PERGAMON

Pattern Recognition 35 (2002) 2479–2488

PATTERN RECOGNITION

THE JOURNAL OF THE PATTERN RECOGNITION SOCIETY

www.elsevier.com/locate/patcog

Image database indexing using JPEG coefficients

Sharlee Climer, Sanjiv K. Bhatia*

Department of Mathematics and Computer Science, University of Missouri—St. Louis, St. Louis, MO 63121, USA

Received 4 December 2000; received in revised form 5 September 2001

Abstract

Image database indexing is used for efficient retrieval of images in response to a query expressed as an example image. The query image is processed to extract information that is matched against the index to provide pointers to similar images. We present a technique that facilitates content similarity-based retrieval of JPEG-compressed images without first having to uncompress them. The technique is based on an index developed from a subset of JPEG coefficients and a similarity measure to determine the difference between the query image and the images in the database. This method offers substantial efficiency as images are processed in compressed format, information that was derived during the original compression of the images is reused, and extensive early pruning is possible. Initial experiments with the index have provided encouraging results. The system outputs a set of ranked images in the database with respect to the query using the similarity measure, and can be limited to output a specified number of matched images by changing the threshold match. © 2002 Pattern Recognition Society. Published by Elsevier Science Ltd. All rights reserved.

Keywords: Image database systems; Content based image retrieval; Indexing

1. Introduction

The adage “A picture’s worth a thousand words” reflects the rich semantic meaning conveyed by an image. This semantic complexity confounds the efforts to automate image indexing and retrieval. While humans can easily discern objects in an image, we are not able to unambiguously describe its full semantic content in a language that can be tokenized for automatic indexing and retrieval. At the present time, the technology has not advanced to the point for a computer to discern an image or to unambiguously describe its contents. This is due to the limitations of current edge detection and object recognition techniques. Yet, the need for efficient automated retrieval systems has dramatically increased in recent years due to the world-wide web and multimedia technology.

In the past, small image data bases were maintained manually. A domain expert would identify the objects in an image and enter relevant text to be associated with the image. This method has several drawbacks. The semantic complexity of an image can lead to different descriptions, resulting in content and/or language mismatches [1,2]. A content mismatch occurs when a seemingly insignificant object or characteristic is omitted in an annotation and, later, a user searches for that omitted information. Language mismatches occur when different words are used to describe the same object. Another drawback is that many applications, such as weather forecasting or law enforcement, involve a significant number of similar images that would result in an unmanageable number of matches for a given query. Finally, this method is severely limited as it is time intensive and thus impractical for large data bases.

In this paper, we present a technique to develop an index for an image database using the JPEG coefficients of the compressed image. The use of JPEG coefficients

* Corresponding author. Tel.: +1-314-516-6520; fax: +1-314-516-5400.

E-mail address: sanjiv@cs.umsl.edu (S.K. Bhatia).

obviates the need for uncompressing the image and only requires decoding the run-length encoded strings of coefficients. The index is based on partitioning the JPEG coefficients of an image into a quad-tree structure [3].

In the next section, we present the problem background in detail and review the literature. In Section 3, we present the basics of JPEG compression, with an emphasis on the phase where we pick up the coefficients. In Section 4, we show the development of the quad tree corresponding to the DCT coefficients and the information to be retained at each node of the quad tree. In Section 5, we illustrate the query process. Section 6 describes the implementation status of the system. We conclude by presenting a summary and future plans regarding the system in Section 7.

2. Background

In recent years, many content-based image retrieval (CBIR) methods have been developed to address the growing need for efficient image management systems. Generally, CBIR systems define methods for extracting image characteristics, known as *signatures*, from images and employ rules for comparing images based on these signatures. In most of these systems, the user will supply or construct a query image or enter text to initiate a search for matching images. Generally, these systems return a number of images with the closest match to the query.

Most CBIR systems can be roughly categorized into one of three areas: histogram-based, color layout-based, or region-based. Some systems, such as QBIC [4], WALRUS [5], and our system, have characteristics of more than one category. In the remainder of this section, we present descriptions of each of the three categories and conclude with an overview of our techniques.

2.1. Histogram-based systems

A histogram of an image conveys the relative quantities of colors in an image. Inside the machine, each pixel in the image is represented as a weighted amount of three colors: red, green, and blue. Each of these three color channels is quantized into m divisions. Thus, we can have m^3 composite colors, or bins. The total number of pixels that correspond to each bin are tallied and the signature of the image is a vector containing these totals. The signature is then used to build an index. When a query image is presented, its signature is extracted and compared with entries in the index.

The histogram-based systems suffer from several limitations. First, these systems disregard the shape, texture, and object location information in an image, leading to a high rate of return of semantically unrelated images. For instance, the histograms of the two images in Fig. 1 are identical, yet the images are obviously different. Furthermore, the color quantization step leads to additional

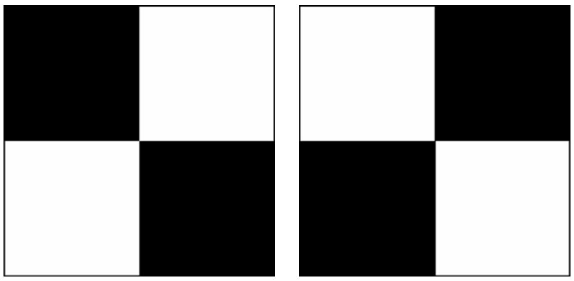
Fig. 1. Two images with opposite colors, yet identical histograms and average intensities.

sources of error [6]. For efficiency, there usually are far fewer bins than colors, so many colors may get quantized into a given bin. The first problem is that similar colors that are near the division line may get quantized into different bins and for a given bin, the extreme colors may be quite different. Second, given a set of three color channels, perceptual sensitivity to variations within colors is not equal for all three channels [7]. However, histogram quantization incorrectly uses a uniform divisor for all three channels. Finally, a query image may contain colors that are similar to the colors of a particular image in the index, but a large distance may result if they are not close enough to fall into the same bins [6].

2.2. Color layout-based systems

Color layout-based systems extract signatures from images that are similar to low resolution copies of the images. The image is divided into a number of small blocks and the average color of each block is stored. Some systems, such as WBIIS [8] and WALRUS [5], utilize significant wavelet coefficients instead of average values in order to capture sharp color variations within a block.

Signatures derived in the WBIIS system include coefficients derived from a fast wavelet transform with Daubechies' wavelets and their standard deviation [8]. The query process involves a filtering step followed by distance computations. In the first step, the standard deviations of the query image and each of the indexed images are compared. Next, a weighted Euclidean distance measurement is calculated on a 192 dimensional vector within each signature of the remaining images. Finally, the images with distances within a specified threshold are used to calculate a more precise distance measurement on a 768 dimensional vector [8].

A limitation of traditional color layout systems is their intolerance of object translation and scaling. Object location is frequently helpful in identifying semantically similar images. For instance, the histogram of a query image containing green grass and blue sky may have a close distance to a blue house with a green roof, while a color layout scheme would discard this image. However, tolerance of object translation is often desirable. The WALRUS system uses a wavelet-based color layout method to overcome the translation and scaling limitation [5]. For each image in the index, a variable number of signatures (typically thousands) are computed and clustered. Each signature is computed on a square area of the image, with the size and location of each square varying according to prescribed parameters. Clustering is performed on squares with similar signatures in order to reduce the number of signatures to be searched during a query. While this system is tolerant of image translation and scaling, the computational complexity is dramatically increased. WALRUS is essentially a color layout scheme, however, it incorporates some region-based attributes.

2.3. Region-based systems

Region-based systems use local properties of regions (ideally objects) as opposed to the use of global properties of the entire image. A fundamental stumbling block for these systems is that objects are frequently divided into multiple regions, each of which inadequately identifies the object. Examples of region-based systems include QBIC [4], SaFe [9], Blobworld [10], and SIMPLIcity [11].

The QBIC system uses both local and global properties and incorporates both region-based and histogram properties. Objects are identified in images using semiautomatic outlining tools [4]. SaFe is a complex system that automatically extracts regions and allows queries based on specified arrangements of the regions. Regions are automatically extracted using a color set back-projection method [9]. Characteristics such as color, shape, texture, area, and location, are stored for each region. A separate search is performed for each region in the query image. Blobworld is a region-based system that automatically defines regions, or *blobs*, within an image using the Expectation-Maximization algorithm on 6-D vectors containing color and texture information for each pixel [10]. For each blob, the anisotropy, orientation, contrast, and two dominant colors are stored. SIMPLIcity is a region-based system that partitions images into predetermined semantic classes prior to extracting the signature [11]. Signature construction and distance formulations are varied according to the semantic class. The k-means algorithm and Haar wavelet are used to segment the image into regions [11].

2.4. Our system

Despite their complexity, all of these systems miss relevant images in the database and may return a number of irrelevant images. An ideal system will be one that provides high value for *precision* and *recall* [12]. These terms originate in the information retrieval literature. *Precision* is defined as the ratio of the number of correct images retrieved to the number of images in the retrieved set. *Recall* is the ratio of the number of correct images retrieved to the number of relevant images in the database. It is easy to see that a recall of 1 is achieved if we retrieve *all* the images in the database. However, that will have an adverse effect on precision which is akin to signal-to-noise ratio. Maximizing precision may adversely affect recall by omitting some relevant images. Precision and recall capture the subjective judgement of the user and may provide different values for different users of a system. It has been commented that even manual browsing is not error-free [13].

Our system is based on using the JPEG coefficients of a compressed image. Each image in our database is represented by a quad-tree structure with leaves that contain relevant JPEG coefficients. We compute statistics concerning each node in the quad tree and include those in the index along with the identification and location information. The quads at any level in the tree, starting from the root, are of the same size. All the quads at a given level in the tree are collected in a relational database table, with quads at different levels collected in separate relations.

Just like the images in the database, a given query image is partitioned into the quad-tree structure and statistics from different nodes are compared against the statistics from the quads of same size from the index relations. The comparison is quantified as a distance measure that can be used to determine the similarity of the query to different images in the data base. Using a threshold, some of the images can be ignored from further comparison yielding further improvements in retrieval efficiency. Finally, an adjustment of threshold yields the desired number of images that match the query image.

The efficiency of our system stems from three factors. First of all, signatures are extracted from compressed images. Since images are transmitted and stored in compressed format and decompression is a CPU-intensive operation, this feature offers substantial savings. Second, this system takes advantage of the work already performed during the initial compression of the image. Compression methods, such as JPEG, have been heavily researched, and offer color compression ratios in the order of 25 : 1 [14]. Despite the vast reduction in size, the JPEG-compressed data contains all the information necessary to reproduce a perceptually identical match of the original image. We take advantage of this invested work and extract the fundamental elements of the image's signature with ease. Finally, we use these elements as the leaves of a quad-tree structure that allows extensive early pruning.

This concept is general enough that it could be extended to other compression standards. Once a standard is set for a system, images of other formats could be incorporated by converting them to the desired format.

Our system uses an extremely efficient color layout method with region-based characteristics. The signature of the query image is based on its global characteristics, however, regional properties of the indexed images are searched. At the present time, it is tolerant of limited object translation. As explained in Section 7, this system can be extended to tolerate scaling and arbitrary object translation.

One of the major advantages of region-based searching is the ability to perform multi-object searches disregarding the relative locations of the objects. For instance, a user may desire an image containing a horse and a car, irrespective of their locations. Using our system, a query for a horse can be performed, followed by a query for a car. The two sets of returned images can be easily checked for matches. This system offers desirable features within an extremely efficient framework.

3. JPEG compression of images

JPEG derives its name from Joint Photographic Experts Group and is a well-established standard for compression of color and grayscale images for the purpose of storage and transmission [7,14,15]. The JPEG compression results in an image that is considerably similar in quality to the original image while using far less bytes, typically from 20 : 1 to 25 : 1 levels of compression without a perceivable quality degradation. The compression itself is lossy which means that some information in the image may be lost depending on the desired amount of compression.

The minimal subset of the JPEG compression standard, known as the *baseline* JPEG, is based on the discrete cosine transform (DCT). To apply DCT, each pixel in the image is level-shifted by 128 by subtracting 128 from each value. Then, the image is divided into fixed size blocks and a DCT is applied to each block, yielding DCT coefficients for the block. These coefficients are quantized using weighting functions optimized for the human eye. The resulting coefficients are encoded using a Huffman variable word-length algorithm to remove redundancies [7].

The compression process is started by dividing the rectangular image canvas into 8×8 blocks and the DCT is applied to each block to separate the high- and low-frequency information in the block. Application of DCT results in the average value (or DC component) in location $(0,0)$ of the 8×8 block while the other locations of the 8×8 block contain the AC terms. The AC terms are made up of higher frequency components of the block. The maximum value for an AC term, the *dominant coefficient*, represents the highest change in the cosine wave between any two adjacent pixels in the block.

The DCT coefficients in each block are quantized to get scaled coefficients by dividing each value with a quantization coefficient from the quantization table developed by the ISO JPEG [14,15]. The coefficients in the quantized block are rearranged using a zigzag ordering to create a vector. The zigzag pattern approximately orders the basis functions from low to high spatial frequencies [14].

The vectors resulting from the zigzag ordering contain the DC coefficient corresponding to the original image in the first location. The baseline JPEG standard requires these vectors of DCT coefficients to be run-length encoded, using Huffman encoding, for storage and transmission. However, we want to use the vectors without encoding and extract some useful data from each vector to facilitate comparison of images. In the remainder of this paper, we will only be interested in the DC component and the dominant coefficient, as these two values give us the most important information about the 8×8 block. The DC component is the average value of the block and the dominant coefficient indicates the degree of uniformity within the block. We build our index efficiently as these two values have already been calculated and are readily available in the compressed image data.

4. Quad tree structure representation of DCT coefficients

In the previous section, we showed how to extract the DCT coefficients for each 8×8 block of the image. This effectively implies that the entire image is considered to be constructed of 8×8 pixel blocks. We consider the 8×8 DCT-encoded block to be the smallest addressable unit of the image, instead of each pixel. In this section, we show the construction of quad tree structure using the information from each 8×8 DCT-encoded block.

The quad tree in our system is a full tree such that each node contains exactly four children or none (Fig. 2). Moreover, all the leaf nodes are at the same level in the tree, and represent the 8×8 pixel block in the original uncompressed image.

To develop the quad tree, we consider an image of size $r\times c$ with r rows and c columns of pixels. After the application of DCT, we obtain a two-dimensional set of vectors that can be described by r' rows and c' columns of vectors, such that

$$r' = \left\lceil \frac{r}{8} \right\rceil,$$

$$c' = \left\lceil \frac{c}{8} \right\rceil,$$

where $\lceil x \rceil$ indicates the smallest integer such that $x \leqslant \lceil x \rceil < x + 1$, that is, the ceiling function. The two-dimensional set of vectors of size $r'\times c'$ is then superimposed on a square array of size $R\times C$, where

$$R = C = 2^{\lceil \lg(\max(r',c')) \rceil}.$$

Thus, the length of each side of the new square image is a power of 2. As shown in Fig. 3, a quad tree can be derived from the square block by recursively dividing each side by 2. In the figure, the input image is superimposed on

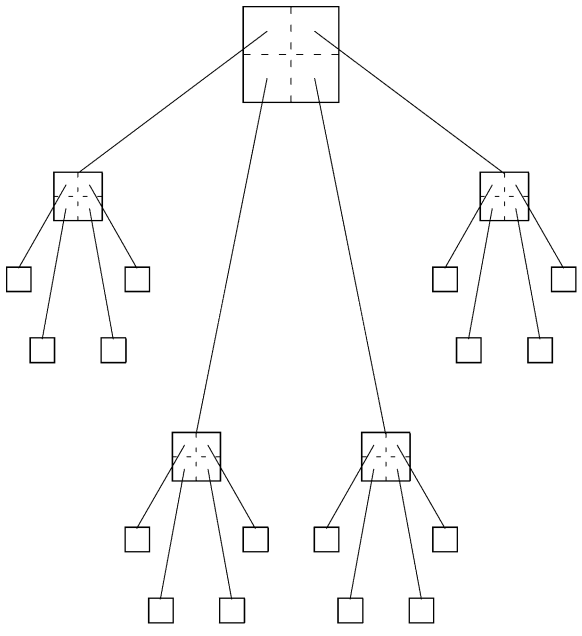
Fig. 2. Quad tree in the system.

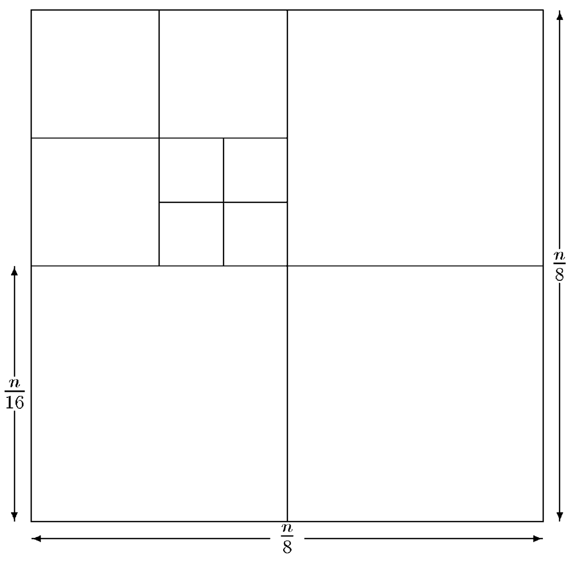

Fig. 3. Quad tree of an image.

an $n\times n$ image, with the maximum dimension in quad tree being $n/8\times n/8$ due to application of DCT. Next, we describe the contents of different nodes in the quad tree.

4.1. Leaf nodes of the quad tree

Each leaf node of the quad tree corresponds to an 8×8 vector of DCT-encoded coefficients that has been computed from an 8×8 pixel block in the original image. The information of interest is the location of the original 8×8 pixel block in the image, the DC coefficient in the DCT-encoded vector (the first element in the vector), and the dominant coefficient in the vector. The location of the pixel block is taken to be the location of pixel in the top left corner in the block. This does not lead to any loss of information as the image, at this level, is described in block coordinates, using 8×8 blocks, instead of pixel coordinates. In addition, we also retain the name of the image for identification purpose.

```
struct jpeg_node
{
  loc block_start;        /* Starting coordinates of block */
  int block_height;       /* Height of the block */
  int block_width;        /* Width of the block */
  int size;               /* Size of the enclosing square block side */
  int avg_dcc;            /* Average DC coefficient */
  int dom_coeff;          /* Dominant coefficient in the block */
  loc dom_loc;            /* Location of dominant coefficient */
  char img[33];           /* Image name (for identification) */
  struct jpeg_node *ul,   /* Children */
         *ur, *ll, *lr;
};
```

Fig. 4. Structure of a node in the quad tree.

4.2. Non-leaf nodes and the quad tree

The non-leaf nodes contain information extracted from aggregates of 8×8 DCT-encoded vectors. They will be used to make similarity comparisons between images at a *higher* level than the individual blocks. Each non-leaf node in the quad tree contains the location of the corresponding aggregate block in the original image, the average DC value of its children, and the dominant coefficient in the entire aggregate block of DCT-encoded coefficient vectors. Just like in leaf nodes, we retain the name of the image for identification purpose. Lastly, the non-leaf nodes also contain the links to their four children, identified respectively as `ul` (upper left), `ur` (upper right), `ll` (lower left), and `lr` (lower right). The complete structure of each node is presented in Fig. 4.

It may be noted that a leaf node uses the same structure as in Fig. 4 by assigning a null value to the four children. The quad tree itself is illustrated in Fig. 3. In this figure, we start with the set of 8×8 DCT-encoded coefficients forming a $n/8\times n/8$ square corresponding to an $n\times n$ image at the root. The set of coefficients is divided into four sets of $n/16\times n/16$ coefficients each. We continue to subdivide each set into four subsets recursively until we get a set containing only one 8×8 DCT-encoded coefficient. At each node in the tree, we calculate the information in the node (Fig. 4).

The nodes in the tree are at distinct levels that can be identified by the size of the image. We save the nodes depending upon their level in the tree in a file named as `indx`*nnn*`.idx` where *nnn* is indicative of the size of the DCT-encoded segment at that level. It is easy to see that all the segments at a given level are of the same size. The nodes are saved by performing a level-order traversal of the quad tree with the following collating sequence: `ul`, `ur`, `ll`, and `lr`.

5. The query process

Once the quad tree has been developed for each image in the database and the required information saved in the index files, the query process is fairly simple. The query in this case is considered to be a JPEG image. The image is processed to develop the quad tree of its DCT-encoded coefficient blocks as described in Section 4. Further steps in the query rely completely on the quad tree.

The root of the quad tree contains information about the size of the image. Since each index file contains information about the nodes of the same size, we select the index file that contains nodes with size equal to the size of the image. It may be noted that the size of the node is represented in terms of an exponent of 2 and therefore, the image is also fit into a larger *template* with size rounded to the next exponent of 2. The overall result is that we only consider those images in the database that are larger than or equal to the query image. The root node of the quad tree constructed from the query is compared against each node in the selected index file using the expression

$$\mathrm{diff}(c_q - c_n) + \frac{\mathrm{diff}(d_q - d_n)}{m},$$

where c_q and c_n are the average DC coefficients for the nodes in the query and the index file, respectively, d_q and d_n are the corresponding dominant coefficients, and m is a constant. This formula is a work in progress. We are currently looking into the computation of an optimal value for m and an appropriate diff function. Furthermore, we have considered deleting the second term of the formula while comparing non-leaf nodes in order to reduce error due to noise.

It is easy to see that in the case of a perfect match, e.g., the same image in query and the database, the above comparison will evaluate to zero. At any time, the images that result in the comparison greater than a prespecified threshold can be ignored from further consideration. However, it may be the case that two images which are not similar to each other in content but are similar in average intensity (the DC component for the image), may result in the comparison evaluation of zero. An example of this case is depicted in Fig. 1. These images need to be ranked using the information in the lower nodes in the quad tree.

While comparing the lower nodes, we must keep track of the relative location of each subblock at the node in the query as well as the database and must compare the corresponding nodes only. The index has been structured such that each index file contains the nodes at a given level within the quad tree. Traversing the quad tree corresponding to the query in level order, each node is compared with the corresponding node in the image database, with the root node as the reference, and the evaluation of the comparison (the distance between query and reference for the subblock) added to the previous evaluation. At any stage, if the summation of distances during the level-order traversal-comparison becomes too large, the image is removed from further consideration. We can also limit the number of images that should be kept under consideration by keeping only the top-ranked images in consideration.

The overall results of the query are summarized in a ranked list of images that are sorted on the basis of their distance from the query. The ranking in the list allows us to present the resulting images in the decreasing order of relevance with respect to the query. In addition, we can also limit the number of images that are to be presented to the user.

6. Implementation status

The image database system described in this paper has been implemented on a Sun SPARC system in C. We are using multiple databases, including one from the Smithsonian Institute. The Smithsonian database contains 747 images divided into five different categories, with each category containing between 33 and 216 images. We used gray scale images (the luminance component) for testing the system.

A quad tree was computed for each image and index files were developed using the quad trees. The query program was given an image as input with the result being a ranked list of images from the database. Currently, the program just gives the name of the image and a number to specify its distance from the query image. The diff function used is simply $|c_q - c_n|$, or the difference between the DC component of corresponding blocks.

The results from a query are displayed in Fig. 5. The database used for this query contains 94 Smithsonian air-space images and 27 other images. These images show the query image (top left), four images with the smallest distance, and the image with the largest distance. Surprisingly, a picture of the sun is the third closest match. The sun image is 527×475 pixels, while the query image is 635×380 pixels. Consequently, the dark lower edge of the sun and the right-hand edge of the jet image weren't compared. Furthermore, the dark right-hand edge of the sun happens to line up with the tail of the jet. Clearly, the distance between these images would be

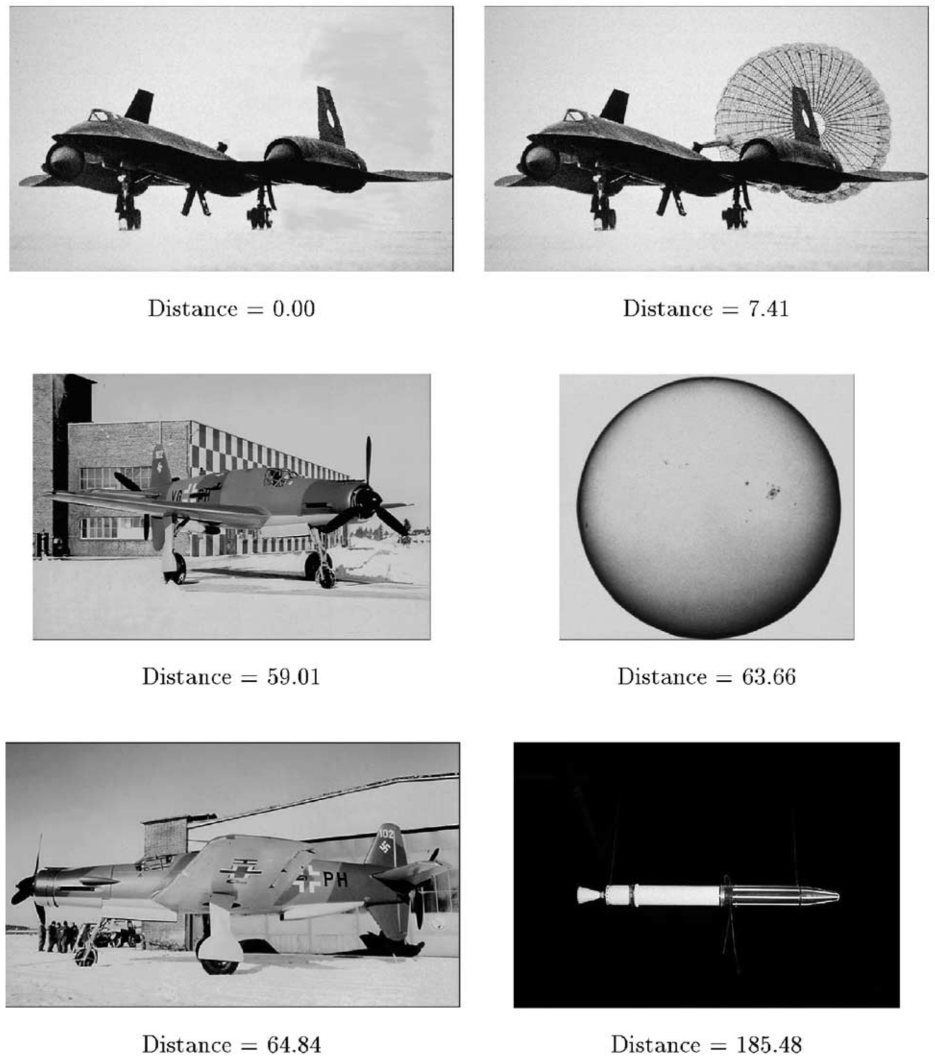

Fig. 5. Results from a query, with the distance of the picture from the query.

dramatically increased if the dominant coefficient term were utilized in the distance formula.

To evaluate the system, we conducted an experiment by asking human subjects to rank the images with respect to the query. Six individuals independently ranked 14 images according to the amount of work necessary to change each of the images into the query image. When we queried the same image using our system, the results were very close to the human ranking as shown in Figs. 6 and 7.

7. Future work

For the immediate future, we are working on optimizing our code and creating a GUI front end for the system under the X windows environment. In addition, we are working on a system that will work with color images. We are also looking into more complex distance computation functions in compressed domain, in particular the techniques proposed by Skodras [16].

For the long-term goal, we would like to extend our system to tolerate arbitrary translation and scaling. Translational tolerance could be achieved by increasing the number of non-leaf nodes in the quad tree. The number of nodes on level n would correspond to the number of possible locations of an $n \times n$ block within the image. Scaling tolerance could be achieved by querying each level between the largest block size and a specified smaller block size. Another promising project is to adapt this system to JPEG 2000 by utilizing wavelet coefficients. Finally,

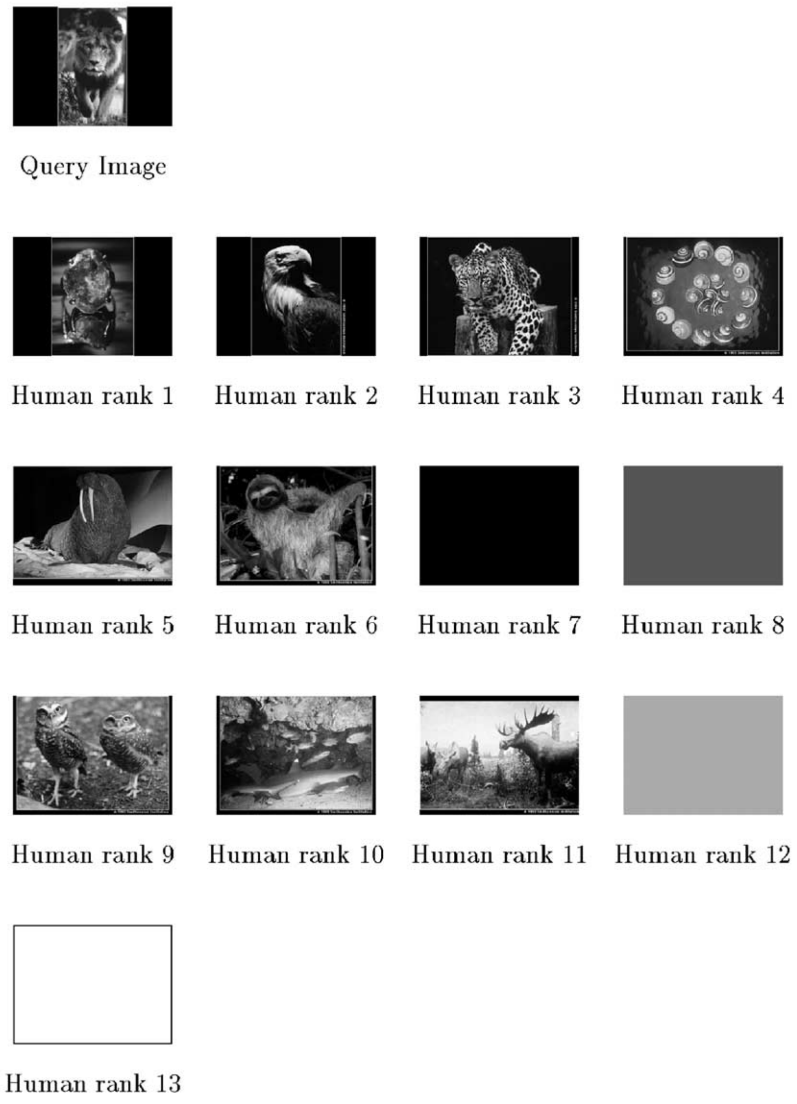

Fig. 6. Human ranking of images.

another enhancement will be provided by using clustering methods within the index organization.

The efficiency of this system results from indexing images in compressed format, utilizing work that was previously performed, and creating a tree that allows extensive early pruning. These concepts could be applied to other CBIR projects.

8. Summary

In this paper, we present an image database indexing system for efficient storage and retrieval of images in response to a query expressed as an example image. Our system can be classified as a content-based image retrieval system and is exceptionally efficient. The efficiency stems from indexing images while in compressed format, by utilizing work that was already performed (during compression), and using a quad tree for the image signature. The system also allows extensive early pruning during query so that any images that are not promising for the query are eliminated from consideration at an early stage.

The system accepts JPEG images, extracting the average DC coefficients from the compressed data. (When an image is compressed, it is divided into 8×8 blocks of pixels, and a discrete cosine transform is applied to each block. The average DC coefficient corresponds to the

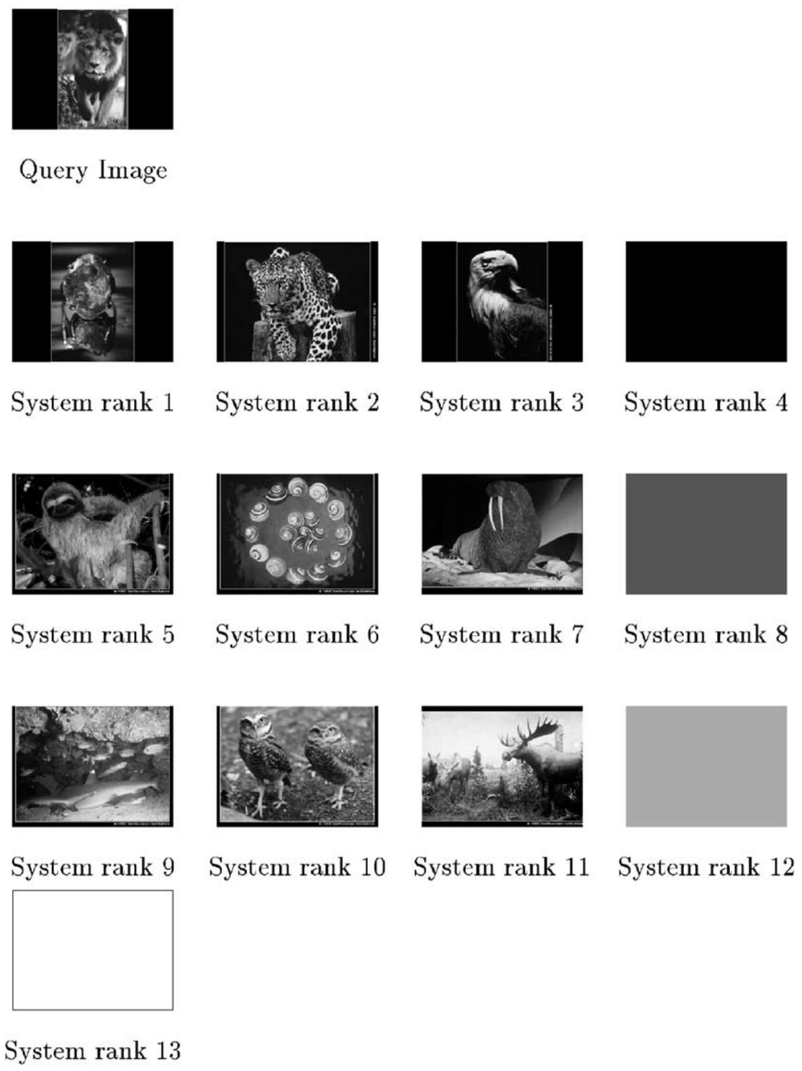

Fig. 7. System ranking of images.

average value of the 8×8 block.) These values become the leaf nodes of the quad tree. Four 8×8 blocks comprise a 16×16 block of the image. The nodes on the level next to the leaf nodes each contain the average of four leaf nodes (corresponding to a 16×16 block). This procedure is repeated until the root node is constructed, containing the average value for the entire image. This tree is the "signature" of the image that is stored in the index. A query image is processed to produce a quad tree, and the roots of the trees in the index are compared with the query root using a distance formula. For distances within a given tolerance, the second level of the corresponding trees are compared. This process repeats down to the leaf nodes, and selected images are ordered and returned to the user.

The system has been implemented in C on a Sun workstation and has been tested using images from the Smithsonian database.

References

[1] Shih-Fu Chang, J.R. Smith, M. Beigi, A. Benitez, Visual information retrieval from large distributed on-line repositories, Commun. ACM 40 (12) (1997) 63–71.

[2] J.R. Smith, Shih-Fu Chang, Automated image retrieval using color and texture, Technical Report 414-95-20, Columbia University, Department of Electrical Engineering and Telecommunications Research, New York, NY 10027, July 1995.

[3] H. Samet, The quadtree and related hierarchical data structures, ACM Comput. Surv. 16 (2) (1984) 187–260.
[4] M. Flickner et al., Query by image and video content: the QBIC system, IEEE Comput. 28 (9) (1995) 23–32.
[5] A. Netsev, R. Rastogi, K. Shim, WALRUS: a similarity retrieval algorithm for image databases, in: A. Delis, C. Faloutsos, S. Ghandeharizadeh (Eds.), SIGMOD 1999: Proceedings of the ACM SIGMOD International Conference on Management of Data, ACM Press, Philadelphia, PA, June 1999, pp. 395–406.
[6] G. Lu, B. Williams, An integrated WWW image retrieval system, In Australian WWW Conference, April 1999.
[7] J.D. Murray, W. vanRyper, Encyclopedia of Graphics File Formats, 2nd Edition, O'Reilly, Sebastopol, CA, 1996.
[8] James Ze Wang, G. Wiederhold, O. Firschein, Sha Xin Wei, Content-based image indexing and searching using Daubechies' wavelets, Int. J. Digital Libraries 1 (4) (1997) 311–328.
[9] J.R. Smith, Shih-Fu Chang, Integrated spatial and feature image query, Int. J. Multimedia Systems 7 (2) (1999) 129–140.
[10] S. Belongie, C. Carson, H. Greenspan, J. Malik, Color- and texture-based image segmentation using the expectation-maximization algorithm and its application to content-based image retrieval, in: ICCV98: Proceedings of the International Conference on Computer Vision, 1998, pp. 675–682.
[11] J. Ze Wang, J. Li, G. Wiederhold, SIMPLIcity: semantics-sensitive integrated matching for picture libraries, IEEE Trans. Pattern Anal. Mach. Intell. 23 (9) (2001) 947–963.
[12] G. Salton, M.J. McGill, Introduction to Modern Information Retrieval, McGraw-Hill, New York, 1983.
[13] D. Forsyth, J. Malik, R. Wilensky, Searching for digital pictures, Sci. Am. 276 (6) (1997) 88–93.
[14] W.B. Pennebaker, J.L. Mitchell, JPEG Still Image Data Compression Standard, van Nostrand Reinhold, New York, 1993.
[15] R.C. Gonzalez, R.E. Woods, Digital Image Processing, Addison-Wesley, Reading, MA, 1992.
[16] A.N. Skodras, Direct transform to transform computation, IEEE Signal Process. Lett. 6 (8) (1999) 202–204.

About the Author—SHARLEE CLIMER received her B.S. in Engineering from Washington University in 1994, B.A. in Physics from St. Louis University in 1995, and B.S. in Computer Science from the University of Missouri—St. Louis in 1999. She is currently a graduate student and teaching assistant at the University of Missouri—St. Louis. Her research interests include Algorithms, Artificial Intelligence, and Computer Vision.

About the Author—SANJIV BHATIA received his B.E. degree in Computer Science Engineering from Motilal Nehru Regional Engineering College, Allahabad, India in 1983. He then worked with Engineers India Limited, New Delhi, India for three years. He received his M.S. (Computer Science) from the University of Arkansas, Fayetteville in 1987, and Ph.D. (Computer Science) from the University of Nebraska, Lincoln in 1991. Since Fall 1991, he has been with the University of Missouri—St. Louis. He has published in the areas of Image Databases, Knowledge Acquisition, Machine Learning, and Information Retrieval. He is also active in software design for flight simulator visuals. He is a member of ACM and AAAI.