate objects for r-range search
//A_{r-search}:the final set of objects for r-range search

Procedure Filtering_r-Search(Q, r)
Begin Procedure
for all o_i such that 1 ≤ i ≤ n do
  apxDist = 0;
  for all k such that 1 ≤ k ≤ l do
    apxDist = apxDist + ||I^k_middle||
      ×CountNumOf11(Q^k XOR B^k(o_i));
  end for
  if apxDist < r then
    tmpSet = tmpSet ∪ o_i;
  end if
end for
return (tmpSet);
End Procedure

Procedure Exact_r-Search(q, C_{r-search}, r)
Begin Procedure
tmpSet = {};
for all o_i such that o_i ∈ C_{r-search} do
  realDist = L_p(q, o_i);
  if realDist < r then
    tmpSet = tmpSet ∪ {o_i};
  end if
end for
return (tmpSet);
End Procedure

Procedure r-RangeSearch(q, r)
Begin Procedure
for all I^k such that 1 ≤ k ≤ l do
  Q^k=MakeBitmap(q,I^k_low,I^k_middle,I^k_high);
end for
C_{r-search}=Filtering_r-Search(Q, r);
A_{r-search}=Exact_r-Search(q,C_{r-search},r);
return (A_{r-search});
End Procedure
```

Figure 5: An r-range Search Algorithm with HBI

Equation 4 implies that the lower bound of the L_p-distance between q_{i+1} and o_k, $L_p(q_{i+1}, o_k)$ could be computed by subtracting the distance between the successive queries, $L_p(q_i, q_{i+1})$, from the distance between q_i and o_k, $L_p(q_i, o_k)$, that was calculated in the previous search iteration. This lower bound could be used to filter-out the irrelevant objects in 2nd and later searches. For example, an object o_k that satisfies the above condition (*i.e.*, $L_p(q_i, o_k) - L_p(q_i, q_{i+1}) < r$) could be filtered-out in the r-range CBMR search without computing the actual distance to the revised query($L_p(q_{i+1}, o_k)$). This simple filtering algorithm helps to reduce the search

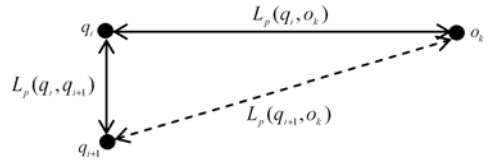

Figure 6: The Relationship between $L_p(q_i, o_k)$, $L_p(q_{i+1}, o_k)$, and $L_p(q_i, q_{i+1})$

time of CBMR with *relevance feedback* in general, and could be used to any CBMR with relevance feedback regardless of its indexing mechanism. Let us show how to apply this optimized filtering algorithm to CBMR with relevance feedback.

In the case of CBMR with relevance feedback using HBI, since $B_p(q_i, o_k) \leq L_p(q_i, o_k)$ is always true by **Theorem 1**, we can derive following equation from **Equation 4**;

$$B_p(q_i, o_k) - L_p(q_i, q_{i+1}) \leq L_p(q_{i+1}, o_k) \quad (5)$$

Equation 5 implies that the lower bound of the L_p-distance between q_{i+1} and o_k,$L_p(q_{i+1}, o_k)$, could be approximated by $B_p(q_i, o_k) - L_p(q_i, q_{i+1})$ in filtering process using HBI, and it could be used to filter-out the irrelevant objects in 2^{nd} and later searches. Since this filtering requires only one subtraction, it could be filter-out the irrelevant objects very efficiently. Of course, since $B_p(q_i, o_k) - L_p(q_i, q_{i+1}) \leq L_p(q_i, o_k) - L_p(q_i, q_{i+1})$, the filtering rate would be lowered with FQD along. This problem could be resolved if the filtering using HBI is applied again to the relevant objects that were not filtered-out by FQD. We will experimentally show that HBI together with FQD could be used to quickly filter-out the irrelevant objects, and this filtering mechanism helps to reduce the total search time of CBMR's with relevance feedback.

4. 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 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)[2] and three synthetic data sets (S1, S2, S3) whose parameters are summarized in **Table 3**. The problem used in the experiments is the r-range similarity search, and all experiments are tested under Microsoft Windows XP on an Intel Pentium 4 (3.0 GHz) with 1 GB main memory.

4.1 Number of Bitmaps and Filtering Rates

Let us show some experiments on the filtering rates of HBI with the data sets shown in **Table 3**. As shown in **Figure 7**, the filtering rates of HBI are increased as the number of bitmaps used for indexing is being increased for the real image data sets and S1. In the case of S2, in which the feature values are skewed to some ranges and these ranges are different 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 S3, a high filtering rate could be achieved with HBI although a small number of bitmaps is used for indexing. Note that, in order to get a filtering rate more than 0.9 for real image databases (R1, R2, R3), 4 bits are required to each dimension in VA-File (*i.e.*, $b \approx 4$), whereas 10 bitmaps are required to each dimension in HBI.(*i.e.*, $l \approx 10$). It means that about five times more bits ($\frac{2 \cdot l}{b} = \frac{2 \cdot 10}{4} = 5$) are averagely required for indexing with HBI to get the similar filtering rate as VA-File. However, because both of these bits for indexing (*i.e.*, the compact approximations of an object in VA-File and HBI) could be usually retrieved with one "read operation" and the I/O time for the index is relatively small compared to the CPU time for indexing, the effect of I/O times for index to the filtering time is slight as shown in the experiments on the total elapsed search times.

Table 3: Multimedia Data Sets used in the Experiments

Data Set	Feature	Distribution	# of Dim	# of Objects
R1	MPEG-7 Color Structure [11]	Extracted from Real Image Sets	256	25,160
R2	MPEG-7 Edge Histogram [11]		80	25,160
R3	HSV Histogram		32	68,040
S1	Synthetically Generated [10]	Uniformly	256	100,000
S2		Skewed		
S3		Clustered		

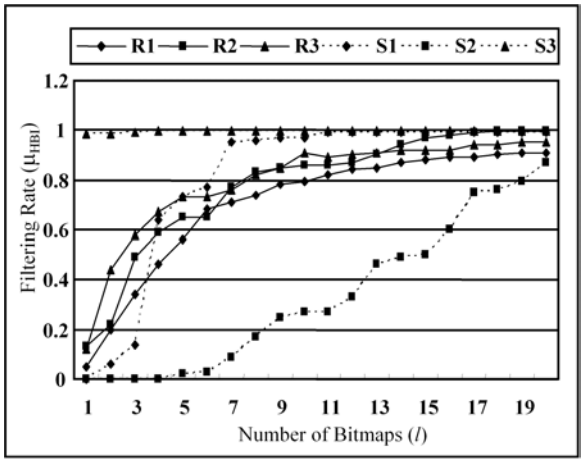

Figure 7: Filtering Rates of HBI

4.2 Experimental Analysis and Comparison of Search Times

Figure 8 shows the elapsed total search times of HBI for S1 while increasing the number of bitmaps for indexing. In the case of S1, the filtering rate is almost 1.0 when 11

[2]R1 and R2 are extracted from UC Berkeley Landscape Images (http://clib.cs.berkeley.edu/photos/use.html) using MPEG-7 XM. R3 is downloadable at UCI KDD archive (http://kdd.ics.uci.edu/databases/CorelFeatures/CorelFeatures.data.html). $S1/simS3$ are synthetic data sets generated as in [10].

bitmaps are used because the feature vectors are clustered, and the total search time is minimized when 7 bitmaps are used. For this data set, the search time is 2.5 $\sim$ 4.0 times faster than BFS (Brute Force Search, or Sequential Search without indexing) when an optimal number of bitmaps are used as shown in **Figure 8**. From these experiments, we find that there is an optimal number of bitmaps that minimizes the total search times. 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.

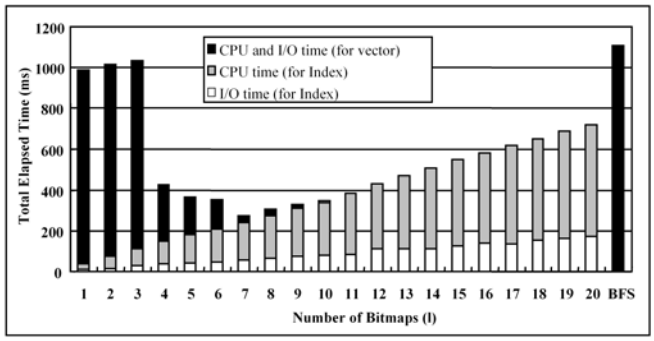

Figure 8: Experimental Search Time of HBI with respect to the Number of Bitmaps

Now, let us experimentally compare the search times of HBI, VA-File, and BFS. **Figure 9** shows the times for indexing and the times for -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 than the ones with VA-File because of its lower filtering rate. However, since the time for filtering (especially, the CPU time for B_p-distance calculations) with HBI is much less than the ones with VA-File, the total search time with HBI is 2.0 $\sim$ 3.0 times faster than the ones with VA-File, and 2.5 $\sim$ 4.0 times faster than the ones with BFS, in the case of experiments with R1, R2, R3, and S1. In the experiment with S2, 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 S3, 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 that the similarity search with HBI is averagely 2.0 $\sim$ 3.0 times faster than the one with VA-File, and could produce the best performance in the clustered data set as S3, and the comparable performance in the skewed data set as S2.

Let us finally present the experimental search times of

[3]Note that the numbers of bitmaps used in the experiments are 6 for R1, 15 for R2, 15 for R3, 7 for S1, 20 for S2, and 1 for S3. 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. These indexing parameters are selected to produce the best performances.

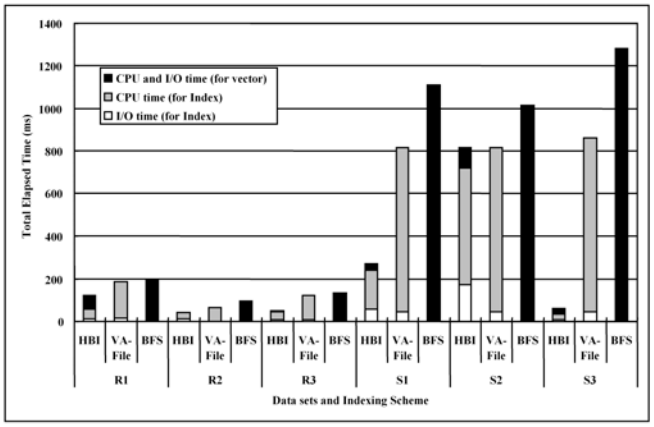

Figure 9: Experimental Comparisons of the Search Times with BFS, VA-File, and HBI

CBMR with relevance feedback with HBI indexing and FQD. **Figure 10** shows the filtering rates of FQD itself in CBMR with relevance feedback for the data sets in **Table 3** when α=0.5, β=0.25, and γ=0.25, in which the half of the search results of previous search are randomly selected and checked as the relevant ones to generate a new query for the next search.

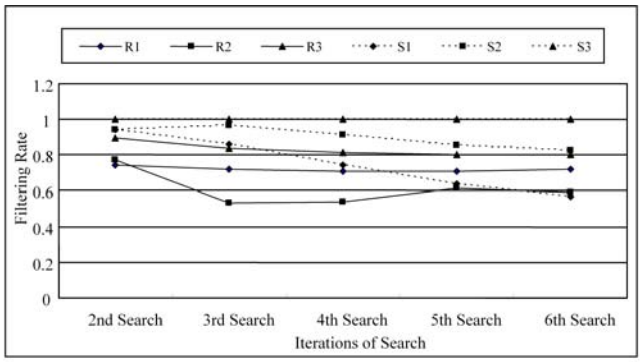

Figure 10: Filtering Rates of FQD

As shown in this experiment, the filtering rates of FQD itself are lowered as the searches with relevance feedback proceed. It is because the lower bound of the distance to the revised query is calculated by subtracting the relative difference between successive two queries from the distance calculated in the 1st search so that the new lower bound is always decreased as the search proceeds. Note that the filtering rates of FQD itself in 2nd and later searches are not high enough to get a satisfiable performance although its filtering time is negligible. It forces us to use the filtering algorithm using HBI to the objects that are not filtered-out by FQD. By filtering the objects by FQD and then HBI, the filtering rates as shown in **Figure 7** could be obtained in 2nd and later searches while reducing their total filtering time. **Figure 11** shows the experimental search times of CBMR with relevance feedback for the data set R3, in which the filtering time includes the time for filtering with FQD and then HBI, and the exact search time includes the time to find the exact solutions from the objects that are not filtered-out

with FQD and HBI. As shown in this figure, the time for 2^{nd} and later search could be $2 \sim 2.5$ times faster than the one for 1^{st} search, because a lot of irrelevant objects could be pre-filtered-out by FQD with a negligible overhead and the remaining objects are tested and filtered-out again with HBI that helps to keep the filtering rate as high as possible. Note that the exact solutions could be also found with this combined filtering algorithm because the relevant objects are never filtered-out by HBI as well as FQD. From this experiment, we can argue that the combination of FQD (for reducing filtering time) and HBI (for a high filtering rate) is a good filtering algorithm that is efficient enough to be used in CBMR with relevance feedback.

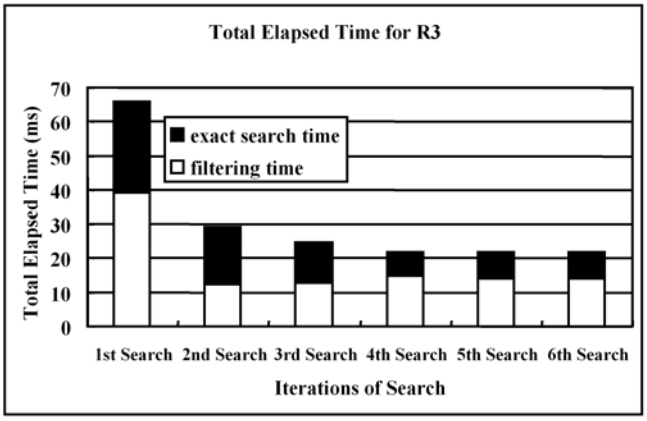

Figure 11: Search Time of CBMR with Relevance Feedback by Filtering FQD and then HBI

5. CONCLUDING REMARKS

In order to build a powerful multimedia archive system supporting CBMR, an efficient indexing scheme that helps to quickly search the multimedia objects similar to the query is highly required. Unfortunately, the conventional indexing schemes based on the data partitioning could not be used for indexing of multimedia objects because of the problem called "dimensionality curse". Recently, some indexing schemes, called filtering approach, such as VA-File and LPC-File were proposed to solve this problem. Although they could index the multimedia objects efficiently by representing their feature vectors with some compact approximations and filter-out the irrelevant objects by calculating approximated distance with these compact representations, they still require a lot of CPU time to filter-out the irrelevant objects because they use the same metrics in the filtering process. This paper proposes a robust indexing scheme, called HBI, 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. With this encoding scheme, many irrelevant objects could be filtered-out by the approximated distance calculated by a simple XOR operation. Because a set of two bits is still too simple to represent the feature value of objects precisely although they are hierarchically organized, the filtering rate of HBI is somewhat smaller than that of VA-File. However, as the distance approximation based on XOR operation is efficient enough, the similarity search based on HBI would be efficient than that of VA-File. This paper also presents a filtering algorithm, called FQD, for the CBMR with relevance feedback that filter-out the irrelevant objects using the previously calculated distances and the distance between two successive queries. Upon experimental results, we found that the total search time based on HBI was averagely $2 \sim 3$ 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, the combination of FQD and HBI for filtering of irrelevant objects in CBMR with relevance feedback could further reduce the search times of 2^{nd} and later searches. The proposed indexing mechanism could be used to build an efficient CBMR system that guarantees a quick response time.

6. REFERENCES

[1] A. Guttman. R-Trees : A dynamic index structure for spatial searching. In *Proc. of ACM SIGMOD International Conf. on Management of Data*, pages 47–57, 1984.

[2] N. Beckmann and H. Kriegel. The R*-Tree : An efficient and robust access method for points and rectangles. In *Proc. of ACM SIGMOD International Conf. on Management of Data*, pages 322–331, 1990.

[3] S. Berchtold, D. Keim, and H. Kriegel. The X-Tree : An index structure for high dimensional data. In *Proc. of International Conf. on VLDB*, pages 28–39, 1996.

[4] P. Yianilos. Data structures and algorithms for nearest neighbor search in general metric space. In *Proc. of the ACM-SIAM Symposium on Discrete Algorithms*, pages 311–321, 1993.

[5] P. Ciaccia, M. Patella, and P. Zezula. M-Tree : An efficient access method for similarity search in metric space. In *Proc. of the International Conf. on VLDB*, pages 426–435, 1997.

[6] S. Berchtold, C. Bohm, and H. Kriegel. The Pyramid Technique: Towards breaking the course of dimensionality. In *Proc. of ACM SIGMOD International Conf. on Management of Data*, pages 142–153, 1998.

[7] R. Weber, H. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high dimensional spaces. In *Proc. of the VLDB Conf.*, pages 194–205, 1998.

[8] G. Cha, X. Zhu, D. Petkovic, and C. Chung. An efficient indexing methods for nearest neighbor search in high dimensional image databases. *IEEE Transaction on Multimedia*, 4(1):76–87, 2002.

[9] I. Ruthven and M. Lalmas. A survey on the use of relevance feedback for information access systems. *The Knowledge Engineering Review*, 18(2):99–145, 2003.

[10] T. Bozkaya and M. Ozsoyoglu. Distance based indexing for high dimensional metric spaces. In *Proc. Of ACM SIGMOD Conf. on Management of Data*, pages 357–368, 1997.

[11] ISO/IEC JTC1/SC29/WG11. *Information Technology Multimedia Content Description Interface-Part3: Visual*. 2001.