$2D$-h Trees: An Index Scheme for Content-Based Retrieval of Images in Multimedia Systems*

Youping Niu, M. Tamer Özsu, Xiaobo Li
Laboratory for Database Systems Research
Department of Computing Science
University of Alberta
Edmonton, Alberta
Canada T6G 2H1
{niu,ozsu,li}@cs.ualberta.ca

Abstract – **An important feature to be considered in the design of a Multimedia Database Systems (MMDBS) is content-based retrieval of images. Spatial features represent the spatial relationships among objects in an image. The *salient objects* (interesting objects) can be organized in an object hierarchy, based on object-oriented concepts. This paper proposes a new indexing scheme, called 2d-h trees, for content-based retrieval of images. This scheme organizes the representations of the spatial relationships among objects in images and the hierarchical relationships among objects efficiently for query optimization. Our performance analysis indicates that the $2D$-h-tree is an efficient index scheme for content-based retrieval of images.**

1 Introduction

Current hardware technology enables us to acquire, store, manipulate and transmit various types of data such as *image, audio*, and *video*. This has stimulated a great deal of interest in *Multimedia Database Systems* (MMDBS). An important feature to be considered in the design of a MMDBS is content-based retrieval of images. The effectiveness of still image retrieval in a MMDBS ultimately depends on image content representations, the types of image queries allowed, and the search and retrieval strategies. One important consideration in selecting appropriate types of image content representations is their computational efficiency in the search procedures in order to meet the performance requirements of MMDBS applications.

Other than the traditional types of queries, such as query by command, query by identifier, and range query, most MMDBS applications allow queries by example. In this case, the query image or its sketch is provided while the system must analyze and extract the appropriate features. These features are then compared with the image content representations stored in the database to find matching images. Among all the existing image features, *texture*, *color*, *shape*, and *spatial* features are often selected, because they have broad, intuitive applicability [1].

The search and retrieval of images by content, using indexing methods, mainly relies on image content representations (i.e., the representations of the chosen features) and similarity measures used in comparing these features. This type of querying is well understood [2]. Another important method of search is based on the spatial relationships of objects in these images. The spatial relationships, such as relative positioning, adjacency, overlap, and containment, enable users to ask queries of the type "show all the images where a car is on the left of a $building$." Spatial querying is not as well studied as feature-based querying. Furthermore, based on object-oriented concepts, the interesting image components, the *salient* objects, can be organized in an object hierarchy. Coupled with searches on other features, powerful query systems can be developed that can allow queries of the type "show all the images where a $red\ car$ is in front of an $animal$."

In this paper, we present a new indexing scheme for content-based retrieval of images. The new indexing scheme, which we call $2D$-h trees, organizes the representations of the spatial relationships among objects in images and the hierarchical relationships among objects efficiently for query optimization. Our performance analysis indicates that the $2D$-h-tree is an efficient index scheme for content-based retrieval of images.

The remainder of the paper is organized as follows. In Section 2, we present the proposed indexing scheme. The performance analysis of our indexing scheme is presented in Section 3. Section 4 presents our conclusions and future work.

*This research has been supported by a grant from the Canadian Institute for Telecommunication Research (CITR), a Federal Network of Centre of Excellence funded by the Government of Canada and by a Strategic Grant from the Natural Science and Engineering Research Council (NSERC) of Canada.

2 The Proposed Indexing Scheme: $2D$-h Trees

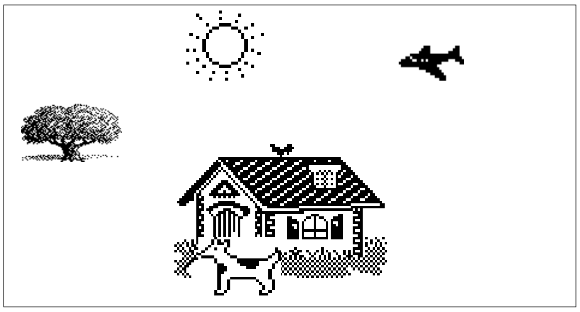

Figure 1: An Image.

For each image in the database, we first use edge detection, combined with other detection techniques, to extract the contours of objects. We then label (automatically or manually) each salient object (interesting object) with a symbol which can be used for indexing and querying. For example, in the image in Figure 1, the following salient objects would be identified: *house, tree, dog, plane* and *sun*. Other objects such as the *grass* (on the ground) and a bird (on the roof) may be ignored since they are not so interesting for this image from indexing and searching points of view. Then we use 2-D string scheme [3], which is the most commonly used data structure for spatial reasoning and spatial similarity computing, to present the spatial relationships among those salient objects. The 2-D string representation of the image in Figure 1 is: $(tree < house : dog = sun < plane,\ house : dog < tree < sun = plane)$, where the symbol "$=$" denotes the spatial relation "at the same location as", the symbol "$<$" denotes the spatial relations "left of/right of" and "below/above", and the symbol ":" denotes the spatial relation "in the same set as". These relationships are specified in both x and y dimensions resulting in a two dimensional string. Refer to [3] for more detailed discussion about the 2-D string representation.

Furthermore, all the salient objects identified (for all images in the database) are classified by using object-oriented modeling method since some objects can be viewed as the constituent objects for an object. This forms an *is-part-of* hierarchy. For example, the objects *cat, dog, horse* ..., can be viewed as the objects (or subclasses) in (of) the class of *Animal*, while the objects *car, truck, van* ..., as the objects (or subclasses) in (of) the class of *Vehicle*.

The $2D$-h trees consist of two indexing structures: one for the organization of the hierarchical relationships among objects, and another one for the organization of the spatial representations among objects in images.

2.1 The Indexing Structure for The Spatial Representations

The 2-D string has been an effective approach to represent the spatial relationships among objects in images. The problem of the approach, though, is that for a large database the costs of the sequential subsequence matching is too high since the 2-D string specified for the query must match against all the 2-D strings (for all images) in the database sequentially by using the subsequence matching algorithm [3]. To solve the problem, we have developed a new indexing structure, called the 2-D-S-tree, to organize 2-D strings for query efficiency. The 2-D-S-tree supports the following types of queries very efficiently:

1. Find all images containing at least one cat, two dogs, and a $duck$. ($type$-0 query)

2. Find all images containing a $plane$ to the right of a $tree$. ($type$-1 query)

3. Find all images containing a $plane$ immediately to the right of a $tree$. ($type$-2 query)

When a query is mainly interested in finding all the images containing specified symbols (objects) with some exclusive spatial conditions, $type$-0 subsequence matching should be conducted. Given a key, for example, $a = b$, $type$-0 queries treat images with relationships $a = b$, $a : b$ and $a < b$ as satisfactory. But it excludes images where $b < a$ from the answer set. For a query "Find all images containing a $tree$ and a $house$ as long as the $house$ is not to the left of the $tree$", the image in Figure 1 would be retrieved by the $type$-0 query.

A $type$-1 query is interested in finding all the images in which the symbols in the pattern maintain alignments between each other, but the "distance" between any two symbols is insignificant and may be different than in the original pattern. For example, "Find all images containing a $plane$ to the right of a $tree$". An image containing a plane to the right of tree, but with a house between them, would be retrieved by a $type$-1 query.

A $type$-2 subsequence matching is more precise than both $type$-0 and $type$-1 subsequence matching. If a local substring u_1 of u matches a local substring v_1 of v, then the local substring u_i of u must match the local substring v_i of v for any $i \geq 1$. Thus, the "distance" between objects is important. For example, $tree < plane$ is not a $type$-2 subsequence of $tree < house : dog = sun < plane$. Therefore, an image containing a plane to the right of tree but with a house between them is not a resulting image of a $type$-2 query: "Find all images containing a $plane$ immediately to the right of a $tree$".

Readers may refer to [3, 8] for more detailed discussion about $type$-0, $type$-1 and $type$-2 queries.

The 2-D-S-tree is a set grouping index whose non-leaf nodes have a layout similar to the B^+ tree [9], while the leaf node layout is different. Figure 2 shows the layout of a 2-D-S-tree leaf node. The fields of "overflow page" and "next page"

no. of records	overflow page	next page	key1	record of key1	key2	record of key2	......

Figure 2: Layout of the leaf node records of a 2-D-S-tree.

```
struct xy_set {
        int             id;
        int             **rank;
        int             r_off;
};

struct Record {
        int             x_off;
        int             y_off;
        struct xy_set   *x_set;
        struct xy_set   *y_set;
};
```

Table 1: Internal Structure of A Record

are pointers pointing to the overflow page and the next node (sibling leaf node next to the current one), respectively. The key value here is a form of $A\theta B$, where A and B are symbols in V and $\theta \in \{<, =, :\}$.

The internal structure of a record is defined in Table 1, where x_off and y_off are the offsets of the sets of x_set and y_set, respectively. id is the identifier for each 2-D string (image) and r_off is the offset of $rank$. The $rank$ is a set of pairs of ranks for the two symbols in the key which are used to distinguish different types of queries.

The search algorithm first traverses the tree using pairs of consecutive symbols in the query 2-D string as the keys to the leaf node, where all their rank pairs reside. It then calculates rank pairs to determine the results for different types ($type$-0, $type$-1, and $type$-2) of queries. The search algorithm returns all the ids of the resultant images. A more detailed presentation of the 2-D-S-tree is provided in [8].

2.2 The Indexing Structure for The Salient Object Hierarchy

Depending on the image types, the salient objects are identified from the images by segmentation software or expert identification, and represented in the vocabulary set V. Objects in V can be further classified by using object-oriented modeling method. For example, for a vocabulary set $V = \{car, van, cat, dog\}$, it can be organized in the class hierarchy shown in Figure 3.

Each object in a class can have any or all of the following attributes (Table 2):

A number of indexing techniques have been proposed that exploit the class hierarchies [4, 5, 6, 7]. The scope of an object-oriented query can be restricted to a particular class (accessing the *shallow extent*), for example, targeting at cat only, or expanded to include some or all classes in the hierarchy rooted in that class (accessing the *deep extent*), targeting at $Animal$ and all its subclasses: cat and dog. Depending on the types of features, such as texture, shape and color, we have developed an indexing structure, h-structure, to organize the hierarchical relationships among objects. The basic idea behind the structure is to keep only one non-leaf level for all the classes with a *directory* (Figure 4) in it, which are used to store the information of the location of each individual class' leaf node pages. In this way, we are able to speed up the associative search not only for efficient retrieval of the instances from a single class, but also for efficient retrieval of the instances from a class and all its subclasses in the hierarchy. If the type of the indexing attribute (feature) is a one-dimensional vector (integer or character), the h-structure is similar to the CH-tree [4] (a directory is built in each leaf node of a B^+ like indexing structure). However, if the type of the indexing attribute (feature) is an n-dimensional vector, directories need to be integrated into leaf nodes of a R^*-tree [10] like structure.

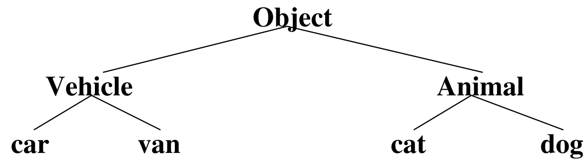

Figure 3: A Class Hierarchy.

```
Class Object {
        char*           name;
        void*           texture;
        void*           shape;
        void*           color;
        list*           ids; {all the image ids
                         which contain the object}
};
```

Table 2: An Abstract Object.

2.3 The $2D$-h Scheme

The $2D$-h scheme provides the functionalities to support the search over both the 2-D-S-tree and the h-structure. For ex-

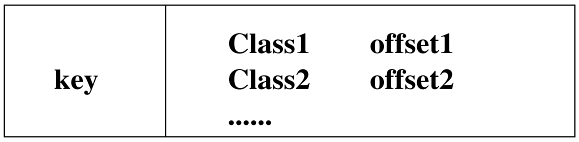

Figure 4: Class Directory.

ample, if we use the h-structure to index the salient object hierarchy using the feature "color" as the key, and use the 2-D-S-tree to index all the spatial relationships between two salient objects, the query systems allow queries of the types:

1. show all the images where a car is to the right of a $black$ dog.
2. show all the images where a car is to the right of any $black\ animal$.

For the first query, we use the h-structure to access the shallow extent to get all the images with a $black\ dog$, and use the 2-D-S-tree to get all the images where a car is to the right of a dog. The intersection of the resultant sets presents the final results.

For the second query, we do the same as we do for query 1 except we use the h-structure to access the deep extent to get all the images not only with a $black\ dog$ but with a $black\ cat$ as well.

The query can be represented as a 2-D string as well. For example, the following two 2-D strings can be used to search the image in Figure 1: $(tree < house : dog < plane,\ house : dog < tree < plane)$, and $(tree < house : Animal < plane,\ house : Animal < tree < plane)$. In this way, we can conduct queries on both shallow extent and deep extent in the class hierarchy.

Before we present the search procedure, let's use some examples to show what 2-D-h trees would look like and how it works. Suppose we have the following two 2-D strings to present two images:
$\{1, (car{<}van{=}cat{<}dog,\ car{<}cat{<}dog{=}van)\}$
$\{2, (car{<}dog{=}car{<}cat,\ cat{<}car{<}car{=}dog)\}$
and we also know the colors for the objects, say
$\{1, (car(b) < van(w) = cat(w) < dog(w),\ car(b) < cat(w) < dog(w){=}van)(w)\}$
$\{2, (car(w) < dog(b) = car(b) < cat(b),\ cat(b) < car(b) < car(w){=}dog)(b)\}$
The h-structure index and the 2-D-S tree for the two images are drawn in Figure 5 and 6, respectively.

Suppose a $type$-1 query is $(car(w) < dog(b),\ car(b) < dog(b))$. We first use $car{<}dog$ as the key to traverse the 2-D-S tree and find the leaf node to hold the record with $car{<}dog$ as its key in Figure 6. Then we fetch all the ids in x_set. Next, similarly we use $car{<}dog$ as the key to fetch all the ids in its y_set. The results should then be $x\_set \cap y\_set$, and they are $S_1 = \{1, 2\}$. We then go to the h-Structure to fetch all the image ids for $black\ car$, $white\ car$ and $black\ dog$. The result is $S_2 = \{2\}$. The final result for the query is $S_1 \cap S_2 = \{2\}$.

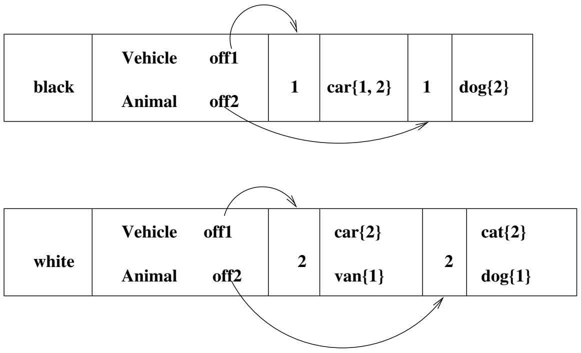

Figure 5: An Indexing Example of h-Structure.

Searching Procedure
$begin$
For a given query, $generate$ all its corresponding 2-D strings without the features attached to any symbols and without class names in them (by replacing class names with object names);
$Call$ the 2-D-S tree searching procedure for the above 2-D strings;
Put the results into a set S;
$Call$ the h-structure searching procedure
For each symbol which has a feature labeled {
$Call$ the h-structure searching procedure;
Put the results into a set S';
$S = S \cap S'$ };
$return\ S$;
end

Table 3: The Searching Procedure.

If a $type$-1 query is $(car(w) < Animal(b),\ car(b) < Animal(b))$, we then have to replace the class name $Animal$ with its all objects to generate corresponding query 2-D strings. They are:
$(car(w){<}cat(b),\ car(b){<}cat(b))$
$(car(w){<}cat(b),\ car(b){<}dog(b))$
$(car(w){<}dog(b),\ car(b){<}cat(b))$
$(car(w){<}dog(b),\ car(b){<}dog(b))$
Next we use the same procedure, as in the above example, to get all the image ids for each query 2-D string. The final result is $\{2\}$.

The searching procedure is outlined in Table 3.

3 Performance Analysis

We implemented the 2-D-h-trees using the EXODUS [11] storage manager $V3.0$. The experimental environment con-

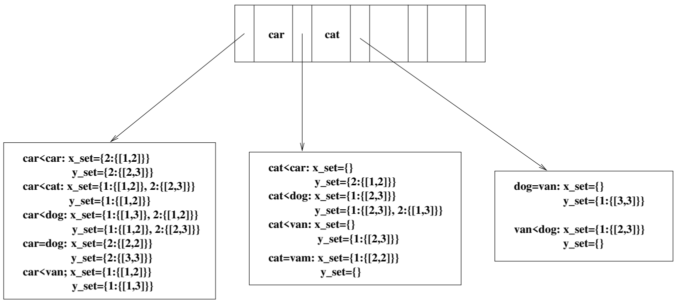

Figure 6: An Indexing Example of 2-D-S Tree.

Database Size	*type-0*	*type-1*	*type-2*
2000	3.550	1.060	0.970
5000	4.100	1.130	1.120

Table 5: Effect of The Database Size (seconds)

sisted of two Sun SPARCstation 5 in a client-server configuration. The implementation was done in C^{++} (eg^{++}). We conducted a number of experiments to study the query efficiency. Table 4 lists the parameters used in the cost analysis and the values assigned to those parameters used in the experimental evaluation. The default values for D, S and L are set to be 5000, 40 and 10, respectively. We generated the symbols S in all the experiments according to *normal* distribution. This means that some of the symbols would appear in more 2-D strings than others to reflect the reality that some of the objects do appear in more images than others. All the symbols are randomly grouped in 8 classes with a 2-level class hierarchy. For the h-structure, we used *hashing* instead of B^+ tree like structure to avoid unnecessary overheads since we only used 30 different colors as the indexing keys. Due to the space constraint, we would only report the query efficiency experiments for the second type of queries above since they are more general.

Table 5, 6, 7 show the results of effects of the database size (i.e., number of images), the number of different symbols and the 2-D string length. It seems that the performance is more sensitive to the 2-D string length. The longer 2-D string length indicates that more salient objects are identified for each image. This increases the length of the image ids' list (Table 2) and the number of the elements in the x_set and y_set (Table 1). This, in turn, increases the calculation costs of the intersections.

We should point out that the 2-D string scheme was primarily designed for iconic indexing [3], not for exactly depicting the original images. From this perspective, for a given image, if a 2-D string of the image can provide sufficient discrimination from other images in the database, it is considered "good". In other words, given the fuzziness of multimedia queries, we are generally satisfied if the system generates some false hits. For example, for the image in Figure 1, the 2-D string $(tree < house : dog = sun < plane,\ house : dog < tree < sun = plane)$ is good enough, since it presents the most important spatial relationships among objects in the image, and provides sufficient information to discriminate the image from other images. Notice that we ignored, in the 2-D string representation, some objects and their spatial relationships with others in the image, such as the $window$ and the $door$ on the wall of the house, the $grass$ on the ground, and the $bird$ on the roof of the house, since they are not as important as the others from indexing point of view. In general, if the vocabulary size is S, and the 2-D string length is L, then $(3^{L-1} * S^L)^2$ 2-D strings can be composed, which, in turn, can represent $(3^{L-1} * S^L)^2$ different images. For example, if a 2-D string length is 5, the 2-D string can discriminate itself from $(3 * S^2)^2 + (3^2 * S^3)^2 + (3^3 * S^4)^2 + (3^4 * S^5)^2$ 2-D strings. Therefore, the 2-D string length doesn't need to be very long when using 2-D-S-tree for indexing, if S is not too small (say less than 10).

The length of the query 2-D string is also an important factor for the efficiency of the 2-D-h-tree, especially when it accesses the deep extent of the class hierarchy. In [8], we proved that query 2-D strings do not need to be long, and their short forms (the first few symbols and their relationships) return the results as good as their original ones do, but with much better efficiency. Using the short form of a query 2-D string also yields less 2-D strings when it access the deep extent of the hierarchy and visits the h-structure with less number of objects. This enhances the efficiency of 2-D-h trees significantly for the following reasons.

First, the basic requirement of indexing in the image database is that it provides sufficient discrimination to prevent the retrieval of a large fraction of the database, although it may not produce only the exact match. Thus, partial queries are common.

Second, the main distinction between pattern matching and searching in the image database is that while exact pattern matching is the main criterion in pattern recognition, in database retrieval the objective is to help the user navigate through a large number of images. In fact, the users may not even be interested in the best match at all! They might use the retrieved patterns to further modify the search specifications (incremental queries). Thus, the constraints on pattern recognition are somewhat relaxed in the database environment, and efficiency in related to reducing the retrieval time, and robustness in terms of not missing targets (allowing some *false hits*, but no *false dismissals*) is a larger concern. Notice that "false hits" are acceptable, since they can be discarded easily through a post-processing step.

Finally, the possibility of false hits is very small if S is sufficiently large (say larger than 10), since the hits themselves are small.

Parameters	*Meaning*	*Values*
D	number of images (2-D strings) in database	2000, 5000, 10000
S	number of different symbols in V ($\|V\|$)	20, 40, 60, 200
L	maximum number of symbols in a 2-D string	7, 10, 15
$sz(N)$	node size (page size)	$4K$ bytes
$sz(P)$	number of bytes for a node pointer	16 bytes
$sz(K)$	number of bytes for the key	20 bytes
$sz(Ctr)$	number of bytes for the counter	4 bytes
$sz(Cid)$	number of bytes for the 2-D string id	4 bytes
$sz(Off)$	number of bytes for specifying the offset	4 bytes

Table 4: Parameters and notation

Vocabulary Size	*type-0*	*type-1*	*type-2*
20	4.750	1.800	1.700
40	4.100	1.130	1.120
60	4.020	1.110	1.110

Table 6: Effect of The Vocabulary Size (seconds)

Length	*type-0*	*type-1*	*type-2*
7	2.750	0.880	0.760
10	4.100	1.130	1.120
15	9.020	3.720	2.110

Table 7: Effect of The 2-D String Length (seconds)

4 Conclusions

Spatial features represent the spatial relationships among objects in an image, such as directional relationships, adjacency, overlap, and containment involving two or more objects. These are very important for content-based retrieval of images in an image database. In this paper, we present a new indexing scheme for content-based retrieval of images. The scheme, which we call $2D$-h trees, organizes the representations of the spatial relationships among objects in images and the hierarchical relationships among objects efficiently for query optimization. Our performance analysis indicates that the $2D$-h-tree is an efficient index scheme for content-based retrieval of images.

It is preferable to conduct the experiments on real image repositories. In the image database that is under development, we have collected more than 10,000 images, and we will implement the index scheme in the system to test its effectiveness in the future.

References

[1] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic and P. Yanker, The QBIC project: Querying images by content using color, texture, and shape, In *Storage and Retrieval for Image and Video Database*, v(1908), 173-187, SPIE, 1993.

[2] C. Faloutsos, *Searching Multimedia Databases by Content*, Kluwer Publisher, 1996.

[3] S. K. Chang, Q. Y. Shi, and C. W. Yan, Iconic Indexing by $2D$ Strings, *IEEE Trans. Pattern Analysis and Machine Intelligence*, 9(3), 413-428, 1987.

[4] W. Kim, K. C. Kim, A. Dale, *Indexing techniques for Object-Oriented Database*, In *Object-Oriented Concepts, Databases, and Applications*, Addison-Wesley, 371-394, 1989.

[5] C. C. Low, B. C. Ooi, and H. Lu, H-trees: A Dynamic Associative Search Index for OODB, In *Proc. of SIGMOD* 134-143, 1992.

[6] B. Sreenath and S. Seshadri, The hcC-tree: An efficient Index Structure For Object Oriented Databases, In *Proc. Intl. Conf. on Very Large Data Bases*, 203-213, 1994.

[7] C. Kilger and G. Moerkotte, Indexing Multiple Sets, In *Proc. Intl. Conf. on Very Large Data Bases*, 180-191,1994.

[8] Y. Niu, M. T. Özsu, and X. Li, 2-D-S-tree: An Index Structure for Content-Based Retrieval of Images, submitted to *The Fifth ACM International Multimedia Conference*, 1997.

[9] D. Comer, The Ubiquitous B-tree, *ACM Computing Surveys*, 11(2), June 1979.

[10] N. Beckmann, H.P. Keiegel, R. Schneider, and B. Seeger, The R^*-tree: An Efficient and Robust Access Method for Points and Rectangles, *Proc. ACM SIGMOD int'l Conf. on the Management of Data*, 322-331, 1990.

[11] M. J. Carey, D. J. DeWitt, J. E. Richardson and E. J. Shekita, Object and file management in the EXODUS extensible database system. In *Int'l Conf. on Very Large Database*, 91-100, 1986.