Object-Oriented Retrieval Mechanism for Semistructured Image Collections

Guang-Ho Cha
Department of Multimedia Engineering
Tongmyong University of Information Technology
Pusan 608-711, South Korea
+82-51-629-7269
ghcha@tmic.tit.ac.kr

Chin-Wan Chung
Department of Computer Science
Korea Advanced Institute of Science and Technology
Taejon 305-701, South Korea
+82-2-958-3316
chungcw@ngis.kaist.ac.kr

1. ABSTRACT

This paper presents the object-oriented image retrieval mechanism which provides the content model, the indexing scheme, and the query processing techniques *as a whole*. Three types of image content, i.e., visual features, semantic features, and keywords, are defined, and they are represented using the object-oriented data model. Three types of index structures corresponding to the identified features are elaborated to facilitate the search. The query processing techniques to process complex queries which use multiple types of indexes are also described. Experiments have been carried out on large image collections to demonstrate the effectiveness of the proposed retrieval mechanism.

1.1 Keywords

content-based retrieval, image indexing, object-oriented model, multimedia database

2. INTRODUCTION

In everyday life, a large amount of images is produced in various domains and their contents are diverse - news footage, medical imagery, art collections, weather photos, movie images, and more. An important research issue caused from this widespread use of images is content-based image retrieval which helps users retrieve relevant images based on their contents. To provide such a facility images are analyzed so that their content descriptions can be extracted and stored in a database. The descriptions are then used to search the database and to determine which images satisfy the user's query selection criteria. These descriptions are called *metadata*. The effectiveness of content-based retrieval depends largely on the availability of rich metadata. The discriminating power of the retrieval system increases as more information describing the image content is included in the metadata. Thus, many kinds of information may be included in the metadata, and some of them are inherently *unstructured* and some others have certain structures such as tree or graph. The motivation behind our work is to unify the structured and unstructured metadata through object-oriented data model.

Let us discuss the effect of the unstructured modeling, which does not use any specific data model to describe and represent the image content, on the image retrieval system. The most obvious advantage of the unstructured approach is that it is easy to query or browse for novice and infrequent users, because they need not to know about the schema structure of the image database. However, the database administrators or frequent users may want to query in precise manner to restrict the search scope. The second advantage of the unstructured approach is that it is simple to insert, delete, and update data objects, because there is not a complex schema structure. However, this unstructured approach loses many advantages that can be acquired from the structured approach.

We model the metadata using object-oriented data model. The reasons why we use the object-oriented approach to construct our content-based image retrieval system are as follows:

- Using the structure of metadata promotes the clustering of objects with similar characteristics on the secondary storage when they are stored in the database;
- Queries using the schema restrict the search space only to the given class or class hierarchy, and therefore, the search performance may be improved and more exact results can be obtained.
- Schemas make efficient to browse the database due to their structural organization;
- It is convenient to manage the objects with common properties because they are grouped by inheritance relationship;

ACM Multimedia'98, Bristol, UK
© 1998 ACM 1-58113-036-8/98/0008 $5.00

- The facilities, e.g., query by example and query by keywords, allow novice users to query the database without knowing about the schema structure.

We are developing the content-based image retrieval system based on the object-oriented model. The work has three goals: (1) to develop the content model of image to describe the image content, (2) to develop the indexing scheme to index the metadata, (3) to investigate the processing techniques for complex queries.

The first goal is to find what kinds of information are necessary to make the image retrieval system effective. To achieve the second goal we need separate indexes for the feature sets describing the image content, because each feature set has its own intrinsic properties which cannot be mixed with others. For example, the visual features such as colors and textures are quantifiable, while the features which can be obtained only by human sense cannot be quantified by any metric function. The third goal is to develop the processing techniques for queries using multiple feature sets. Efficient query processing is an essential requirement in any information system.

3. OBJECT-ORIENTED CONTENT MODEL

Conceptually, an object has a set of attributes and a set of methods. An attribute corresponds to the instance variable and can take object as its value. The methods are functions act as an interface to other objects. In the following, even if we mention only attributes it is assumed that there are methods associated to the attributes.

3.1 An Image Object

The basic system unit that will be stored and retrieved is an *image object*. An *image object I* consists of a *body B* and *a header H*. The body is a binary bitmap image. The header is a metadata that describes the content of the image. We model the header as a triplet $H = (C_v, C_s, C_k)$, where

- C_v is a class which consists of a fixed number of *visual feature attributes*;
- C_s is a class which consists of a fixed number of *semantic feature attributes*;
- C_k is a class which consists of a variable number of *keywords*.

An object O_v instantiated from class C_v includes the visual features which are extracted automatically by the image processing subsystem. The visual features would be colors, textures, shapes, positions, and so on. We regard the O_v as a point in an n-dimensional visual feature space, where n is the number of visual features extracted. Each visual feature represents one dimension and therefore n represents the dimensionality of the visual feature space.

An object O_s instantiated from class C_s has a fixed number of semantic (i.e., non-quantifiable or abstract) features that should be extracted manually by human sense or that can be deduced by the system when the image is inserted into the database. The semantic features contain higher level of abstract information than the visual features. For example we can enumerate the following common simple semantic attributes of an image:

- title: title of image,
- subject: peace, love, architecture, nature, and so on,
- type: painting, scenery, portrait, and so on,
- perspective: aerial, ground, or close-up,
- orientation: horizontal or vertical,
- date: date when the picture is shot,

An object O_k instantiated from class C_k consists of a variable number of keywords. The set of keywords has no certain structure. Keywords give the gist of an image. They are words or sequence of words which describe the content of an image that can not be easily described using only simple common attributes. Keywords may contain the highest level of abstract information among other features. Figure 1 shows an example image class hierarchy constructed by our content model.

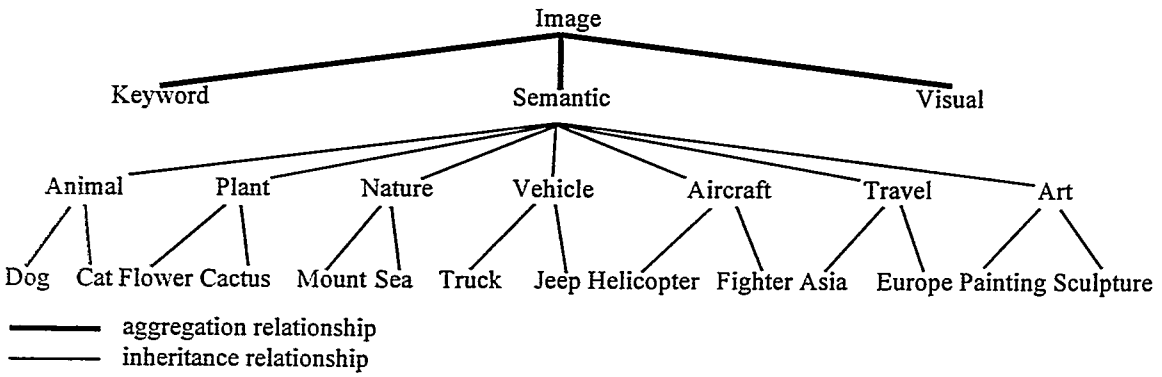


Figure 1. An example image class hierarchy

3.2 Queries

A *user query* is the specification of a header that closely corresponds to the information known about the image. Users can query the image database based on the three types of class attributes: the semantic, keyword, and visual. Some attribute values may be omitted or given a specific values or range of values. Keywords are specified by providing a set of words that describe the image. Visual features are given by example images, user-sketched drawings, or selected color and texture patterns.

To summarize the different types of queries, we have the following:

- *Simple queries* that specify a single attribute value for each possible attribute and require the exact match;
- *Range queries* that either explicitly specify a range of values for some of the attributes, as in (10% ≤ red ≤ 30%) and (30% ≤ green ≤ 40%) and (80% ≤ blue ≤ 90%), or implicitly specify a range of values by leaving one or more attributes values unspecified.
- *Similarity or nearest neighbor queries* that give an example image or user-sketched drawing and require to find most similar images to a given one.
- *Complex similarity queries* that consist of one or more similarity queries and other types of queries. For example, retrieve five images which are most similar to a given image and whose subject is *nature*.

Since the complex similarity query covers all other types of queries, we will focus our attention on the complex similarity query. The other types of query processing methods can be found in [4].

4. INDEXING SCHEME

This section describes the indexing scheme that facilitate the search. Our indexing scheme consists of three types of index structures which correspond to visual class, semantic class, and keyword class.

4.1 Visual Indexes

We assume that a set of n visual features have been extracted (semi-)automatically from each image. They may be dominant colors, textures, shapes, and so on. A set of n visual features is represented by a visual object $O_v = (f_1, f_2, ..., f_n)$, where f_i , $1 \le i \le n$, corresponds to one feature value, and is mapped to a point in an n-dimensional visual feature space. We use the HG-tree [4] as our underlying index structure for organizing the visual feature indexes. It is one of the most successful multidimensional point index structures. The HG-tree guarantees the storage utilization of 66.7% (2/3) in worst case and typically achieves more than 80%. It was shown that the HG-tree is fairly robust even in high dimensions and it achieves good response time.

Many other image retrieval systems used some other index structures. The QBIC system adopted the R*-tree [1]. Petrakis and Faloutsos [24] used R-tree [14]. Mehrotra and Gary [20] used the K-D-B-tree [26]. The systems, CAFIIR [29], STAR [30], and Zhang and Zhong [32] employed the iconic index tree based on the Self-Organizing Map [17]. Compared with the other index structures, the performance of the spatial index structures such as R-tree and R*-tree degenerates drastically with an increase in the dimensionality of the underlying feature space, because their fanout decreases in inversely proportional to the dimensionality. *Fanout* gives the number of entries expected within an index node. All current spatial index structures suffer from this *dimensionality curse*.

The iconic index trees based on the SOM simplify the multidimensional problem by converting it to a one-dimensional clustering problem based on similarities. The major problem of these index structures is that they are *static*. Another problem is that they are constructed only for nearest neighbor queries. Thus, it is difficult to process range queries. In fact, most of the index structures, e.g., VP-tree [6] and GNAT [3], designed only for nearest neighbor queries have these common problems, i.e., they are static and appropriate only for similarity search.

The HG-tree, which is one of the multidimensional point index structures, avoid all above problems. It is less influenced by the increase of the dimensionality than the spatial index structures, because it represents each directory region covered by the data set by only two Hilbert values regardless of the dimensionality. In addition, the HG-tree is completely dynamic. Due to these reasons we adopted the HG-tree as our underlying index structure. However, other recent index structures such as M-tree [7], X-tree [2], and SS-tree [28] may be applied to the system instead of the HG-tree if they show better performance than the HG-tree.

In the HG-tree, all n-dimensional values are transformed into 1-dimensional points using *space-filling curve*, and specifically *Hilbert curve* [15], before they can be used. This promotes the *deferred node splitting* to be used when node overflow occurs, and therefore guarantees high storage utilization. A *space-filling curve* is a mapping that maps the unit interval onto the n-dimensional unit hyper-rectangle continuously. While there are other space-filling curves such as the Peano curve [23] and the Gray-code curve [11], it was shown that the Hilbert curve achieves better clustering than the others [12, 16]. The desirable features of the Hilbert curve are that the points close on the Hilbert curve are close in the domain space, and the points close in the domain space are likely to be close on the Hilbert curve.

To minimize the dead space (i.e., the space which does not include any actual data but covered by the directory region) of the index node, the HG-tree encloses a set of

entries in a node by *minimum bounding interval* (MBI). MBI is the smallest interval on the Hilbert curve, which completely encloses all the regions at lower level. The MBI I is represented by two Hilbert values at both ends of interval, $I = (H_1, H_2)$, where H_1 is the starting point and H_2 is the ending point on the MBI.

The HG-tree consists of internal and leaf nodes as in other index trees. A *leaf node* contains at most C_l entries of the form (*oid, H*), where C_l is the capacity of the leaf node, *oid* is a pointer to the object in database, and H is the Hilbert value for the feature vector. An *internal node* contains at most C_n entries of the form (*ptr, I*), where C_n is the capacity of an internal node, *ptr* is a pointer to the child node, and I is the MBI. The entries in nodes are maintained in Hilbert order.

4.2 Semantic Indexes

Queries in object-oriented database are formulated with reference to a target class c with two possible interpretations: the query can be evaluated on the set of all objects which belong exclusively to the target class c, or it may be evaluated on the set of all objects belonging to any class in the class hierarchy rooted at c. To support concurrently both types of queries, we provide more efficient alternative implementation based on the χ-tree [5]. The central idea of the χ-tree is to transform the object indexing problem into a 2-dimensional range search problem by imposing an appropriate linear order on classes in a class hierarchy. With this transformation, every query Q can be mapped to a 2-dimensional search space with the class ordering as one dimension and the indexed attribute as the other dimension. This search space takes the form of a rectangular region R_Q. The answers to a query Q consists of all the data points which fall within the region R_Q.

Example 1. Consider the class hierarchy shown in Figure 2, which is mapped to the sequence A, B, E, F, G, C, D using the preorder traversal of the class hierarchy. The query is represented by rectangular region $([c, c'], [r, r'])$, where $[c, c']$ defines the class range and $[r, r']$ defines the range of attribute values. As illustrated at (b) in Figure 2, the query $Q1$ corresponds to a class hierarchy query on the class B on the attribute range [1, 5]. Query $Q2$ is a query on the entire class hierarchy with the indexed attribute restricted to value 3. Query $Q3$ corresponds to a single class query on class C with range [4, 6].

Unlike the χ-tree which uses the K-D-B-tree-like structure as underlying structure, we employ the HG-tree as our underlying structure for indexing the hierarchy of semantic features, because of the performance advantages of the HG-tree over the K-D-B-tree. We construct one $(n+1)$-dimensional semantic index tree, which consists of n common semantic feature dimensions and one class ordering dimension, rather than n 2-dimensional χ-trees. Thus the class hierarchy indexing problem is transformed into a $(n+1)$-dimensional range search problem.

The advantages of using single multi-attribute (SMA) index over using multiple single-attribute (MSA) indexes are obvious. First, the clustering of index pages and data pages on disk can reduce significantly the number of I/O operations needed for database accesses. Second, to process multi-attribute queries in MSA index, multiple independent accesses to separate indexes and the intersection of the multiple partial results are needed to get a final result. Third, when new object is inserted into or deleted from a database, SMA index organization needs only single update for its index, while MSA index requires multiple updates.

4.3 Keyword Indexes

The *signature file* has proved to be a convenient indexing technique for keywords [8, 18, 19, 25, 27, 33]. Multidimensional index structures are not appropriate for indexing keywords, because they assume that the dimensionality of the domain space is small and constant. However, the number of keywords given by users to query may be variable. Moreover, most of the multidimensional index structures suffer from the dimensionality curse.

The main idea of the signature file is to derive properties of data objects, called *signatures*, and store them in a separate file. Signatures are hash-coded binary words of fixed length. A collection of the derived signatures is called *signature file*. Although a lot of research has been done on the improvement of the performance of a signature file, most

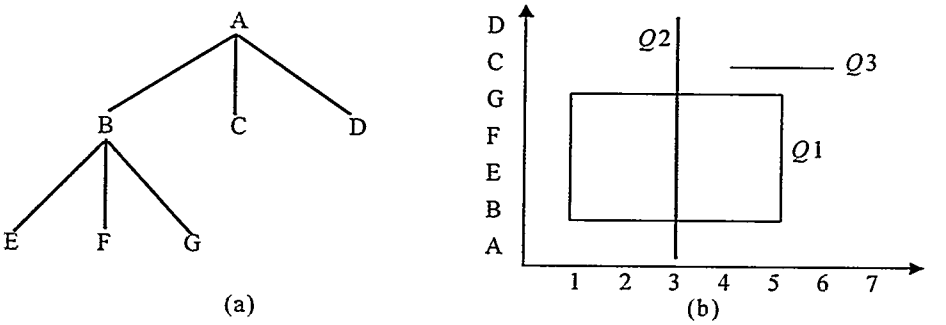


Figure 2. Transformation of class indexing to a 2-dimensional indexing problem

of the researches have been performed for *static* environment where update operations are rarely occurred. Our indexing scheme requires a *dynamic* environment. The two representative dynamic signature files are S-tree [8] and Quick filter [33]. The main idea of the S-tree is to group adjacent signatures and build a B-tree on top of them. The major problem of the S-tree is that the performance is degenerated as the query signature weight becomes lower. The number of 1's in a signature is called the *signature weight*. In the Quick filter, a signature file is partitioned by a hash function and the partitions are organized by linear hashing. Therefore, it is appropriate for the dynamic environment and results in good performance in the queries with high signature weights. However, if the distribution of signatures is nonuniform, then similar signatures are frequently generated and therefore the overflow rate increases and the storage utilization decreases. These degenerate the performance of the Quick filter.

To attack the disadvantages of existing dynamic signature files, we combine the concepts of the HG-tree and the frame-sliced signature file [19]. Using the HG-tree, which is a complete dynamic index structure, we solve the problem caused by high overflow and low storage utilization. We also tackle the problem caused by light weight signatures by adopting of frame-sliced signature method. The leaf nodes of the HG-tree are built using the concept of the frame-sliced signature file. The directory regions in the nodes are represented by the image signatures.

At first, a signature is divided into s *frames*, and c frames are selected out of a total of s frames using one hash function h_1. To make up the *word signature* (i.e., the signature corresponding one keyword) m bits are set to "1" in the selected c frames using the second hash function h_2. The *frame signature* is constructed by superimposing the parts belong to the corresponding frame of word signatures. At last, the *image signature* describing the content of an image is constructed by combining the frame signatures.

Example 2. Figure 3 shows the procedure of constructing an image signature based on the frames when an image consists of a set of four keywords, {'*sky*', '*sea*', '*bridge*', '*ship*'}. In Figure 3, it is assumed that the length of an image signature is 12 bits, the number of total frames is 3, the number of frames to be selected is 2 and the number of bits to be set is 2. The word signatures of keywords '*sky*', '*sea*', '*bridge*' and '*ship*' are assumed to be 0010 1000 0000, 0000 0100 0100, 0100 0001 0000, and 0010 0000 1000, respectively. The image signature becomes 0110 1101 1100 when we concatenate the frame signatures 0110, 1101 and 1100.

a set of keywords O_k = { '*sky*', '*sea*', '*bridge*', '*ship*' }

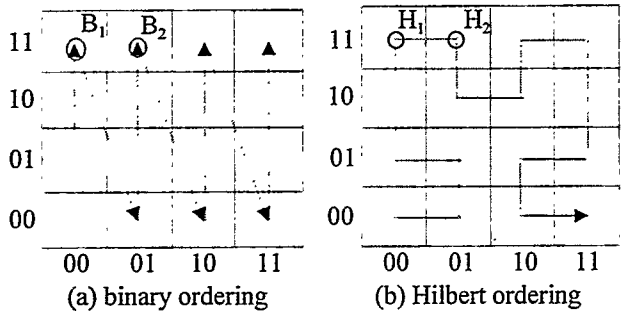
keywords	frame1	frame2	frame3
sky	0010	1000	
sea		0100	0100
bridge	0100	0001	
ship	0010		1000
image signature	0110	1101	1100

Figure 3. The procedure constructing the image signature

The leaf and internal node structures of the HG-tree for indexing keyword signatures are slightly modified to accommodate the keyword indexes. The entry in a leaf node has the form (*oid*, F), where *oid* is a pointer to the raw image in the database and F is an image signature which consists of s frames (F_1, F_2, ..., F_s). The entry in an internal node has the form (*ptr*, S), where *ptr* is a pointer to the child node and S is a signature made by superimposing all the image signatures in the corresponding child node.

When we construct the HG-tree with image signatures, the values of the image signatures are interpreted as the Hilbert values and they are inserted in the order of Hilbert values. This interpretation procedure corresponds to the mapping a point in s-dimensional space into a point in a linear Hilbert curve. The number of frames, s, constituting an image signature determines the dimensionality of the keyword domain space and the image signature used in a node of the HG-tree determines a directory region in the domain space. The reason of using Hilbert mapping instead of simply concatenating s frames is to place the similar signatures into the same disk pages. Example 3 illustrates this property.

Example 3. Figure 4 shows (a) the binary number sequence and (b) the Hilbert number sequence in a 4×4 grid space. Since the two points B_1 and B_2 are adjacent, the corresponding signatures may be quite similar when we concatenate the codes in two directions. The signatures (or numeric values) of B_1 and B_2 are 0011 and 0111, respectively, and the signature suffixes are very similar. However, the probability of placing the two signatures into same page



(a) binary ordering (b) Hilbert ordering

Figure 4. Illustration of binary number sequence vs. Hilbert number sequence

is very low, because the numeric difference of two signatures is relatively large. This may result in many random accesses on disk. On the other hand, The signatures (or Hilbert values) of H_1 and H_2 are 0101 and 0110, respectively. Thus the probability of placing the two signatures into same page may be very high, because the numeric difference of them is only 1. The points close on the Hilbert curve are also close in the domain space. This characteristic can achieve a better clustering of pages and can avoid expensive random disk accesses.

5. QUERY PROCESSING

The result of a k-nearest neighbor query based on the visual features is a sorted list with k most similar objects. On the other hand, the results of queries based on the semantic features and keywords are sets of objects which satisfy the queries. When the complex similarity queries are issued to retrieve, the search procedure must synthesize the results of different types of queries in a certain consistent manner.

Example 4. Let us consider the query Q_1 where a user wants to retrieve 5 most similar image to a given image I and whose *subject* is '*animal*':

Q_1: $(C_v(color\text{-}histogram) = 'I') \wedge (C_s(subject) = 'animal')$

In this case, the query result is probably a list with 5 objects sorted by the visual similarity to the image I, where the value of *subject* attribute in C_s is *animal*. A reasonable way to evaluate these types of queries, in which the result of one query is a set and the result of the other query is a sorted list, would be to first evaluate the query whose result becomes a set, and then to find the sorted list consisting of the required number of objects from the set. Therefore, to evaluate the query Q_1 in this example we determine all objects that satisfy the predicate $C_s(subject) = 'animal'$, and then obtain similarity scores of the objects, and finally return 5 objects which have highest similarity scores.

Example 5. Let us now consider the query Q_2 with two similarity predicates where a user wants to retrieve k best matches which are similar to both of images I_1 and I_2:

Q_2: $(C_{v1}(color\text{-}histogram) = 'I_1')$

$$\wedge (C_{v2}(color\text{-}histogram) = 'I_2')$$

There are several ways to deal with this kind of queries. An obvious fact is that it is not correct to retrieve the best match only for single predicate. In the query where several sample images are given, we can transform it to a query containing an image which is best matched to all query images. In the query Q_2, we can find a point P which is closest to both points P_1 and P_2 corresponding to the images I_1 and I_2, respectively. Then we retrieve k objects closest to P.

Another solution to this kind of query is Fagin's A_0 algorithm [9], which independently evaluates the predicates in the query and computes the overall similarity scores of the objects in each result set based on a rule combining the similarity scores. Finally it returns k objects in the order of highest similarity scores.

It should be noted that the query results from the above two methods, i.e., the method transforming the features in several query images into the features of one best-matched image in advance and Fagin's A_0 algorithm, may be different. In the former, the search target is changed to the object most common to the given query objects, while in the latter method, the most similar objects among the separately selected best matches are chosen. This difference for best matches may make the final results different. To determine which method produces more exact results is not trivial and may be dependent on the user's viewpoint. One obvious fact is that the former is far more efficient than the latter.

Example 6. Consider a database where images are characterized globally by colors and textures and also characterized by shapes and locations of the components within the image. Let us assume that the two sets of visual feature attributes, i.e., C_{v1} = {color, texture}, C_{v2} = {shape, location}, are indexed separately. Consider the query Q_3:

Q_3: $(C_{v1}(color, texture) = 'I_1')$

$$\wedge (C_{v2}(shape, location) = 'I_1').$$

In this case, since the domain space of two predicates are made different, the evaluation of each predicate must be performed independently and the algorithm such as Fagin's A_0 should be incorporated to combine two result sets. However, it is important to note that the visual features should be integrated in a single index if possible, because the independent evaluation of the predicates and the combination of the separate results are very expensive.

Fagin used the standard rules of fuzzy logic [31] for evaluating Boolean combinations of atomic formulas:

Conjunction rule: $s_{A\wedge B}(x) = \min\{s_A(x), s_B(x)\}$

Disjunction rule: $s_{A\vee B}(x) = \max\{s_A(x), s_B(x)\}$

Negation rule: $s_{\neg A}(x) = 1 - s_A(x)$

where $s_A(x)$ is the similarity score of object x under the query A. Although these rules have some attractive points as shown in [9], the drawback is that it depends on only the single operand, that is, $s_{A\wedge B}(x)$ (as well as $s_{A\vee B}(x)$) is always equal either to $s_A(x)$ or $s_B(x)$. This does not reflect the effect of all predicates. With this observation, we use other rule to synthesize the results evaluated independently, which will depend on all predicates in the query.

The first step towards an evaluation of complex similarity queries concerns how to compute the similarity score. The next step is how to combine the similarity scores when they are computed independently. As mentioned before, it is desirable to build a single index which covers all features if possible. However, the method to synthesize other types

of queries should be invented in the cases that data reside in multiple systems or features describing the image have inherently different characteristics, e.g., some features are quantifiable and the others are not.

In general, evaluating the similarity of an object with respect to a query value can be done through the *distance function* measuring the distance between feature values. Definition 1 gives the distance function used in our work.

Definition 1. Distance function

A *distance function*, d, for any pair of feature values (x, y) from the domain space D yields a non-negative real value between 0 and 1, which shows the normalized distance between x and y.

$$d: D^2 \rightarrow [0, 1]$$

The 0 distance denotes the exact match and the 1 distance shows the maximum difference.

Definition 2 gives the similarity function which assigns maximum similarity (i.e., 1) in case of 0 distance and makes the similarity inversely related to the distance.

Definition 2. Similarity function

We define the *similarity function*, s, for any pair of feature values (x, y) as

$$s(x, y) = 1 - d(x, y),$$

where d is a distance function. The similarity score is also a real value in the range [0, 1].

Let us now consider how to combine the similarity scores computed in independent predicate evaluations. Fagin used the standard rule of fuzzy logic for this purpose. Although it is not clear which is the best rule, it is important to realize that any specific model has some advantages and drawbacks. We do not consider the Fagin's rule as our combination method, because, in that rule, the best match only for a certain specific predicate determines the overall best match. We use the *probability function* on the independent predicates as our combining rule. Then the overall similarity score will be determined by the following definition.

Definition 3. Overall similarity function s

overall similarity score of object x

$$= s_{A1 \wedge \ldots \wedge Am}(x) = \prod_{i=1}^{m} s_i(x)$$

where s_i the similarity function over the i-th predicate A_i and m is the number of conjuncts, i.e., the number of similarity predicates. This overall similarity function gives the combined degree of similarity.

With these distance and similarity functions, the algorithm **ComplexNNSearch** based on the Fagin's A_0 algorithm for processing the complex k-nearest neighbor query can be given. Note that the procedure **NNSearch**(A_i, k) receives the predicate A_i and the number, k, of objects to retrieve, and returns k objects with highest similarities.

Algorithm ComplexNNSearch(m, A, k)

// m is the number of conjuncts (or predicates) in the query, A is the set which consists of m predicates A_i, i.e., $A = \{A_i\}$, $i = 1 .. m$, and k is the number of objects to retrieve //

1. [Initialize]
   for $i \leftarrow 1$ to m do
   $X^i \leftarrow \phi$
   // X^i is initialized to empty set. X^i will contain the objects returned from the k-nearest neighbor search. //
2. [Find the k matches]
   do {
   for $i \leftarrow 1$ to m do {
   $G^i \leftarrow$ **NNSearch**(A_i, k)
   // G^i is the set which will contain the objects and their similarity values in the output from the processing of the i-th k-nearest neighbor search conjunct A_i. //
   $X^i \leftarrow X^i \cup G^i$
   }
   $L \leftarrow \bigcap_{i=1}^{m} X^i$
   // L is a set which has the intersection of X^is. //
   } while ($|L| < k$)
   // Perform loop until L has at least k objects.
   $|L|$ denotes the number of objects in the set L. //
3. [Compute the similarity scores for the candidates]
   $Y \leftarrow \{ x \mid x \in \bigcup_{i=1}^{m} X^i \}$
   $j \leftarrow$ the number of objects in Y
   for $i = 1$ to j do {
   Compute the similarity score c_i of x_i in Y using the overall similarity function.
   Restore (x_i, c_i) into Y.
   }
4. [Find the most similar k objects]
   Return the sorted list of k objects with highest similarity scores.

6. EXPERIMENTS

To test the effectiveness of our content-based image retrieval mechanism, we have constructed an image database that has a 1,064 images. The images are 256-color bitmaps with a variety of contents.

6.1 Visual Feature Extraction

To acquire the visual features we used statistical color moments of the histogram of an image. Since most histogram bins of an image are sparsely populated and only a small number of bins have the majority of pixel counts, we used only the largest 32 bins (in terms of pixel counts) as the

representative bins of the histogram. We used first two moments of the histogram as descriptors of an image:

$$\mu_i = \frac{1}{n}\sum_{j=1}^{k} f_{ij}x_{ij},$$

$$\sigma_i = \left(\frac{1}{n}\sum_{j=1}^{k} f_{ij}\left(x_{ij}-\mu_i\right)^2\right)^{\frac{1}{2}}, i=1,2,3$$

where x_{ij} is the value of color component of j-th bin, f_{ij} is the frequency of x_{ij}, k is the number of total bins, i.e. 32, and n is the total number of pixels in the histogram. Since we use the RGB color model, the i-th color component corresponds to one of red, green, and blue. The first moment, μ_i, defines the average intensity of each color component. The second moment, σ_i, is a measure of contrast that can be used to represent relative smoothness.

Measures of global color statistics using only histograms suffer from the limitation that they carry no information regarding the relative position of pixels. To overcome this limitation to some extent, we divided the image into 4 sub-areas and computed 2 moments for each sub-area, resulting in a 24 (= 2 moments × 3 color components × 4 sub-areas) visual features for an image. Using this 24-dimensional feature vector, we estimate the similarity, $s(S, T)$, between two color histograms S and T as follows:

$$s(S,T) = 1 - \frac{\sum_{k=1}^{4}\left(\sum_{i=1}^{3}\left(|\mu_{ik}(S)-\mu_{ik}(T)|+|\sigma_{ik}(S)-\sigma_{ik}(T)|\right)\right)}{\Delta}$$

where Δ is a normalizing factor.

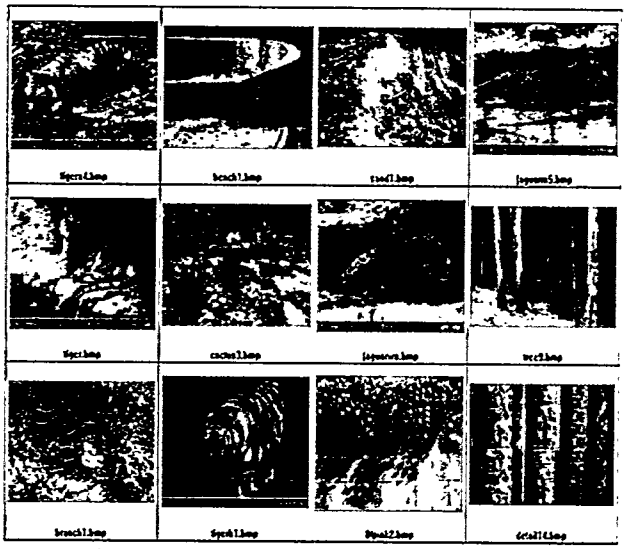
6.2 Sample *k*-Nearest Neighbor Queries

Figure 5 shows the results of two sample 12-nearest neighbor queries:

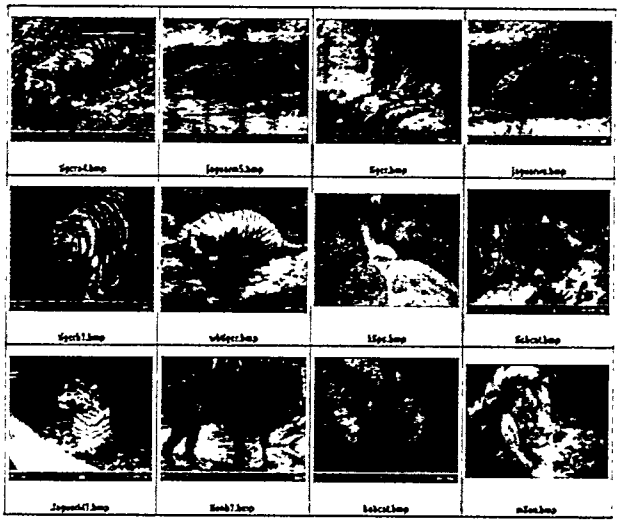
(a) Query 1: "Find 12 images most similar to image *tigera4.bmp*"

(b) Query 2: "Find 12 images most similar to image *tigera4.bmp* and whose keyword is *animal*".

The image on the upper-left corner in Figure 5(a) and 5(b) is the query image *tigera4.bmp* and 12 most similar images are retrieved in left-right and top-down sequence: In Figure 5(a), the query image, tigera4.bmp is, of course, the most similar image, bench1.bmp is the second similar image, and detail14.bmp is the 12nd similar image. The result of Figure 5(a) is obtained only using visual features. Obviously, all images retrieved from a real image database have similar color properties to the given query image. On the other hand, the keyword *animal* is used together with the color visual features in the processing of Query2. As you can see, the more the features are specified in the query, the higher the *selectivity*, i.e., the ratio of the expected number of answers over the total number of data in the database, is increased.



(a) Queries using only color visual features



(b) Queries using color visual features together with the keyword *animal*

Figure 5. Two sample 12-nearest neighbor queries

7. CONCLUSIONS

In this paper, we have considered the issues concerned with the content model, the indexing scheme, and the query processing techniques as a whole in content-based image retrieval system. The content of an image consists of a set of visual features, a set of semantic features, and a set of keywords. Each component of the content may have different structural characteristics. We model these multiple types of information using object-oriented approach. To index three kinds of feature sets, three types of index

structures are proposed. The underlying index structure is the HG-tree. The performance advantages of the HG-tree make our indexing scheme efficient. The techniques for processing complex similarity queries are also provided. These three issues are very important things to construct the effective and efficient image retrieval system. In the future, we plan to consider whether the object-oriented data model is still effective when more complex types of information coexist and how the different types of index structures and query evaluation strategies can improve the retrieval performance of the system.

8. ACKNOWLEDGMENTS

This research was supported by the grant for the Korea Science and Engineering Foundation with grant number KOSEF 95-0100-23-04-3.

9. REFERENCES

[1] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An efficient and robust access method for points and rectangles," *Proc. of ACM SIGMOD Conf.*, 1990, 322-331.

[2] S. Berchtold, D.A. Keim, H.-P. Kriegel, "The X-tree: An Index Structure for High-Dimensional Data," *Proc. of the 22nd VLDB Conf.*, 1996, 28-39.

[3] S. Brin, "Near Neighbor Search in Large Metric Spaces," *Proc. of the 21st VLDB Conf.*, 1995, 574-584.

[4] G.-H. Cha and C.-W. Chung, "A New Indexing Scheme for Content-Based Image Retrieval," *Multimedia Tools and Applications*, Vol. 6, No. 3, May 1998, 263-288.

[5] C.Y. Chan, C.H. Goh, and B.C. Ooi, "Indexing OODB Instances Based on Access Proximity," *Proc. of the 13th IEEE Int. Conf. on Data Engineering*, 1997, 14-21.

[6] T.-C. Chiueh, "Content-Based Image Indexing," *Proc. of the 20th VLDB Conference*, 1994, 582-593.

[7] P. Ciaccia, M. Patella, and P. Zezula, "M-tree: An efficient access method for similarity search in metric spaces," *Proc. of the 23rd VLDB Int. Conf.*, 1997, 426-435.

[8] U. Deppisch, "S-Tree: A Dynamic Balanced Signature Index for Office Retrieval," *Proc. of the ACM SIGIR Conf.*, 1986, 77-87.

[9] R. Fagin, "Combining Fuzzy Informatin from Multiple Systems," *Proc. of the 15th ACM Symposium on PODS*, 1996, 216-226.

[10] R. Fagin, N. Nievergelt, N. Pippenger, and H. R. Strong, "Entendible hashing - A fast access method for dynamic files," *ACM Transactions on Database Systems*, 4(3), 1979, 315-344.

[11] C. Faloutsos, "Gray codes for partial match and range queries," *IEEE Transactions on Software Engineering*, 14(10), 1988, 1381-1393.

[12] C. Faloutsos and S. Roseman, "Fractals for secondary key retrieval," *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on PODS*, 1989, 247-252.

[13] M. Flickner et al., "Query by Image and Video Content: The QBIC System," *IEEE Computer*, 28(9), 1995, 23-32.

[14] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," *Proc. of the ACM SIGMOD Conf.*, 1984, 47-57.

[15] D. Hilbert, "Uber die stetige Abbildung einer Linie auf ein Flachenstuck," *Math. Annalen*, 38, 1891.

[16] Jagadish, "Linear Clustering of Objects with Multiple Attributes," *Proc. of the ACM SIGMOD Conf.*, 1990. 332-342.

[17] T. Kohonen, "The Self-Organizing Map," *Proc. of the IEEE*, 78(9), 1990, 1464-1480.

[18] D.L. Lee and C.-W. Leng, "Partitioned Signature Files: Design Issues and Performance Evaluation," *ACM Transactions on Office Information Systems*, 7(2), 1989, 158-180.

[19] Z. Lin and C. Faloutsos, "Frame-Sliced Signature Files," *IEEE Transactions on Knowledge and Data Engineering*, 4(3), 1992, 281-289.

[20] R. Mehrotra and J.E. Gary, "Similar-Shape Retrieval in Shape Data Management," *IEEE Computer*, 28(9), 1995, 57-62.

[21] J. Nievergelt, H. Hinterberger, and K.C. Sevcik, "The grid file: an adaptable, symmetric multikey file structure," *ACM Transactions on Database Systems*, 9(1), 1984, 38-71.

[22] V.E. Ogle and M. Stonebraker, "Chabot: Retrieval from a Relational Database of Images," *IEEE Computer*, 28(9), 1995, 40-48.

[23] G. Peano, "Sur une courbe qui remplit toute une aire plane," *Math. Annalen*, 36, 1890, 157-160.

[24] E.G.M. Petrakis and C. Faloutsos, "Similar Searching in Large Image Databases," Technical Report CS-TR-3388, University of Maryland, 1994.

[25] C. S. Roberts, "Partial-Match Retrieval via the Method of Superimposed Codes," *Proc. of the IEEE*, 67(12),1979, 1624-1642.

[26] J. T. Robinson, "The K-D-B-Tree: A Search Structure for Large Multidimensional Dynamic Indexes," *Proc. of the ACM SIGMOD Conf.*, 1981, 10-18.

[27] R. Sacks-Davis, A. Kent, and K. Ramamohanarao, "Multikey Access Methods Based on Superimposed Coding Techniques," *ACM Transactions on Database Systems*, 12(4), 1987, 655-696.

[28] D.A. White and R. Jain, "Similarity Indexing with the SS-tree," *Proc. of the IEEE Int. Conf. on Data Engineering*, 1996, 516-523.

[29] J.K. Wu, Y.H. Ang, P. Lam, H.H. Loh, and A.D. Narasimhalu, "Inference and retrieval of facial images," *Multimedia Systems*, 2(1), 1994, 1-14.

[30] J.K. Wu, A.D. Narasimhalu, B.M. Mehtre, C.P. Lam, and Y.J. Gao, "CORE: a content-based retrieval engine for multimedia information systems," *Multimedia Systems*, 3(1), 1995, 25-41.

[31] L.A. Zadeh, Fuzzy Sets, Information and Control. 8, 1965, 115-124.

[32] H.J. Zhang and D. Zhong, "A Scheme for Visual Feature based Image Indexing," *Proc. of the IS&T/SPIE Conf. on Storage and Retrieval for Image and Video Databases III*, 1995, 36-46.

[33] P. Zezula, F. Rabitti, and P. Tiberio, "Dynamic Partitioning of Signature Files," *ACM Transactions of Information Systems*, 9(4), 1991, 336-369.