RIME: A Replicated Image Detector for the World-Wide Web

Edward Chang, James Ze Wang, Chen Li, and Gio Wiederhold
Department of Computer Science
Stanford University
{echang,wangz,chenli,gio}@cs.stanford.edu

Abstract

This paper describes RIME (Replicated IMage dEtector), an alternative approach to watermarking for detecting unauthorized image copying on the Internet. RIME profiles internet images and stores the feature vectors of the images and their URLs in its repository. When a copy detection request is received, RIME matches the requested image's feature vector with the vectors stored in the repository and returns a list of suspect URLs. RIME characterizes each image using Daubechies' wavelets. The wavelet coefficients are stored as the feature vector. RIME uses a multidimensional extensible hashing scheme to index these high-dimensional feature vectors. Our preliminary result shows that it can detect image copies effectively: It can find the top suspects and copes well with image format conversion, resampling, and requantization.
Keywords: copy detection, wavelets, multidimensional indexes.

1 Introduction

Advancement in the internet and world wide web technology has led to the growth of information dissemination at an unprecedent pace. The resulting more and more efficient distribution of data, however, has increased the problems associated with copyright enforcement [8].

Many studies have proposed schemes for detecting copies of *text* documents [3]. In this paper, we present an *image* copy detection service being developed at Stanford University. We call this service RIME, for Replicated IMage dEtector. Given an image registered by its creator or distributor, RIME checks if copies of the image exist on the internet and returns a list of *suspect* URLs. The creator or distributor of the image can then check if the suspect images are indeed unlawful copies (for instance by using watermarks).

RIME faces two conflicting performance requirements: *accuracy* and *speed*. On the one hand, we want to achieve high *accuracy* and high *recall*, which often means high computational cost.

On the other hand, we would like to reduce processing time so that more images can be compared given limited time and computing resources. To meet these performance requirements, RIME faces two design challenges: First, it needs a good feature vector that can tell apart different images. Second, it needs an efficient indexing scheme that can find the suspect images accurately and quickly (i.e., with minimum number of IOs)

For the feature vector, RIME characterizes each image using Daubechies' Discrete Wavelet Transform (DWT) for each of the three opponent color components. Wang, Wilderhold, and Firschein have demonstrated in the Wavelet Image Search Engine (WISE) [22, 21] that DWT may characterize images better than other known techniques for content-based image retrieval. DWT offers good spatial and frequency localization. In the spatial domain, it is able to capture the color changes from pixel to pixel. In the frequency domain, DWT can capture the intensity of the minutest color changes by applying wavelet transform recursively. Moreover, subsets of the frequency bands can be used as filters that quickly screen out irrelevant images to narrow down the search space.

Since each image is represented by a multi-dimensional feature vector, searching replicas of an image means essentially searching vectors in the neighborhood of the given vector. Many trees (e.g., R-tree, k-d-b-tree, SS-tree, SR-tree, and etc) have been proposed to perform nearest-neighbor search for high-dimensional data [2, 12, 15]. These trees, however, often suffer from *the curse of dimensionality*, in that both the memory requirement and search speed can grow superlinearly with the data dimensions as well as the database size. Instead, RIME employs a multidimensional extensible hashing index structure. The hashing scheme requires less memory; more importantly, it performs the search through computation (i.e., computing the nearest buckets via a hashing function) rather than through index structure traversal. When the index structure cannot fit in main memory (e.g., because either the number of data dimensions or the database is very large), the hashing scheme enjoys significantly faster searching speed.

In addition to the challenges that we have described, the feature vector must be able to survive image format conversion (e.g., from JPEG to Gif and vice versa), resampling, and requantization. Furthermore, RIME must also be able to cope with geometric transformations such as translation, scaling, and rotation of the images. The first version of RIME, which was built upon WISE, can discover most suspect images (with format conversion, resampling, requantization, and geometric transformation) in the database. We are currently implementing the hashing scheme we proposed and building a much larger image database to verify RIME's scalability.

The rest of the paper is organized as follows: Section 2 describes RIME's architecture. Section 3 depicts how RIME prepares for the feature vectors and selects filters. Section 4 compares the multidimensional extensible hashing scheme that RIME employs with the traditional tree structures. Section 5 shows our preliminary experimental results. Finally, we offer our conclusions in Section 6.

1.1 Related Work

Many studies have proposed using watermark schemes to safeguard image copyright. These schemes add to the images the creator's or the distributor's identity. A watermark, however, is vulnerable to image processing, geometric distortions, and subterfuge attacks [5]. In addition, working with a large variety of watermark schemes increases the detection time exponentially. Safeguarding copyright by checking the image directly has two distinct advantages: although a copied image can also be altered, it is unlikely that a pirate will change the image substantially and thereby loses its original appearance, and image copy detection is more efficient. We believe that the watermark still plays an important role in authenticating images in courts of laws. After RIME provides the creator or distributor with a suspect list, the actual owner of the images can use watermark or other authentication techniques to prove ownership.

At a first glance, RIME's functionality may look similar to that of some content-based image retrieval databases such as QBIC [7], Virage [9], WISE [21], and VisualSEEk [18] which retrieve images similar to a given one. RIME, however differs from these systems in many ways. The fundamental difference lies in the measure of search accuracy. RIME searches for all the copies of an image, while a content-based image retrieval system searches for similar images, whereas judging if an image is a copy of another is objective. But, judging if two images are similar can be very subjective. For instance, two images with the same green mountain and blue sky background but different small objects (e.g., flowers of different colors) in the foreground

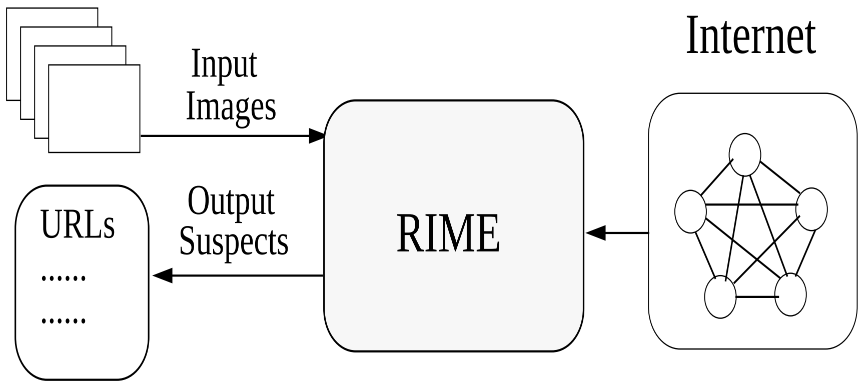

Figure 1: RIME's Components

may be considered similar by one person but different by another. In fact, as pointed out in the study of [23], a content-based image retrieval system may answer a similarity query only *approximately*. Under this relaxation, a nearest neighbor search can be implemented much more efficiently: Instead of looking for the images in all adjacent bins (or buckets, cells, etc. depending on the index structure used) within a range, returning all images in the bin where the given image resides can satisfy the similarity search. On the other hand, making the same relaxation in RIME's searching algorithm may not be as forgiving. To achieve accurate copy detection, RIME requires a good feature vector and an efficient index structure.

2 Architecture

Figure 1 is a bird's-eye view of RIME. RIME receives registered images from users, profiles the images, and finds matching ones in its repository. If matches are found, RIME returns a list of suspect URLs. To build the repository that stores the image profiles, RIME crawls the web periodically, profiles newly discovered images, and then indexes them based on their features.

Figure 2 shows RIME's components. They are:

1. The Profiler, which extracts feature vectors from images.
2. The Registered Image Repository (RIR), which contains the feature vectors and URLs of the images registered by the owner or distributor for copy detection. RIME performs copy detection periodically (e.g., after a new web crawl) for these registered images.
3. The Internet Image Repository (IIR), which stores the feature vectors and URLs of the

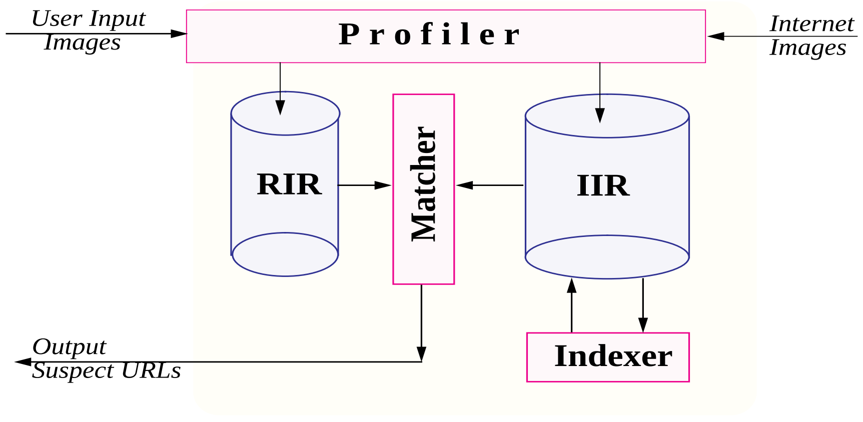

Figure 2: The Components of RIME

crawled web images. Notice that RIR and IIR may be two logical databases on top of one physical database.

4. The Indexer, which indexes the feature vectors in a high-dimensional structure for efficient searching and storage.

5. The Matcher, which performs nearest-neighbor search in the index structure to return suspect URLs.

RIME does not store any images in its repository. Instead, after profiling the images, it stores only the resulting feature vectors and the URLs of the images. In Section 3, we define feature vectors and describe in detail how they are created. Not storing the images not only saves a tremendous amount of storage space, but also avoids copyright issues.

3 Image Feature Vectors

A feature vector should be small and good at representing the image. It must also be able to tell different images apart in a very large image database. As in WISE, RIME also applies Daubechies' Discrete Wavelet Transform to an image and extracts the feature vector based on the wavelet coefficients. The current implementation of the system uses the same feature extraction scheme as that used in WISE. In our future implementations, some linear and non-linear combinations of the wavelet coefficients will be used as the image's feature vector.

Figure 3 presents the three steps to prepare for an image's feature vector. First, we convert the image into a canonical size and format. Second, we apply Daubechies' wavelet transforma-

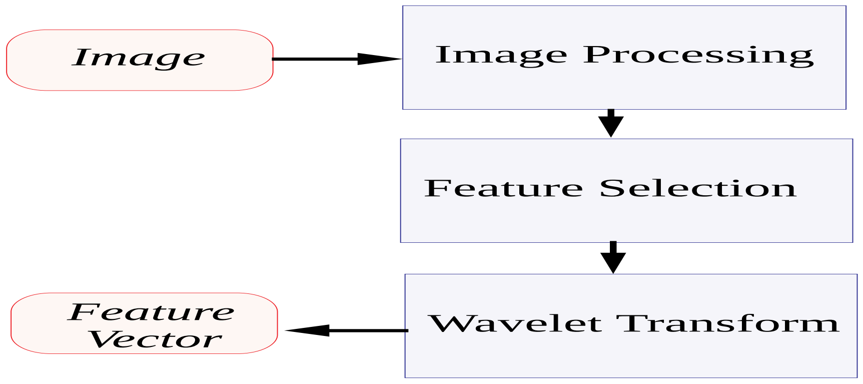

Figure 3: Feature Vector Preparation Steps

tion on the image to produce coefficients. Finally, we store the selected wavelet coefficients as the image's feature vector. In the remainder of this section, we describe these steps in detail and show that RIME's feature vector is superior to that produced by other methods.

3.1 Preprocessing the Images

Many color image formats exist, GIF, JPEG, and TIFF being the ones most widely used. Because images can have different formats and different sizes, we first normalize the data. We rescale images to thumbnails of 256×256 pixels in the RGB (Red-Green-Blue) color space. We choose the 256×256 image size because it is the size and aspect ratio of most internet images and the wavelet transform requires the pixel number of each side to be a power of two.

To rescale an image, we use bilinear interpolation. This method resamples the input image by overlaying the image with our grid of 256×256 points. The input image is sampled at each grid point to determine the pixel colors of the output image. The sample process shifts the grid both vertically and horizontally. The value at each grid point is the linear interpolation of the pixels it samples. This rescaling process is more effective than a Haar-like rescaling (i.e., grouping many pixels into one to decrease image size or replicating one pixel into many to increase image size), especially for the images that have sharp color changes.

Next, we transform the rescaled image to a different color space. Since what's needed is actual human perceptual color distance, which differs from that of RGB color space [7], we convert and store the image in a component color space with intensity and perceived contrasts.

We define the new values of a color pixel based on the RGB values of an original pixel as follows:

$$\begin{cases} C_1 = (R + G + B)/3 \\ C_2 = (R + (max - B))/2 \\ C_3 = (R + 2 * (max - G) + B)/4 \end{cases} \tag{1}$$

Here max is the maximum possible value for each color component in the RGB color space. For a standard 24-bit color image, $max = 255$. Each color component in the new color space also ranges from 0 to 255. This color space is similar to the opponent color axes

$$\begin{cases} RG = R - 2 * G + B \\ BY = -R - G + 2 * B \\ WB = R + G + B \end{cases} \tag{2}$$

defined in [6] and [19].

In addition to providing the perception correlation properties [11] of such an opponent color space, another important advantage of this alternative space is that the C_1 axis, or the intensity, can be more coarsely sampled than the other two axes for color correlation. This helps reduce the size of the feature vector.

3.2 Characterizing Images Using Wavelets

Traditionally, a color histogram has been used to characterize an image. However, while a global histogram preserves the color information contained in the image, it does not preserve information about the colors' locations. For instance, an image with the sun setting into the sea contains orange and blue. A global color histogram may capture the fractions of orange and blue correctly but cannot tell where the colors are concentrated. Thus, two images having similar color histograms may have very different semantics.

Storing color layout information is another way to describe the content of an image. In traditional color layout image indexing, we divide the image into equal-sized blocks, compute the average color on the pixels in each block, and store the values for image matching using the Euclidean metric or variations of the Euclidean metric. It is also possible to compute the values based on statistical analysis of the pixels in the block. Both techniques are very similar to image rescaling or subsampling. However, these approaches do not perform well when the image contains high frequency information such as sharp color changes. For example, if there are pixels of various colors ranging from black to white in one block, these techniques may not be effective.

Shape-based and texture-based detection and coding algorithms are other techniques of characterizing images. Both have substantial limitations for general-purpose image databases. For example, current shape detection algorithms only work effectively on images with relatively uniform backgrounds, and texture coding is not appropriate for non-textural images. In addition, shape recognition is computation intensive and hence not suitable for processing a large number of images.

RIME uses Daubechies' wavelet coefficients proposed in [21] to represent image semantics, including object configuration and local color variation. A fast wavelet transform with Daubechies' wavelet is applied to a preprocessed image for each of the three color components (i.e., C_1, C_2, and C_3). The low frequency coefficients of the wavelet transforms are stored as the feature vector of the image. Contrary to the techniques mentioned above, the Daubechies' wavelet offers good spatial and frequency localization. In the spatial domain, it is able to capture the color changes from pixel to pixel. In the frequency domain, it can capture the intensity of the color changes to the minutest detail by applying wavelet transform recursively.

3.3 Selecting Features

RIME uses wavelet coefficients as the features to compare images. However, using all 256×256 coefficients in three color space to perform image matching requires too large an index structure ($256 \times 256 \times 3$ dimensions) and too much storage space. Instead, we apply wavelet transform recursively to reduce the wavelets to 16×16 coefficients to produce some useful coarser features. Figure 4 shows an example where wavelet transformation is applied to one color space of the image three times to reduce the number of coefficients down to one eighth (at the upper-left corner of the figure). The idea here is that we use some coarser features as filters to screen out irrelevant images. After this filtering step, we then use the total number of coefficients to do the fine-grained and more computationally intensive copy detection in a much-reduced sample.

Rime uses two filters: a color filter and a shape filter:

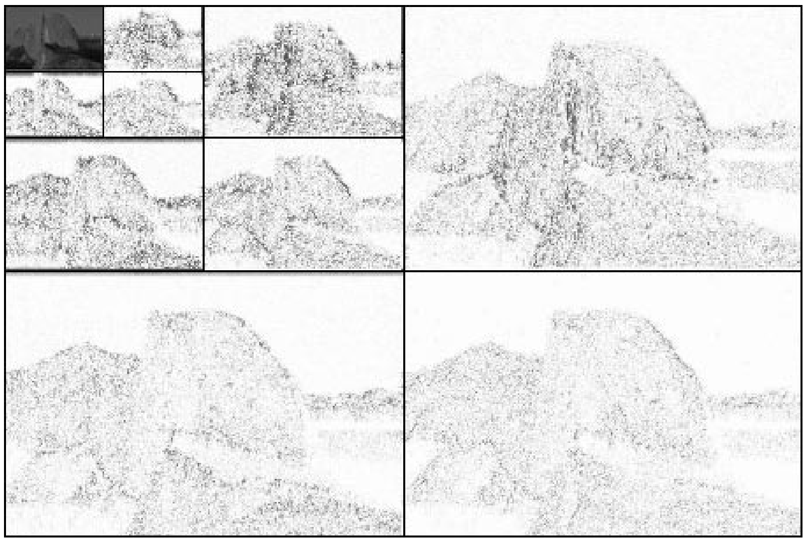

Figure 4: A Wavelet Transform Example

- Color filter: We first perform wavelet transform recursively to reduce the number of wavelets to 16×16 in three color space as shown in Figure 5. We pick the 8×8 low frequency wavelets in the upper-left quadrant in Figure 5 in three color space as the color filter. In Section 4 we show how we index these wavelets to perform fast search to reduce the space for the fine-grained matching.

- Shape filter: the shape filter is the sum of the remaining high frequency wavelets in the table shown in Figure 5. The 8×8 coefficients in the bottom-left quadrant represent the high horizontal frequency components; the upper-right quadrant the high vertical frequency components; and the bottom-right quadrant the high diagonal components. Intuitively, we see that vertical lines in an image produce high frequency horizontal wavelets, horizontal lines produce high frequency vertical wavelets, and diagonal lines diagonal wavelets. Two similar images should have similar shapes and hence similar high frequency components in these three quadrants in C_1, C_2, and C_3, respectively.

  Since most of these high frequency components are zeros, we sum them up in three quadrants of three color space to produce nine values. We then pick a threshold value to decide if an image has *significant* high frequency components in these nine quadrants. If two images have objects of the same shapes, they should have significant high frequency components in the same quadrants. Therefore, images having different high frequency

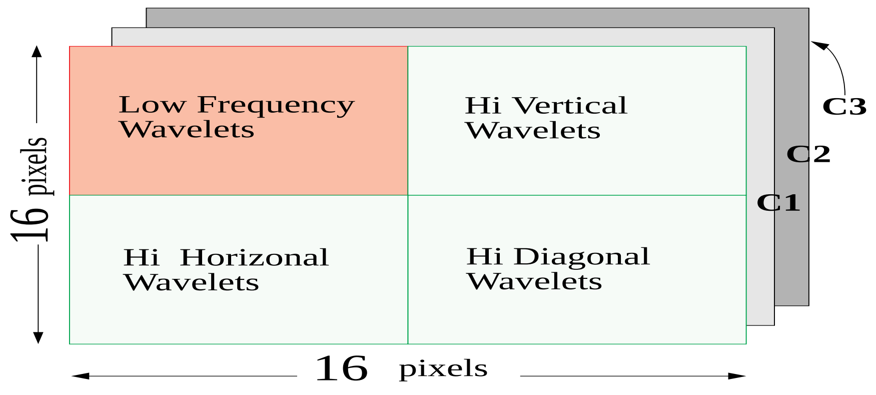

Figure 5: 16 × 16 Coefficient Table

quadrants can be discarded in this filtering step.

Although identifying the objects and their shapes in an image is difficult and expensive, determining that the image has horizontal, vertical, and diagonal lines is fairly simple using this approach.

3.4 Filtering and Matching

Given a query image, the search is carried out in two steps: 1) filtering and 2) fine-grained matching. In the first step, RIME indexes into a subset of found images by using the coarser wavelets as the color filter. In the second step, we compute a weighted version of the Euclidean distance between the feature coefficients of the remaining images and those of the querying image [21]. We then select and sort the images with the smallest distances and present them as the images matching the query.

4 Index Structure

RIME uses a high-dimensional index structure to organize feature vectors so that both storing and looking up these vectors in the high-dimensional space can be efficient. The index structure divides the high-dimensional space into a number of regions (buckets, bins, or quadrants, depending on the index structure used), each containing a subset of feature vectors that can be stored in a small number of disk blocks.

Given an image vector, RIME searches for similar vectors in the high-dimensional space in the following steps:

1. It performs a *where-am-I* search to find in which region the given vector resides.

2. It then performs a *nearest-neighbor* search to locate the neighboring regions where similar vectors may reside. This search is often converted into a *range* search, which locates all the regions that overlap, with the search sphere centered the the given vector and having a diameter d.

3. Finally, RIME computes the Euclidean distances between the vectors in the nearby regions (obtained from the previous step) and the given vector. The search result includes all the vectors that are within distance d from the given vector.

The performance bottleneck of RIME lies in the first two steps. If the index structure does not fit in main memory and the search algorithm is inefficient, RIME needs to traverse a large portion of the index structure on the disk. Also, the number of neighboring regions can grow exponentially with respect to the dimensions of the feature vectors. Reading in the data from all neighboring regions can thus take an exponential number of IOs. A high-dimensional index structure must be designed to minimize these IO costs to achieve reasonable performance.

There are two categories of index structures, *tree-like* and *hash-table-like* [20]. In the remainder of this section, we discuss their pros and cons and present the structures with which we are currently experimenting to satisfy RIME's performance objectives.

4.1 Tree-like Structures

Many tree-like structures have been proposed to index high-dimensional data (e.g., R*-tree [1, 10], SS-tree [24], SR-tree [12], TV-tree [14], X-tree [2], M-tree [4], and K-D-B-tree [16]). At the interior nodes of these tree structures, key and pointer pairs are stored. Figure 6 shows a typical interior node with three keys and four pointers. Each $\{key, pointer\}$ pair records the splitting values of the attributes and the pointer to the child nodes. Given a fixed number of vectors to index, the depth of the tree depends on the fanout of the interior nodes, i.e., the larger the fanout, the larger the depth of the tree. The fanout of the tree in turn depends on the size of these $\{key, pointer\}$ pairs. The higher the data's dimensions, the more splitting

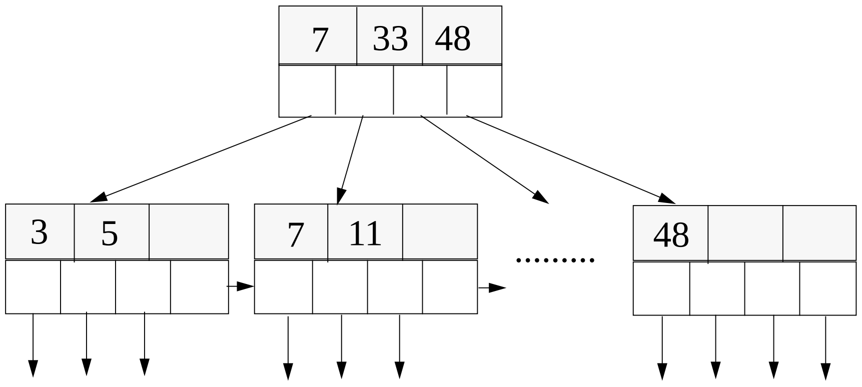

Figure 6: Tree-like Structures

attribute values must be recorded, and hence the smaller the fanout of the interior nodes. As the height of the tree structure increases, so does the memory requirement for staging the index structure. The memory requirement of the tree structure can thus grow more than linearly with respect to the dimensionality of the data.

When an interior node splits into child nodes, the space represented by the parent node is divided into subregions of different shapes.[1] The coordinates of these bounding shapes must also be recorded in the interior nodes for speeding up range queries (discussed below). This further aggravates the memory requirement for the tree-like index structures. If the index structure is too large to fit in main memory, the search performance will suffer.

In addition, these bounding shapes often overlap. The study of [23] shows that when the data dimensions go beyond 16, the percentage to which a bounding shape overlaps its neighboring shapes can approach 100% for the R-tree. Because siblings can overlap, the number of buckets to look up for the *where-am-I* search can grow exponentially.

Finally, most tree-like structures employ the branch-and-bound algorithm [17] to perform nearest-neighbor queries. Since the algorithm can traverse a large portion of the tree to look for the candidate regions, it is critical for the index structure to fit in main memory, or the on-disk search can degrade the performance significantly. This leads to our proposal of using a hash-table-like structure.

[1]For instance, the R-tree and the K-D-tree use bounding rectangles, the SS-tree uses bounding spheres, and the SR-tree uses both.

4.2 Hash-table-like Structures

To conserve memory space, we propose using a hashing approach, which avoids recording $\{key, pointer\}$ pairs. The main ideas are the followings:

- We divide the i^{th} dimension into 2^{k_i} evenly divided stripes, where the value of k_i is determined by how the values of the i^{th} attribute cluster. This way, using a minimum number of bits (k_i bits in the i^{th} dimension) we can encode to which region a feature vector belongs.
- To perform the *where-am-I* query for a given vector, we hash the vector to a region (bucket) via a function. This effectively replaces searching in a high-dimensional structure with *computation*.
- To make the *range* query mimic a *nearest-neighbor* query, we add a distance to the given vector in each dimension to compute a range. We then use the hash function again to *compute* the candidate buckets.

This index structure conserves memory space since it does not need to record any $\{key, pointer\}$ pairs (i.e., performing binary splits at each dimension). It also uses a hash function to compute the candidate buckets to avoid potential on-disk lookup. However, this structure is suboptimal in choosing the splitting points at each dimension compared to a scheme like VAM-split [23]. Consequently, it can incur some storage overhead and increase the number of candidate buckets it must load in to compute vector distances during the final phase of the nearest-neighbor search. We remedy this shortcoming through bucket grouping. The details of the index structure are described in [13]. We are currently integrating this hashing scheme into RIME to enhance its searching speed and accuracy.

5 Experiments

The primary focus of our experiments is to make sure that the wavelet feature vectors are proper for image copy detection. We conduct our experiments using a modified version of WISE without the extensible hashing index.

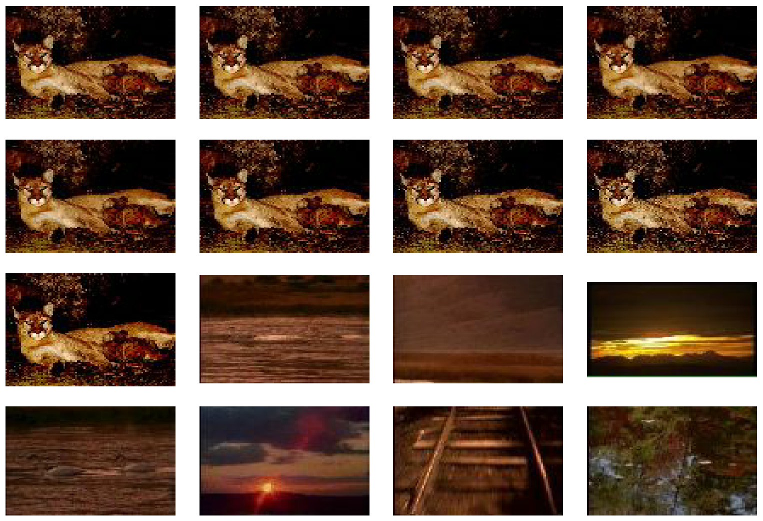

Figure 7: A Sample Query Result

The WISE system uses wavelet transform and tree-structure vector quantization to perform content-based image search for very large image databases. It was implemented using C on UNIX platforms. The discrete fast wavelet transforms were performed on a Pentium Pro 300MHz LINUX workstations. As reported in [21], to compute the feature vectors for more than 30,000 color photos in our database requires approximately 2 hours of CPU time. It takes about 40 seconds for the feature space classification module to process all the 30,000 feature vectors in the database and to build the tree structure.

WISE was built initially to perform similarity search. RIME adopts its feature selection module, but has to change its filters to fit the purpose of image copy detection. Specifically, we removed the variance filter from WISE because it could filter out images with significant requantization. Instead, the 8×8 wavelet color filter worked well, locating most modified images.

For testing the accuracy, we inserted a sequence of 10 images modified from the same image to the database of 30,000 images. The image is processed through sharpening, softening, despeckling, posterizing and watercoloring. Figure 7 shows a sample query result obtained from the system. With the current system, we detected eight out of the ten copies but missed two. We are satisfied with the detection accuracy that wavelets can provide. However, we feel enhancements can be performed in at least three areas:

1. Some images with no resembling semantics are also included as the suspects. We think this is due to the lack of the shape filter we proposed in Section 3.3.

2. Although the query performance is satisfactory since the index structure can fit in main memory, we have not adequately stressed the index structure.

3. Some copies were not detected because we performed the similarity search only *approximately*: we searched only in the bucket where the query image resided but not in all the neighboring buckets.

Section 6 concludes with our plans for future enhancement.

6 Conclusions and Future Work

In this paper we have shown a new approach to detect unauthorized image copies on the internet. We proposed RIME, which uses wavelets to characterize images and indexes the images with an extensible hashing scheme. Our preliminary experimental results show that the wavelets are proper features for doing image copy detection work.

We are currently crawling more internet images to build a much larger database to test out the robustness of the feature vector and index structure. Specifically, we are doing the following:

1. Experiment with various color and shape filters to enhance the search result.

2. Implement the extensible multidimensional index structure for RIME.

3. Build a large image database to stress the performance of the index structure.

Acknowledgements

We would like to thank professor Shih-Fu Chang at Columbia University for his providing links to internet images.

References

[1] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: an efficient and robust access method for points and rectangles. *Proceedings of ACM Sigmod*, May 1990.

[2] S. Berchtold. The x-tree: An index structure for high-dimensional data. *Proceedings of the 22nd VLDB*, August 1996.

[3] S. Brin and H. Garcia-Molina. Copy detection mechanisms for digital documents. *Proceedings of ACM SIGMOD*, May 1995.

[4] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. *Proceedings of the 23rd VLDB*, August 1997.

[5] I. Cox, J. Kilian, F. Thomson, and T. Shamoon. Secure spread spectrum watermarking for multimedia. *IEEE Transaction on Image Processing*, 6(12):1673–86, December 1997.

[6] C. M. B. Dana H. Ballard. Computer vision. *Prentice-Hall*, 1982.

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

[8] H. Garcia-Molina, S. Ketchpel, and N. Shivakumar. Safeguarding and charging for information on the internet. *Proceedings of ICDE*, 1998.

[9] A. Gupta and R. Jain. Visual information retrieval. *Communications of the ACM*, 40(5):69–79, 1997.

[10] A. Guttman. R-trees: a dynamic index structure for spatial searching. *Proceedings of ACM Sigmod*, June 1984.

[11] L. M. Hurvich and D. Jameson. An opponent-process theory of color vision. *Psychological Review*, 64:384–90, 1957.

[12] N. Katayama and S. Satoh. The sr-tree: An index structure for high-dimensional nearest neighbor queries. *Proceedings of ACM SIGMOD*, May 1997.

[13] C. Li, E. Chang, and J. Z. Wang. An extensible hashing index for high-dimensional similarity search. *Stanford Technical Report*, August 1998.

[14] K.-L. Lin, H. V. Jagadish, and C. Faloutsos. The tv-tree: an index structure for high-dimensional data. *VLDB Journal*, 3(4), 1994.

[15] J. T. Robinson. The k-d-b-tree: A search structure for large multidimensional dynamic indexes. *Proceedings of ACM SIGMOD*, April 1981.

[16] J. T. Robinson. The k-d-b-tree: a search structure for large multidimensional dynamic indexes. *Proceedings of ACM Sigmod*, April 1981.

[17] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. *Proceedings of ACM Sigmod*, May 1995.

[18] J. R. Smith and S.-F. Chang. Visualseek: A fully automated content-based image query system. *ACM Multimedia Conference*, 1996.

[19] M. Swain and D. H. Ballard. Color indexing. *Int. Journal of Computer Vision*, 7(1):11–32, 1991.

[20] J. Ullman, H. Garcia-Molina, and J. Widon. Database system primciples lecture notes. 1998.

[21] J. Z. Wang, J. Li, and G. Wiederhold. Wise: A wavelet-based image search engine with efficient feature vector clustering and classification.

[22] J. Z. Wang, G. Wiederhold, O. Firschein, and S. X. Wei. Content-based image indexing and searching using daubechies' wavelets. *International Journal of Digital Libraries*, 1(4):311–28, 1998.

[23] D. A. White and R. Jain. Similarity indexing: Algorithms and performance. *Proc. SPIE Vol.2670, San Diego*, 1996.

[24] D. A. White and R. Jain. Similarity indexing with the ss-tree. *Proceedings of the 12th ICDE*, Feb. 1996.