WALRUS: A Similarity Retrieval Algorithm for Image Databases

Apostol Natsev*	Rajeev Rastogi	Kyuseok Shim
Duke University	Bell Laboratories	Bell Laboratories
Durham, NC 27708	Murray Hill, NJ 07974	Murray Hill, NJ 07974
natsev@cs.duke.edu	rastogi@bell-labs.com	shim@bell-labs.com

Abstract

Traditional approaches for *content-based image querying* typically compute a single signature for each image based on color histograms, texture, wavelet transforms etc., and return as the query result, images whose signatures are closest to the signature of the query image. Therefore, most traditional methods break down when images contain similar objects that are scaled differently or at different locations, or only certain regions of the image match.

In this paper, we propose WALRUS (WAveLet-based Retrieval of User-specified Scenes), a novel similarity retrieval algorithm that is robust to scaling and translation of objects within an image. WALRUS employs a novel similarity model in which each image is first decomposed into its regions, and the similarity measure between a pair of images is then defined to be the fraction of the area of the two images covered by matching regions from the images. In order to extract regions for an image, WALRUS considers sliding windows of varying sizes and then clusters them based on the proximity of their signatures. An efficient dynamic programming algorithm is used to compute wavelet-based signatures for the sliding windows. Experimental results on real-life data sets corroborate the effectiveness of WALRUS's similarity model that performs similarity matching at a region rather than an image granularity.

1 Introduction

Advances in image processing techniques, processor speeds and graphics capabilities of modern computers, coupled with the proliferation of the internet, have made hundreds of thousands of digital images easily accessible to users. Consequently, applications requiring content-based querying and searching of images abound, and can be found in a number of different domains that include data mining, multimedia messaging, medical imaging, weather prediction, insurance, TV production, satellite image databases and E-commerce.

*The work was done while the author was visiting Bell Laboratories.

1.1 Drawbacks of Existing Approaches

Traditionally, the problem of "query by content", or alternately, that of retrieving images that match a given query image from a large database of images has been solved by computing a *feature signature* for each image, mapping all signatures to d-dimensional points in some metric space (usually reducing dimensionality in the process), and building an index on all signatures for fast retrieval. An appropriate distance function (e.g., euclidean distance) is defined for each pair of signatures, and given a query, the index is used to efficiently locate signatures close to the query point. The set of images corresponding to the signatures are then returned to the user and constitute the result of the query.

Typical methods for computing signatures include color histograms, which can be used to characterize the color composition of an image, regardless of its scale or orientation [Nib93, FSN+95]. The problem with color histograms, however, is that they do not contain any shape, location or texture information. As a result, two images with similar color composition may in fact contain very different shapes, and thus be completely unrelated semantically. One approach to solving this problem is to define separate distance functions for color, shape, and texture, and subsequently combine them to derive the overall result. An alternate approach, proposed in [JFS95, WWFW98], is to use the dominant *wavelets* coefficients for an image as its signature – since wavelets capture shape, texture and location information in a single unified framework, their use ameliorates some of the problems with earlier algorithms.

A drawback of the schemes from [Nib93, JFS95, WWFW98] mentioned above is that they compute a single signature for the entire image. As a result, the methods usually fail when images contain similar objects, but at different locations or in varying sizes. For example, consider the two images in Figure 1 for which matching objects (enclosed in dotted rectangles) comprise more than 50% of each image. However, since the similar objects are at different locations in the two images, and there is very little similarity among the remaining objects, both wavelet signatures as well as

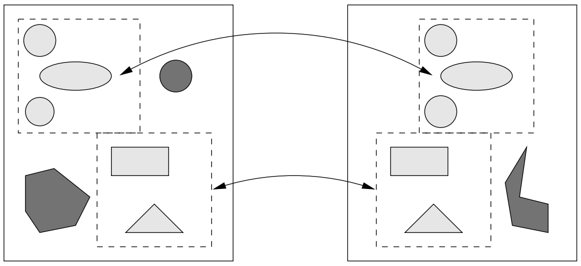


Figure 1: Images with Similar Objects

color histograms for the two images could be vastly different. Thus, the algorithms from [Nib93, JFS95, WWFW98] typically fail with respect to scaling and translation of objects within images, because a single signature computed for the whole image cannot sufficiently capture the important properties of individual objects. In contrast, we would like to build a system that is robust with respect to resolution changes, dithering effects, color shifts, orientation, size, and location, not only of the whole image, but of its individual objects as well.

A number of strategies for decomposing an image into its individual objects have been proposed in the literature [Smi97]. However, extracting regions from an image is a very hard problem to solve. Approaches that involve manually extracting objects can be extremely tedious and time-consuming, and are therefore impractical for large image collections. Consequently, most image segmentation techniques rely on being able to identify region boundaries, sharp edges between objects, and on a number of other factors, such as color, shape, connectivity, etc. However, besides being computationally expensive, the schemes are frequently inaccurate in identifying objects and the used methods are generally not robust with respect to object granularity. The reason for this is that the definition of an object is largely subjective – as a result, a single method cannot successfully identify the "correct" objects for all applications, and may decompose what the user perceives as a single object into several smaller objects. A number of image segmentation techniques, therefore, utilize domain-specific constraints and are thus application-specific.

1.2 Our Contributions

In this paper, we propose the WALRUS (WAveLet-based Retrieval of User-specified Scenes) similarity retrieval algorithm for the querying by content problem. Below, we present the main contributions of WALRUS over existing approaches.

Novel Similarity Model for Matching Images. WALRUS's image similarity model is designed specifically to address shortcomings with existing approaches for the cases when images contain similar regions but the region in one image is a translation or a scaling of the matching region in the other. WALRUS achieves this by computing and comparing signatures at the granularity of regions and not the entire image. Thus, instead of storing a single signature for each image as in [Nib93, JFS95, WWFW98], in WALRUS, we build a set of a variable number of signatures for an image, one signature for each image region. We then define the similarity measure between the query image Q and a target image T in terms of the fraction of the area of the two images covered by matching pairs of regions from Q and T (matching regions are determined based on the distance between their signatures). Our experimental study with real-life data sets indicates that, compared to existing approaches, WALRUS's similarity model results in significant improvements in the quality of images retrieved.

Extraction of Regions and their Signatures. WALRUS employs novel techniques for region extraction that are computationally efficient, domain-independent and avoid the complexity of traditional image segmentation algorithms based on identification of region boundaries, connectivity etc. To extract image regions and compute their signatures, in WALRUS, we consider *sliding* windows of varying sizes within the image. For each window, we compute signatures by mapping it to some d-dimensional space, and then perform clustering on the signatures in order to group windows with similar characteristics (e.g., color composition, texture) in a single cluster. Each cluster thus identifies a region of the image with related pixel values and we use the cluster representative (e.g., centroid) as the signature for the region. Since our similarity model compares images at the granularity of regions and regions represent variable-sized windows at varying locations, WALRUS effectively eliminates the scaling and translation problem, not only at the image level, but also at the object level.

Computation of Signatures for Windows. In WALRUS, we use the Haar wavelet [SDS96] transform for feature extraction and dimensionality reduction for all windows. (See Section 3.)

Even though computing the wavelet transform for an image requires time linear in the size of the image, the computation could become expensive when applied to thousands of windows within an image. WALRUS employs a dynamic programming algorithm for efficiently computing wavelets in the sliding variable-sized window framework by reusing computation as much as possible. Compared to naive algorithms, we show that the dynamic programming algorithm can speed up computation of wavelet signatures by more than an order of magnitude.

2 Related Work

Initial systems tackled the content-based image retrieval problem by using color histograms, texture and shape features. The most common systems in this class are IBM's QBIC system [Nib93, FSN$^+$95], the Virage system [GJ97] by Virage Inc., and the Photobook system [PPS95] from the MIT Media Lab. These systems allow the user to specify a particular color composition, texture, some shape feature, and perhaps a partial sketch or painting. The user is also required to place relative weights on these attributes in order to arrive at the combined similarity measure. The systems typically use a collection of features and distance metrics, and usually require several indices, which can increase both space and time requirements. In addition, the correct weighing of the individual feature distances is hard to do automatically and imposes a certain burden on the user.

Jacobs *et al.* [JFS95] addressed the problem by using wavelets to capture color, texture and shape in a unified framework, where the user is not required to specify any parameters. They used the simple Haar wavelet transform to compute a feature vector for each image. The wavelet representation is truncated retaining only forty to sixty of the largest magnitude coefficients, and harshly quantized so that the magnitude of the coefficients is essentially discarded and only their presence or absence is recorded. The authors experimented with different color spaces and image metrics, and found that the YIQ color space and a weighted version of the Huffman bitmap distance metric give best results for them. They also report experiments on the performance of their method for rotated, scaled, translated and color shifted images, which indicate a small tolerance to these operations.

The WBIIS system developed by Wang *et al.* [WWFW98] improved on the wavelet method of [JFS95] by using Daubechies' wavelets, a better distance metric, and a three-step search process. They generalized the distance metric to allow different weighing of the color components and the different subbands but their approach also suffers from the problem of correctly estimating these weights. The search process is performed in three steps: first a crude selection is done based on variances, then a refinement of the search is performed by comparing wavelet feature vectors based on a 4-level Daubechies' transform, and in the final stage, feature vectors from a 5-level transform are compared. The authors report better performance results than the other approaches described but their system still cannot handle region queries, translation, and scaling.

John Smith [Smi97] considered image query systems that integrate spatial and feature information, both at the image and region level. His system allows the user to specify the spatial location of regions, both in absolute terms as well as relative to each other. Each image is decomposed into regions by reverse-mapping region features from a finite library of patterns to the image. The library is obtained off-line, does not change dynamically, and it contains the feature representations for a fixed set of regions. The drawbacks of this system are the inconsistent quality of region segmentation and the limitations of the region mapping. Each image can be decomposed only into a pre-specified set of regions, so if the pattern library does not contain the correct regions, performance could suffer. After identifying the regions in an image, they are compared with each other, and the image similarity is computed based on the location of regions, their size, and relative position.

Guibas, Rogoff, and Tomasi [GRT95] considered a quadtree partitioning of an image where only windows with uniformly consistent features are preserved and other windows are discarded as containing multiple region information. For each window a Fourier signature is computed and compared to that of neighboring windows to determine if they are sufficiently alike. This process results in a primitive segmentation of the image, which the authors argue can be used for image retrieval but do not give any experiments to support it. The paper does consider three different spectral distance measures, though, and compares them for segmentation purposes. The drawback of this approach is again the fact that windows' sizes and locations are restricted, and thus it will not be able to handle scaling and translation of objects well.

3 Wavelets

Wavelets are useful for hierarchically decomposing functions in ways that are both efficient and theoretically sound. Broadly speaking, a wavelet representation of a function consists of a coarse overall approximation together with detail coefficients that influence the function at various scales [SDS96]. The wavelet transform has excellent energy compaction and de-correlation properties, which can be used to effectively generate compact representations that exploit the structure of data. By using wavelet subband decomposition, and storing only the most important subbands (that is, the top coefficients), we can compute fixed-size low-dimensional feature vectors independent of resolution, image size and dithering effects. Also, wavelets are robust with respect to color intensity shifts, and can capture both texture and shape information efficiently. Furthermore, wavelet transforms can generally be computed in linear time, thus allowing for very fast algorithms.

In this paper, we use Haar wavelets to compute feature signatures because they are the fastest to compute and have been found to perform well in practice [JFS95]. Haar wavelets enable us to speed up the wavelet computation phase for thousands of sliding windows of varying sizes in an image, and also facilitate the development of efficient incremental algorithms for computing wavelet transforms for larger windows in terms of the ones for smaller windows. One disadvantage of Haar wavelets is that it tends to produce blocky image artifacts for the most important subbands. However, in our application, the images constructed using signatures are never viewed and thus this is not our concern. We discuss the Haar wavelet transform for one and two dimensions in this section.

3.1 One-Dimensional Haar Wavelets

Suppose we are given a one-dimensional pixel image I = [2, 2, 5, 7]. The Haar wavelet transform for the above image can be calculated as follows. We first average the values together pairwise to get a new lower resolution image [2, 6]. Obviously, some information has been lost in this averaging process. To be able to restore the original four values of the image, we need to store some *detail coefficients*, that capture the missing information. In Haar wavelets, the difference of the (second of the) averaged values from the average itself constitutes the detail coefficients. Thus, for the first pair of averaged values, the detail coefficient is 0 since 2-2 =0, while for the second we need to store 1 since 7-6 = 1. Note that it is possible to reconstruct the 4 pixels of the original image from the lower resolution image containing the two averages and the two detail coefficients. By repeating the above process on the lower resolution image containing the averages recursively, we get the following full decomposition:

Resolution Level	Averages	Detail Coefficients
2	[2, 2, 5, 7]	
1	[2, 6]	[0, 1]
0	[4]	[2]

We define the *wavelet transform* of the original image with four pixels to be the single coefficient representing the overall average of the pixel values followed by the detail coefficients in the order of increasing resolution. Thus, the one-dimensional Haar wavelet transform for the original image is given by I' = [4, 2, 0, 1].

Each entry in I' is called a *wavelet coefficient*. Using the wavelet transform of an image, rather than the image itself has a number of advantages. One advantage is that a large number of detail coefficients tend to be very small values. Thus, truncating these small coefficients from the transform introduces only small errors in the reconstructed image, giving a form of "lossy" image compression.

Intuitively, the wavelet coefficients in the above example carry different weights with respect to their importance for the reconstructed image. For example, the overall average of the whole data set is more important than any of the detail coefficients because it affects the whole range of reconstructed values. In order to equalize the importance of all coefficients, we need to *normalize* the final wavelet coefficients appropriately. We achieve this by dividing each wavelet coefficient by $\sqrt{2}^i$, where i denotes the index of the approximation level the coefficient appears in (where level 0 is the finest resolution level). Thus, the wavelet transform for the previous example becomes I' = [4, 2, 0, 1/$\sqrt{2}$].

3.2 Two-Dimensional Haar Wavelets

There are two ways in which wavelets can be used to transform the pixel values in a two-dimensional image. Each of these transforms is a two-dimensional generalization of the one-dimensional wavelet transform described above. The first is called *standard decomposition*. We do not describe this method here because we use the second approach to compute wavelet transforms in WALRUS.

The second transform is called *non-standard decomposition*. In this method, we perform one step of horizontal pairwise averaging and differencing on the pixel values in each row of the image. Next, we apply vertical pairwise averaging and differencing to each column of the result. We then repeat this process recursively only on the quadrant containing averages in both directions. The procedure for computing the wavelet transform W for a $w \times w$ image I using non-standard decomposition is illustrated in Figure 2. In our coordinate system, the coordinate of the upper left corner pixel of I is [1,1], the lower left corner pixel is $[1, n]$ and the upper right corner pixel is $[n, 1]$. Each horizontal followed by vertical pairwise averaging and differencing involves pixels in the 2×2 box rooted at coordinates $[2i-1, 2j-1]$ for $1 \leq i, j \leq \frac{w}{2}$. The horizontal and vertical averaging of pixels in each 2×2 box rooted at $[2i-1, 2j-1]$ results in 4 new pixel values and these are computed in Steps 3–6. The upper left value (computed in Step 3) is the average of the 4 pixel values and is stored in a new $\frac{w}{2} \times \frac{w}{2}$ temporary matrix A whose primary purpose is to store averages on which the above averaging and differencing process will be recursively applied. The remaining 3 new pixel values denoting the upper right (Step 4), lower left (Step 5) and lower right (Step 6) pixels are assigned to pixel $[i, j]$ in the upper right, lower left and lower right $\frac{w}{2} \times \frac{w}{2}$ quadrants of W, respectively. These are the detail coefficients. Once the averages for all the 2×2 boxes have been computed, the computeWavelet procedure is recursively invoked on A in Step 9. For the two-dimensional Haar transform, the normalization factor is 2^i.

4 Image Similarity Model

Computing signatures at the granularity of an entire image may fail to capture the similarity between regions of two images. WALRUS's image similarity model is designed specifically to address the above shortcomings with existing approaches for the cases when images contain similar regions but the region in one image is a translation or a scaling of the matching region in the other. WALRUS achieves this by computing and comparing signatures at the granularity of regions and not the entire image. The similarity measure between a pair of images is then defined in terms of the fraction of the area covered by matching regions of the two images. Note that we do permit overlap between regions of an image.

We are now in a position to formally define WALRUS's similarity metric between two images Q and T consisting of regions $Q_1, \ldots, Q_m$ and $T_1, \ldots, T_n$, respectively. The following definitions make use of two user-specified parameters ϵ and ξ.

Definition 4.1: (Region similarity) A pair of regions is considered to be similar if one's signature lies within an envelope of ϵ width around the other. ∎

Earlier we mentioned that the signature of a region is the centroid of the cluster of signatures for the region. Alternately, we can also consider as the signature of a

```
procedure computeWavelet(I, W, w)
begin
for i = 1 to w/2 do
    for j = 1 to w/2 do {
        A[i,j] := (I[2i-1,2j-1] + I[2i,2j-1] + I[2i-1,2j] + I[2i,2j])/4
        W[w/2+i,j] := (-I[2i-1,2j-1] + I[2i,2j-1] - I[2i-1,2j] + I[2i,2j])/4
        W[i,w/2+j] := (-I[2i-1,2j-1] - I[2i,2j-1] + I[2i-1,2j] + I[2i,2j])/4
        W[w/2+i,w/2+j] := (I[2i-1,2j-1] - I[2i,2j-1] - I[2i-1,2j] + I[2i,2j])/4
    }
if w > 2
    computeWavelet(A, W, w/2)
else
    W[1,1] = A[1,1]
end
```

Figure 2: Procedure for Computing the Wavelet Transform

region, the bounding box of all signatures in the cluster for the region. As a consequence of Definition 4.1, when we use bounding boxes instead of centroids as signatures for regions, two regions are defined to be similar if the bounding rectangle for one after extending it by distance ϵ overlaps with the rectangle of the other.

Definition 4.2: (Similar region pair set) For images Q and T, the set of ordered pairs $\{(Q_1, T_1), \ldots, (Q_l, T_l)\}$ is referred to as a similar region pair set for Q and T if Q_i is similar to T_i and for $i \neq j$, $Q_i \neq Q_j$, $T_i \neq T_j$. ∎

Definition 4.3 (Image similarity) Two images Q and T are said to be similar if there exists a similar region pair set for Q and T $\{(Q_1, T_1), \ldots, (Q_l, T_l)\}$ such that:

$$\frac{area(\cup_{i=1}^{l}(Q_i)) + area(\cup_{i=1}^{l}(T_i))}{area(Q) + area(T)} \geq \xi$$

∎

In the above definition, $area(\cup_{i=1}^{l}(Q_i))$ is the number of pixels in Q covered by regions $Q_1, \ldots, Q_l$ considered together. Thus, two images are considered to be similar if the fraction of *match area* compared to the *total area* of the two images is above the user-specified threshold ξ. Note that, depending on the application, one could consider other variations for the similarity measure between images. One would be to simply measure the fraction of the query image Q covered by matching regions in Q; another, for images with different sizes would be to use twice the area of the smaller image in the denominator instead of the sum of the areas of the two images.

5 The WALRUS Similarity Retrieval Algorithm

5.1 Overview

Below, we provide a brief overview of each step involved in processing a query image Q. Indexing of images is done only once at the beginning and when new images are added to the database, while the steps for querying need to be repeated for each query image.

- **Generating Signatures for Sliding Windows**: In the first step, each image is broken into sliding windows (which could overlap) with different sizes ranging from $\omega_{min} \times \omega_{min}$ to $\omega_{max} \times \omega_{max}$. As the signature for each window, we use the s^2 coefficients from the lowest frequency band of the Haar wavelet transform for the window.

- **Clustering the Sliding Windows**: We next cluster the windows in the image using as the distance metric between a pair of windows, the euclidean distance between their signatures. Each cluster thus contains a set of similar windows which together define a region. The centroid of the cluster can be used as the signature for the region (alternately, the bounding box of all signatures in the cluster can be used as the signature, too). The query image is thus decomposed into a number of regions.

- **Region Matching**: For each region of the query image, we use a spatial index (R*-tree) to find *all* regions in the database that are similar, that is, regions whose signatures are within ϵ distance of a region of the query. (In the image indexing phase, the regions of each image in the database computed in the previous step are indexed using their signatures).

- **Image Matching**: The previous step computes all the pairs of matching regions (Q_i, T_j) for the query image Q and each target image T in the database. This information is used to compute the best similar region pair set for Q and T, that is, the one that covers the maximum area in the two images and thus maximizes the similarity measure between Q and T as defined in Definition 4.3.

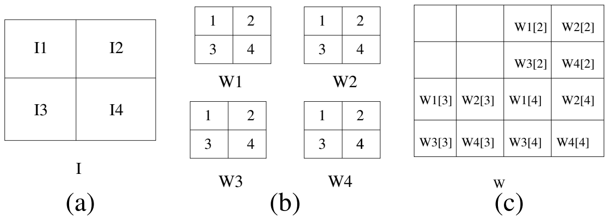


Figure 3: copyBlocks Procedure

procedure computeSingleWindow($W_1, W_2, W_3, W_4, W, \omega$)
begin
1. **if** $\omega = 2$ {
2. $W[1,1] := (W_1[1,1] + W_2[1,1] + W_3[1,1] + W_4[1,1])/4$
3. $W[2,1] := (-W_1[1,1] + W_2[1,1] - W_3[1,1] + W_4[1,1])/4$
4. $W[1,2] := (-W_1[1,1] - W_2[1,1] + W_3[1,1] + W_4[1,1])/4$
5. $W[2,2] := (W_1[1,1] - W_2[1,1] - W_3[1,1] + W_4[1,1])/4$
6. **return**
7. }
8. copyBlocks($W_1, W_2, W_3, W_4, W, \omega$)
9. computeSingleWindow($W_1, W_2, W_3, W_4, W, \frac{\omega}{2}$)
end

Figure 4: Procedure for Computing Wavelets for a Single Window

5.2 Computation of Wavelet Signatures for Windows

In this subsection, we present a dynamic programming algorithm to efficiently compute wavelet signatures for sliding windows in an image. An image is a two dimensional array of pixel values. Images can be represented using a number of different color spaces: RGB, HSV and YIQ, each of which store 3 color components for every pixel. The development in this section is for a single color value per pixel – the extension to multiple color channels is straightforward.

The complexity of naively computing wavelet coefficients of $\omega \times \omega$ windows rooted at every pixel in an $n_1 \times n_2$ image is $O(\omega^2(n_1 - \omega)(n_2 - \omega))$, which could be prohibitive. Instead, in WALRUS, we use a dynamic programming algorithm that incrementally computes the coefficients for larger windows using those computed for smaller windows – our algorithm thus reuses the computation performed for smaller windows. In particular, assuming that we have computed signatures for windows of size $\frac{\omega}{2} \times \frac{\omega}{2}$, we can compute signatures for windows of size $\omega \times \omega$ using the signatures of the smaller windows of size $\frac{\omega}{2} \times \frac{\omega}{2}$.

Intuition: Let I be the $\omega \times \omega$ window whose wavelet transform, W, we would like to compute and let I_1, I_2, I_3 and I_4 denote the four smaller $\frac{\omega}{2} \times \frac{\omega}{2}$ windows contained in it (see Figure 3(a)). Let W_1, W_2, W_3 and W_4 be the wavelet transforms for I_1, I_2, I_3 and I_4.

Consider a further decomposition of each of W_1, W_2, W_3 and W_4 into quadrants, and label them by 1, 2, 3 and 4 as shown in Figure 3(b). The quadrants 2, 3 and 4 denote the upper right, lower left and lower right detail coefficients generated by the first round of horizontal and vertical averaging and differencing on pixels in each of I_1, I_2, I_3 and I_4 by the computeWavelets procedure described in Section 3.2. Thus, were the computeWavelets procedure applied to I, then after the first round of averaging and differencing, the detail coefficients in the upper right, lower left and lower right quadrants of W would have values as shown in Figure 3(c).

Suppose A is the $\frac{\omega}{2} \times \frac{\omega}{2}$ matrix of average values of 2×2 boxes of pixels in I after the first round of averaging and differencing. Thus, the upper left quadrant of W is the wavelet transform of A, and A consists of averages of 2×2 boxes from I_1, I_2, I_3 and I_4 after the first round of averaging. Then, since quadrant 1 in each of W_1, W_2, W_3 and W_4 are the wavelet transforms of the averages of 2×2 boxes in I_1, I_2, I_3 and I_4, respectively, after the first round of averaging, $W_1[1], \ldots, W_4[1]$ are the wavelet transforms for the four quadrants of A. Thus, since the upper left quadrant of W is the wavelet transform for A, we simply need to repeat the earlier steps with $W_1[1], W_2[1], W_3[1]$ and $W_4[1]$ as the wavelet transforms for the four quadrants that comprise A and the upper left quadrant of W as the target where the wavelet transform for A is to be stored.

The above-described recursive process terminates when $W_1[1], W_2[1], W_3[1]$ and $W_4[1]$ each contain only a single value which is the average of all the pixel values in I_1, I_2, I_3 and I_4, respectively. At this point, values for the four upper left pixels of W, $W[1,1], W[1,2], W[2,1]$ and $W[2,2]$ can be computed by performing horizontal and vertical averaging over the four averages in $W_1[1], \ldots, W_4[1]$ as described in the computeWavelets procedure.

Algorithm for Computing Wavelets for Single Window: The procedure for computing the wavelet transform for a $\omega \times \omega$ window from the wavelets for its four subwindows is as shown in Figure 4. W_1, W_2, W_3 and W_4 are the wavelet coefficients for the four $\frac{\omega}{2} \times \frac{\omega}{2}$ subwindows of the window whose wavelet transform we wish to compute, and W is the target for the computed wavelet coefficients for the $\omega \times \omega$ window. If ω is 2, then in steps 2–5, we simply perform horizontal and vertical averaging and differencing on the upper left pixel of each of W_1, W_2, W_3 and W_4. Otherwise, the copyBlocks procedure is invoked which copies the upper right, lower left and lower right quadrants of W_1, W_2, W_3 and W_4 to the corresponding quadrants of W as described in Figure 3. The procedure then calls itself recursively to compute the coefficients for the upper left $\frac{\omega}{2} \times \frac{\omega}{2}$ quadrant of W using the coefficients in the upper left $\frac{\omega}{4} \times \frac{\omega}{4}$ quadrants of W_1, W_2, W_3 and W_4.

Note that since we are only interested in the $s \times s$ signature for each window, we are only interested in computing wavelet coefficients for the upper left $s \times s$ matrix of W. From our earlier discussion on the computation of the wavelet transform for a single window, it follows that this can be computed by recursively invoking copyBlocks on the $\frac{s}{2} \times \frac{s}{2}$ upper left coefficients of the wavelet transform for the four subwindows in the window. Thus, invoking procedure computeSingleWindow with $\omega = s$ computes the upper left $s \times s$ wavelet matrix for W with the upper left $\frac{s}{2} \times \frac{s}{2}$ wavelet matrices of its four subwindows W_1, W_2, W_3 and W_4 as the starting point.

Algorithm for Computing Wavelets Signatures for Sliding Windows: We next show how we can use dynamic programming to compute signatures for multiple sliding windows in an $n_1 \times n_2$ image. Procedure computeSlidingWindows in Figure 5 computes $s \times s$ signatures for all sliding windows in an image whose sizes are a power of 2 and do not exceed ω_{max}. Procedure computeSlidingWindows also accepts as an input parameter the number of pixels t by which to slide each window (that is, t is the horizontal/vertical distance between the upper left pixels of any two adjacent windows). The parameters s, ω_{max} and t are all required to be powers of 2. In a single iteration of the outermost for loop in Step 1, wavelet signatures are computed for all $\omega \times \omega$ windows in the image. In each successive iteration, ω is doubled, beginning with an initial value of 2 for ω until it reaches the maximum window size ω_{max}. The wavelet signature for a window of size ω and whose upper left pixel is at $[i, j]$ is stored in $W^{\omega}[i, j]$. The wavelet signature for each $\omega \times \omega$ window is computed using those computed in the previous iteration for the four $\frac{\omega}{2} \times \frac{\omega}{2}$ subwindows in it. $W^1[i, j]$ for every 1×1 window rooted at pixel $[i, j]$ is initialized to the value of the pixel $[i, j]$ (that is, the raw image intensity at a pixel is the signature for the 1×1 window containing the pixel).

In Step 2, we set $dist$, the distance between any two successive sliding windows to be the minimum of t and the window width ω. This is for alignment reasons and to ensure that for any $\omega \times \omega$ window, the wavelet signatures for its four $\frac{w}{2} \times \frac{w}{2}$ subwindows have been computed in the previous iteration.

In the for loops in steps 3 and 4, the coordinates $[x, y]$ for the upper left corner pixel for each window is varied from 1 to $n_1 - \omega + 1$ with increments of $dist$ in the horizontal direction, and from 1 to $n_2 - \omega + 1$ with increments of $dist$ in the vertical direction. Finally, in Step 7, procedure computeSingleWindow is invoked to compute the signature for the $\omega \times \omega$ window rooted at $[x, y]$ from signatures for its four $\frac{\omega}{2} \times \frac{\omega}{2}$ subwindows and the size of the signature to be computed $\min(\omega, s)$ is passed as a parameter.

Therefore, if we want to generate signatures for all windows with sizes that are a power of 2 and range from $\omega_{min} \times \omega_{min}$ to $\omega_{max} \times \omega_{max}$, passing ω_{max} as input parameter to the dynamic programming algorithm generates all the desired signatures.

Time and Space Complexity: The overall time complexity of computing signatures for all the sliding windows can be shown to be $O(NS \log_2 \omega_{max})$ where $N = n_1 n_2$ and $S = s^2$. The total auxiliary memory space required by procedure computeSlidingWindows can also be shown to be exactly NS, which is desirable. (See [NRS98] for details.)

procedure ComputeSlidingWindows(s, ω_{max}, t)
begin
1. **for each** $\omega \in [2, \omega_{max}]$ and that is a power of 2 **do** $\{$
2. $dist := \min(\omega, t)$
3. **for** $i = 0$ **to** $\frac{n_1 - \omega}{dist}$ **do**
4. **for** $j = 0$ **to** $\frac{n_2 - \omega}{dist}$ **do** $\{$
5. $x := i * dist + 1$
6. $y := j * dist + 1$
7. computeSingleWindow($W^{\frac{\omega}{2}}[x, y], W^{\frac{\omega}{2}}[x + \frac{\omega}{2}, y],$
8. $\frac{\omega}{2}[x, y + \frac{\omega}{2}], W^{\frac{\omega}{2}}[x + \frac{\omega}{2}, y + \frac{\omega}{2}], W^{\omega}[x, y], \min(\omega, s))$
9. $\}$
9. $\}$
end

Figure 5: Dynamic Programming Algorithm

Discussion: The naive scheme for computing the wavelet signatures individually for all windows needs only ω^2 space compared to the NS space required by the dynamic programming algorithm. However, the overall time complexity of the naive algorithm is $O(N\omega_{max}^2)$ which can be much worse than the $O(NS \log_2 \omega_{max})$ complexity for our dynamic programming algorithm since typically, signatures are much smaller than the windows themselves and thus $s << \omega_{max}$.

5.3 Clustering the Sliding Windows for an Image

Since an image may generate a sizeable number of sliding windows, in order to guarantee low response times, we are interested in clustering algorithms with linear time complexity (most of the clustering algorithms in the literature have at least quadratic complexity [DH73]). Also, since we would like to ensure that a cluster contains windows that are fairly alike, we would like to be able to specify a threshold on the radius of the cluster – that is, the maximum distance between the center of the cluster and a point in it. The pre-clustering phase of BIRCH [ZRL96], one of the state-of-the-art clustering algorithms for large data sets meets all of our requirements. The pre-clustering phase has time complexity that is linear in the input size and accepts an ϵ_c parameter which is the threshold on the cluster size.

The threshold parameter ϵ_c and the signatures of all the sliding windows in an image are given as inputs to BIRCH's pre-clustering algorithm. BIRCH, then, generates a set of clusters each of whose radius is generally within ϵ_c. The number of clusters typically increases with image complexity.

Each cluster defines a region of the image. Either the centroid or the bounding box of the signatures for windows in the cluster can be used as the signature for the corresponding region. For each region, we also compute a bitmap for the pixels of the image covered by windows in its cluster. This information is needed in the final step by our similarity metric to compute the area of an image covered by multiple matching (and possibly overlapping) regions. For each region, its signature along with its bitmap is stored in an R^*-tree [BKSS90] which is an efficient disk-based index structure for the storage and retrieval of high-dimensional data points. Note that in the bitmap for a region, we don't need to store a bit for each pixel – instead we could use a coarse representation in which a single bit is kept for each $k \times k$ array of pixels, thus decreasing the storage overhead by a factor of k^2.

5.4 Region Matching Algorithm

The regions for database images, generated in the previous step, are stored in a disk-based spatial index. Each region is indexed using either the centroid or the bounding box for signatures in its cluster. Given a query image Q, its regions are extracted using steps identical to those used for the database images. Then, the index is probed to find all regions in the database whose signatures are within ϵ distance of any of the query's regions. Thus, if signatures are bounding rectangles, then the bounding rectangles of regions in the query image are extended by ϵ and then the index is used to find all overlapping bounding rectangles in it. Since each bounding rectangle in the index represents a region belonging to an image in the database, this step effectively retrieves all database images that have at least one similar region with the query.

5.5 Image Matching Algorithm

For a query image Q, the previous region matching step retrieves all the regions in the database that match every region of Q. For a target image T, let $(Q_1, T_1), \ldots, (Q_n, T_n)$ be the matching pair of regions belonging to Q and T. Note that a single region from Q can match a number of regions from T and vice versa. We use this information on matching regions and the bitmap of the pixels covered by each region (stored in the index along with the signature of each region) to compute the overall similarity measure between images Q and T as defined in Section 4.

The quickest similarity metric to compute is one in which we simply union the bitmaps for the matching regions from Q and T, and then compute the area covered by the regions. The similarity is then the fraction of the total area of the two images that the computed area for the regions represents, as described in Definition 4.3. This is very fast (linear time complexity in n) and corresponds to relaxing the requirement that each region appear only once in a similar region pair set (see Definition 4.2). A drawback of this approach, however, is that the same query region may match a number of different regions in the target making the covered area in the target image large since all the matched regions get included in the computation of the area. However, this may be undesirable since very few regions from the query image match regions in the target, and thus, the covered area in the query itself could be small.

The above situation can be remedied by adopting the strict definition of a similar region pair set (see Definition 4.2) which restricts the relationship between regions of Q and T to be a one-to-one correspondence, that is, it prohibits a region from appearing multiple times in the similar region pair set. Ideally, we would like to compute the similar region pair set that results in the maximum value for the similarity measure. However, since we permit overlap between regions, the problem of determining such matching pairs of distinct regions from Q and T that maximize the covered area is an NP-hard problem.

Theorem 5.1: *Given matching pairs of regions (Q_1, T_1) $\cdots (Q_n, T_n)$, the problem of computing a similar region pair set with the maximum covered area is NP-hard.* ∎

Proof: See [NRS98]. ∎

Therefore, we also propose a greedy heuristic for computing the similar region pair set with the maximum area. The basic idea is to iteratively choose the best pair of matching regions that maximizes the area covered by the regions. A detailed description of our greedy heuristic can be found in [NRS98]. The time complexity of the greedy algorithm is $O(n^2)$, where n is the number of matching pairs of regions from Q and T.

For images T whose similarity to the query image Q exceeds the threshold ξ, we can perform an additional refined matching phase with more detailed signatures if the resulting increase in response time is acceptable.

6 Experimental Results

In this section, we study the performance of WALRUS and demonstrate its effectiveness for similarity retrieval on images. We first show that our dynamic programming algorithm for computing wavelet signatures for sliding windows is more than an order of magnitude faster than the naive scheme. We then compare the quality of the images retrieved by WALRUS and WBIIS[1] [WWFW98], an image indexing and retrieval system developed at Stanford University. Our results indicate that, in general, the images returned by WALRUS are semantically more similar to the query image that those returned by WBIIS.

In all of our experiments, we used euclidean distance as the distance metric. We performed experiments using a Sun Ultra-2/200 machine with 512 MB of RAM and running Solaris 2.5.

6.1 Algorithms

The WBIIS algorithm uses Daubechies wavelets to compute a single signature for each image, and a brief description of it was presented earlier in Section 2.

Our WALRUS implementation currently handles several color spaces for images including YCC and RGB. We present the result with YCC space only in the paper due to the lack of space. Additional results with different color spaces can be found in [NRS98]. We implemented both

[1] http://www-asd-test.stanford.edu/žwang/imsearch.

the naive algorithm as well as our dynamic programming algorithm for computing wavelet signatures for sliding windows. In our implementation of WALRUS, to develop portions of our code, we used the following libraries that are available for free on the internet.

BIRCH Clustering Algorithm: We used the pre-clustering phase of the BIRCH implementation[2] provided to us by the authors of [ZRL96]. We set parameter values to the default values suggested in [ZRL96]. (See [NRS98] for details.)

$R^\star$-tree packages: To build the disk-based $R^\star$-tree index, we used the GiST C++ library[3] that is an implementation of the Generalized Search Tree, a template index structure that makes it easy to implement any type of hierarchical access method. Currently, the libgist distribution comes prepackaged with a B-tree and an R-tree extension.

ImageMagick Library: The ImageMagick[4] image display program can read and write many of the more popular image formats including JPEG, TIFF, PNM, GIF, and Photo CD. In addition, it can convert image files into different color spaces, resize, rotate, sharpen, color reduce, or add special effects to an image. The ImageMagick library can handle a number of color spaces including RGB, XYZ, YCC, YIQ and YUV. WALRUS uses the ImageMagick library to display images as well as to convert them to specific formats and color spaces.

6.2 Data Sets

In order to evaluate the quality of images retrieved by WALRUS, we used a real-life image dataset called *misc* from the web site for the image search engine WBIIS at Stanford University. This dataset was provided to us by James Wang at Stanford University who downloaded it from VIRAGE's web site a few years ago. The misc database contains 10000 images stored in JPEG format, each of whose sizes is either 85×128, 96×128 or 128×85.

6.3 Effectiveness of Dynamic Programming Algorithm

We measured the execution times of our algorithm and the naive algorithm for a wide range of parameter values. Three important factors that affect the performance of the algorithms are: image size, sliding window size and signature size. We fixed the image size to be 256×256, and varied the sliding window size and signature size (we set the distance between any two adjacent sliding windows to be 1 pixel). We excluded from our measurements of the execution time, the time spent for reading the image; thus, the running times represent the actual wavelet computation times for all sliding windows in the image.

Figure 6(a) plots the running times of both algorithms for computing 2×2 signatures as the sliding window size is varied from 2×2 to 128×128. The numbers 2^i along the x-axis represent a window size of $2^i \times 2^i$. Since the sliding window size is always a power of two, the x-axis is shown in log scale. Observe that the execution time increases slowly for the dynamic programming algorithm as the window size increases. However, the performance of the naive algorithm degrades significantly as the window size grows. The reason for this is that the time complexity of the naive algorithm is proportional to the square of window size, while the execution time of the dynamic programming algorithm is proportional to the logarithm of the window size. For a window size of 128×128, the naive algorithm is about 17 times slower than our dynamic programming algorithm, and its running time is close to 25 seconds.

Figure 6(b) shows the execution times for the algorithms as we increase signature sizes for a fixed sliding window size of 128×128. From the graph, it follows that the execution time of the dynamic programming algorithm increases slowly as the signature size is increased, while that of the naive algorithm stays almost constant at 25 seconds. However, even for signature sizes as large as 32×32, the dynamic programming algorithm is almost 5 times faster than the naive algorithm. Since typical signature sizes can be expected to be between 2×2 and 8×8 (due to the inability of existing indices to handle high-dimensional data), in most practical situations, our dynamic programming algorithm should be far superior to the naive algorithm.

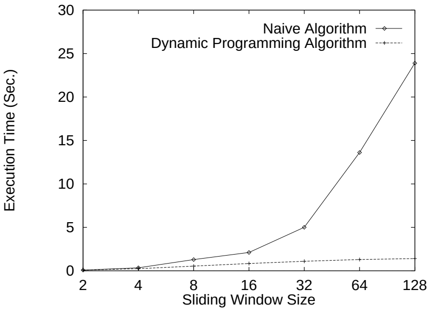
6.4 Effectiveness of WALRUS's Similarity Model

In order to determine the effectiveness of WALRUS's new similarity model, we compared how semantically related images retrieved by WALRUS and WBIIS are, to a given query image. Due to space constraints, we show here only the results obtained for one query image. The results with other query images are presented in [NRS98].

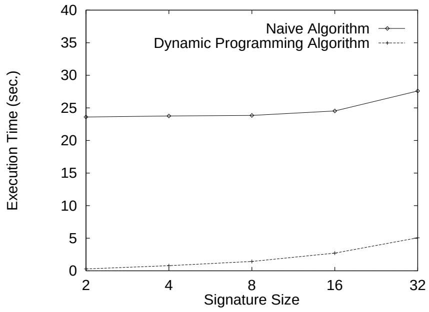
The query image that we consider contains red flowers with green leaves in the background (see Figure 7(a)). The query image has an Id of 866 in the misc database. The top 14 similar images found by WBIIS are as shown in Figure 7 in the increasing order of rank. As we can see from the figure, some of the returned images are very different from the query image. For example, the image in Figure 7(d) is a wall with orange and dark brown bricks. The image in Figure 7(g) is a picture of sunset on the ocean, while Figure 7(k) contains a yellow dog sitting on a green lawn.

In all, about 7 images are semantically unrelated to the query image. One of the reasons for this is that WBIIS generates a single wavelet signature for each image in the database. Since wavelets capture location, color and texture information, a number of images returned by WBIIS contain either green or red in similar locations as the query image. For instance, figures 7(h) and 7(k) have green in the background, while figures 7(d) and 7(g) have red/orange towards the center of the image. All of these images are

[2] http://www.cs.wisc.edu/z̃hang/birch.html.
[3] http://epoch.CS.Berkeley.EDU:8000/gist/libgistv1.
[4] http://www.wizards.dupont.com/cristy/ImageMagick.html.



(a) Window Size

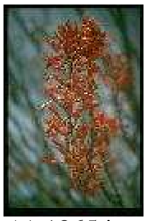


(b) Signature Size

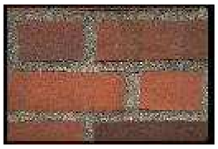
Figure 6: Computing Wavelet Signatures for Sliding Windows

(a) Query Image

(b) 3763.jpg



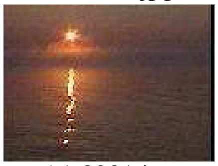
(c) 1865.jpg



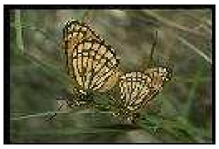
(d) 1002.jpg

(e) 881.jpg

(f) 812.jpg



(g) 8301.jpg



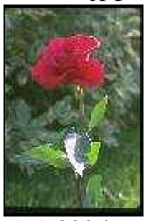
(h) 86.jpg

(i) 845.jpg

(j) 1610.jpg

(k) 3566.jpg

(l) 3427.jpg



(m) 822.jpg

(n) 826.jpg

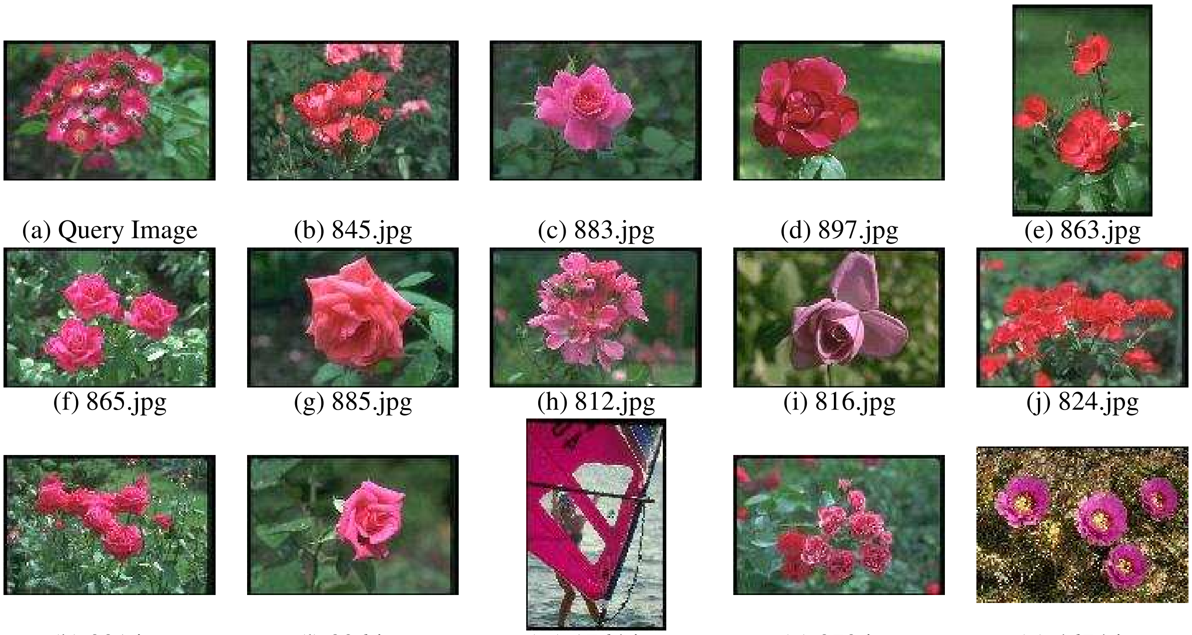
(o) 825.jpg

Figure 7: Images Found by WBIIS

very different from the original image having red flowers with green leaves in the background.

In contrast, the top 14 matching images retrieved by WALRUS are as shown in Figure 8. As the images illustrate, all images except the image in Figure 8(m) contain either red or pink flowers. Even the image in Figure 8(m) that is unrelated to the query image has a girl windsurfing with a red color sail in the picture. One of the reasons why WALRUS performs so well is that in the matching images containing red/pink flowers, the flowers appear at different locations in the images and in different sizes. In the query image, the fairly large bunch of red flowers appear somewhat in the center, slightly shifted to the left. However, in figures 8(f), 8(g) and 8(o), the flowers are placed more towards the right, while in figures 8(b), 8(e), 8(j), 8(k) and 8(n), flowers are distributed all over the image. Also, figures 8(c), 8(d), 8(i) and 8(l) contain a single flower (and not a large bunch) close to the center of the picture. Thus, since WALRUS's similarity model is more robust with respect to the scale and position of image objects, it is more effective in identifying related images (containing similar regions but at different locations or in varying sizes).

In WALRUS, the images shown in Figure 8 were retrieved using the following values for the various parameters. A fixed window size of 64×64 was used for sliding windows and 0.05 was employed for ϵ_c, the epsilon value for clustering. A 2×2 signature was used for each color space, and the cluster centroid was chosen as the representative for the corresponding region. Thus, the resulting signature for each image region was a 12-dimensional point. Also, with each region, we stored a 16×16 (32 byte) bitmap



(a) Query Image (b) 845.jpg (c) 883.jpg (d) 897.jpg (e) 863.jpg
(f) 865.jpg (g) 885.jpg (h) 812.jpg (i) 816.jpg (j) 824.jpg
(k) 881.jpg (l) 896.jpg (m) 1764.jpg (n) 878.jpg (o) 4674.jpg

Figure 8: Images Found by WALRUS (YCC)

in the index. The epsilon value, ϵ, which is the distance between the signatures of matching regions was set to 0.085 and the color space used was YCC. In the image matching step, we used the quick algorithm for computing the similarity measure between images that simply unions the matching regions between the query image and the target image retrieved using the disk-based index, and then computes the matching area fraction. Thus, even with very simple matching algorithms, using WALRUS, we were able find similar images effectively.

6.5 Query Response Time

To get a feel for query response times with WALRUS, we measured the time WALRUS takes to retrieve similar images for the query image in Figure 8(a) when the querying epsilon, ϵ, is varied from 0.05 to 0.09. Two important factors that affect query response times in WALRUS are 1) the region selectivity, that is, the number of matching regions retrieved for a query, and 2) the number of database images containing the matching regions. In the experiments we performed to see the effects these parameters have on query response times, we set the cluster epsilon, ϵ_c, to be 0.05 and stored images using YCC. With this cluster epsilon, WALRUS extracted 18 regions from the query image. We used 64×64 as the sliding window size, 2×2 signatures (for each color channel) and used the cluster centroid as the representative signature for each region.

We list in Table 1 the response time, the average number of regions retrieved per region of the query image and the total number of distinct images that contain regions matching at least one of the query image's regions, as the querying epsilon is varied from 0.05 to 0.09. The response time includes the time to perform all steps of WALRUS – this includes reading the image file from disk, converting it into YCC format, computing wavelet signatures for the 64×64 sliding windows, clustering the windows using BIRCH, retrieving similar regions for each query region from the disk-based R*-tree index, and matching images based on the fraction of the area covered by similar regions.

From Table 1, it follows that as ϵ is increased, the average number of matching regions per region of the query image increases (which is not surprising). As a result, more images need to be processed and consequently, the query response times increase, too. However, the query response times range from 5 seconds to about 20 seconds, which are quite reasonable. The results thus indicate that WALRUS is practical to use even though it uses a very general similarity model.

6.6 Number of Regions Per Image

We also computed the average number of regions generated for the query image (in Figure 8(a)) as ϵ_c is varied from 0.025 to 0.1. The average number of clusters generated with RGB is typically four times more than the corresponding number for YCC. Also, the number of clusters generated decreases as ϵ_c is increased. Due to the lack of space, we present details in [NRS98].

7 Conclusions

WALRUS employs a novel similarity model in which each image is first decomposed into its regions, and the similarity measure between a pair of images is then defined to be the

Querying Epsilon (ϵ)	Response Time (sec)	Avg. No. of Regions Retrieved	No. of Distinct Images
0.05	5.19	15	65
0.06	6.67	49.9	153
0.07	9.42	148.3	344
0.08	13.61	834.9	718
0.09	19.86	890.7	1287

Table 1: Query Response Time (Selectivity)

fraction of the area of the two images covered by matching regions from the images. In order to extract regions for an image, WALRUS considers sliding windows of varying sizes and then clusters them based on the proximity of their signatures. An efficient dynamic programming algorithm is used to compute wavelet-based signatures for the sliding windows. Thus, unlike traditional approaches for *content-based image querying* that typically compute a single signature for each image, in WALRUS, we build a set of a variable number of signatures for an image, one signature for each image region. Furthermore, by performing similarity matching at the region rather than image granularity, WALRUS's image similarity model can handle the cases when images contain similar regions but the region in one image is a translation or a scaling of the matching region in the other. Experimental results on real-life data sets suggest that the images retrieved by WALRUS are semantically more related to the query image than those retrieved by traditional methods. Our experiments also indicate that it is possible to easily obtain more than an order of magnitude speedup with the dynamic programming algorithm for computing wavelet signatures compared to naive algorithms.

Acknowledgments: We would like to thank Narain Gehani, Hank Korth and Avi Silberschatz for their encouragement for this work. Without the support of Yesook Shim, it would have been impossible to complete this work.

References

[BKSS90] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R^*-tree: an efficient and robust access method for points and rectangles. In *Proceedings of ACM SIGMOD*, pages 322–331, Atlantic City, NJ, May 1990.

[DH73] Richard O. Duda and Peter E. Hard. *Pattern Classification and Scene Analysis*. A Wiley-Interscience Publication, New York, 1973.

[FSN+95] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, B. Dom, Q. Huang, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker. Query by image and video content: the qbic system. *IEEE Computer*, 28(9):23–32, 1995.

[GJ97] Amarnath Gupta and Ramesh Jain. Visual information retrieval. *Communications of the ACM*, 40(5):69–79, 1997.

[GRT95] L. J. Guibas, B. Rogoff, and C. Tomasi. Fixed-window image descriptors for image retrieval. In *Storage and Retrieval for Image and Video Databases III*, volume 2420 of *SPIE Proceeding Series*, pages 352–362, Feb. 1995. Available at http://vision.stanford.edu/public/publication/guibas/guibasSrivd95.ps.gz.

[JFS95] C. E. Jacobs, A. Finkelstein, and D. H. Salesin. Fast multiresolution image querying. In *Proc. of SIGGRAPH 95*, Annual Conference Series, pages 277–286, August 1995. Available at http://www.cs.washington.edu/research/projects/grail2/www/pub/abstracts.html.

[Nib93] W. Niblack *et al.* The qbic project: Query image by content using color, texture and shape. In *Storage and Retrieval for Image and Video Databases*, pages 173–187, San Jose, 1993. SPIE.

[NRS98] P. Natsev, R. Rastogi, and K. Shim. WALRUS: A similarity matching algorithm for image databases. Technical report, Bell Laboratories, Murray Hill, 1998.

[PPS95] A. Pentland, R. W. Picard, and S. Sclaroff. Photobook: Content-based manipulation of image databases. In *SPIE Storage and Retrieval Image and Video Databases II*, San Jose, 1995.

[SDS96] E. J. Stollnitz, T. D. DeRose, and D. H. Salesin. *Wavelets for Computer Graphics: Theory and Applications*. Morgan Kaufmann, 1996.

[Smi97] J. R. Smith. *Integrated Spatial and Feature Image Systems: Retrieval, Compression and Analysis*. PhD thesis, Graduate School of Arts and Sciences, Columbia University, Feb. 1997. Available at http://www.ctr.columbia.edu/∼jrsmith/publications.html.

[WWFW98] James Ze Wang, Gio Wiederhold, Oscar Firschein, and Sha Xin Wei. Content-based image indexing and searching using daubechies' wavelets. *Intl. Journal of Digital Libraries (IJODL)*, 1(4):311–328, 1998. Available at http://www-db.stanford.edu/∼zwang/project/imsearch/IJODL97/.

[ZRL96] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: An efficient data clustering method for very large databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 103–114, Montreal, Canada, June 1996.