Tile-based Image Visual Codewords Extraction for Efficient Indexing and Retrieval

Zhiyong Zhang and Olfa Nasraoui
University of Louisville
US

1. Introduction

Inspired by the success of inverted-file indexing in text search engines, content based image retrieval (CBIR) researchers have begin to consider using inverted indexing for image content search Squire et al. (1999) Jing et al. (2004). For inverted indexing to be effective in image content search, two issues need to be addressed. First, the image content features need to follow a power-law distribution the same way as textual document terms do. Second, a mechanism is needed to *textualize* the low-level image content features. We addressed these two constraints. First of all, we tested the sparseness of image content features and our results confirmed the sparseness assumption. This makes the inverted file indexing on image content well grounded. However, to translate to image content feature into texts, a vector quantization (VQ) scheme is needed. K-means (or Generalized Lloyd Algorithm) clustering is usually used for vector quantization. But for sparse data, K-means algorithm may generate cluster centers overcrowded in high density area. To solve this problem, we developed a cluster-merge algorithm to guarantee the quality of cluster centers. Based on these preliminary efforts, we are able to *textualize* the image content features. More specifically, we developed the *textualization* procedures for image content features, which include both codebook generation and codewords extraction process. To better describe image textures and image shapes, we defined three types of tiles: *inner tiles*, *bordering tiles*, and *crossing tiles*. Furthermore, to reduce the dependence on accurate image segmentation for shape-based retrieval, we propose to use more basic elements, *boundary angles*, to represent the shape feature. With these efforts and methods, we are able to enjoy high level visual codewords representation of low-level image features, thus making efficient and scalable image content search realizable.
This chapter is organized as follows: in section 2, we discuss related work. In section 3 and section 4, we discuss the sparseness of image content features and the cluster-merge algorithm respectively. We focus on image content codewords representation in section 5. After that, we present our experimental results in section 6, and finally we draw conclusions and give future works in section 7.

2. Related Works

QBIC Flickner et al. (1995), Blobworld Carson et al. (1999), WALRUS Natsev et al. (2004), and WBIIS Wang et al. (1997), which uses the QBIC system, all use *R*-trees* Beckmann et al. (1990) for indexing. The *R*-tree* is a variant of the *R-tree* by Guttman Guttman (1984). These

tree-based method claim to have good performance for nearest neighbor search. However, *curse of dimensionality* spells potential trouble for these tree-based methods. In Weber et al. (1998), Weber et al showed that the performance of these tree-based methods degrade significantly as the number of dimensions increases and is even outperformed by a sequential scan whenever the dimensionality is above 10. Weber et al then proposed a scheme named *vector approximation file* (or '*VA-file*') and demonstrated that *VA-file* can offer better performance for high dimensional similarity search. An *OVA-file* (*Ordered VA-file*) structure, which places *approximations* that are close to each other in close positions for later fast access, was later used in Lu et al. (2006) for video retrieval. The *VA-file* structure adopts a signature-file like filtering method, thus trying to build a mechanism for fast scanning for nearest neighbors search. However, in Witten et al. (1999) Zobel et al. (1998), Zobel et al showed that signature files are distinctly inferior to inverted files as data structure methods for indexing. At the same time, the success of inverted files in texts search engine and scalability concerns have drawn attention to the potential of using inverted files for multimedia indexing. Viper Squire et al. (1999) had attempted to use inverted files for indexing image databases and their experiment results showed their inverted indexing scheme had better performance than the vector space system used before by the same authors. In Jing et al. (2004), Jing et al tried to use a modified inverted file for image indexing and showed through experiments that inverted file indexing is much more efficient than sequential search without much loss of accuracy. In Rahman et al. (2006), Rahman et al also used inverted files for image indexing and observed comparable accuracy results for inverted file indexing and sequential search method. In order to build an inverted index, image content features need to be transformed into textual words. Vector Quantization (VQ) is usually used to achieve such goal. A Uniform Quantizer has been used in a image indexing and search system Zhang et al. (2006). However, since there are several advantages of using Nonuniform Quantization such as an improved Signal to Noise Ratio (SNR) (see Gersho & Gray (1992)) over Uniform Quantization, Nonuniform quantization are more widely used. Among the Nonuniform Quantization methods, the Generalized Lloyd Algorithm (GLA), also known as k-means or LBG algorithm, is widely used for generating image codebooks for indexing or retrieval. See Ma & Manjunath (1998) Sivic & Zisserman (2003) Jegou et al. (2007) Mojsilovic et al. (2000) and Malik et al. (1999).
As image segmentation technology develops from theoretical Ncut Shi & Malik (2000) to practical efficient Kruskal style realization Felzenszwalb & Huttenlocher (2004), people can expect the successful landing of shape-based image retrieval in real life in the near future. Shape based image retrieval are divided into two categories: *boundary-based* and *region-based* Safar et al. (2000). Among several different approaches, the grid-based method Lu & Sajjanhar (1999), which belongs to region-based methods, was later used by Prasad et al. (2004). There are several factors that make shape-based image retrieval very challenging and that have limited development of shape-based image retrieval. First, shape-based retrieval relies too much on an accurate image segmentation, which is not a solved problem yet. Second, it is very hard to build a shape-based retrieval system which is invariant to transformation, rotation, and scaling. A few systems like Lu & Sajjanhar (1999) have considered these factors, however they paid high computation costs to solve these problems. Third, humans don't have a uniform definition of the shapes of different objects other than some very common and simple shapes such as triangle, rectangle, and circle, etc. Several working system of query by shape include CIRES Iqbal & Aggarwal (2002), which used perceptual grouping and several image structural boundary types like "L" junctions, "U" junctions, "polygons", etc to differentiate

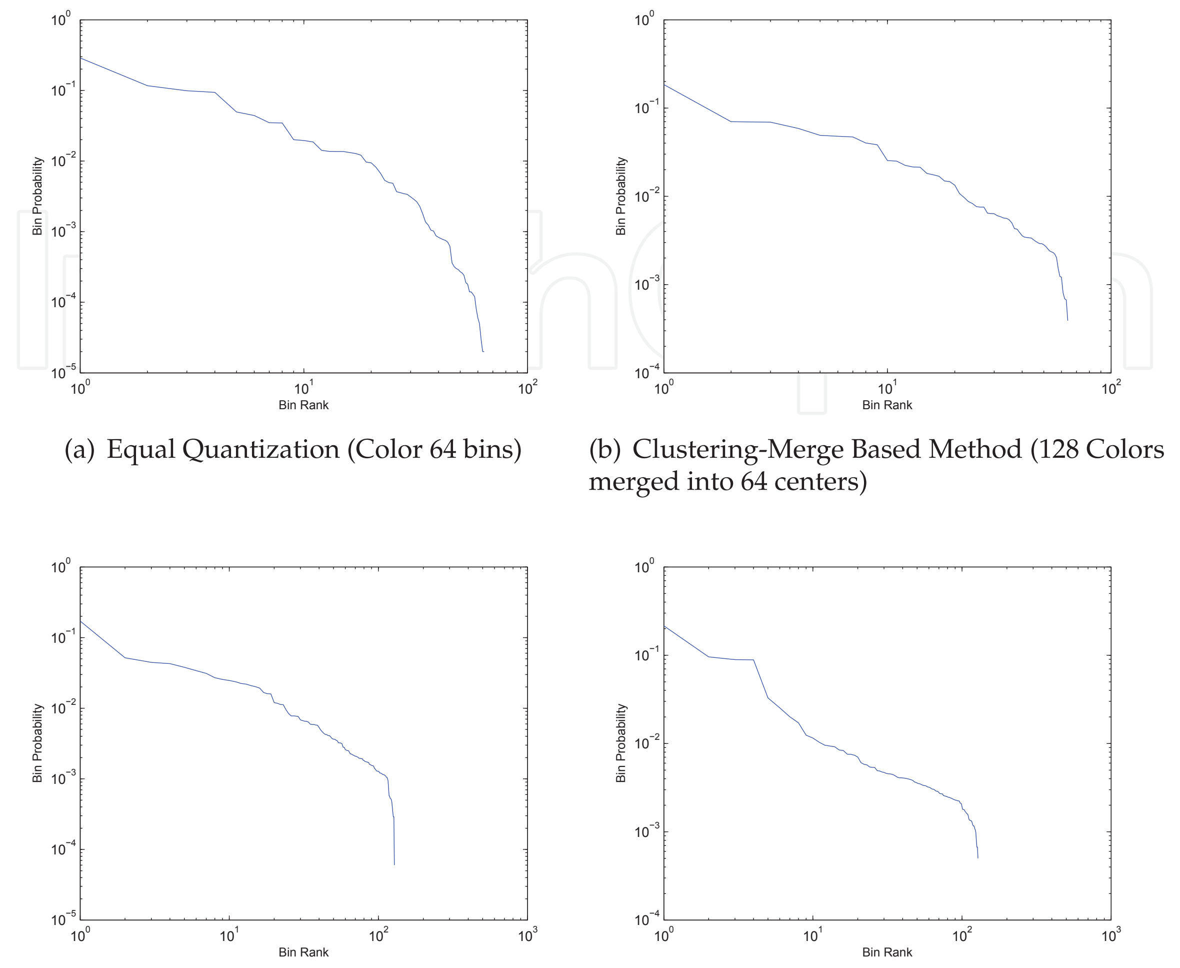
manmade objects from outdoor images. No image segmentation was used in their methods. Another work is *query by sketch* such as Retrievr Jacobs et al. (1995).

3. Sparseness of Image Features

One assumption and reason behind the success of the inverted file structure for indexing text and documents is the sparseness of terms in documents. The total number of terms in a large document collection is very high with dimensionality $O(10^4)$ Squire et al. (1999) Westerveld (2000), while the number of terms that occur in a single document is comparatively low $O(10^2)$. In other words, human have accumulated a big enough vocabulary to cover each field of our everyday life, yet for a specific topic or article, a small portion suffices. However for the case of images, since "a picture is worth a thousand words" and since pictures tend to be self-explanatory, it can be hard to develop a complete thesaurus for images. The result is that a *sparse* dictionary comparable to documents does not exist. However, an analysis of images based on its color, texture, and shape, shows that although a term-document like high sparseness is not achieved, a slightly lower sparseness could be reached with an appropriate content to codeword mapping. For instance, a picture may contain only a few dominant colors when using a color histogram containing hundreds of bins. The sparseness also depends on the visual word representation.
Figure 1 gives the log-log distribution of the resulting colorwords (mapped from original colors) using different methods to represent 10,000 actual images randomly selected from flickr.com. For equal color quantization, we use $4 \times 4 \times 4$ quantization on the RGB color channels to get 64 bins. For the clustering-merge based method, we use 1000 images as the training set to obtain the global color or texture codebook, then use the global codebook to represent these 10,000 images.
We can see that these representations roughly follow a power law distribution. In Jurie & Triggs (2005), Frederic et al showed that texture features followed a power law distribution. Their results match our experimental results, and lay the foundation for our inverted indexing scheme. In the following, we will discuss our clustering-merge method to compute the global codebook.

4. GLA and Cluster-Merge Algorithm

There are two reasons for us to use GLA (Generalized Lloyd Algorithm) vector quantization instead of uniform quantization. The first reason is that the latter is not feasible for high-dimensional features, whereas GLA can be used in such cases. The second reason is that the latter does not take the actual feature distribution into account and can lose some accuracy in representing the images, while GLA does better.
One issue with most K-mean-based clustering algorithm such as GLA for codebook generation is that for power law distributed data, more cluster centers are generated around high density areas Jurie & Triggs (2005). Our experiments also confirmed such phenomenon, see figure 3. This can cause the codewords to be overpopulated in the high density area while the sparse area not properly represented. To solve this problem, we propose to merge these very close cluster centers after obtaining the K-means clustering result. We first use K-means to generate more centers than what we expect, then we merge these centers into the number of centers we desire. Through merging close centers, we can avoid cluster centers to crowd into high density areas. At the same time, we fully take advantage of the efficiency of K-means.



(a) Equal Quantization (Color 64 bins)

(b) Clustering-Merge Based Method (128 Colors merged into 64 centers)

(c) Clustering-Merge Based Method (256 Colors merged into 128 centers)

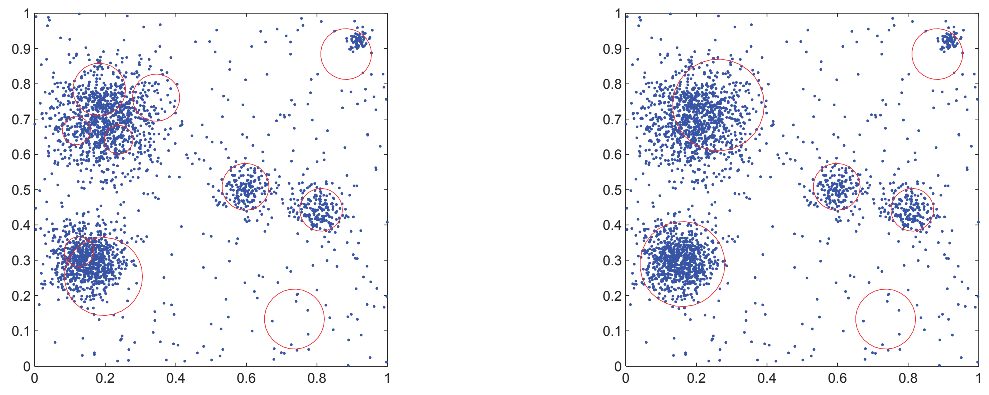
(d) Clustering-Merge Based Method (256 Textures merge into 128 centers

Fig. 1. Log-log plots of different quantization methods.

Thus, our clustering merge algorithm will be much more efficient than purely hierarchical clustering algorithm.
The pseudocode for merging clusters is shown in pseudocode 2. In pseudocode 2, we check two constraints, the number of clusters and the distance between two closest clusters. Once we obtain the desired number of clusters or if the distance between two closest clusters becomes greater than a preset threshold value, we terminate the merging and return the results.
Figure 3 gives a two-dimension visualization of how our cluster-merge algorithm works. For Figure 3, we use 2,541 manually synthesized 2D data points for K-means clustering test. We manipulate these data points to make it roughly follow certain degree of sparseness. The two big clusters on the left are used to simulate the high-density area, while the three small clusters in the right are used to simulate the sparse area. For visualization purpose, we scale the red circle proportionally smaller to cover only a portion of the clusters while the center of the red circles show the real cluster centers. The left figure shows the k-means clustering result by setting k equals to 10, we can see more cluster centers are crowded into the high density area. After our cluster-merge algorithm, the resulting 6 clusters are satisfactory to our demand.

```
**Input**: n clusters
**Output**: k clusters
**MergeCluster**(n,k,r)
if(k>=n)
    return n; //already <= k clusters
while(k<n)
    do
        find two closest centers c1, c2 among n clusters
        if(dist(c1,c2) > r)   //radius limit
            break;
        merge c1, c2 into c
        insert c into the list of cluster centers
        delete c1 and c2 from the list of cluster centers
    n = n - 1
return n
```

Fig. 2. Pseudocode for merging clusters.



(a) k-means cluster (k=10) (b) cluster-merge result

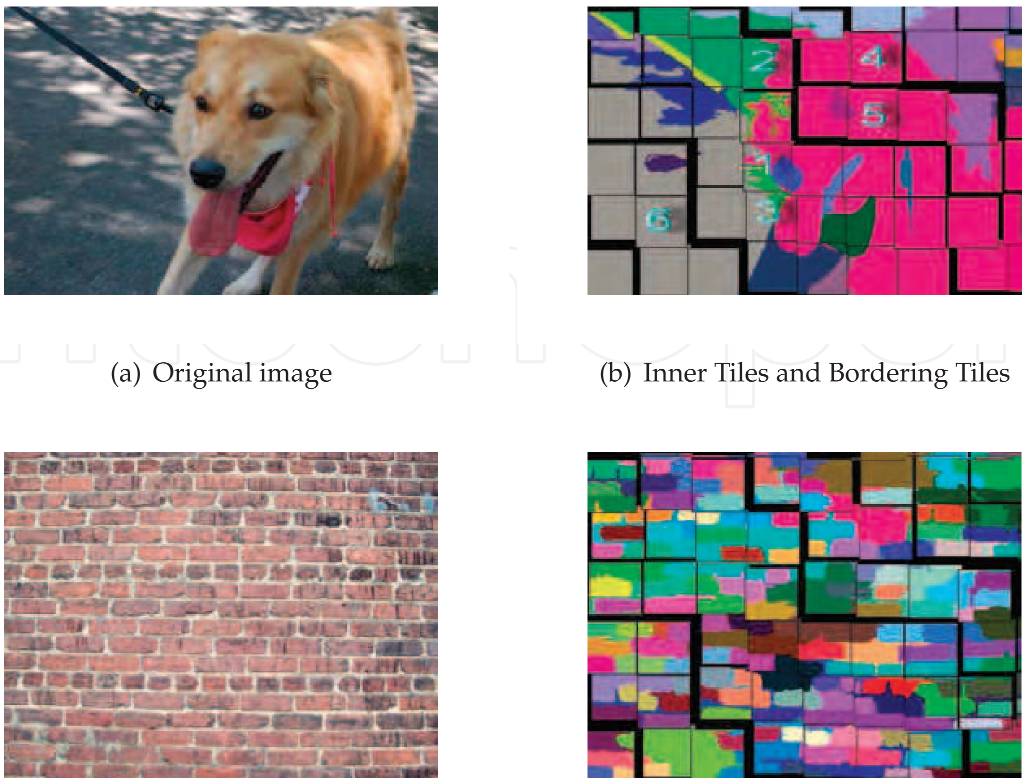
Fig. 3. Merge close clusters (setting radius < 0.1732).

5. Codebook Design and Feature Extraction

In this section, we are going to use the clustering-merge algorithm discussed in seciton 4 in image codebook design. As image color codebook design is very straightforward, we are not going to include it in this chapter due to page limit. We only discuss image texture and shape codebooks.

5.1 Inner, Bordering and Crossing Tiles

Before we discuss texture words and shape words, we will define several relevant tile types that will support extracting such features. Figure 4 shows how we divide an actual image from into small (64×64) tiles for further feature extraction. Notice that if we divide the original image into small grids (as in the top right image), some small tiles (tile 4, 5, and 6, etc) will have uniform texture patterns since they are totally inside one object or inside the background. We will name these tiles *inner tiles* as they lay totally inside an object or the background. On the other side, tiles such as tile 1, 2, and 3 don't have uniform texture patterns as they

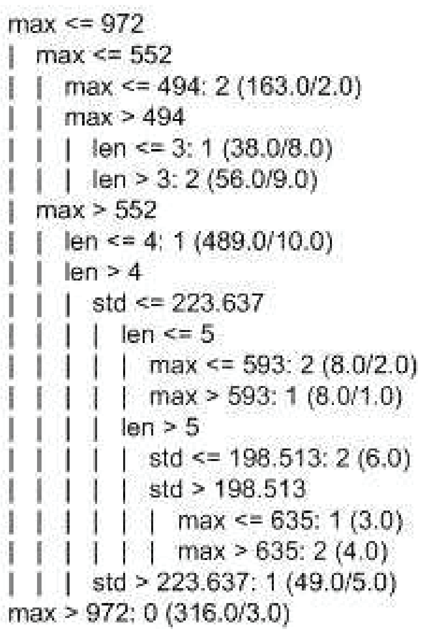


(a) Original image (b) Inner Tiles and Bordering Tiles

(c) Original Image (d) Crossing Tiles

Fig. 4. Inner tiles (4, 5, 6), Bordering tiles (1, 2, 3), and Crossing tile (right bottom image).

contain multiple conceptual areas (the dog fur and the road). We name these tiles *bordering tiles* as they tend to border a big block and some other small blocks. *Inner tiles* are very helpful in identifying the texture pattern of an object, while *bordering tiles* are useful for identifying shapes or contours of objects. *Inner tiles* and *bordering tiles* are not sufficient to characterize image tiles. For instance, when we segment and divide the brick wall image (in Figure 4), we cannot get even one tile (right figure) which purely belongs to one big block or borders one big block. To deal with such case, we define another type of tile: *crossing tile*. By definition, it contains many small blocks without any pixel belonging to any big block. Such segmentation results often mean that the image is a texture image. *Crossing tiles* will be very meaningful in identifying texture patterns. For the above three types of tiles, we will select *inner tiles* and *crossing tiles* for image texture extraction and select *only bordering tiles* for shape analysis.
In order to take advantage of tiling, we need to be able to decide whether an image tile is an inner tile, bordering tile, or crossing tile. To differentiate between these three kinds of tiles, we train a simple decision tree classifier (C4.5) to learn how to automatically classify a tile into the correct type. We use three features: number of components (*len*), number of pixels of the maximum component (*max*), and standard deviation (*std*) of the number of pixels of each component in the segmented tile, and three class labels: inner tile (0), bordering tile (1), and crossing tile (2). We first use 1000 tiles as a training set. For these 1000 color segmented tiles, we manually label each segmented tile as inner tile, bordering tile, or crossing tile. After training and 10-fold cross-validation, we get the following classification model (with 93.8596



```
max <= 972
|   max <= 552
|   |   max <= 494: 2 (163.0/2.0)
|   |   max > 494
|   |   |   len <= 3: 1 (38.0/8.0)
|   |   |   len > 3: 2 (56.0/9.0)
|   max > 552
|   |   len <= 4: 1 (489.0/10.0)
|   |   len > 4
|   |   |   std <= 223.637
|   |   |   |   len <= 5
|   |   |   |   |   max <= 593: 2 (8.0/2.0)
|   |   |   |   |   max > 593: 1 (8.0/1.0)
|   |   |   |   len > 5
|   |   |   |   |   std <= 198.513: 2 (6.0)
|   |   |   |   |   std > 198.513
|   |   |   |   |   |   max <= 635: 1 (3.0)
|   |   |   |   |   |   max > 635: 2 (4.0)
|   |   |   std > 223.637: 1 (49.0/5.0)
max > 972: 0 (316.0/3.0)
```

Fig. 5. Decision Tree Model for Classifying Tiles.

% Correctly Classified Instances). For image segmentation, we use the graph based method proposed by Felzenszwalb & Huttenlocher (2004).

5.2 Image Texture Codebook

We use Gabor texture feature extraction to generate the image texture codebook. However, because we first need to get the global image texture dictionary from a set of training texture images, instead of clustering the filtered image output *pixel-wise* for only one image as in Malik et al. (1999), we will perform clustering *tile-wise* on an entire *set* of training images. This way, each texture image can be subdivided into small *tiles* which corresponds to certain *texture words*. We analyze texture tile-wise instead of textel-wise because this corresponds to our visual understanding of images: textures can make sense to us only when they are displayed in a *region* and thus form a pattern.

Furthermore, we do not use all the segmented components for texture analysis. Instead we use only the inner tiles and crossing tiles. This is because first of all, image segmentation is not very accurate especially in the bordering part separating two components or blocks. Another reason is that sampling inner tiles instead of all the segmented blocks can help reduce the computational cost and raises the possibility of saving even more computations by only computing the texture features of those tiles that are not similar to any other analyzed tiles. For instance, before we compute one tile's texture feature, we can check whether most of its pixel values are similar to the previous neighboring tile in the same block by taking the pixel value difference. If we find that the tiles are very similar, then we can simply classify this tile into the same pattern as the previous tile and avoid computing its texture features.

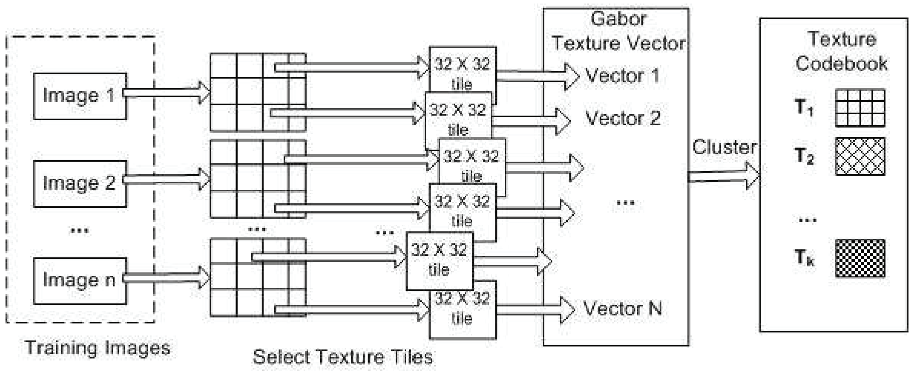


Fig. 6. Process of Generating Image Texture Codebook.

To compute the texture codebook, we divide each image into 32 × 32 pixel blocks or tiles, select texture tiles (inner tiles and crossing tiles) according to image segmentation results, then apply Gabor filters to each texture tile, and finally record the corresponding texture features. The process for generating the image texture codebook is summarized in Figure 6. We use 1000 texture pictures for training (each picture roughly contains 160 (32 × 32 pixel) tiles with more than half of the tiles are selected as texture tiles). This process yield 128 texture words (compared to Ma & Manjunath (1998) which yields 950 codewords).

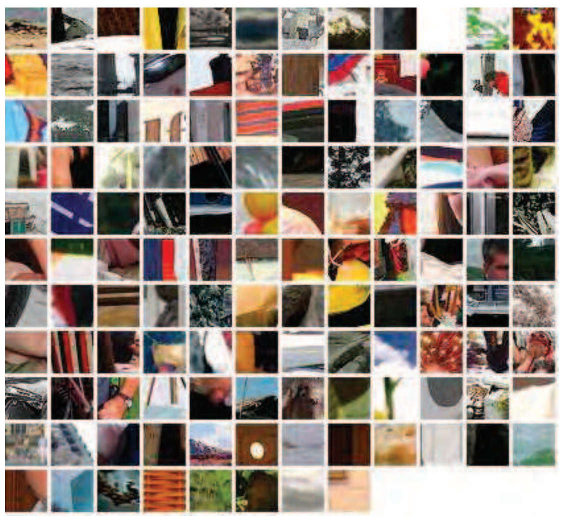


Fig. 7. Example Texture Codebook.

The clustering method used is GLA followed by the cluster-merge algorithm, as discussed above. Figure 7 shows an example texture book generated using the above procedure.

5.3 Extracting and Indexing Texture Features

When we try to extract image texture words, we apply Gabor filters to small 32 × 32 image texture tiles. After extracting the image texture feature vector, we use Nearest Neighbor (accelerated by a Kd-tree structure) to determine the corresponding texture codeword from the texture codebook.

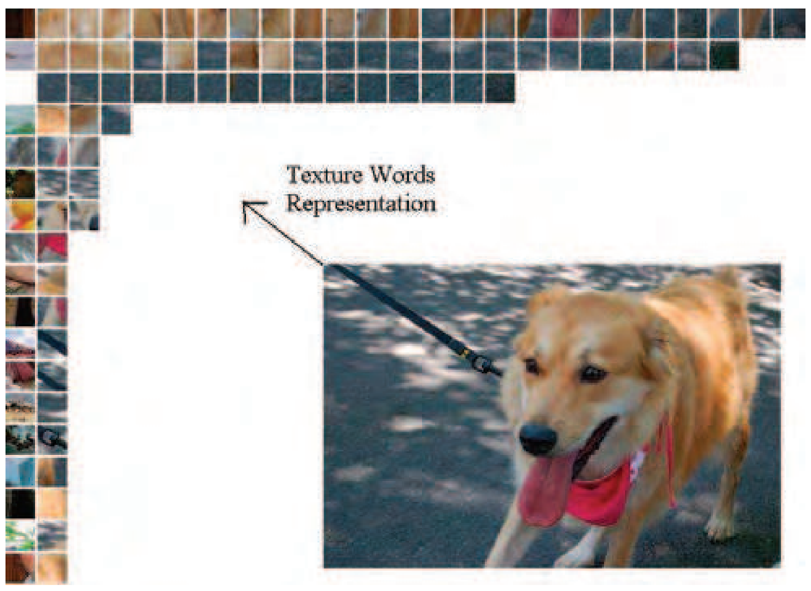
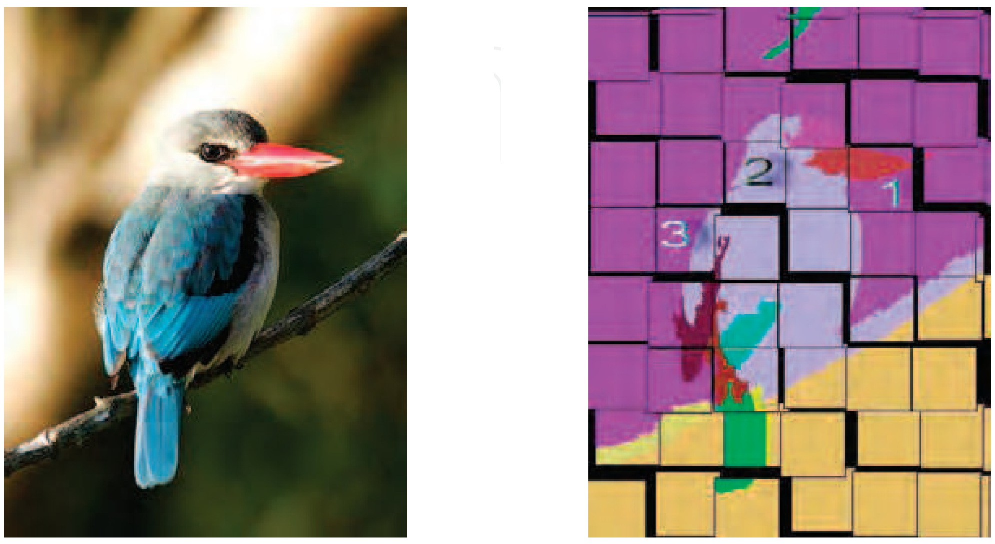


Fig. 8. Extracting Texturewords from An Image.

Figure 8 gives an example of extracting texture words from an actual image. The leftmost column contains the representative tiles selected from the codebook in Figure 7, while the remaining tiles are the corresponding image tiles extracted from the actual image. In the process of extracting texturewords from an actual image, we adopt a similar procedure, where we segment the image and select texture tiles (inner tiles and crossing tiles) according to the segmentation results using the decision tree model show in model 5. We use nearest neighbor search method to find the representative texture word in the texture codebook to represent each texture tile.

5.4 Image Boundary Angle Codebook

In this section, we propose a novel method of image retrieval that is very similar to shape-based image retrieval. We name this new type of retrieval *boundary angle-based image retrieval* or *BABIR*. In section 2, we have reviewed several shape-base image retrieval systems, and their limitations. Considering all of the drawbacks, we propose to use an important part of the shape boundaries, *boundary angles*, to represent a shape. For instance, we can use two 90 degree angles or even four straight lines to represent a rectangle. These boundary angles seem to agree with human's atomic understanding of shapes, and constitute the basic elements in drawing a picture. This simplification has important consequences. First, while it can be hard for an image segmentation system to identify the accurate shape, most image segmentation systems would easily identify the boundary lines or angles. Second, boundary-angle-based retrieval relies on basic elements like straight lines or turning angles or arcs which are actually scale and rotation invariant. As discussed in section 5.3, we will use *bordering tiles*, for boundary-angle-based retrieval. Figure 9 gives another example showing these tiles, where we can see that it is very hard for an image segmentation scheme to render the whole bird shape as one component. However, we are able to segment image for the boundary-angle contours to extract boundary-angle words.



(a) Original image (b) Segmented Tiles

Fig. 9. sample boundary angles tiles (1, 2, 3).

For shape feature, we will adopt the region-based shape representation proposed in Lu & Sajjanhar (1999) and Prasad et al. (2004) with slight modification. The region based method

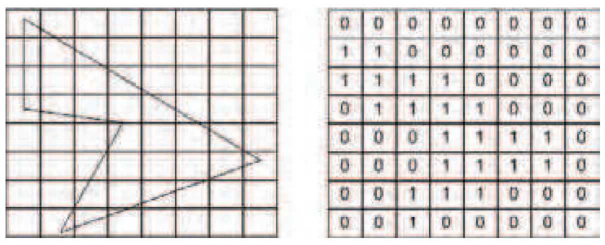


Fig. 10. Region based shape representation.

can be illustrated in Figure 10, where a grid with fixed-size square cells was used to cover the shape region. "1" and "0" are assigned to the small cells in which the shape covers above or below 25% of the cell respectively. This allows representing the shape using the binary sequence: 00000000 11000000 11110000 01111000 00011110 00011110 00111000 00100000. To adapt the grid-based method to our application, we further grid each image tile into $8 \times 8 = 64$ small pixel cells. Instead of using "0" and "1" to represent each cell, we use integer value ranging from "0" to "16" to represent each cell, where the value of the cell is a count of image pixels belonging to the dominant component of the image tile. This sequence of 64 counts will represent the shape of each boundary angle tile, and is called the *shape vector* of the image tile. Then we perform clustering and cluster-merging in the same way as we did for image

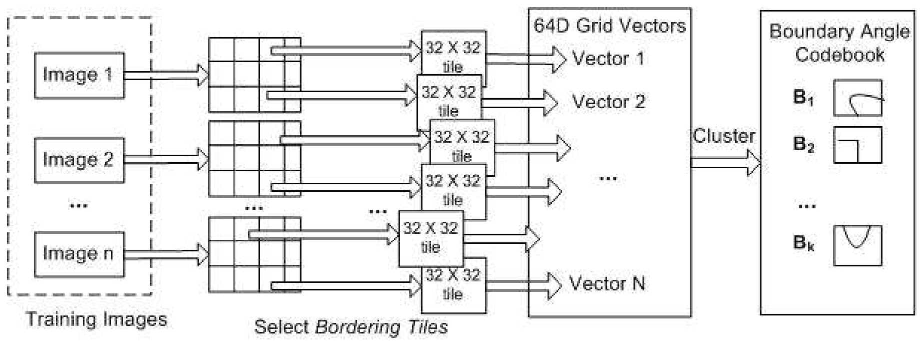


Fig. 11. Process of Generating the Boundary-Angle Codebook.

textures. Here, as well, we use a training set (e.g., 1000 shape images) and use tile selection, GLA method and the cluster-merging algorithm to generate a shape codebook. This process is summarized in Figure 11, while Figure 12 shows an example of a boundary angle codebook generated using our method.
Once we have obtained the boundary angle codebook, during the image crawling phase, we adopt a procedure that is similar to extracting texture codewords (Kd-tree-based Nearest Neighbor mapping) to extract image-boundary-angle code words. Figure 13 shows an example of extracting boundary angle words from an image.

6. Experimental Results

6.1 Effectiveness of Texture Word Representation

To demonstrate the effectiveness of our tile-based image representation, we will compare the retrieval precision of using image texture tiles with that of using the whole image texture vectors. Our assumption is that although we use a much faster boolean query consisting of several representative image texture words, we can get a comparable retrieval precision to that of using the whole image texture vectors for similarity search, which the general CBIR systems would use. To evaluate the precision, we adopt a similar strategy as was used in SIMPLIcity Wang et al. (2001). We use a subset of COREL database with 10 categories shown in table 1, where each category contains 100 semantically coherent images. Altogether, there are 1000 images in total for testing. In the codebook generation stage, we collect 100 images

Africa	Beach	Buildings	Buses	Dinosaurs
Horses	Flowers	Elephants	Food	Mountains

Table 1. COREL Categories of Images Tested

by randomly selecting 10 images from each category. We use these 100 images to generate a global image texture codebook. After we build the image search by implementing Nutch[1]

[1] http://lucene.apache.org/nutch/

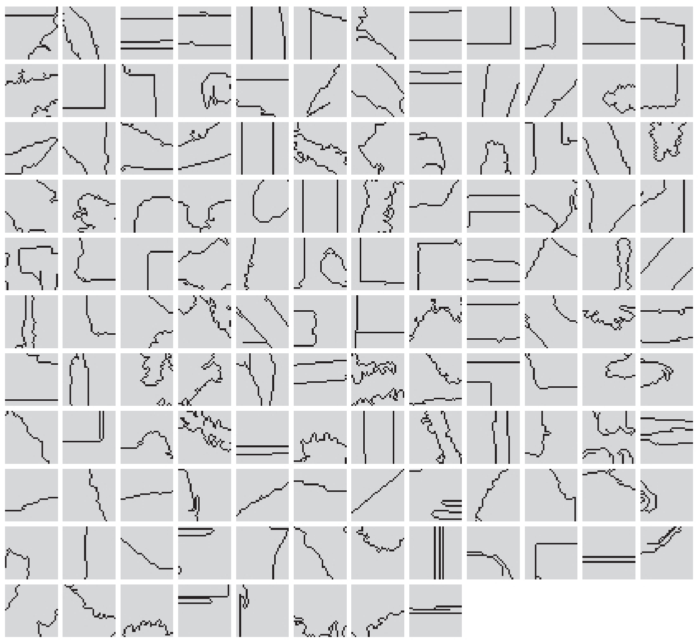


Fig. 12. Example Boundary Angle Codebook.

on image texture feature, we conduct our testing by randomly selecting three images from each category as query images (30 queries for each case). To form the actual queries, for the texture words representation case, we select top-N (N ranges from 1 to 10) texture words for the query image to form the boolean query and we use Nutch's default $TF \times IDF$ ranking. For the texture vector query case, we take the whole 48 dimension texture vector for the query image to form the query and use Euclidean distance measure for ranking. We show 10 results in each page and examine the number of category matches in the first page. In case the total number of results is less than 10, we show all the results in the first page. We then calculate the precision as the number of category matches in the first page divided by the number of results in the same page. We then average the precision of the 30 result-sets and compare the two types of methods. The results is shown in Figure 14.
As shown in Figure 14, the precision for Texture words boolean query case increases as the number of query terms increases. As the number of query terms approaches 5 or 6, the average precision of boolean query comes close to the vector similarity query case. On the other hand, the number of returned results drops dramatically as the number of query terms increases as shown in Figure 15. For instance, the average number of returned results drops from 13.5 to 6.6 as the number of terms in the query increases from 5 to 6. Although the average precision is high as we include much more terms (say 9 or 10), the very few number of results returned should prevent us from using too many terms. Combining Figure 14 and Figure 15, we can see that selecting around 5 query terms would give us a balanced results of desirable precision and total number of results.
On the other end, the retrieval efficiency benefit of using boolean query over vector similarity query is obvious, as shown in Figure 16 and the experiments are with the Linux server (with

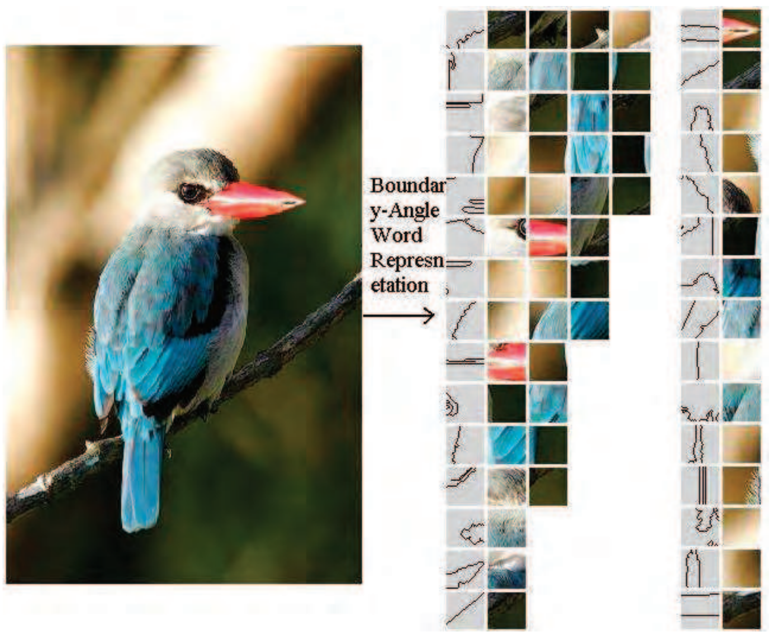


Fig. 13. Extract Boundary-Angle Words.

Intel(R) Xeon(TM) CPU 2.80GHz, 2cpu, and 2G memory). We also can see that as the number of terms contained in the boolean query increases, the retrieval speed does not fluctuate much. This is guaranteed by the inverted indexing structure.

6.2 Effectiveness of on Boundary-angle Based Retrieval

Since we have argued for using boundary-angle instead of whole shape for image retrieval, we need to empirically justify this choice. For this experiment, we use the Columbia image testing set, which contains 7200 images of 100 objects traditionally used to evaluate shape-based retrieval, and measure the precision of retrieval. We adopt a similar procedure to get the global boundary-angle codebooks. We first randomly select 7 images for each object and get 700 images in total for training the boundary-angle codebook. Then we use Nutch to build the inverted index and conduct the testing by two types of search. For each object, we randomly choose 2 images to form the query images and altogether we get 200 query images. To form the boundary-angle boolean query, we use top-N (N ranges from 1 to 5) boundary-angle words and to form the shape vector query, we use the 64D shape grid vector. Again, we consider the returned results that correspond to the same object as accurate matches. We compare the precision of using the two methods. We can see from Figure 17 and Figure 18 that when we choose a boolean query that contains 3 or 4 terms, we can get very close precision as the vector nearest-neighbor query case without losing too much recall. Note the average number of returned results is 29.2 for 3-term boolean query and 6.6 for 4-term boolean query. Also, we can see the obvious efficiency benefit of using boolean queries from Figure 19.

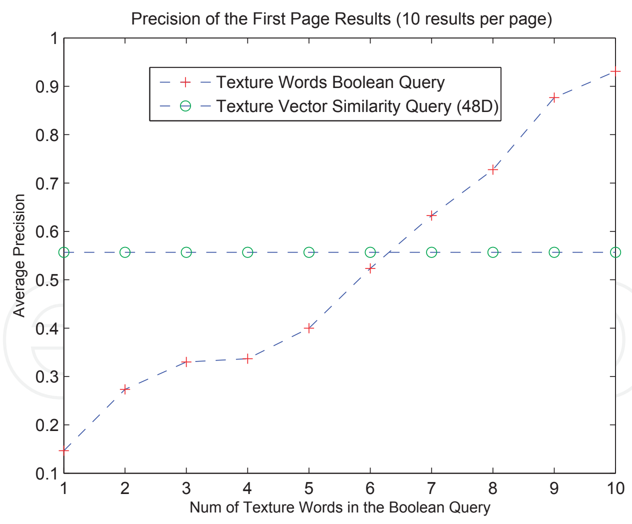


Fig. 14. Texture Query Precision Comparison.

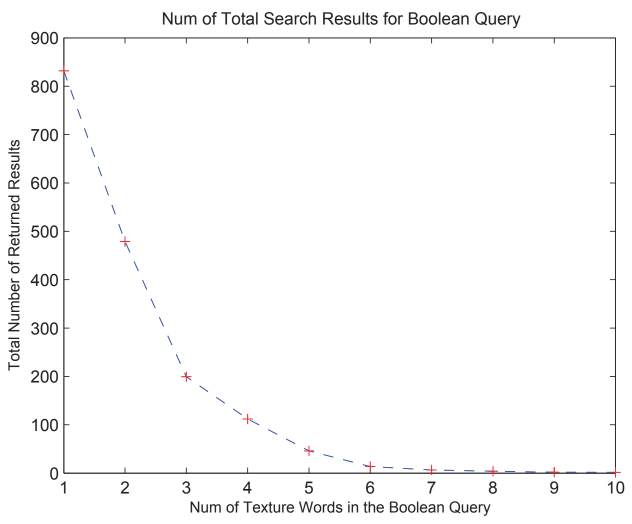


Fig. 15. Number of Results Returned (Texture).

7. Conclusions and Future Works

We have shown through our experiments that image content features follow the power-law spare distribution. Then we take advantage of such sparseness for inverted file indexing. Our clustering-merge algorithm gave us better cluster center representations over GLA clustering, which generated cluster centers over-crowdedness in high density area for sparse data. Our experimental results shows our tile-based image *textualization* process for image texture and shape features are effective. Such techniques can be used for scalable and efficient content-based image retrieval and search systems.

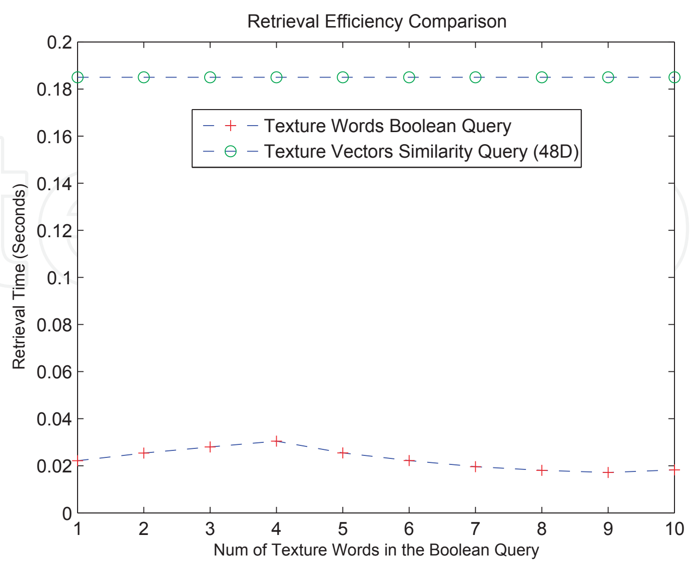


Fig. 16. Texture Retrieval Efficiency Comparison.

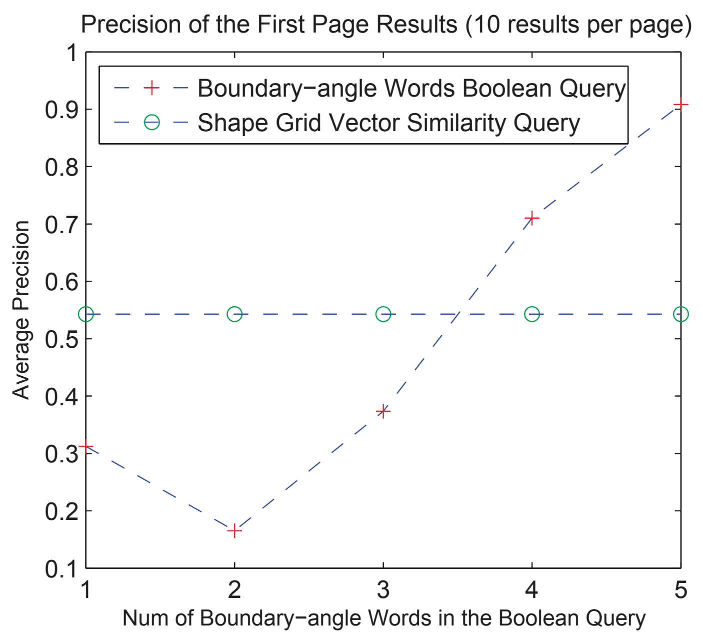


Fig. 17. Shape Query Precision Comparison.

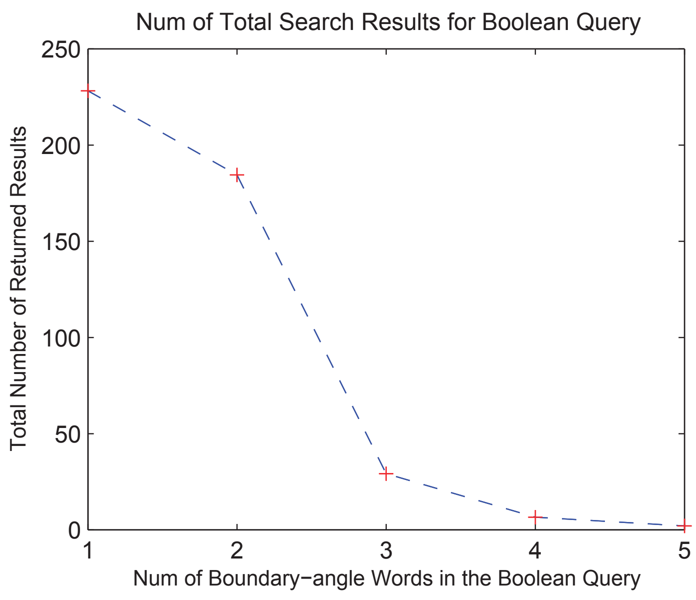


Fig. 18. Number of Results Returned (Shape).

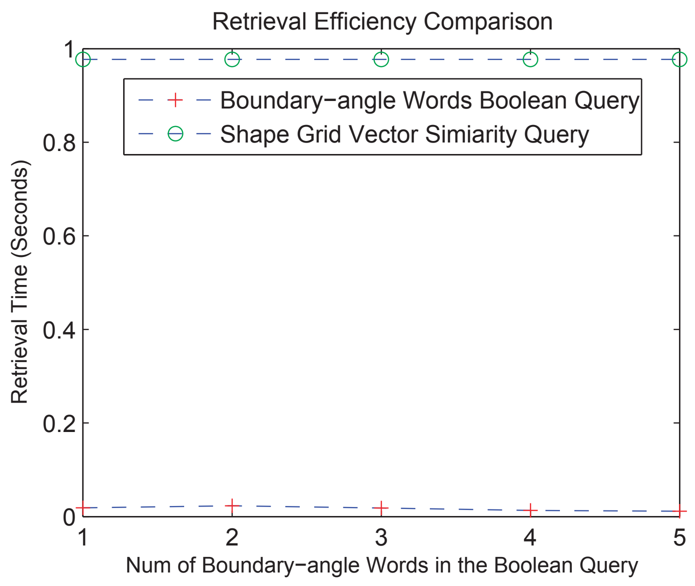


Fig. 19. Shape Retrieval Efficiency Comparison.

8. References

Beckmann, N., Kriegel, H. P., Schneider, R. & Seeger:, B. (1990). The r*-tree: An efficient and robust access method for points and rectangles, *SIGMOD Conference*, pp. 322–331.

Carson, C., Thomas, M., Belongie, S., Hellerstein, J. & Malik, J. (1999). Blobworld: A system for region-based image indexing and retrieval, *Proc. Third International Conf. Visual Information Systems*, pp. 509–516.

Felzenszwalb, P. F. & Huttenlocher, D. P. (2004). Efficient graph-based image segmentation, *Int. J. Comput. Vision* **59**(2): 167–181.

Flickner, M., Sawhney, H., Niblack, W., Ashley, J. & et al (1995). Query by image and video content: the qbic system, *IEEE computer* **28**(9): 23–32.

Gersho, A. & Gray, R. M. (1992). Kluwer Academic Publishers.

Guttman, A. (1984). R-trees: A dynamic index structure for spatial searching, *ACM SIGMOD International Conference on Management of Data*, pp. 47–57.

Iqbal, Q. & Aggarwal, J. (2002). Retrieval by classification of images containing large man-made objects using perceptual grouping, *Pattern Recognition* **35**: 1463–1479.

Jacobs, C. E., Finkelstein, A. & Salesin, D. H. (1995). Fast multiresolution image querying, *Computer Graphics* **29**(Annual Conference Series): 277–286.

Jegou, H., Harzallah, H. & Schmid, C. (2007). A contextual dissimilarity measure for accurate and efficient image search, *IEEE Conference on Computer Vision & Pattern Recognition*.

Jing, F., Li, M., Zhang, H. & Zhang, B. (2004). An efficient and effective region-based image retrieval framework, *IEEE TRANSACTIONS ON IMAGE PROCESSING* **13**(5).

Jurie, F. & Triggs, B. (2005). Creating efficient codebooks for visual recognition, *ICCV '05: Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, IEEE Computer Society, Washington, DC, USA, pp. 604–610.

Lu, G. & Sajjanhar, A. (1999). Region-based shape representation and similarity measure suitable for content-based image retrieval, *Multimedia Syst.* **7**(2): 165–174.

Lu, H., Ooi, B. C., Shen, H. T. & Xue, X. (2006). Hierarchical indexing structure for efficient similarity search in video retrieval, *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING,* **18**(11): 1544–1559.

Ma, W.-Y. & Manjunath, B. S. (1998). A texture thesaurus for browsing large aerial photographs, *Journal of the American Society for Information Science* **49**(7): 633–48.

Malik, J., Belongie, S., Shi, J. & Leung, T. K. (1999). Textons, contours and regions: Cue integration in image segmentation, *ICCV (2)*, pp. 918–925.

Mojsilovic, A., Kovacevic, J., Hu, J., Safranek, R. J. & Ganapathy, K. (2000). Matching and retrieval based on the vocabulary and grammar of color patterns, *IEEE Trans. on Image Processing* **9**(1): 38–54.

Natsev, A., Rastogi, R. & Shim, K. (2004). Walrus: A similarity retrieval algorithm for image databases, *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING* **16**(3): 301–316.

Prasad, B. G., Biswas, K. K. & Gupta, S. K. (2004). Region-based image retrieval using integrated color, shape, and location index, *Comput. Vis. Image Underst.* **94**(1-3): 193–233.

Rahman, M. M., Desai, B. C. & Bhattacharya, P. (2006). Visual keyword-based image retrieval using latent semantic indexing, correlation-enhanced similarity matching and query expansion in inverted index, *10th International Database Engineering and Applications Symposium (IDEAS'06)*, pp. 201–208.

Safar, M., Shahabi, C. & Sun, X. (2000). Image retrieval by shape: A comparative study, *IEEE International Conference on Multimedia and Expo (I)*, pp. 141–144.

Shi, J. & Malik, J. (2000). Normalized cuts and image segmentation, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **22**(8): 888–905.

Sivic, J. & Zisserman, A. (2003). Video Google: A text retrieval approach to object matching in videos, *Proceedings of the International Conference on Computer Vision*, Vol. 2, pp. 1470–1477.

Squire, D., Muller, W., Muller, H. & Raki, J. (1999). Content-based query of image databases, inspirations from text retrieval: inverted files, frequency-based weights and relevance feedback.

Wang, J. Z., Li, J. & Wiederhold, G. (2001). SIMPLIcity: Semantics-sensitive integrated matching for picture LIbraries, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **23**(9): 947–963.

Wang, J. Z., Wiederhold, G., Firschein, O. & Wei, S. X. (1997). Content-based image indexing and searching using daubechies' wavelets, *International Journal on Digital Libraries* **1**: 311–328.

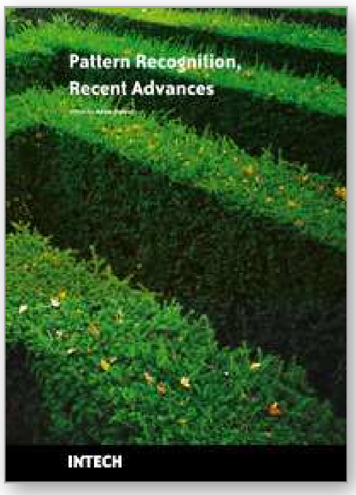
Weber, R., Schek, H.-J. & Blott, S. (1998). A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces, *Proc. 24th Int. Conf. Very Large Data Bases, VLDB*, pp. 194–205.

Westerveld, T. (2000). Image retrieval: Content versus context, *Content-Based Multimedia Information Access, RIAO*, p. 276ÍC284.

Witten, I. H., Moffat, A. & Bell, T. C. (1999). *Managing gigabytes (2nd ed.): compressing and indexing documents and images*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Zhang, Z., Rojas, C., Nasraoui, O. & Frigui, H. (2006). Show and tell: A seamlessly integrated tool for searching with image content and text, *In Proceedings of the ACM-SIGIR Open Source Information Retrieval workshop*.

Zobel, J., Moffat, A. & Ramamohanarao, K. (1998). Inverted files versus signature files for text indexing, *ACM Transactions on Database Systems* **23**(4): 453–490.



Pattern Recognition Recent Advances
Edited by Adam Herout

ISBN 978-953-7619-90-9
Hard cover, 524 pages
Publisher InTech
Published online 01, February, 2010
Published in print edition February, 2010