Region-based Indexing in an Image Database

Niels Nes, Martin Kersten
University of Amsterdam
{niels,mk}@wins.uva.nl

Abstract

Image retrieval systems based on the *image-query-by-example* paradigm locate their answer set using a similarity measure of the query image with all images stored in the database. Although this approach generally works for quick re-location of 'identical' or partly occluded images, it does not support the more interesting query type aimed at finding images with a particular image fragment. In this paper we introduce a region-based indexing scheme to support retrieval of images on the basis of both global and local image features.
keywords: Image Retrieval, Similarity measures, Region Features, and DBMS.

1 Introduction

With the advent of large image databases becoming readily available for inspection and browsing, it becomes mandatory to improve image database query support beyond the classical textual annotation and domain specific solutions, e.g. [?, ?]. An ideal Image DBMS would provide a data model to describe the image domain features and it would provide a query language to study domain specific algorithms with respect to their precision and recall capabilities. However, it is still largely unknown how to construct such a generic image database system.

Prototype image database systems, such as QBIC [?] and VisualSeek[?], have demonstrated some success in supporting domain-independent queries using global image properties. Their approach to query formulation (after restricting the search using using textual categories) is based on presenting a small sample of random images taken from the target set and to enable the user to express the query (Q_1) "find me images similar to this one" by clicking one of the images provided. Subsequently the DBMS locates all images using its build-in metrics over the color distribution, texture, or shape sets maintained.

This query evaluation technique is bound to fail for several reasons. First, it presupposes that the target image is stored in the database and that progressing from a small sample will lead to it quickly. This won't be true when the databases becomes large, such as envisioned for the Acoi image database[1]. Random sample sets to steer the query process will become confusing, because they likely lack an evident color, texture and shape relationship with the semantic domain of interest.

Second, the global image properties alone are not sufficient to prune false hits. Image databases are rarely used to answer the query Q_1. Instead, the user formulates a query (Q_2) :"find me an image that contains (part of) the one selected" where the containment relationship is expressed as a user controlled metric over selected features or directly (Q_3) :" find me an image that contains specific features using my own metric". In addition to color distribution and texture, spatial information about object locality embedded in the image is needed.

For example, in a large image database one could be interested to locate all images that contain part of the Coca-cola logo. This query could be formulated by clipping part of a sample Coca-colo logo to derive its reddish (R) and white (W) color and to formulate a query of the form:

select display(i)
from image_region r1,r2, image i
where distance(r1.avghue, R) < 0.2 **and** distance(r2.avghue, W) < 0.2
and r1.area **overlaps** r2.area **and** r1 **in** i
sort by distance(r1.avghue, R), distance(r2.avghue, W)

This query uses two primitive parameterized metric functions. The function *distance* calculates a distance in the color space and *overlaps* determines region containment. The former is defined as part of the **color** data type and the latter for the **region** data type. In principle, the DBMS should support overloading and refinement of this function by the user.

[1]Acoi is the experimental base for the national project on multi-media indexing and search (AMIS)

Finally, in an ideal image retrieval system, querying should be expressed in terms of semantic objects embedded. For example, a sketch should be sufficient to locate the objects of interest. However, automatic object detection and matching is beyond the scope of this project.

A challenge for database designers is to identify the minimal set of features and operators to accomodate such image query retrieval. In particular, since their derivation from the source image is often time consumings. This problem becomes even more accute when the envisioned database is to contain a million images.

In [?] we introduce a hierarchical indexing scheme to support localization of images according to query type Q_1. It confirmed that the average color of successively partitioned regions provide for a good handle to steer the query process. In this paper we extend this techique to accommodate queries of type Q_2 using both a split and a merge image indexing algorithm. The former recursively splits an image into smaller regions until their feature vectors dissimilarities fall below a cut-off point. The image objects thus considered are rectangular in shape. The latter uses a bottom-up strategy, i.e. rectangular regions are merged as long as their feature vectors are closely related.

The remainder of this report is organized as follows. In Section 2 we position our approach within the context of other prototype image database systems and provide a short introduction of the Monet database system for image retrieval. Section 3 explains the data model and the indexing algorithms. The experimental results are described in Section 4.

2 Image retrieval research

In this section we review several image database project to provide the background information and to identify the requirements imposed on image DBMS.

2.1 Image retrieval projects

The QBIC project [?] studies methods to query image databases based on the image content. The QBIC-web demo contains approximately 1,900 images. This demo supports queries on keywords and on the image contents color layout, color percentage and texture. Similarity is based on the Euclidean distance between two feature vectors. Only global features are used by this system. To improve efficiency, the search space is reduced using a lower bound metric on the color histogram distance. It has been proven that the average color is a lower bound of the color histogram distance [?], which does not result in missing actual hits, though extra false hits will be found.

The Photobook [?] provides a large amount of image processing functionality useful for content-based image retrieval. An example is the semantics preserving image compression technique, which reduces images to a small set of perceptually-significant coefficients. Using a training set of images, the "eigenimage" vectors are computed. These vectors are used to compress the image content information. The similarity between two images is computed using the distance in this compressed "eigenimage" space. This has been succesfull in face recognition.

The approach taken by Gevers [?] is to build histograms of the local Hue, the dominant Hue edges and Hue corners. The Hue color component is invariant to surface specularities like shadow and highlights. The similarity measure is the color histogram intersection [?], which is less variant to occlusion and less dependent on the view point. A web demo is available with about 500 images.

The VISUALSeek image retrieval system, as described in [?] segments the image into objects which have equal binary color content. The spatial information about these objects is also stored. Using both the spatial and color properties the user can query this database. A large database of 12,000 images is used in their web demo.

This short review of projects stresses the need for further experiments in storage and indexing image database. The experimentation platforms are rather poor with respect to scalability. Even the largest database considered contains information of just a few thousand images. Moreover, all systems are focussed on query type Q_1.

2.2 Image indexing techniques

Data structures for image feature indexing has received quite some research attention. The baseline is to replace the search key of an ordinary index structure by a feature vector and to include a proper comparison operator. For example, quad-trees can be easily extended to multilevel color histograms. Signature files, originally developed for textual information retrieval, have been extended by Faloutsos [?]. The trick is to use the important image features as signatures for the images. So fast retrieval can be achieved using the signature files.

Chang et al.[?] proposed a "2D-string representation" to encode the objects and their spatial relationships. Similarity retrieval of images encoded in 2D strings is mapped to substring matching. Nabil et.al. [?] use a graph-based encoding of the objects and their spatial relationships. Subsequently, retrieval is turned into a weighted graph-matching problem. A disadvantage of the approaches is that they do not easily fit a (relational) model. The challenge is to get a better representation and operators to support a broad spectrum of query types.

2.3 Image Database Systems

Our implementation efforts are focussed on Monet, a novel database kernel under development at CWI and UvA since 1993. Monet has been designed as a next generation system, anticipating market trends in database server technology. It relies on a network of workstations with affordable large main memories (> 128 MB) per processor and high-performant processors (> 50 MIPS). These hardware trends pose new rules to computer software – and to database systems – as to what algorithms are efficient. Another trend has been the evolution of operating system functionality towards micro-kernels, i.e. those that make part of the Operating System functionality accessible to customized applications.

Given this background, Monet was designed along the following ideas:

- *Binary relation storage model.* Monet vertically partitions all multi-attribute relationships in Binary Association Tables (BATs), consisting of `[OID,attribute]` pairs. This Decomposed Storage Model (DSM) [?] facilitates table evolution. And it provides a cannonical representation for a variety of data models, including an object-oriented model [?]. Moreover, it leads to a simplified database kernel implementation, which enables readily inclusion of additional data types, storage representations and search accellerators.

- *Main memory algorithms.* Monet makes aggressive use of main memory by assuming that the database hot-set fits into its main memory. For large databases, Monet relies on virtual memory management by mapping files into it. This way Monet avoids introducing code to 'improve' or 'replace' the operating system facilities for memory/buffer management. Instead, it gives advice to the lower level OS-primitives on the intended behavior[2] and lets the MMU do the job in hardware. Experiments in the area of Geographical Information Systems[?] and large object-oriented applications [?] have confirmed that this approach is performance wise justified.

- *Monet's extensible algebra.* Monet's Interface Language (MIL) is an interpreted algebraic language to manipulate the BATs. In line with extensible database systems, such as Postgres, Jasmine and Starburst, Monet provides a Monet Extension Language (MEL). MEL allows you to specify extension modules to contain specifications of new atomic types, new instance- or set-primitives and new search accelerators. Implementations have to be supplied in C/C++ compliant object code.

3 System overview

In this section we explain the data model, indexing schemes, and query primitives for the image retrieval applicaton. The input for the image database is gathered from the Web using a robot aimed at finding multi-media objects.

3.1 Acoi database scheme

The core of the Acoi database schema is illustrated in Table 1. The basic building block is the Binary Association Table (BAT), which provides the data structure to maintain functional mappings between values from arbitrary domains.

The first BAT group in Table 1 illustrate the administration of multi-media objects located on the Web. For each object keywords are automatically extracted from their context. It is used for pre-selection based on textual information. Observe that their URL is sufficient to gain access upon need. The second BAT group contains features obtained from the source, i.e. information part of the image representation format. The final group contains features derived to support region-based querying. This encompasses both primary (average hue) and secondary features. The img_region BAT enumerates the regions in each image. Their spatial relationships are reflected in an R-tree. The remaining BATs represent features derived to support image querying.

To facilitate ease of manipulation, Monet was extended with an atomic type image and with image processing functionality for feature extraction.

3.2 Region Indexing

The key challenge is to develop an efficient algorithm to locate the regions of interest for a given image. Two kinds of algorithms are considered; a top-down method based on recursive splitting, called R-split, and a bottom-up method based on successive merging, called R-merge. No attempt is made to detect or infer hidden faces. Neither

[2]This functionality is achieved with e.g. `mmap()`, `madvise()`,and `mlock()` Unix system calls.

# BAT **name**	**Comments**	# BAT **name**	**Comments**
mmo_url	document resource locator	img_size	image width x height
mmo_name	document base name	img_depth	pixel depth
mmo_kind	object kind {audio,image,video}	img_stamp	derived icon
mmo_type	object type {gif,tiff,jpeg}	img_icon	user defined icon
mmo_cntxt	link to enclosing document	img_region	image-region mapping
mmo_time	last access time	ir_color	region average hue
mmo_censor	flags (copyrighted, X-rated)	ir_hue	dominant hue
txt_keyword	keywords for an MMO	ir_texture	dominant angle
txt_phrase	key phrase for an MMO	ir_area	region area

Table 1: Database schema

do we consider a search for optimal segmentation schemes common in image recognition research. We conceive the index primarilly as a filter for applications dealing with image understanding.

The algorithm R-split finds the collection of discriminating regions by recursively splitting the image into two sub-images. Splitting is attempted both horizontally and vertically. Sub-images are chosen such that their dis-similarity in average Hue is maximal. This improves the selectivity of the individual regions.

The recursive process is controlled by several stop criteria as follows.

- Let I_i be an image split into two regions $I_{i,1}$ and $I_{i,2}$, Then both new regions are added to the *img_region* index provided their average Hue differs from I_i more then a given minimal threshold $H_{threshold}$. This guards against storing redundant information into the database.

- The maximal number of regions per image is limited by a system parameter, $H_{objects}$. This guards against repeatedly splitting images up to the pixel level. Instead, we assume that a limited number of regions (possible dependent on the image size) is often sufficient.

The complexity of this algorithm is $O(d*n^2)$ with n the maximum image width or height and d is equal to $H_{objects}$. This is the worst case complexity. Usually, d will be less than $H_{objects}$ because of the first stop condition.

The algorithm R-merge attempts to merge regions into larger units. The algorithm starts by dividing the original image into equal sized regions using a grid layout. Each grid element is a candidate region for inclusion in the *img_region* index. The minimal grid size considered is H_{grid} pixels. Subsequently, we repeatedly attempt to combine regions into larger units as follows. Let I_i and I_j be two regions. Then they are merged into a single region I_k if the following criteria hold.

- The average Hue of both regions I_i and I_j differ at most by a given constant $H_{threshold}$.

- Both regions share at least one edge.

- The merge is locally optimal.

This process continues until no more regions can be merged. The complexity of this algorithm is $O(n^2 ln(n))$, with n the number of regions to start with.

There are large differences between the two algorithms considered. The R-split algorithm uses a simple region representation, since splitting a rectangle always results in two new rectangles. Conversely, R-split needs a polygon to follow the object boundaries.

An advantage of R-merge is that it enables reuse the hue average. The nature of splitting does not allow us for such reuse of intermediate results at all. At each stage we have to inspect all pixels. A potential disadvantage of R-merge would be the large number of polygons to start with. The split algorithm starts with only one rectangle.

In Section 4 we study the performance cost to gain a better understanding of the inscalability. Both algorithms are based on the same a similarity function. They merely differ in its interpretation. The similarity function calculates the weighted distance between the features in that region. The similarity for a single feature of two regions is calculated using the following function, $S(R_1, R_2) = (\frac{(F_{R_1} - F_{R_2})}{weight})^2$. The function $S(R_1, R_2)$ computes the similarity between the regions R_1 and R_2 using the primary feature F.

4 Experimental results

4.1 Database construction

Construction of the Acoi database with one million images requires a step-wise approach, because its construction is both cpu and storage intensive. Therefore, we conducted two kinds of experiments. First, we determine the resource requirements for the indexing algorithms on a small 500-image database. Second, the web-robot is used to take a sample to assess scalability.

The 500-image database is our standard database for image analysis research. As such is provides a reference point for the algorithms in terms of precision later on. We fed a sample of 100 256x256 sized images to both R-split and R-merge to determine the average number of regions in an image. This depends on the algorithmic parameters $H_{threshold}$, $H_{objects}$ and H_{grid}.

The left of figure 1 illustrates that R-merge should start with not too small a grid size. The right figure illustrates that $H_{objects}$ should be set to 32, since splitting deeper will generally not result in more regions. It also illustrates that R-split finds more regions.

Based on these experiments we can predict the storage and processing requirements for the complete database. The region administration consumes 18 bytes in the current implementation. This should be multiplied by the min$\{ H_{objects}, R\}$ where R is the actual number of regions determined by the algorithm. With a starting grid size of a single pixel the average number of regions found by R-merge is less than 200, i.e. approximately 4Kb to store the region features and index structures. R-split leads to many more objects and requires about 9Kb per image. This leads to an index size of about 4 Gbyte.

In addition, we need space for the global features, e.g. URL, keywords and key-phrases, and secondary features, e.g. histogram and texture. To assess the size and to confirm the index resource requirements, we used the web-robot to obtain the first sample of about 1K gif images from the NL domain. The last table of 2 shows the BAT sizes of this 1000 images large database. It indicates that far less than 200 regions are found per internet image. This means that our 500-sample is an upperbound for the storage requirements. The table also shows the number of keywords and the distribution of multi-media objects. The storage requirements of the icon is 7.5 K and about 2.5 for the remainding features, leading to a total of about 15 Gbyte.

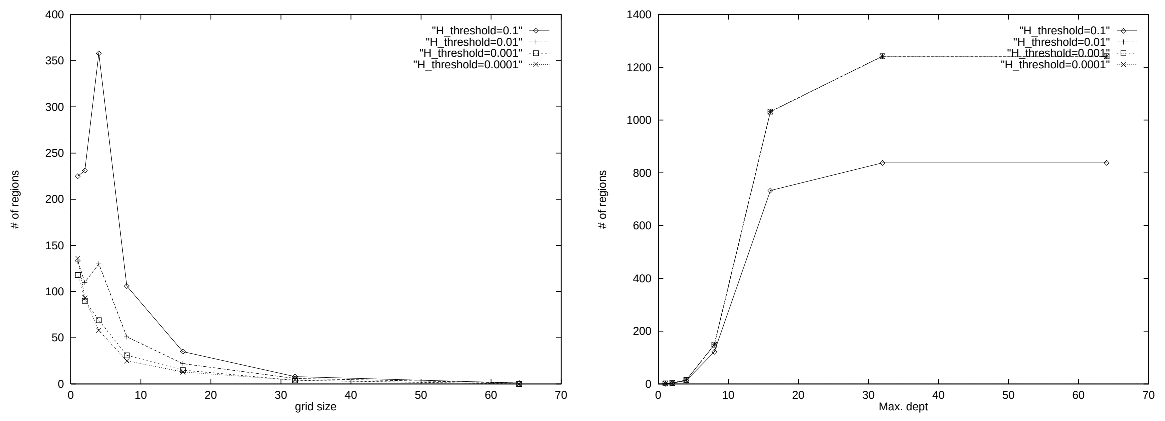

Figure 1: The number of regions using the R-merge and R-split algorithm

The final question to consider at this stage is whether creation of the Acoi database won't take forever. To quantify this, we ran a small experiment on 100 images to determine the wall-clock for the complete process. The first two tables of table 2 show the timing results for the R-merge and R-split using different threshold values.

R-split and R-merge dominate the insertion cost, e.g. with a grid size of 4x4 pixels R-merge takes less 3 seconds. Since localization and download of the candidate images can take place in the background this enables downloading of 30K images per day.

5 Conclusions

In this paper we have introduced the necessary data structures and operators to build a high performance image database system aimed at supporting embedded image querying. We have experimentally demonstrated that a bottom-up index construction outperforms a top-down approach terms of storage requirements and performance. The storage overhead for the region feature index of an image is about 4 Kbytes.

# of	**Time(s)**	
H_{grid}	0.1	0.0001
1	69.2	35.9
2	12.3	7.9
4	2.3	2.2
8	0.9	0.9
16	0.4	0.5
32	0.2	0.3
64	0.1	0.1

# of	**Time(s)**	
$H_{objects}$	0.1	0.0001
1	1.9	1.8
2	2.1	2.1
4	2.5	2.6
8	3.3	3.5
16	4.2	4.9
32	4.3	5.2
64	4.3	5.0

# BAT **name**	**count**
mmo_url	1002
mmo_name	1002
txt_keyword	17340
txt_phrase	612
img_region	9042
ir_hue	9042
ir_texture	9042
ir_area	9042

Table 2: Experimental results

Based on these figures we expect that a full-blown incarnation of the Acoi image database takes 15 Gbyte disk space and about one month of Web scouting. This database will become publically available as a national resource for further studies in image indexing and image database technology in Spring 1997.