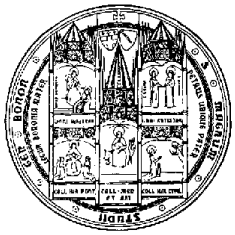
Università degli Studi di Bologna

Dipartimento di Elettronica Informatica e Sistemistica

Dottorato di Ricerca in Ingegneria Elettronica ed Informatica

XIV Ciclo



Efficient and Effective Similarity Search in Image Databases

(Tecniche Efficienti ed Efficaci di Ricerca per Similarità in Database di Immagini)

Tesi di:
Dott. Ilaria Bartolini

Coordinatore:
Chiar.mo Prof. Ing. Fabio Filicori

Relatore:
Chiar.mo Prof. Ing. Paolo Ciaccia

"The mediocre teacher tells,
The good teacher explains,
The superior teacher demonstrates,
The great teacher inspires."

William Arthur Ward

Contents

Chapter 1

Introduction

This thesis presents efficient and effective techniques for similarity search in image databases (DBs).

1.1 Motivation

First of all, let us explain what the terms "efficient" and "effective" exactly mean in our view.

Definition 1.1 (Efficiency)
The ability to obtain results in real time and with low cost in terms of storage overhead.

Definition 1.2 (Effectiveness)
The ability to obtain "good" results satisfying the user's expectations.

The effectiveness concept represents one of the basic objectives of research in the area of pattern recognition, where specific assumptions on the application domain are done in order to obtain a more powerful description of image content. Let us give an example: A fingerprint retrieval system for criminal investigations can base the retrieval on the specific properties that characterize a fingerprint (e.g. ridges, minutiae, etc.). On the other hand, efficiency is a typical goal of the DB community research, that provides similarity search methods not relying on any assumption on the application domain. This is the case, for example, of a general-purpose image retrieval system, which is exactly what we aim to.

Starting from above observations, the challenge of this work is to let both efficiency and effectiveness aspects coexist within the context of DB research, that is:

> *Define efficient and effective techniques for image similarity search without any assumption on the specific domain.*

From state-of-the-art of image similarity search, it is possible to observe that image retrieval systems characterize images by means of relevant properties (*features*), such as color distribution, texture information, and shape descriptors, and then use such feature values to determine those images which are most similar to a given query image. However, this approach is not adequate when images have a complex, not homogeneous, structure, in which case using *global features* leads to inaccurate content descriptions. A more effective way to characterize image content is based on extraction of *local features* together with the definition of correct query processing algorithms able to support efficient retrieval.

In recent years, it has been demonstrated that the interaction between the system and the user can further increase the quality of results. Thus, several methods have been proposed for implementing interactive similarity queries: Common to all these methods is the idea to exploit user judgments in order to progressively refine the query parameters and to eventually converge to an "optimal" parameter setting. The new query (with the new parameters) is then compared to the collection images, returning an improved set of images to the user. However, all these methods also share the feature to "forget" user preferences across multiple query sessions, thus requiring the feedback loop to be restarted for every new query, which is frustrating from the user's point of view and constitutes a significant waste of system resources. So, a solution able to overcome these limitations, by skipping, or cutting the tedious process of feedback interaction, should be appreciated to speed up the efficiency of the retrieval.

1.2 Summary of Contributions

The main contributions of this thesis in defining efficient and effective similarity search techniques in image DBs are summarized as follows:

1. We present WINDSURF (Wavelet-based INDexing of imageS Using Region Fragmentation), a new general approach to content-based image retrieval relying on a wavelet-based local features extraction supported by image fragmentation. So doing, WINDSURF is able to characterize in an effective way the content of each image. From the query processing point of view, WINDSURF then defines the first *correct* index-based algorithm for region-based image similarity that ensures, in an efficient way, that all objects satisfying the query appear in the result set.

2. We describe FeedbackBypass, a new and efficient approach to interactive similarity query processing. It complements the role of relevance feedback engines by storing and maintaining the query parameters, determined with feedback loops over time, using a wavelet-based data structure. For each query, a favorable set of query

parameters can be determined and used to either "bypass" the feedback loop completely for already-seen queries, or to start the search process from a near-to-optimal configuration. It has to be noted that FeedbackBypass can be well combined with all state-of-the-art relevance feedback techniques working in high-dimensional vector spaces. Its storage requirements scale linearly with the dimensionality of the query space, thus making even sophisticated query spaces amenable.

3. We formalize a relevance feedback model for region-based image retrieval systems, able to further increase the effectiveness of WINDSURF results, and more in general, of all region-based image retrieval systems. It is important to note that, prior to this, to the best of our knowledge no relevance feedback model has been proposed for the region-based image retrieval.

1.3 Thesis Outline

The work of the present thesis is organized in six Chapters and two Appendices as follows: In Chapter 2, we explore the state-of-the-art on similarity search, and interactive similarity search in image DBs, pointing out major limits suffered by current image retrieval systems and by strategies of present user-system interactions.[1] Chapter 3 presents the solution we propose to improve the effectiveness and the efficiency of content-based image retrieval: Such solution is represented by the WINDSURF system. Having shown the wavelet-based modality used by WINDSURF to extract local properties from each image, we detail its similarity model, together with its correct index-based query processing algorithm. Experimental results, demonstrating both the effectiveness and the efficiency of the proposed techniques, are reported. In Chapter 4, we describe the efficient approach to complement the role of relevance feedback engines represented by FeedbackBypass. After presenting the main principles on which FeedbackBypass relies, we detail the wavelet-based structure representing the heart of the system. Experimental results demonstrate both effectiveness and efficiency of our technique. In the first part of Chapter 5, we then present the definition of the *first* relevance feedback model for region-based image retrieval systems, that we studied for WINDSURF, but that can be applied to every other system that fragments images into regions. In the second part, a shape-based retrieval is explored. Note that, even if the WINDSURF system, in its originally version, implicity extracts shape information, it does not use it in the retrieval phase. Details and experimental results on the proposed Fourier-based approach, that is scale-, translation- and rotation-invariant, are shown. In Chapter 6, we conclude our work and present some open problems that we

[1] Examples of interactive similarity search are shown in Appendix B.

plan to investigate in future research. Finally, Appendix A reports mathematical details and explanative examples on the Wavelet Transform and Appendix B presents examples of interactive similarity search by using a simple prototype application.

Chapter 2

Background on Similarity Search in Image Databases

The advent of multimedia age poses a number of new challenges to DB researchers. In particular, digital image libraries require effective and efficient automated retrieval based on the "semantic" content of images. The boost of graphics capabilities in modern computer systems and the growing of Internet have further contributed to the increased availability of digital images. In classical DBs, given a query object, where most of the attributes are either textual or numerical, the system has to determine which DB object is the "same" as the query. Results for this kind of searches is the set of DB objects whose attributes match those specified in the query. Traditional approaches to characterize the content of images rely on text surrogates, where human experts manually annotating each image with a textual description, so that text-based information retrieval techniques can be applied [Sal89]. This approach has the advantage of the inheritance of efficient technology developed for text retrieval, but is clearly impracticable in case of very large image DBs. Moreover, its effectiveness is highly dependent on the subjective opinions of the experts, who are also likely to supply different descriptions for the same image [OS95]. Even if the *matching* search paradigm has been proven to be an efficient method to retrieve data of interest in classical DB systems, it can not be successfully applied within the context of image DBs, and more in general, in multimedia DBs due to the complexity of multimedia objects for which matching is not expressive enough. Quoting from [SJ96]:

> "We believe that future image databases should abandon the matching paradigm, and rely instead on similarity searches. In similarity search we do not postulate the existence of a target image in the database. Rather, we order the images with respect to similarity with the query, given a fixed similarity criterion."

This prediction was right: Today, similarity queries arise naturally in a large variety of applications, like:

- E-commerce (e.g. electronic catalogues).
- Medical databases (e.g. ECG, X-ray and TAC).
- Edu-tainment (e.g. video clips, art images, space photographs and geological maps).
- Weather prediction.
- Criminal investigations.

As estimated from above sentence, similarity search can overcome drawbacks of traditional approaches by using numerical features computed by direct analysis of the information content. Content-Base Image Retrieval (CBIR) has been proposed in the early 1990's. CBIR systems use visual features to represent the image content. This approach is favorable since features can be computed automatically, and the information used during the retrieval process is always consistent, not depending on human interpretation. In detail, the user sketches the query image, or select a prototype image, searching for something similar. The result of this kind of queries is a list of images sorted by decreasing values of similarity to the query image. It is immediate, hence, the need for similarity search to define an appropriate similarity criterion, able to measure the grade of similarity between two images using only low level image properties (i.e. no human experts should provide additional information). Moreover, an efficient way to obtain the most similar DB images to the query has to be defined. This goal is usually performed using index structures on the image content descriptors. In other words, each of these image content descriptors, represented by a feature vector, is stored and indexed in the DB so that, at query time, the feature vector of the query image is computed and the DB is searched for the most similar feature vectors.

In the rest of this introductory Chapter, we illustrate possible approaches to define the image content (giving some concrete examples) and describe which similarity query models and modalities of query processing can be used. Various academic and commercial CBIR systems are then presented. We conclude describing which are the advantages of interaction between the user and a CBIR system, discussing the basic feedback approaches.

2.1 Feature Extraction, Similarity Models and Query Processing

To characterize DB images, modern CBIR systems define a set of low level relevant properties (*features*) able to effectively characterize the content of the images and then use such features for retrieval purposes [GR95, SWS+00]. The features should be "simple enough" to allow the design of automatic extraction algorithms, yet "meaningful enough" to capture the image content. To this end, recent studies have highlighted the fact that *global features*, like color and texture, indeed possess a rich semantic value, and as such they are used by several CBIR systems [FSN+95, SO95, PPS96]. Under this view, each image is typically represented by a high-dimensional *feature vector*, whose dimensionality depends on the number and on the type of extracted features, and similarity between images is assessed by defining a suitable distance function on the resulting feature space [Fal96].

It is a fact that CBIR systems that rely on global features cannot support queries like, say, "Find all the images containing a small red region under a big blue region" that refer to *local* properties of the images. Thus, the need to extract not only global but also *local features* has emerged, and a number of *region-based* image retrieval systems, that fragment each image into a set of "homogeneous regions", have been presented [SC96, MM99, CTB+99, NRS99, WLW01]. In region-based systems, similarity assessment between images is performed by associating regions in the query image with those contained in the DB image and by taking into account similarity between associated regions. To this end, features are extracted for each region and a distance function is used to compare regions' descriptors. Existing systems, however, either consider a scenario, which is beyond the scope of the present work, where also spatial constraints are taken into account [BDV99], or use naïve heuristic matching algorithms which are not guaranteed to return the correct results. As an example, suppose that a user looks for images containing two tigers: In this case the query image will contain (at least) two regions, each representing a tiger. If a DB image contains a single "tiger" region, clearly it is not correct to associate both query regions to the single tiger region of the DB image. However, as we will argument in Section 2.2, this can easily happen with existing region-based systems.

In the present work we focus our attention on the processing of k nearest neighbors (best-matches) queries, where the user asks for the k images in the DB which are most similar, according to the similarity measure implemented by the CBIR system, to the query image. Range queries, where the user has to specify a minimum similarity threshold α that images have to exceed in order to be part of the result, are not well suited for

the scenario we envision. In fact, since the user has no a priori knowledge on the distribution of similarities between images, he/she has no way to guess the "right" value for α. Indeed, a high value of α can easily lead to an empty result, whereas slightly decreasing α could result in an overwhelming number of returned images. This situation is further complicated in region-based retrieval, where more than one threshold could be required (see Section 2.2.1).

In the following we describe in detail examples of similarity search for image environments.

2.1.1 Similarity Search Examples

As we saw in previous Section, CBIR systems provide access to content of images extracting features like color, shape and texture. All these systems then use feature-based approaches to index image information [SWS+00]. Note that feature extraction is a complex process, that cannot be accurately discussed in the context of the present thesis; detailed information can be found in [Smi97].

Image Retrieval by Color Representation

The distribution of colors in an image is usually represented by an histogram. Each pixel of an image $O[x, y]$ consists of three *color channels* $O = (O_R, O_G, O_B)$, representing red, green, and blue components. These channels are transformed, by way of a *transformation matrix* T_c, into the natural components of color perception, that is, hue, brightness, and saturation (HSV color space). Finally, the three latter channels are quantized, through a quantization matrix Q_c, into a space consisting of a finite number M of colors. The m-th component of the histogram, $h_c[m]$ is given by:

$$h_c[m] = \sum_x \sum_y \begin{cases} 1 & \text{if } Q_c(T_c O[x, y]) = m \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

Each image is, therefore, represented by a point in a M-dimensional space. The simplest case, shown in Figure 2.1, is represented by color histograms with only three reference colors (e.g. red, green, and blue). In detail, two color histograms are computed starting from the two images; then, similarity comparison between images is performed on the color vectors $\mathbf{p}_1$ and $\mathbf{p}_2$.

Common approaches, however, usually define a much larger number of color bins, e.g. 64, 116, or 256. In all cases, to compare histograms of different images (e.g. $\mathbf{p}$ and $\mathbf{q}$), a distance function on such a space is needed. Relevant examples of distance functions

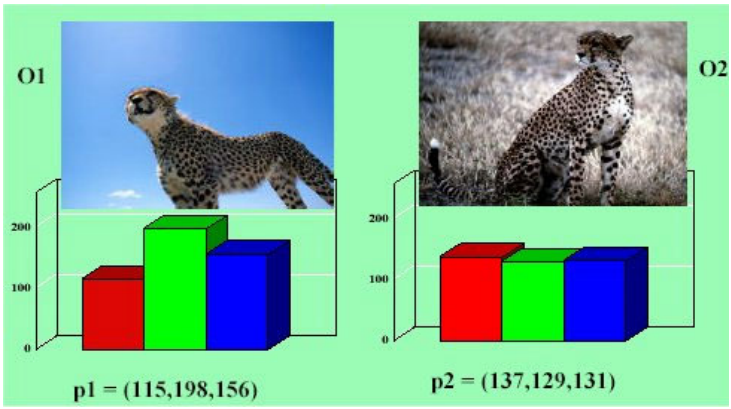


Figure 2.1: Color histogram extractions using 3 colors.

include L_p norms

$$L_p(\mathbf{p},\mathbf{q}) = \left(\sum_{i=1}^{D}(|p_i - q_i|)^p\right)^{1/p} \qquad 1 \leq p < \infty \tag{2.2}$$

$$L_\infty(\mathbf{p},\mathbf{q}) = \max_i\{|p_i - q_i|\} \qquad p = \infty \tag{2.3}$$

(L_1 is the Manhattan distance, L_2 is the Euclidean norm, L_∞ is the "max-metric") and their weighted versions. For instance, the weighted Euclidean distance is:

$$L_{2,W}(\mathbf{p},\mathbf{q};\mathbf{W}) = \left(\sum_{i=1}^{D} w_i\,(p_i - q_i)^2\right)^{1/2} \tag{2.4}$$

where $W = (w_1, ..., w_D)$ is a vector of weights that reflect the relative importance of each coordinate of the space.

Quadratic distances can also be used to capture correlations between different coordinates of the feature vectors [FEF+94]. The quadratic distance is defined as:

$$d^2(\mathbf{p},\mathbf{q};\mathbf{W}) = \sum_{i=1}^{D}\sum_{j=1}^{D} w_{i,j}(p_i - q_i)(p_j - q_j) \tag{2.5}$$

and leads to arbitrarily-oriented ellipsoidal iso-distant surfaces in feature space [SK97]. Note that this distance is indeed a "rotated" weighted Euclidean norm. The well-known Mahalanobis distance is obtained when each $w_{i,j}$ is a coefficient of the covariance matrix. In Figure 2.2 the geometrical interpretation of above distance functions is shown.

An alternate method of color representation is that of color moments [SO95, SD96]. To overcome the quantization effects of color histograms, a 9-dimensional vector, consisting

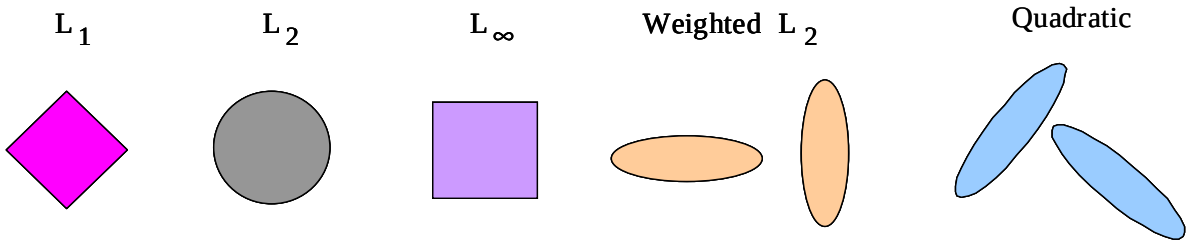


Figure 2.2: Iso-surfaces for different distance functions.

of mean, variance, and skewness of the hue, saturation, and brightness components for all the pixels, is extracted from each image. On these vectors, a weighted Euclidean distance function or a Manhattan distance is then used to compare images. The weights are proportional inverse to the standard deviation of the value along the dimensions. The effectiveness of color moments ha proven to be far better than color histograms [SO95, SD96].

Image Retrieval by Texture Representation

Textures are homogeneous patterns or spatial arrangements of pixels that cannot be sufficiently described by regional intensity or color features [SWS+00]. The simplest way to globally represent texture properties is based on the extraction of information on *coarseness*, *contrast*, and *direction* [FSN+95].

A more powerful method to characterize the image texture follows the same color histogram approach. Image texture is first decomposed into spatial-frequency sub-bands, by way of a wavelet filter bank. Then, a texture channel generator is used to produce a channel for each sub-band. Again, these texture channels can be transformed (by way of a transformation matrix T_t) and quantized (by way of a quantization matrix Q_t) to produce the final histogram representing the image. The representation of texture as an histogram allows us to use, for texture similarity, the same metrics used for color similarity. In particular, in [Smi97] it is shown that L_1 and L_2 metrics perform extremely well in retrieving images having texture similar to that of the query image.

An alternate method to represent texture properties is based on the use of Gabor filters [SD96]. In detail, a Gabor filter measures the presence of patterns in different directions and scales. So, for each scale and direction, the luminance information is transformed with the corresponding Gabor filter and mean and variance are computed. A common Gabor filter approach uses 5 directions and 3 scales determining a feature vector defined in a 30-D dimensional space. On these vectors, a weighted Manhattan distance is used to compare images.

Image Retrieval by Shape Representation

Shape representation techniques fall in two major categories: The feature vector approach, and the shape through transformation approach [Del99]. The choice of a particular representation is driven by application needs, like characteristics of the shapes being analyzed, robustness again noise, and possibility of indexing.

The feature vector approach is widely employed in information retrieval and allows effective indexing. In detail, a shape is represented as a numerical vector using a parametric internal method (where the region enclosed by the object contour is represented), or a parametric external method [RSH96, MM99] (where the external boundary of the object is represented). The Euclidean distance is the most used distance function to compare two shapes. To have an idea of how this approach works, refer to Section 5.2.

On the other hand, shapes can be also compared computing the effort needed to transform one shape into the other. In this case, similarity is computed by way of a transformational distance. The main disadvantage of this approach, however, is that it does not support indexing, due to the fact that the method used to assess similarity does not satisfies metric postulates.

2.2 Current Content-Based Image Retrieval Systems

Many CBIR systems have been designed and developed over the last years [SWS+00]. What can be called the first generation of CBIR systems used *global features* to characterize the images content. For example, QBIC [FSN+95], developed at the IBM Almaden Research Center, extracts from each image a number of features, namely color, texture, and shape descriptors. Color is represented by means of histograms that are compared using a quadratic distance function that also takes into account the similarity between different colors (*cross-talk*). Texture is analyzed globally by extracting information on *coarseness*, *contrast*, and *direction*. The shape feature contains information about the *curvature*, *moment invariants*, *circularity*, and *eccentricity*. The query retrieval system supports the comparison of each of the features separately. Similarity between images is then computed using a weighted Euclidean distance on the overall extracted vector.

Stricker and Orengo [SO95] propose a different approach to color similarity, where the first three moments of the distribution of each color channel are considered. Thus, each image is represented by a 9-dimensional feature vector, and a simple weighted Manhattan distance is used to compare images.

The Photobook system developed at the MIT Media Lab [PPS96] uses a stochastic model (*Wold-decomposition*) to assess the similarity between images based on texture.

Techniques operating in the time-frequency domain, such as the Wavelet Transform [Dau92] (see also Appendix A), have also been proposed to obtain a multi-resolution image representation. As an example, the WBIIS system [WWFW97] uses Daubechies' wavelets [Dau92] to derive a 768-dimensional vector of wavelet coefficients that preserve spatial image information. Although this approach offers a better frequency location with respect to other techniques, it leads to poor results for queries where spatial location and scale of objects is not requested [NRS99].

All above described approaches (as well as many others not covered here) use global features to represent image semantics, thus they are not adequate to support queries looking for images where specific "objects" having particular colors and/or texture (and possibly spatially arranged in a particular way) are present, "partial-match" queries (where only a part of the query image is specified), and shift/scale-invariant queries, where the position and/or the dimension of the seeked objects is not deemed relevant. *Region-based* image retrieval systems aim to overcome such limitations by fragmenting image into a set of "homogeneous" regions, which can then be described by means of *local features* [SC96, CTB$^+$99, NRS99]. Note that the concept of "homogeneity" is by no means easy to define. For instance, if one considers each pixel separately, texture information is lost and only "color homogeneity" can be assessed. For a more complex example, consider an image where a flag with red and blue vertical stripes appears: A human would certainly recognize this as a homogeneous region, even if it contains pixels with rather different colors, since he/she sees a repeated pattern, but this can be difficult for automatic retrieval systems.

VisualSEEk, developed by Smith and Chang [SC96] at the Columbia University, is an example of region-based system that considers information from both the spatial and the frequency domains in order to decompose each image into regions. The similarity between two images is computed by taking into account color, location, size, and relative positioning of regions. Query processing, however, is carried out by using a simple heuristic algorithm: First, for each region of the query image, a range query on color, location, and size is issued with similarity thresholds provided by the user; then, a *candidate set* of images is built, by taking into account only those images that have one region in all the result regions sets; finally, the optimum match is computed on the set of candidate images. It is clear that the use of similarity thresholds has no direct counterparts in a user's mind, and cannot guarantee that the images most similar to the the query image are retrieved.

Another example of region-based system is SIMPLIcity [WLW01]. SIMPLIcity combines the region-based approach to the semantic classification technique; so doing, it is

able to perform an automatic partition of the DB reducing the cardinality of the data set where result images are to be found. In detail, to segment an image into regions, SIMPLIcity partitions the image into blocks of 4×4 pixels and extracts a feature vector of six features from each block. Three features represent the average color components, the remaining three representing texture information. The k-means algorithm is then used to cluster the feature vectors into several classes, with each class corresponding to one region in the segmented image. It has to be noted that SIMPLIcity performs only global search (i.e. uses overall properties of all regions of images) and does not allow the retrieval based on a particular region of the image.

NeTra [MM99] is a region-based system developed at the UCSB that uses color, texture, and shape information to organize and search the DB. It automatically segments each image into regions and uses local information to index images of the DB. The fragmentation of images is performed using a technique based on the *Edge Flow* algorithm [MM99] that is able to discover the neighborhood area between regions using the color property, the texture feature, or both of them.

In the following we concentrate on a major description of the two most widely known region-based image retrieval systems, i.e. WALRUS and Blobworld.

2.2.1 WALRUS

WALRUS (WAveLet-based Retrieval of User-specified Scenes) [NRS99] is a region-based image retrieval system where the similarity measure between a pair of images is defined to be the fraction of area covered by matching regions of the two images.

WALRUS pre-processes an image in two steps: First, it generates a set of *sliding windows* with different sizes and computes a "signature" (local feature vector) for each sliding window, consisting of all the coefficients from the lowest frequency band of the Haar Wavelet Transform (see Appendix A) applied to the pixels in the window. The next step is to cluster together the sliding windows by computing the similarity between their signatures. Each cluster, thus, consists of a set of windows with similar characteristics (i.e. color and texture), which together define a region. Wavelet signatures of the windows in a cluster are then averaged to obtain the region feature vector. To speed-up the retrieval, WALRUS indexes regions' descriptors using an R*-tree [BKSS90].

In order to submit a query to WALRUS, the user has to specify a query image and two similarity thresholds, ϵ and ξ. After extracting regions from the query image, WALRUS uses the index to find all regions in the DB that are similar enough to a query region, that is, regions whose signatures are within ϵ distance from the signature of a query region. Then, similarity between images is assessed by adding up the sizes of matched regions, as

obtained from the index search step, and the result of the query consists of all the images for which the similarity with the query image is not lower than the ξ threshold.

From the query processing point of view, the main limitation of WALRUS is that it requires the specification of two similarity thresholds: The choice of the parameters ϵ and ξ is not very meaningful, since the user has no clear way to determine what a difference between threshold values actually represents. As already argued at the beginning of the Chapter, we believe that range queries are not suitable for effective image retrieval.

2.2.2 Blobworld

Blobworld [CTB+99] is a system that determines coherent image regions that roughly correspond to objects. Blobworld models an image as a set of regions (*blobs*) which are homogeneous with respect to color and texture. Each blob is described by its color distribution and by its mean texture descriptors, obtaining a 220-D feature vector (a 218-bins color histogram and 2 texture descriptors). Querying is then based on the features of some (typically, one or two) regions of interest, rather than on a description of the whole image.

In the image pre-processing phase, Blobword first extracts pixel features, then it groups similar pixels into blob regions, and finally determines the feature vectors of the blobs. In detail, the pixels distribution is modelled in a 8-D space (L*a*b* descriptors for color, *anisotropy*, *orientation*, and *contrast* for texture, and spatial position of the pixel) using a mixture of two to five Gaussians. To fit the mixture of Gaussian models to the pixel data, the Expectation-Maximization (EM) algorithm is used, whereas the number of Gaussians that best suits the real number of groups contained in the image is determined by means of the Minimum Descriptor Length principle [BCGM98]. Once a model is selected, the system performs a spatial grouping of connected pixels belonging to the same cluster.

At query time, the user composes a query by submitting to the system an image of interest and selecting some of the blobs in the image (an "atomic query" is composed by a single blob, whereas a "compound query" is specified by two or more blobs). When dealing with a compound query, which is the most common case, each blob in the query image is associated to its "best" blob in the DB image under consideration (a quadratic, L_2-like, distance function between the feature vectors is used to this purpose). Then, the overall score is computed by using (weighted) fuzzy-logic operators (conjunctions, disjunction, and negation) applied to the scores of matched blobs. Finally, images are ranked according to their overall score and the k best matches are returned.

In order to speed-up query processing, Blobworld can also use an R-tree-like structure to index the color descriptors of the blob feature vectors [TCH00] (no texture information

is taken into account when the index is used). For each blob in the query image, a predetermined number (in the order of the hundreds) of "best matches" is retrieved by using the index. Note that the use of an index can lead to miss the correct best images, since there is no guarantee that such images will be included within those returned by the index itself, as it will be shown in Section 3.3. Among all the images containing regions obtained in the index-retrieval step, the "true", above described, matching algorithm is then used to obtain the result images. However, since best matches for query blobs are computed by ignoring matches for other blobs, it could be the case that a single blob in a DB image is associated to two distinct query blobs (remind the "two tigers" example described at the beginning of the Chapter).

2.3 Interactive Similarity Search

Like observed in the introduction of this Chapter, similarity search is a powerful way to retrieve interesting information from large image DBs and, more in general, from multimedia repositories [Fal96]. However, the very nature of multimedia objects often complicates the user's task of choosing an appropriate query and a suitable distance criterion to retrieve from the DB the objects which best match his/her needs [SK97]. This can be due both to limitation of the query interface and to the objective difficulty, from the user's point of view, to properly understand how the retrieval process works in high-dimensional spaces, which typically are used to represent the relevant *features* of the multimedia objects. For instance, the user of an image retrieval system will hardly be able to predict the effects that the modification of a single parameter of the distance function used to compare the individual objects can have on the result of his/her query.

To obviate this unpleasant situation, several multimedia systems now incorporate some *feedback* mechanisms so as to allow users to provide an evaluation of the *relevance* of the result objects. By properly analyzing such relevance judgments, the system can then generate a new, refined, query, which will likely improve the quality of the result, as experimental evidence confirms [RHOM98, COPM01]. This interactive retrieval process, which can be iterated several times until the user is satisfied with the results, gives rise to a *feedback loop* during which the default parameters used by the query engine are gradually adjusted to fit the user's needs (see for example [ORC+98]).

Although relevance feedback has been recognized as a method of improving interactive retrieval effectiveness (both in text retrieval and image retrieval systems) [RHOM98], its applicability suffers two major problems:

1. Depending on the query, numerous iterations might occur before an acceptable result

is found, thus convergence can be slow.

2. Once the feedback loop of a query is terminated, no information about this particular query is retained for re-use in further processing. Rather, for further queries, the feedback process is started anew with default values. Even in the case that a query object has already been used in an earlier feedback loop, all iterations have to be repeated.

Note that both problems concern the *efficiency* of the feedback process, whereas the *effectiveness* of retrieval will depend on the specific feedback mechanisms used by the system, on the similarity model, and on the features used to represent the objects.

In the following we briefly introduce the basic concepts of relevance feedback techniques within the context of text-based retrieval, describing, more in detail, feedback methods in image retrieval systems.

2.3.1 Relevance Feedback Techniques

We frame our discussion on feedback methods within the context of *vector space* similarity models, where an object is represented through a D-dimensional vector (i.e. a point in $\Re^D$ vector space), $\mathbf{p} = (\mathbf{p}[1], \ldots, \mathbf{p}[D])$, and the similarity of two points $\mathbf{p}$ and $\mathbf{q}$ is measured by means of some *distance function* on such space (see Section 2.1.1).

Relevance Feedback Technique in Text Retrieval

Relevance feedback was initially introduced in text retrieval to increment the number of relevant documents returned by a query [Sal89]. In relevance feedback, the search through a document collection starts with a user query. Upon receiving result documents, the user gives his/her judgments by choosing which documents are relevant and which are not to the query. Both the positive and negative relevance feedback are used to move the query point towards the relevant points and away from the not relevant objects. In detail, the above algorithm is implemented by way of Rocchio's formula [Sal89], defined as:

$$\mathbf{q}_{new} = \mathbf{q}_{old} + \beta \sum_{i=1}^{n_g} \frac{\mathbf{p}_i}{n_g} - \gamma \sum_{i=1}^{n_b} \frac{\mathbf{o}_i}{n_b} \tag{2.6}$$

where $\mathbf{q}_{new}$ represents the new query point, $\mathbf{q}_{old}$ is the initial query vector, $\mathbf{p}_i$ represents positive objects, and $\mathbf{o}_i$ represents negative objects. Finally, n_g and n_b are the number of "good" and "bad" results, respectively, and β and γ are two constant factors able to determine the grade of attraction towards relevant documents and that of repulsion away from negative objects (experiments demonstrated that, to limit the effect of negative

feedback, setting $\beta = 0.75$ and $\gamma = 0.25$ is the best choice [Sal89]). Thus, the new query point is obtained summing, to the old query vector, the positive correlation vector (i.e. the normalized sum of positive documents $\mathbf{p}_i$) and by subtracting the negative correlation vector (i.e. the sum of negative objects $\mathbf{o}_i$). Note that the new vector does not necessarily correspond to a document from the collection.

Relevance Feedback Technique in Image Retrieval

Like observed in Section 2.3, in more recent years relevance feedback techniques have been associated with CBIR systems to overcome problems like the gap between high-level concepts and low-level features, and the human perception subjectivity of visual content. As a result, a remarkable improvement of the effectiveness of similarity retrieval is obtained [RHOM98].

The typical interaction with a CBIR system, or, more in general, with a multimedia retrieval system that implements relevance feedback mechanisms can be summarized as follows [Sal89]:

Query formulation. The user submits an initial query $Q = (\mathbf{q}, k)$, where $\mathbf{q}$ is called the *query point* and k is a limit on the number of results to be returned by the system.

Query processing. The query point $\mathbf{q}$ is compared with the DB objects by using a (default) distance function d. Then, the k objects which are closest to $\mathbf{q}$ according to d, $Result(Q, d) = \{\mathbf{p}_1, \ldots, \mathbf{p}_k\}$, are returned to the user.

Feedback loop. The user evaluates the relevance of the objects in $Result(Q, d)$ by assigning to each of them a *relevance score*, $Score(\mathbf{p}_j)$. On the basis of such scores a new query, $Q' = (\mathbf{q}', k)$, and a new distance function, d', are computed and used to determine the second round of results.

Termination. After a certain number of iterations, the loop ends, the final result being $Result(Q_{opt}, d_{opt})$, where $Q_{opt} = (\mathbf{q_{opt}}, k)$ is the "optimal" query the user had in mind, and d_{opt} the "optimal" distance function used to retrieve relevant objects for Q_{opt}.

Each interactive retrieval system provides a specific implementation for each of the above steps. For instance, the choice of the initial query point depends on the system interface and, also considering the very nature of multimedia objects, can include a *query-by-sketch* facility, the choice from a random sample of objects, the upload of the query point from a user's file, etc. Many options are also available for implementing the query processing

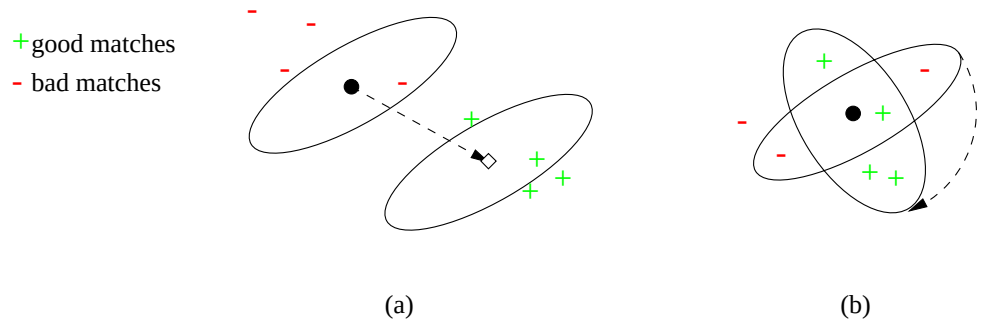


Figure 2.3: The "query point movement" (a) and the "re-weighting" (b) feedback strategies

step, which typically exploits index structures for high-dimensional data, such as the X-tree [BKK96] and the M-tree [CPZ97].

More relevant to the present discussion are the issues concerning the feedback loop. The use of *binary* relevance scores is the simplest one, even from the user's point of view. In this case the user can mark a result object either as "good" or "bad", and implicitly assigns a neutral ("no-opinion") score to non-marked objects. Graded, and even continuous, score levels have also been used to allow for a finer tuning of user's preferences [RHOM98].

The two basic strategies for implementing the feedback loop concern the computation of a new query point (*query point movement*), the change of the distance function, which can be accomplished by modifying the weights (importance) of the feature components (*re-weighting*). A further feedback technique is represented by the expansion of the query point (*query expansion*).

Query point movement. The idea of this strategy, whose implementation dates back to Rocchio's formula [Sal89] reported in Equation 2.6, is to try to move the query point towards the "good" matches (as evaluated by the user), as well as to move it far away from the "bad" result points (see Figure 2.3 (a)). More recently, query point movement has been applied by several image retrieval systems, such as MARS [RHOM98]. Ishikawa et al. [ISF98] have proved that, when using *positive* feedback (scores) and the Mahalanobis distance, the "optimal" query point (based on the available set of results) is a weighted average of the good results, i.e.:

$$\mathbf{q}' = \frac{\sum_j Score(\mathbf{p}_j) \times \mathbf{p}_j}{\sum_j Score(\mathbf{p}_j)} \tag{2.7}$$

Re-weighting. The idea of re-weighting stems from the observation that user feedback can highlight that some feature components are more important than others in determining whether a result point is "good" or not, thus such components should be given

a higher relevance. For simplicity of exposition, let us consider a retrieval model based on weighted Euclidean (see Equation 2.4) and also refer to Figure 2.3 (b). In order to assess the relative importance of the i-th feature vector component, the distribution of the "good" $p_{j,i}$ values, i.e. the values of the good matches along the i-th coordinate, is analyzed. In an earlier version of the MARS system [RHOM98], it was proposed to assign to the i-th coordinate a weight w_i computed as the inverse of the standard deviation of the $p_{j,i}$ values, that is:

$$w_i \propto \frac{1}{\sigma_i} \tag{2.8}$$

Later on, it was proved in [ISF98] that the "optimal" choice of weights is to have:

$$w_i \propto \frac{1}{\sigma_i^2} \tag{2.9}$$

Similar results have been proved for quadratic distance functions [ISF98], as well as for the case where the number of good matches is less than the dimensionality of the feature space [RH00].

In a recent paper [RH00] Rui and Huang have extended the re-weighting strategy to a "hierarchical model" of similarity, where above strategy is first individually applied to each feature separately, and then each feature (rather than each feature component) is assigned a weight which takes into account the overall distance that good matches have from the query point by considering only that feature. Note that for F features this amounts to define the distance between objects $\mathbf{p}$ and $\mathbf{q}$ as a weighted sum of the F feature distances, each of which the authors assume to have a quadratic form [RH00].

In Appendix B, we describe a sample application of feedback-based CBIR system showing the experimental results obtained using different combinations of the two described techniques.

Query expansion. The idea of this strategy, first proposed for the MARS system [PC99], is to employ the relevant objects as new reference images (called *representatives*) for the next search cycle [WFSP00]. This produces a *multipoint* query defined in each feature space. If the number of positive points is too high, a better solution is to select a small number of good representatives to define the multipoint query. This is possible using a clustering algorithm that partitions relevant objects in each feature space and uses the cluster centroids as representatives for such feature space.

Chapter 3

Windsurf

In Section 2.1 we point out the main limitations suffering by current CBIR systems, that can be here summarized as follow:

1. CBIR systems that rely on image global features extraction cannot support queries that refer to *local* properties of the images (e.g. "Find all the images containing a small red region under a big blue region").

2. Existing CBIR systems, that fragment each image into a set of "homogeneous regions" extracting local image properties, however, use naïve heuristic matching algorithms which are not guaranteed to return the correct results.

To overcome such problems, we have presented the WINDSURF system [ABP99, BCP00b], a new region-based image retrieval system. WINDSURF applies the Wavelet Transform to extract color and texture features from an image, and then partitions the image into a set of "homogeneous" regions, each described by a set of *local features*. Similarity between images is then assessed by first computing similarity scores between regions and then combining the results at the image level. From the query processing point of view, WINDSURF provides a *correct* definition of *image similarity* under the region-based retrieval model, and introduce a novel index-based algorithm that can make use of any distance-based access method to retrieve the k best-matching images for a given query. It turns out that this is the *first* correct algorithm for region-based image similarity queries [BCP00a, BP00].

3.1 Feature Extraction

WINDSURF (Wavelet-based INDexing of imageS Using Region Fragmentation) is a region-based system that uses the Discrete Wavelet Transform (DWT) [Dau92] and a *k-means*

clustering algorithm to segment an image into regions. Each region is represented by means of a set of features and the similarity between regions is measured using a specific metric function on such features. Image pre-processing consists of the three steps shown in Figure 3.1, namely:

DWT The image is analyzed in the time-frequency domain using a 2-D DWT.

Clustering The image is fragmented into a set of regions using a k-means clustering algorithm that groups together similar wavelet coefficients (*clustering features*).

Feature Indexing Regions obtained from the clustering phase are represented by means of a set of *similarity features*.

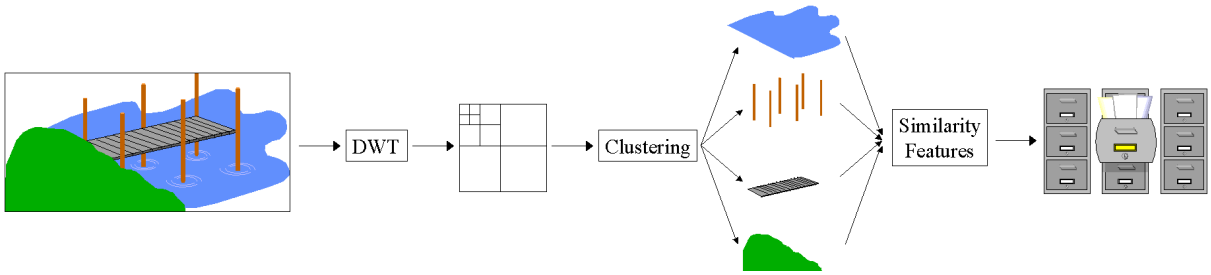


Figure 3.1: Steps of the WINDSURF feature extraction.

3.1.1 DWT

WINDSURF views each image as a 2-D signal to be analyzed by means of a 2-D DWT in the time-frequency domain. More in detail, we use Haar wavelets from the WAILI software library [UVJ+97] and represent images in the HSV color space, because in this space each color component is perceptually independent and uniform [Smi97].

The j-th wavelet coefficient of sub-band B ($B \in \mathcal{B} = \{LL, LH, HL, HH\}$, where L stands for "low" and H for "high") and DWT level l is a 3-D vector, i.e.:

$$w_j^{l;B} = (w_{0_j}^{l;B}, w_{1_j}^{l;B}, w_{2_j}^{l;B}) \tag{3.1}$$

where each component refers to a color channel c ($c \in \{0, 1, 2\}$). The *energy* of $w_j^{l;B}$ on the c and d channels is then defined as:

$$e_{cd_j}^{l;B} = w_{c_j}^{l;B} \cdot w_{d_j}^{l;B} \tag{3.2}$$

When $c = d$, $e^{l;B}_{cc_j}$ is called the *channel energy* of channel c, whereas, when $c \neq d$, $e^{l;B}_{cd_j}$ is termed the *cross-correlation energy* between channels c and d. The energy vector

$$e^{l;B}_j = \left(e^{l;B}_{00_j}, e^{l;B}_{01_j}, e^{l;B}_{02_j}, e^{l;B}_{11_j}, e^{l;B}_{12_j}, e^{l;B}_{22_j}\right) \tag{3.3}$$

captures both color and texture information through channel and cross-correlation energies, respectively. This is similar to the approach described in [Smi97] and is known to be one of the more robust methods for the representation of texture features [CK93, VSLV99].

3.1.2 Clustering

The aim of the clustering phase is to fragment an image into a set of regions by grouping together image pixels that are similar in color and texture features. To this end, we apply a clustering algorithm to the wavelet coefficients (*clustering features*) obtained through the DWT step. In particular, we apply a *k-means* algorithm with a "validity function" which is a variant of the one proposed for the *fuzzy* k-means algorithm [XB91] (see below).

Given a set $X = \{x_1, \ldots, x_N\}$ of N points (wavelet coefficients, in our case) to be clustered, the k-means algorithm starts with a set of k randomly-chosen *centroids*, $\{\mu_1, \ldots, \mu_k\}$, and then assigns each point x_j to its closest centroid μ_i. After this, the algorithm iterates by recomputing centroids and reassigning the points, until either a stable state or a limit to the number of iterations are reached. It can be proved that k-means algorithm leads to minimize the function

$$J = \sum_{i=1}^{k} \sum_{x_j \in C_i} \delta(x_j, \mu_i)^2 \tag{3.4}$$

where $\delta(x_j, \mu_i)$ is the distance between x_j and its closest centroid μ_i. Obviously, both the final value of J and the result of the algorithm depend on the value of k and on the choice of the distance function $\delta()$. As for $\delta()$ we use the *Mahalanobis distance* applied to the 3-D wavelet coefficients of the LL sub-band of the 3-rd DWT level, that is, $x_j \equiv w^{3;LL}_j$. This choice is due to the results of extensive experimental evaluation, which demonstrated that best, most stable, clusters are obtained by taking into account only low frequency descriptors. The Mahalanobis distance between points $w^{3;LL}_i$ and $w^{3;LL}_j$ is given by:

$$\delta(w^{3;LL}_i, w^{3;LL}_j)^2 = (w^{3;LL}_i - w^{3;LL}_j)^T \times \left(\mathcal{C}^{3;LL}\right)^{-1} \times (w^{3;LL}_i - w^{3;LL}_j) \tag{3.5}$$

where $\mathcal{C}^{3;LL} = \left\{cov^{3;LL}_{c,d}\right\}$ is the covariance matrix of the points, that is:

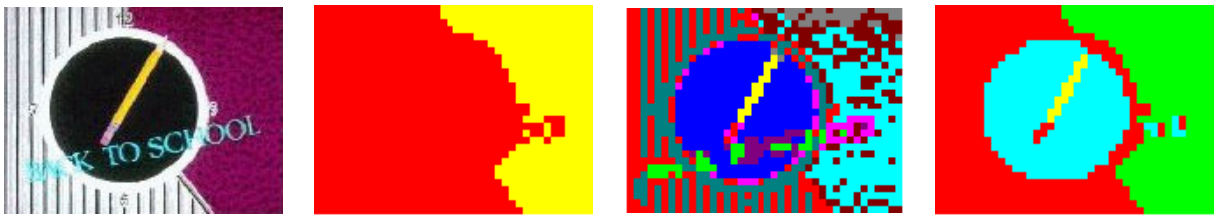
$$cov^{3;LL}_{c,d} = \left(\frac{1}{N}\sum_{j=1}^{N} w^{3;LL}_{c_j} w^{3;LL}_{d_j} - \frac{1}{N^2}\sum_{j=1}^{N} w^{3;LL}_{c_j} \cdot \sum_{j=1}^{N} w^{3;LL}_{d_j}\right) \tag{3.6}$$

By using the Mahalanobis distance, two desirable effects are obtained: First, vectors are automatically normalized (depending on the diagonal elements of the covariance matrix); second, the distance function also considers cross-correlation energies, thus texture characteristics, due to the off-diagonal elements of $\mathcal{C}^{3;LL}$.

Since different values of the k parameter can lead to different results, we iterate the k-means algorithm with $k \in [2, 10]$, and then select the "optimal" k value as the one minimizing the *validity function* V, defined as:

$$V = \frac{J'}{N \cdot \delta_{\min}^2} + \sum_{i=1}^{k'} \frac{1}{1 + |C_i|} \tag{3.7}$$

where k' represents the number of "good" clusters, i.e. clusters that are not too small, J' is as in Equation 3.4, but now it only takes into account good clusters, $\delta_{\min} = \min_{i \neq j}\{\delta(\mu_i, \mu_j)\}$ is the minimum distance between cluster centroids, and $|C_i|$ is the cardinality of cluster C_i. The first term of Equation 3.7 represents the goal function J' divided by $\delta_{\min}^2$, i.e. clusters well separated provide better solutions, whereas the second term represents a penalty factor for small clusters. As an example, Figure 3.2 shows the results of the k-means algorithm applied to the image on the left, when $k = 2$, $k = 10$, and $k = 4$, respectively, the latter being the optimal solution according to the V validity function.



(a) (b) (c) (d)

Figure 3.2: (a) The input image. Clusters obtained for: (b) $k = 2$; (c) $k = 10$; (d) $k = 4$ (optimal solution). In the clustered images, points having the same color belong to the same cluster.

As a final issue, consider the particular case where the optimal solution is to have $k = 1$, which corresponds to images consisting of a uniform pattern, for which no segmentation is appropriate. Since the validity function V is not defined for $k = 1$, we resort to an analysis of the covariance matrix $\mathcal{C}^{3;LL}$. This can be geometrically represented as a 3-D ellipsoid, where each axis has a direction given by a matrix eigenvector and a length determined by its corresponding eigenvalue. Intuitively, when all the eigenvalues are small, then the wavelet coefficients have a small variance, and this can be used as an evidence that the image represents a homogeneous pattern. Since the trace of a matrix equals the

sum of its eigenvalues, by just looking at the trace $\mathcal{T}_{\mathcal{C}^{3;LL}}$ of $\mathcal{C}^{3;LL}$ we can therefore deal with images for which the clustering algorithm should not be applied at all. In practice, if $\mathcal{T}_{\mathcal{C}^{3;LL}}$ is smaller than a given threshold value β, then the image is considered as a homogeneous pattern. In our tests we found that $\beta = 1000$ is an appropriate threshold value. As an example, consider the image in Figure 3.3 (a), whose covariance matrix is given in Figure 3.3 (b). It is clear that the image is a homogeneous pattern, and our perception is confirmed by the analysis of the trace of the covariance matrix whose value is $\mathcal{T}_{\mathcal{C}^{3;LL}} = 5.17 + 357.91 + 237.00 = 600.08 < 1000$.

(a)

$$\mathcal{C}^{3;LL} = \begin{pmatrix} \boxed{5.17} & 16.10 & -3.90 \\ 16.10 & \boxed{357.91} & -152.56 \\ -3.90 & -152.56 & \boxed{237.00} \end{pmatrix}$$

(b)

Figure 3.3: A homogeneous image (a) and its covariance matrix $\mathcal{C}^{3;LL}$ (b).

3.1.3 Feature Indexing

Regions obtained from the clustering phase are described using a set of *similarity features*, which are then used for image retrieval. In detail, when comparing regions, we consider information on size and color-texture as provided by all the frequency sub-bands of the 3-rd DWT level. To this end, the similarity features for a region $R_{s,i}$ of image I_s are defined as a 37-D vector, whose components are:

Size The number of pixels in the region, $size(R_{s,i})$.

Centroid The 12-D centroid of $R_{s,i}$, $\mu_{R_{s,i}} = (\mu^{LL}_{R_{s,i}}, \mu^{LH}_{R_{s,i}}, \mu^{HL}_{R_{s,i}}, \mu^{HH}_{R_{s,i}})$, where each $\mu^{B}_{R_{s,i}}$ is a 3-D point representing the average value for each of the 3 color channels in the B sub-band.

Covariance matrices This is a 24-D vector, denoted $\mathcal{C}^{3}_{R_{s,i}}$, containing the elements of the 4 3×3 covariance matrices, $\mathcal{C}^{3;B}_{R_{s,i}}$, of the points in $R_{s,i}$. Since the covariance matrices are symmetric, only 6 values for each matrix need to be stored.

3.2 Similarity Model

The image similarity model of WINDSURF defines the similarity between two images as a function of the similarities among "matched" regions, as Figure 3.4 suggests.

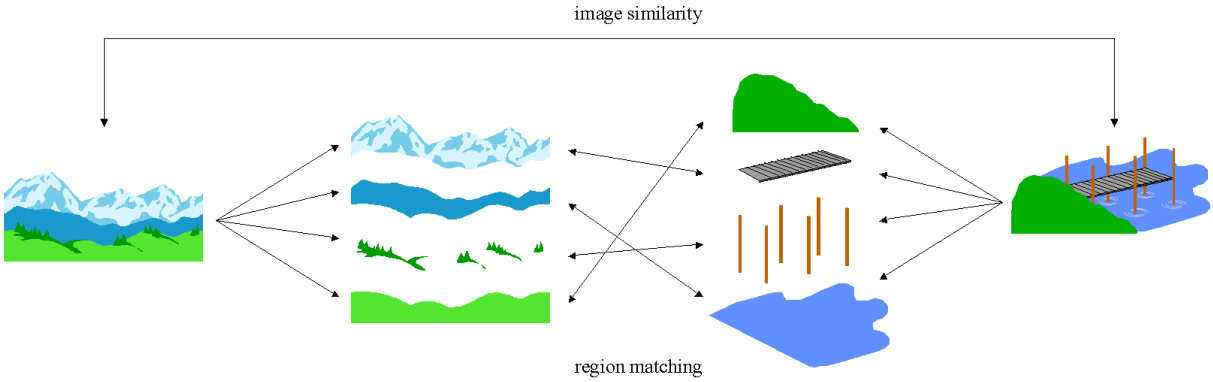


Figure 3.4: Similarity between images is assessed by taking into account similarity between matched regions.

To completely characterize the image similarity model, we have therefore to first specify how similarities among regions are determined, and then how such region-based similarities are combined together to produce the overall similarity score between images.

3.2.1 Region Similarity

The similarity between two regions, $R_{q,i}$ (represented by the feature vector $[\mu_{R_{q,i}}, \mathcal{C}^3_{R_{q,i}}, size(R_{q,i})]$) of a query image I_q and $R_{s,j}$ (with feature vector $[\mu_{R_{s,j}}, \mathcal{C}^3_{R_{s,j}}, size(R_{s,j})]$) of a DB image I_s, is computed by WINDSURF as:

$$r_{sim}(R_{q,i}, R_{s,j}) = h(d(R_{q,i}, R_{s,j})) \tag{3.8}$$

where $d()$ is a distance function, and $h()$ is a so-called *correspondence function* [Fag96, CPZ98] that maps distance values to similarity scores. The function $h : \Re_0^+ \rightarrow [0, 1]$ has to satisfy the two following properties:

$$\begin{aligned} h(0) &= 1 \\ d_1 \leq d_2 &\Rightarrow h(d_1) \geq h(d_2) \qquad \forall d_1, d_2 \in \Re_0^+ \end{aligned}$$

since equal regions should have a similarity score of 1 and the function should map low distance values into higher scores and vice versa.

In all our experiments we use $h(d) = e^{-d/\sigma_d}$, where σ_d is the standard deviation of the distances computed over a sample of DB regions. The distance $d(R_{q,i}, R_{s,j})$ between regions $R_{q,i}$ and $R_{s,j}$ is a weighted sum, taken over the four frequency sub-bands, of the distances between color-texture descriptors, plus an additional term that takes into

account the difference between the relative size of the two regions:

$$d(R_{q,i}, R_{s,j})^2 = \sum_{B \in \mathcal{B}} \gamma_B \cdot d_B(R_{q,i}, R_{s,j})^2 + \\ + \left(\frac{2}{\frac{size(R_{q,i})}{size(I_q)} + \frac{size(R_{s,j})}{size(I_s)}} \right) \cdot \left(\frac{size(R_{q,i})}{size(I_q)} - \frac{size(R_{s,j})}{size(I_s)} \right)^2 \quad (3.9)$$

In our experiments we equally weigh the frequency coefficients, i.e. $\gamma_B = 1$, $\forall B \in \mathcal{B}$. The second term in Equation 3.9 takes into account the difference in size between the regions by multiplying it by a coefficient that favors matches between large regions.

The distance $d_B(R_{q,i}, R_{s,j})$ between two regions on the frequency sub-band B is computed using the *Bhattacharyya* metric [Bas89]:

$$d_B(R_{q,i}, R_{s,j})^2 = \frac{1}{2} \ln \left(\frac{\left| \frac{\mathcal{C}^{3;B}_{R_{q,i}} + \mathcal{C}^{3;B}_{R_{s,j}}}{2} \right|}{\left| \mathcal{C}^{3;B}_{R_{q,i}} \right|^{\frac{1}{2}} \cdot \left| \mathcal{C}^{3;B}_{R_{s,j}} \right|^{\frac{1}{2}}} \right) + \\ + \frac{1}{8} \left[\left(\mu^B_{R_{q,i}} - \mu^B_{R_{s,j}} \right)^T \times \left(\frac{\mathcal{C}^{3;B}_{R_{q,i}} + \mathcal{C}^{3;B}_{R_{s,j}}}{2} \right)^{-1} \times \left(\mu^B_{R_{q,i}} - \mu^B_{R_{s,j}} \right) \right] \quad (3.10)$$

where $|A|$ is the determinant of matrix A. Equation 3.10 is composed of two terms. The second term is the Mahalanobis distance between regions centroids, where an average covariance matrix is used. The first term is used to compare the covariance matrices of the two regions. Note that if the two regions have the same centroid, the second term of Equation 3.10 vanishes, thus the first term measures how similar the two 3-D ellipsoids are (this is the case of regions with similar colors but different texture, see Figure 3.5).

When computing Equation 3.10, we also correctly take into account those particular cases arising from singular covariance matrices. Such situations originate, for instance, from uniform images (e.g. a totally black image), where the covariance matrix is null.

3.2.2 Combining Region-Based Similarities

The basic idea of any region-based image retrieval system is that the similarity between two images depends on the similarities among component regions. What makes WINDSURF different from other systems, such as those described in Section 2.2, is that its similarity model can correctly define the "best matches" for a query image by taking into account *all* the information available from regions' similarities. For this we first need to define what a *matching* is.

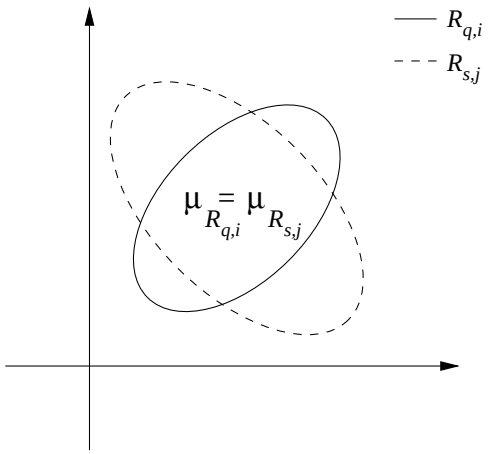


Figure 3.5: Two ellipsoids with different shape and equal centroids.

Definition 3.1 (Matching)
Given a query image I_q, divided into a set of regions $\mathcal{R}_q = \{R_{q,1}, \ldots, R_{q,n_q}\}$, and a DB image I_s, divided into a set of regions $\mathcal{R}_s = \{R_{s,1}, \ldots, R_{s,m_s}\}$, a *matching* between I_q and I_s is an injective function $\Gamma_s : \mathcal{R}_q \rightarrow \mathcal{R}_s \cup \{\perp\}$ that assigns to each region $R_{q,i}$ of the query image either a region of I_s or the "null match" $\perp$.

Note that any matching satisfies, by definition, the two following constraints:

1. A region of I_q cannot match with two different regions of I_s (Figure 3.6 (a)).
2. Two different regions of I_q cannot match with the same region of I_s (Figure 3.6 (b)).

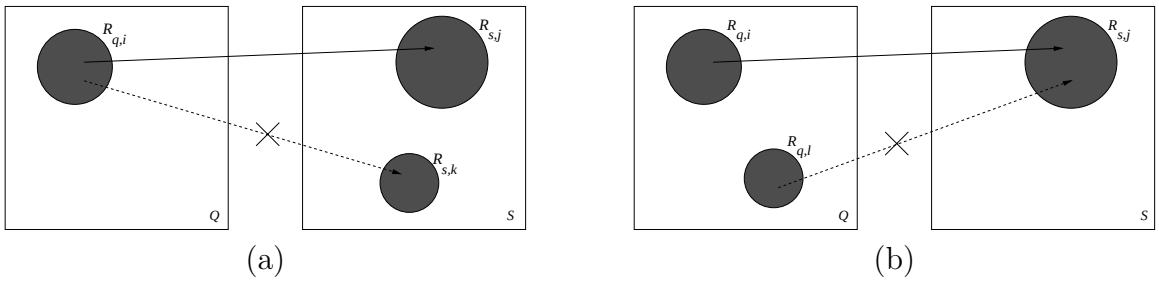


Figure 3.6: A region of I_q cannot match with two regions of I_s (a) and two regions of I_q cannot match with the same region of I_s (b).

Given a matching Γ_s, the corresponding similarity between I_q and I_s is computed by means of the IM_{sim} *combining function*:

$$I_{sim}(I_q, I_s) = IM_{sim}(r_{sim}(R_{q,1}, \Gamma_s(R_{q,1})), \ldots, r_{sim}(R_{q,n_q}, \Gamma_s(R_{q,n_q}))) \tag{3.11}$$

where it is assumed that $r_{sim}(R_{q,i}, \perp) = 0$, in case a match for $R_{q,i}$ is not defined.

The only requirement we put on the IM_{sim} function is that it has to be a monotonic non-decreasing function, i.e.:

$$s_i \leq s'_i \implies IM_{sim}(s_1, \ldots, s_i, \ldots, s_n) \leq IM_{sim}(s_1, \ldots, s'_i, \ldots, s_n) \tag{3.12}$$

This is intuitive, since better matches between regions are expected to increase the overall similarity score between corresponding images. Moreover, for the sake of simplicity, in the following we will assume that IM_{sim} is a symmetric function of its arguments.

Clearly, any different matching leads, according to Equation 3.11, to a different value for the similarity of I_q and I_s. It is natural to define the "true" image similarity by only considering *optimal* matchings.

Definition 3.2 (Optimal matching)
A matching that maximizes Equation 3.11 is called an *optimal* matching between I_q and I_s, and will be denoted as Γ_s^{opt}.

Definition 3.3 (Image similarity)
The similarity between I_q and I_s is defined as the value of IM_{sim} computed from an optimal matching, i.e.:

$$\begin{aligned} I_{sim}(I_q, I_s) &= \max_{\Gamma_s}\{IM_{sim}(r_{sim}(R_{q,1}, \Gamma_s(R_{q,1})), \ldots, r_{sim}(R_{q,n_q}, \Gamma_s(R_{q,n_q})))\} = \\ &= IM_{sim}(r_{sim}(R_{q,1}, \Gamma_s^{opt}(R_{q,1})), \ldots, r_{sim}(R_{q,n_q}, \Gamma_s^{opt}(R_{q,n_q}))) \end{aligned} \tag{3.13}$$

The following is a simple property of optimal matchings, which holds for any combining function IM_{sim}.

Property 3.1 (Maximal and complete matchings)
Let n_q be the number of regions of I_q and m_s the number of regions of I_s. If $r_{sim}(R_{q,i}, R_{s,j}) > 0$ holds for any pair of regions of I_q and I_s, then a matching Γ_s can be optimal only if it is *maximal*, that is, only if $\Gamma_s(R_{q,i})$ is undefined for exactly $\max\{n_q - m_s, 0\}$ regions of I_q. When $n_q \leq m_s$ a maximal matching is also said a *complete* matching, since for all query regions $R_{q,i}$ it is $\Gamma_s(R_{q,i}) \in \mathcal{R}_s$.

3.2.3 Determining the Optimal Matching

Determining the optimal matching for images I_q and I_s can be formulated as a *generalized assignment problem*. For this, let $s_{ij} = r_{sim}(R_{q,i}, R_{s,j})$ be the similarity score between

region $R_{q,i}$ of I_q and region $R_{s,j}$ of I_s, and denote with $\mathcal{H}$ the index set of pairs of matched regions, that is:

$$\mathcal{H} = \{(i,j)|\Gamma_s(R_{q,i}) = R_{s,j}\}$$

Of course, it is $|\mathcal{H}| \leq \min\{n_q, m_s\}$. Then, the goal is to maximize the function

$$IM_{sim}(s_{i_1 j_1}, \ldots, s_{i_{|\mathcal{H}|} j_{|\mathcal{H}|}})$$

with $(i_h j_h), (i_l j_l) \in \mathcal{H}, (i_h j_h) \neq (i_l j_l)$. To this end, we introduce the variables x_{ij}, where $x_{ij} = 1$ if $\Gamma_s(R_{q,i}) = R_{s,j}$ and $x_{ij} = 0$ otherwise. Then, the generalized assignment problem is formulated as follows:

$$I_{sim}(I_q, I_s) = \max\{IM_{sim}(s_{i_1 j_1}, \ldots, s_{i_{|\mathcal{H}|} j_{|\mathcal{H}|}})\} \quad (i_h j_h), (i_l j_l) \in \mathcal{H}, (i_h j_h) \neq (i_l j_l) \tag{3.14}$$

$$\mathcal{H} = \{(i,j)|x_{ij} = 1\} \tag{3.15}$$

$$\sum_{j=1}^{m_s} x_{ij} \leq 1 \ (i = 1, \ldots, n_q) \tag{3.16}$$

$$\sum_{i=1}^{n_q} x_{ij} \leq 1 \ (j = 1, \ldots, m_s) \tag{3.17}$$

$$x_{ij} \in \{0,1\} \ (i = 1, \ldots, n_q; \ j = 1, \ldots, m_s) \tag{3.18}$$

Equation 3.14 means that to determine the overall score $I_{sim}(I_q, I_s)$ we have to consider only the matches in $\mathcal{H}$ (Equation 3.15). Equation 3.16 (respectively Equation 3.17) expresses the constraint that at most one region $R_{s,j}$ of I_s (resp. $R_{q,i}$ of I_q) can be assigned to a region $R_{q,i}$ of I_q (resp. $R_{s,j}$ of I_s).

In order to devise an algorithm to solve the generalized assignment problem, we need to consider specific choices for the IM_{sim} combining function. At present, in WINDSURF we consider the average similarity between pairs of matched regions:

$$I_{sim}(I_q, I_s) = \frac{1}{n_q}\sum_{i=1}^{n_q} r_{sim}(R_{q,i}, \Gamma_s^{opt}(R_{q,i})) = \frac{1}{n_q}\sum_{i=1}^{n_q} h(d(R_{q,i}, \Gamma_s^{opt}(R_{q,i}))) \tag{3.19}$$

This leads to rewrite Equation 3.14 as follows:

$$I_{sim}(I_q, I_s) = \frac{1}{n_q}\max\left\{\sum_{i=1}^{n_q}\sum_{j=1}^{m_s} s_{ij} \cdot x_{ij}\right\} \tag{3.20}$$

The generalized assignment problem, in this case, takes the form of the well known Assignment Problem (AP), a widely studied topic in combinatorial optimization, which can be solved using the Hungarian Algorithm [Kuh55].

In case of sequential evaluation, the ERASE (Exact Region Assignment SEquential) algorithm, shown in Figure 3.7, can be used to determine the k nearest neighbors of the image query I_q within the $\mathcal{C}$ data set. Note that HUNG invokes the Hungarian Algorithm on the $\{s_{ij}\}$ matrix of regions' similarity scores.

ERASE(I_q: query image, k: integer, $\mathcal{C}$: data set)
{ $\forall$ image I_s in the data set $\mathcal{C}$
  { $\forall$ region $R_{s,j}$ of I_s
      $\forall$ region $R_{q,i}$ of I_q compute $s_{ij} = s(R_{q,i}, R_{s,j})$;
    invoke HUNG($\{s_{ij}\}$) obtaining, as the result, the value $I_{sim}(I_q, I_s)$; }
  return the k images having the highest overall similarity scores $I_{sim}(I_q, I_s)$; }

Figure 3.7: The Exact Region Assignment SEquential algorithm.

Resolution of k nearest neighbors queries by means of the ERASE algorithm requires the computation of similarity scores between regions in the query image and *all* the regions contained in the DB images. Algorithm complexity is, hence, linear in the DB size.

To evaluate the goodness of the ERASE solution, we also introduce a simple heuristic method of image matching, called WINDSURF^{app}. WINDSURF^{app} first determines for each query region $R_{q,i}$ the most similar region $\Gamma^*_s(R_{q,i})$ in $\mathcal{R}_s$. Then, in case $\Gamma^*_s(R_{q,i}) = \Gamma^*_s(R_{q,i'})$ holds for two distinct query regions $R_{q,i}$ and $R_{q,i'}$, WINDSURF^{app} only keeps the best of the two assignments and discards the other one (i.e. the corresponding score is set to 0 and the query region remains unmatched). Comparison between the proposed strategies is postponed until Section 3.4.

3.3 Query Processing and Indexing

In this Section we describe an index-based algorithm aiming to speed-up the evaluation of k nearest neighbors queries. This is carried out by reducing the number of *candidate images*, i.e. images for which the overall image similarity needs to be computed.

Since similarity between images is computed by combining distances between regions' features, we use a distance-based access method (DBAM), like the R*-tree [BKSS90] or the M-tree [CPZ97], to index regions extracted from the DB images.[1] Such index structures are able to efficiently answer both range and k nearest neighbors queries, as well as to perform a *sorted access* to the data, i.e. they can output regions one by one in increasing order of distance with respect to a query region [HS99].

[1] In WINDSURF we use the M-tree index [CPZ97], but other choices are possible.

In order to deal with "compound" queries, where multiple query regions are specified, a query processing algorithm based on multiple sorted access index scans is needed. To retrieve the best matches for the query regions, we run a sorted access to the indexed regions for each region in the query image. Clearly, the problem is to devise a suitable condition to stop such *sorted access phase* so that we are guaranteed that the k best images can be correctly determined without looking at the whole data set. More precisely, the stop condition has to guarantee that the k nearest neighbor images of the query image I_q are among the so-called *candidate images*, i.e. those images for which at least one region has been retrieved during the sorted access phase (Figure 3.8).

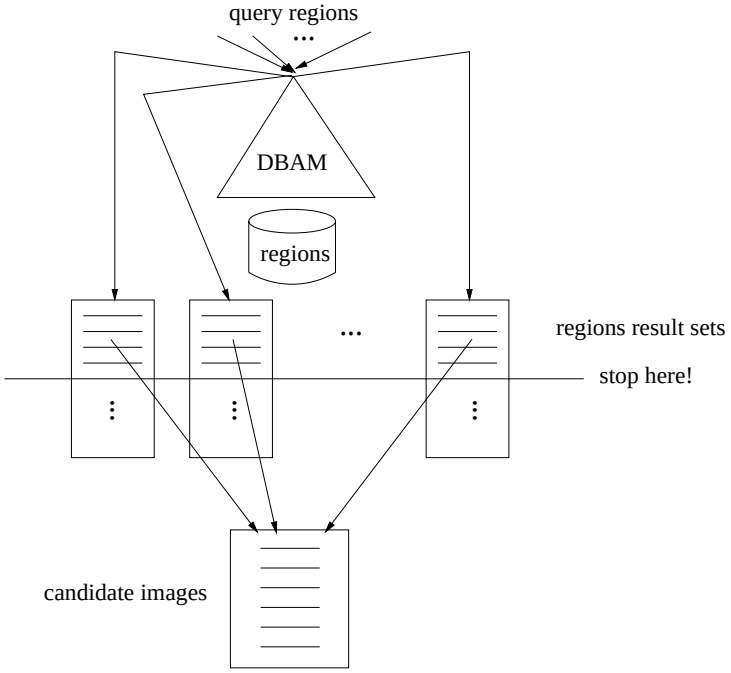


Figure 3.8: Producing the candidate set of images from the sorted access phase.

A first naïve approach to solve compound queries with DBAMs goes as follows. For each region $R_{q,i}$ of the query image I_q, we execute a k nearest neighbors query, that is, we determine the k regions in the data set most similar to $R_{q,i}$. Then, we compute an optimal matching for all the images for which at least one region has been returned by the previous step.[2] This algorithm guarantees that the number of candidate images is not higher than $n_q \cdot k$. Such a solution is indeed quite efficient, but it is *not* correct. As an example, consider the case where $n_q = 2$, $k = 1$, and assume that the regions' similarity scores obtained by the two sorted access scans are as in Table 3.1.

[2]Note that this is, indeed, the query processing approach used by Blobworld.

$R_{q,1}$			$R_{q,2}$		
region	image	similarity	region	image	similarity
$R_{1,1}$	I_1	0.90	$R_{3,2}$	I_3	0.87
$R_{2,2}$	I_2	0.85	$R_{2,1}$	I_2	0.79
$R_{4,1}$	I_4	0.83	$R_{3,3}$	I_3	0.75
$R_{3,3}$	I_3	0.71	$R_{1,1}$	I_1	0.72
$R_{2,1}$	I_2	0.69	$R_{1,2}$	I_1	0.70
⋮	⋮	⋮	⋮	⋮	⋮

Table 3.1: A sorted access example for a query image with two regions $R_{q,1}$ and $R_{q,2}$.

It is plain to see that the image most similar to I_q is the image I_2 (the overall similarity score, computed as the average sum of regions similarities, is $(0.9+0.7)/2 = 0.80$ for image I_1, $(0.85 + 0.79)/2 = 0.82$ for I_2, and $(0.71 + 0.87)/2 = 0.79$ for I_3, with other images leading to lower scores), whereas the candidate set only contains images I_1 and I_3.

In order to find a correct condition to stop the sorted accesses, we start from Fagin's $\mathcal{A}_0$ algorithm [Fag96]. The $\mathcal{A}_0$ algorithm stops the sorted access phase when at least k objects are included in *all* the index scans results. The only requirement for the $\mathcal{A}_0$ algorithm is that the function applied to combine objects' scores (in our case, the IM_{sim} function) has to be monotonic which is our case (see Section 3.2.2). Applying the $\mathcal{A}_0$ algorithm to the optimal image matching problem would be as in Figure 3.9.

$\mathcal{A}_0$(`query image`, k: `integer`, $\mathcal{T}$: `DBAM`)
{ $\forall$ `region` $R_{q,i}$ `of` I_q, `open a` ***sorted access*** `index scan on` $\mathcal{T}$ `and insert images containing result regions in the set` X^i;
`stop the sorted accesses when there are at least` k `images in the intersection` $L = \cap_i X^i$;
`for each image` I_s `in the` ***candidate set*** $\cup_i X^i$, `compute the optimal assignment;` (***random access***)
`return the` k `images having the highest overall similarity scores` $I_{sim}(I_s, I_q)$; }

Figure 3.9: The $\mathcal{A}_0$ algorithm for the optimal image matching problem.

$\mathcal{A}_0$, however, does not guarantee yet that the k best images are included in the candidate set, since its stopping condition does not take into account that assignment of regions has to be a matching. Just consider, as an example, the case depicted in Table 3.2, where $n_q = 2$ and $k = 1$. Here, as opposed to the case of Table 3.1, it is not correct to stop the sorted access phase at the second step, since image I_2 has been found for both query regions with the same region $R_{2,1}$; therefore, we cannot find a matching for image

I_2 by using only regions that have been seen during the sorted access phase.

$R_{q,1}$			$R_{q,2}$		
region	image	similarity	region	image	similarity
$R_{1,1}$	I_1	0.90	$R_{3,2}$	I_3	0.87
$R_{2,1}$	I_2	0.85	$R_{2,1}$	I_2	0.79
$R_{4,1}$	I_4	0.83	$R_{3,3}$	I_3	0.75
$R_{3,3}$	I_3	0.71	$R_{1,1}$	I_1	0.72
$R_{2,3}$	I_2	0.69	$R_{1,2}$	I_1	0.70
⋮	⋮	⋮	⋮	⋮	⋮

Table 3.2: Another sorted access example for a query image with two regions $R_{q,1}$ and $R_{q,2}$.

To ensure that the k best results are included into the set of candidate images, the stopping condition of $\mathcal{A}_0$ algorithm has to be modified to test correctness of regions' assignments (see Definition 3.1). The sorted access phase can be stopped as soon as a complete matching (Property 3.1) is found, by taking into account only regions returned by index scans.[3] In the example of Table 3.2, hence, we stop the sorted access phase after the fourth step, since image I_3 has a complete matching ($\Gamma_3(R_{q,1}) = R_{3,3}$ and $\Gamma_3(R_{q,2}) = R_{3,2}$). It should be noted, however, that this is *not* the best result for I_q (image I_1 leads to the best overall score of 0.8). In other words, the sorted accesses can be stopped as soon as it is guaranteed that each image outside of the candidate set leads to an overall similarity score lower than that of the k-th best image, i.e. when optimal matchings for non-candidate images could only lead to lower scores with respect to the k-th best matching within the candidate set.

Consider again, as an example, the case where $n_q = 2$ and $k = 1$, and refer to Table 3.2. After the first step, the candidate set is $\{I_1, I_3\}$ with overall scores $(0.9+0)/2 = 0.45$ and $(0 + 0.87)/2 = 0.435$, respectively. Since an image outside the candidate set could potentially lead to an overall score of $(0.9 + 0.87)/2 = 0.885$, we have to continue the sorted access phase. After the second step, we add image I_2 to the candidate set with an overall score of $(0.85 + 0)/2 = 0.425$ (remember that region $R_{2,1}$ can match at most one region of I_q); therefore, the sorted accesses cannot be stopped yet. At the third step, also image I_4 is added to the candidate set, with a score of $(0.83 + 0)/2 = 0.415$. Finally, at fourth step, we obtain a complete matching for image I_3 ($\Gamma_3(R_{q,1}) = R_{3,3}$ and $\Gamma_3(R_{q,2}) = R_{3,2}$) with a score of $(0.71 + 0.87)/2 = 0.79$. In this case, the sorted access phase can be stopped, since images outside of the candidate set can only lead to lower

[3]By the way, this is the reason why Blobworld algorithm is not correct, since its stopping condition cannot guarantee the existence of a complete matching.

scores (at most $(0.71 + 0.72)/2 = 0.715$). The monotonicity of the combining function IM_{sim} is used here to ensure algorithm correctness (see below). Note, however, that image I_3 is *not* the best result for I_q, since image I_1 leads to the best overall score of 0.8. In order to solve the optimal image matching problem on the set of candidate images, we need to compute similarity scores between query regions and *all* the regions of candidate images.

From above example, it is clear that the sorted access phase can be stopped as soon as a complete matching is found, by taking into account only regions returned by index scans. This leads to the $\mathcal{A}_0^{WS}$ algorithm shown in Figure 3.10. At this point it is important to note that no assumptions are done on the way the sorted access lists have to be explored (i.e. in parallel or in a differential way). Samet at al. in [HS99] say that, for its intrinsic properties, a DBAM is able to establish in a correct way, at each time, which is the most promising sorted access. This allows a differential way to visit the lists that optimizes the sorted access phase. Thus, also WINDSURF can support not only the simple parallel access but also the more efficient differential modality.

$\mathcal{A}_0^{WS}$(`query image`, k: `integer`, $\mathcal{T}$: `DBAM`)
{ $\forall$ `region` $R_{q,i}$ `of` I_q, `open a` *`sorted access`* `index scan`
  `on` $\mathcal{T}$ `and insert result regions in the set` X^i;
  `stop the sorted accesses when there are at least` k `images for which a` *`complete`*
  `matching exists, considering only regions in` $\cup_i X^i$;
  $\forall$ `image` I_s `having regions in` $\cup_i X^i$,
  $\forall$ `pair` $R_{q,i}, R_{s,j}$
  `if` $R_{s,j} \notin X^i$ `compute score` s_{ij}; (*`random access`*)
  `compute the optimal assignment;` (*`combining phase`*)
  `return the` k `images having the highest overall similarity scores` $I_{sim}(I_q, I_s)$; }

Figure 3.10: The $\mathcal{A}_0^{WS}$ algorithm.

The *random access* phase consists in computing those similarity scores s_{ij} between query regions and regions of candidate images not present in the X^i regions result sets. After that, the *combining phase* determines the optimal matchings for all the candidate images.

Correctness of the $\mathcal{A}_0^{WS}$ algorithm follows from the monotonicity of the IM_{sim} combining function. Without loss of generality, consider the case $k = 1$ and assume by contradiction that the nearest neighbor image, say I_{nn}, of I_q is *not* in the candidate set. Also observe that the candidate set includes (at least) one image, say I_s, for which a complete matching has been obtained. We prove that $I_{sim}(I_q, I_s) \geq I_{sim}(I_q, I_{nn})$. Because

of the monotonicity of IM_{sim} it is enough to show that, for each $i \in [1, n_q]$, it is:

$$r_{sim}(R_{q,i}, \Gamma_s(R_{q,i})) \geq r_{sim}(R_{q,i}, \Gamma_{nn}^{opt}(R_{q,i}))$$

where Γ_s is the matching obtained for I_s from the sorted access phase, and Γ_{nn}^{opt} is the optimal matching for I_{nn}. Assume that there exists a value of i for which it is:

$$r_{sim}(R_{q,i}, \Gamma_{nn}^{opt}(R_{q,i})) > r_{sim}(R_{q,i}, \Gamma_s(R_{q,i}))$$

However, this is impossible since the region $\Gamma_{nn}^{opt}(R_{q,i})$ of I_{nn} does not belong, by hypothesis, to the set X^i obtained from the i-th sorted access scan. This is enough to prove that $I_{sim}(I_q, I_s) \geq I_{sim}(I_q, I_{nn})$.

The index evaluation of compound queries, thus, has a twofold impact on query evaluation: First, the use of an index can reduce the number of distance computations needed for assessing image similarity; second, the number of images on which the Hungarian algorithm has to be run is reduced by considering only images in the candidate set.

3.3.1 Optimizations to $\mathcal{A}_0^{WS}$ Algorithm

In Section 3.3 we have described an index-based algorithm aiming to speed-up the evaluation of k nearest neighbors queries. We have observed that the $\mathcal{A}_0^{WS}$ algorithm, even if more efficient as compared to the sequential one, can be further optimized, reducing the number of distance computations needed to answer a query, using bounds provided by the underlying index structure [GBK00, FLN01]. In this Section we describe the optimized version of $\mathcal{A}_0^{WS}$, called $\mathcal{A}_0^{WS^*}$.

Bound-Based Algorithm

To better understand how to improve the $\mathcal{A}_0^{WS}$ algorithm, we give a geometrical interpretation of the problem. Referring to the geometrical model for combining query results introduced in [PF95], we depict our considered scenario in Figure 3.11. For simplicity, we suppose that images contain only two regions ($n_q = m_s = 2$) and $k = 1$. Each image of the DB can be viewed as a point in a 2-dimensional space and each region can be mapped onto an axis divided into score values (from 0, representing the smallest similarity value, to 1, the highest one). Evaluating results of a subquery (representing a region of a query image) by ranks can be represented as moving a hyperplane orthogonal to its respective axis from 1 to 0. The order in which objects are collected by the hyperplane can be mapped onto the ranks on which they occur. Let X^1 and X^2 be the two sorted access lists defined as:

$$X^1 = [\mathbf{p}_{1,1}, \mathbf{p}_{2,1}, ..., \mathbf{p}_{o,1}] \qquad X^2 = [\mathbf{p}_{1,2}, \mathbf{p}_{2,2}, ..., \mathbf{p}_{o',2}]$$

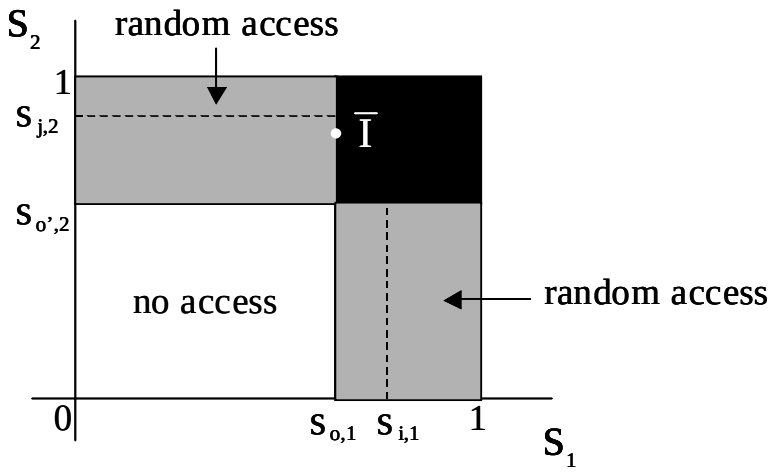


Figure 3.11: Geometrical representation of the $\mathcal{A}_0^{WS}$ algorithm.

where $\mathbf{p}_{i,1}$ and $\mathbf{p}_{j,2}$ represent the labels of two generic regions returned in X^1 and X^2. Let then $s_{i,1}$ and $s_{j,2}$ be the score values associated to the regions labelled as $\mathbf{p}_{i,1}$ and $\mathbf{p}_{j,2}$, respectively. Finally, let $\overline{I}$ be the object satisfying the stopping condition of the sorted access phase of the $\mathcal{A}_0^{WS}$ algorithm (i.e. the image of the data set for which a complete matching exists). By means of the overall similarity score, computed as the average sum of regions similarity, we compute the "local" similarity $S(\overline{I})$ for the object $\overline{I}$.

Definition 3.4 (Local Similarity)
Let $\mathbf{p}_{i,1}$, $\mathbf{p}_{j,2}$ be the two regions of image I with score values $s_{i,1}$ and $s_{j,2}$. The *local similarity* value for I is defined as the average of its local score values (i.e. scores returned during the sorted access phase). In case a score is not present in an index scan its value is set to zero.

$$S(I) = \frac{s_{i,1} + s_{j,2}}{2} \tag{3.21}$$

Because of the monotonicity of the scoring function we can say that $S(\overline{I})$ represents a lower bound for the similarity of the "best" image, i.e. the image having the highest score.

It has to be noted that, following the definition of the stopping condition, objects belonging to the white rectangular area in Figure 3.11, defined by $s_{o,1}$ and $s_{o',2}$, can be discarded. At this point, the random access phase has place and distance values for objects in the grey areas are computed. Finally, following the $\mathcal{A}_0^{WS}$ algorithm, the optimal assignment for all images belonging the candidate set, which is geometrically represented by way of the two grey rectangles shown in Figure 3.11, is obtained. The black area contains objects for which a complete matching has been found (k in the general case).

Maintaining the stopping condition of the sorted access phase, we note that it is possible to further cut down the cardinality of the candidate set obtaining an optimization of the random access: This is the goal of the $\mathcal{A}_0^{WS^*}$ algorithm.

Looking at Figure 3.12, let us observe that the evaluation of the aggregated scores by way of the scoring function can be geometrically represented as moving a hypersurface, collecting objects while moving over the space. Since objects collected first have higher aggregated scores, the hypersurface should be orthogonal to the optimal direction starting at the optimal level. Since image $\overline{I}$ provides a bound on the similarity of the result image, we only have to consider objects located "above" the diagonal line passing in $\overline{I}$, that is in the two light grey triangular regions in Figure 3.12. It is now clear that the cardinality of the candidate set images can be drastically reduced, with no relevant object missed.

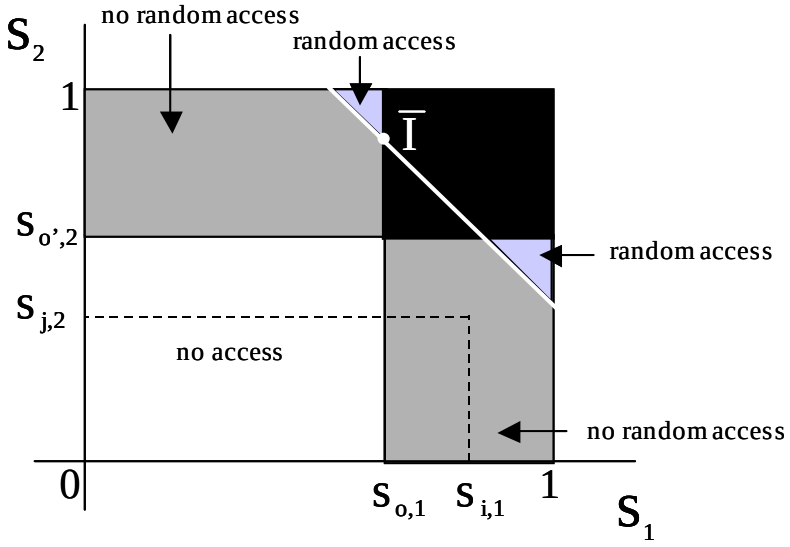


Figure 3.12: Geometrical model the $\mathcal{A}_0^{WS^*}$ algorithm.

From an algorithmic point of view, we can formalize the $\mathcal{A}_0^{WS^*}$ algorithm as follows: Let $\mathbf{p}_{i,1}$ be the region from image I returned from the index scan 1 at time i and let $s_{i,1}$ be its score value, and suppose that, at time i, we have no information for image I from scan 2, i.e. we do not know the value of $s_{j,2}$. A bound on the value of $S(I)$ can be immediately found, by considering that $s_{j,2} \leq s_{o',2}$ holds. The "approximate" similarity $S^+(I)$ for image I can be computed as follows.

Definition 3.5 (Approximate Similarity)
Given an image I with score values $s_{i,1}$ (returned by the index scan 1) and $s_{j,2}$ (not returned from the index scan 2), the *approximate similarity* value for I is computed as

```
A_0^{WS*}(I_q: query image, k: integer, T: DBAM)
{ ∀ region R_{q,i} of I_q, open a sorted access index scan
    on T and insert result regions in the set X^i;
  stop the sorted accesses when there are at least k images for which a complete
    matching exists, considering only regions in ∪_i X^i;
  ∀ image I_s having regions in ∪_i X^i
    compute the local image similarity S_s;
    compute the approximate image similarity S_s^+
  ∀ image I_s having regions in ∪_i X^i
    if S_s^+ ≥ max{S_y}
      ∀ pair R_{q,i}, R_{s,j}
        if R_{s,j} ∉ X^i compute score s_ij; (random access)
      compute the optimal assignment; (combining phase)
  return the k images having the highest overall similarity scores I_sim(I_q, I_s); }
```

Figure 3.13: The $\mathcal{A}_0^{WS^*}$ algorithm.

the average of $s_{i,1}$ and $s_{o',2}$:

$$S^+(I) = \frac{s_{i,1} + s_{o',2}}{2} \tag{3.22}$$

where $s_{o',2}$, the score value associated to $\mathbf{p}_{o',2}$, the last object retrieved by the index scan, represents an upper bound for $s_{j,2}$.

As an immediate consequence, the following holds:

$$S(I) = \frac{s_{i,1} + s_{j,2}}{2} \leq \frac{s_{i,1} + s_{o',2}}{2} = S^+(I)$$

Thus, $S^+(I)$ is an upper bound on the similarity value of image I. Like shown in Figure 3.13, starting from the same candidate set of the $\mathcal{A}_0^{WS}$ algorithm, we do not need to solve the assignment problem for all candidate images but only for images whose upper bound $S^+(I)$ is greater than or equal to the maximum similarity value obtained so far (let us denote such score as $S_{\max}$):

$$S^+(I) \geq S_{\max}$$

Related to the state-of-the-art about query processing algorithms, we would like to stress that the WINDSURF query processing strategy has the same potentialities as the Quick-Combine approach [GBK00]. Moreover, like pointed out in Section 3.3, WINDSURF is also able to optimize the sorted access phase in a better way than Quick-Combine, determining, in a correct way, which is, for each step, the most promising sorted access list and, thus, finding the fastest way to satisfy the stopping condition.[4] Since WINDSURF

[4]The Quick-Combine approach bases this kind of optimization on the use of simple heuristics [GBK00].

uses an index structure for which the sorted access strategy proposed in [HS99] can be used, we are able to establish:

- At which distance is the next entry in the sorted access list.
- A tight bound on such distance, in case the previous information is not available.

In both cases, it is clear that the bounds used from $\mathcal{A}_0^{WS^*}$ can be further cut down, producing further efficiency improvements.

Moreover, it has also to be pointed out that the presented query processing strategy can be used with *any* monotonic combining function and not with just the average.

3.4 Experimental Results

Preliminary experimentation of the proposed techniques has been performed on a sample medium-size data set consisting of about $2,000$ real-life images, yielding more than $10,000$ regions, extracted from a CD-ROM from *IMSI-PHOTOS* [IMS]. The query workload consists of about 100 randomly chosen images not included in the data set. All experiments were performed on a Pentium II 450 MHz PC equipped with 64MB of main memory and running Windows NT 4.0.

3.4.1 Efficiency

The first set of experiments we present concerns the efficiency of the proposed approach. In order to test the performance of the $\mathcal{A}_0^{WS}$ index-based algorithm, in Figure 3.14 we compare the number of candidate images, i.e. the images on which the Hungarian algorithm has to be applied, as a function of the number of query regions.[5] Of course, for the `ERASE` algorithm the number of candidate images equals the number of images in the data set, whereas for the index version this number depends both on values of k and of the number of query regions. As the graph shows, the $\mathcal{A}_0^{WS}$ algorithm is indeed very efficient in reducing the number of candidate images, even if its performance deteriorates as the number of query regions increases. This is intuitive, since the complexity of finding k objects in the intersection of n_q sets augments with n_q.

Another element affecting performance is the number of computed distances between regions. In Figure 3.15 (a) we show the number of computed distances for the `ERASE` and the $\mathcal{A}_0^{WS}$ algorithms, as a function of k, with a number of query regions $n_q = 3$. In order

[5] Unless otherwise specified, all the graphs presented here show numbers averaged over all the images contained in the query workload.

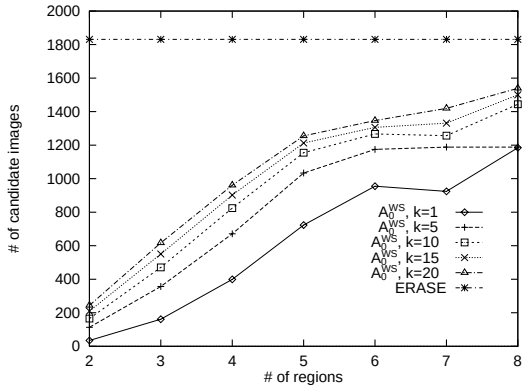


Figure 3.14: Average number of candidate images as a function of the number of query regions.

to reduce the number of distances to be computed for the index-based algorithm, we also considered an approximate version of the $\mathcal{A}_0^{WS}$ algorithm, called $\mathcal{A}_0^{WS_{app}}$. In this case, the random access phase computes the optimal matching for each candidate image by taking into account only regions returned by the sorted access phase, i.e. no new distance is computed. Average number of distance computations for the $\mathcal{A}_0^{WS_{app}}$ algorithm is also shown in Figure 3.15 (a). The graph shows that the index-based approach is not very efficient in reducing the number of computed distances. We believe that this is due to the low cardinality of the data set: Increasing the number of images in the data set would have a beneficial effect on the performance of index-based algorithms (whose search costs grow logarithmically with the number of indexed objects) with respect to that of sequential ones.

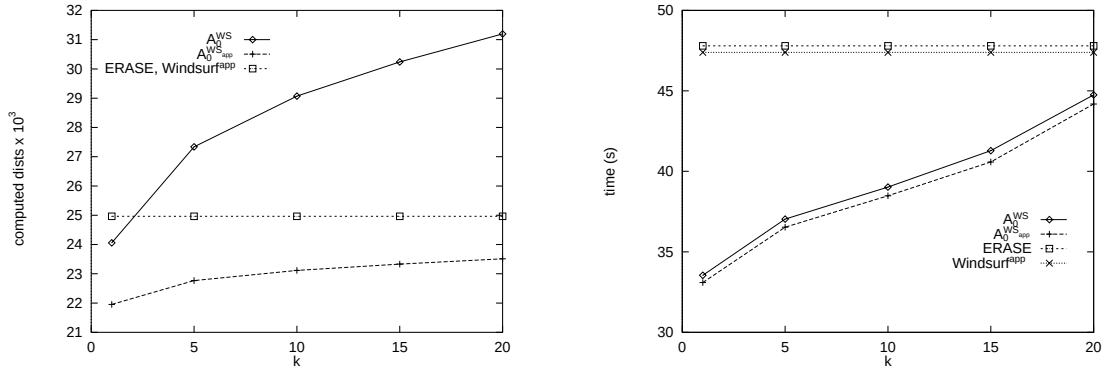
Finally, in Figure 3.15 (b) we compare query response times as a function of k (with a constant value of $n_q = 3$). The graph shows average query evaluation times (in seconds) for the `ERASE` algorithm, the WINDSURF^{app} heuristic matching algorithm, and the two index-based algorithms, $\mathcal{A}_0^{WS}$ and $\mathcal{A}_0^{WS_{app}}$, respectively. From the graph it can be deduced that:

- The lower complexity of image matching for the WINDSURF^{app} algorithm with respect to the `ERASE` algorithm does not pay off in reducing query evaluation times. This is due to the fact that, if n_q is low (as it is in our case), finding the optimal result is very easy.

- The index-based algorithms really succeed in cutting down query resolution times, even if difference in performance reduces with increasing values of k.

- The approximate $\mathcal{A}_0^{WS_{app}}$ algorithm has performance similar to that of the exact

$\mathcal{A}_0^{WS}$ algorithm. This demonstrates that the performance improvement with respect to sequential query evaluation is due to the lower number of candidate images, and that the number of computed distances has a minor impact on performance.



(a) (b)

Figure 3.15: Average number of computed distances (a) and average query resolution time (b) as a function of k ($n_q = 3$).

3.4.2 Effectiveness

In order to compare the "goodness" of results obtained by approximate algorithms (i.e. the WINDSURF^{app} heuristic matching algorithm and $\mathcal{A}_0^{WS_{app}}$) with respect to those obtained by exact ones (`ERASE` and $\mathcal{A}_0^{WS}$), we need a performance measure able to compare results of k nearest neighbors queries. Such measure should compare two sorted lists of results, that is, it should contrast *ranks* (positions) of images in exact and approximate results. Given the i-th image in the approximate result, its rank, rank(i), is given by the position of that image in the exact result. As an example, consider the case when $k = 1$. The "goodness" of an approximate result with respect to the exact one can be obtained by just taking into account rank(1): The lower rank(1) is, the better the approximation works. This measure can be easily extended to the case where $k > 1$ by considering the ranks of all the k images in the approximate result, i.e. rank(1), ..., rank(k).

In [WB00], the *normalized rank sum* (nrs) is used to quantify the loss of result quality when k nearest neighbors queries are approximately evaluated. The nrs is defined as:

$$nrs = \frac{k(k+1)}{2 \cdot \sum_{i=1}^{k} \text{rank}(i)} \tag{3.23}$$

The nrs is computed as the inverse of the sum of all the ranks of the images in the approximate result. Thus, higher values of nrs are to be preferred. This measure, however,

is not able to capture inversions in the result (e.g. when image I_s is ranked higher than image $I_{s'}$ in the approximate result and lower in the exact result), since no difference between ranks of images in the approximate and in the exact results is taken into account.

In [ZSAR98], the *precision of approximation* measure P is introduced, which is defined as:

$$P = \frac{1}{k} \sum_{i=1}^{k} \frac{i}{\text{rank}(i)} \tag{3.24}$$

P, therefore, measures the relative error in ranking for all the images in the approximate result. This measure, however, relies on the assumption that $i \leq \text{rank}(i)$, thus no inversions on results are allowed.

To overcome above limitations in quality measures, we introduce a new measure, the normalized rank difference sum ψ. To compute ψ, we sum differences in rankings for images in the approximate result and normalize by k. Normalization of the measure in the interval $[0, 1]$ leads to the formulation of ψ as follows:

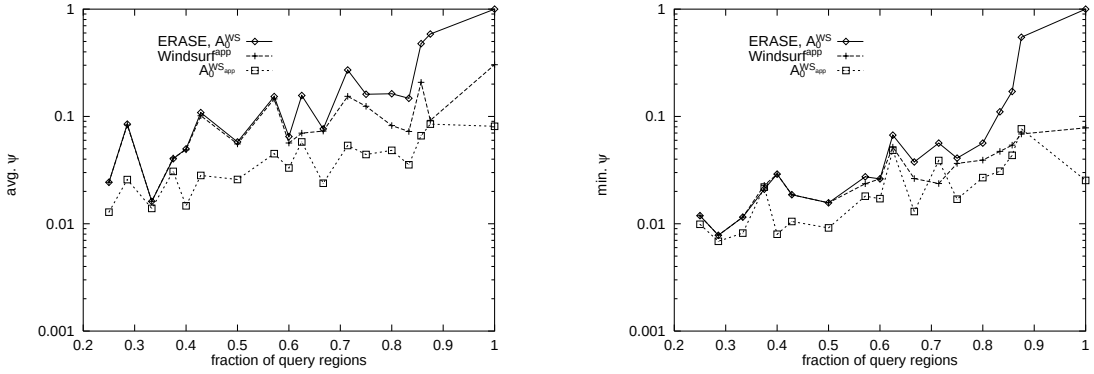
$$\psi = \frac{1}{1 + \frac{1}{k} \left(\sum_{i=1}^{k} \mathbf{1}(\text{rank}(i) - i)^p \right)^{1/p}} \tag{3.25}$$

where $\mathbf{1}()$ is the ramp function ($\mathbf{1}(x) = 0$ if $x < 0$, $\mathbf{1}(x) = x$ if $x \geq 0$), and p is an integer parameter (we used $p = 2$ in our experiments). Values of ψ close to 1 indicate high quality of the approximate result. The use of the ramp function $\mathbf{1}()$ is needed to avoid counting twice the effects of inversions in ranking. For instance, consider the case where $k = 3$ and the exact result is I_1, I_2, and I_3. If the approximate result is I_1, I_3, I_2, it is $rank(1) = 1$, $rank(2) = 3$, and $rank(3) = 2$. Accordingly to Equation 3.25, and setting $p = 2$, the value of ψ is computed as:

$$\psi = \frac{1}{1 + \frac{1}{3} \left(\mathbf{1}(1-1)^2 + \mathbf{1}(3-2)^2 + \mathbf{1}(2-3)^2 \right)^{1/2}} = \frac{1}{1 + \frac{1}{3} \left(0 + 1 + 0 \right)^{1/2}} = 0.75$$

Figure 3.16 shows average (a) and minimum (b) values of ψ for exact and approximate algorithms as a function of the fraction of query regions used to query the DB (the value of k is kept fixed at 20, other values lead to similar results and are omitted here for brevity). This is to show the effectiveness of different approaches when only some regions of the query image are used for the query (this can be done in order to reduce the query response time or just because we are interested only in some objects included in the query image). Both graphs exhibit similar trends: The effectiveness of the $\mathcal{A}_0^{WS_{app}}$ algorithm is almost always the lowest, and, for all curves, ψ only reaches high values when the fraction of query regions is close to 1. Figure 3.16 (b) shows that, in order to find a "good"

result, we have to use *all* the regions in the query image. From Figure 3.16 (a), on the other hand, we see that approximate algorithms lead to a low effectiveness, even if, as we have seen before, they attain slightly better performance with respect to their exact counterparts.



(a) (b)

Figure 3.16: Average (a) and minimum (b) ψ as a function of the fraction of query regions ($k = 20$).

Moreover, to show the effectiveness of our approach, we also compare results obtained by the $\mathcal{A}_0^{WS}$ algorithm when only a fraction of query regions is used to query the DB which leads to *approximate* queries. In Figure 3.17 the tradeoff between quality of the result and query evaluation cost is shown. Quality is measured as the sum of similarity scores for the k best images normalized with respect to the case where all regions of the query are used. Cost is computed as the elapsed time relative to the time needed for resolving the "all regions" query. The graph clearly shows that quality and cost are strictly correlated in that both decrease when the number of query regions reduces. As a further observation, since the major part of the points falls below the "relative cost=quality" line, an effective way to reduce query costs is to use only some of the regions in the query image.

3.4.3 Comparison with Other CBIR Techniques

In order to evaluate the effectiveness of WINDSURF, we compare the results of WINDSURF for a number of image queries with those obtained by using the method proposed in [SO95] (denoted SO) and the IBM QBIC system [FSN+95].

Comparison with SO

From a semantic point of view, results obtained by WINDSURF are considerably better with respect to those obtained by the SO method. As an example, consider Figure 3.18:

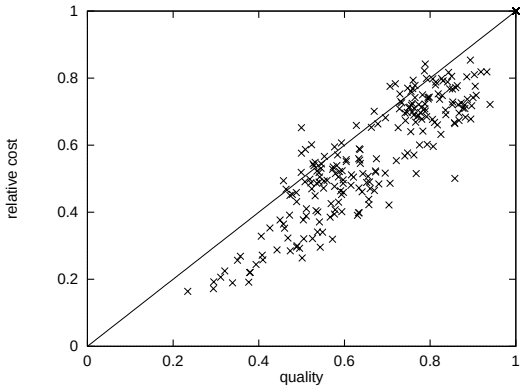


Figure 3.17: Tradeoff between quality and cost for approximate queries ($k = 15$).

Results for SO (SO1 - SO5) contain images semantically uncorrelated to the query image (e.g. image (SO3), a house, and image (SO5), a harbour). As for the results of WINDSURF (WS1 - WS5), all of them present a "sky" region and a darker area.

The superior effectiveness of our approach is confirmed when considering "difficult" queries, i.e. queries having a low number of similar images in the DB. In Figure 3.19 we show the results for a query having only two similar images: For SO, none of the two images is included in the result. WINDSURF, on the other hand, retrieves both images.

Finally, we compared the two approaches when dealing with "partial-match" queries, i.e. queries specifying only a part of the image. As an example, consider Figure 3.20, where the query image is obtained by "cropping" a DB image, namely, the dome of St. Peter in Rome. With WINDSURF all the retrieved images refer to St. Peter, with the only exception of image (WS3), representing the dome of St. Marcus in Venice. Indeed, the query image was extracted from image (WS1). When we analyze the result obtained by using SO, we see that only one image related to the query image is retrieved in third position, whereas other images, with the exception of image (SO2) (again the dome of St. Marcus), are totally uncorrelated to the query image.

Comparison with QBIC

Results obtained by WINDSURF are considerably better also with respect to those obtained by the IBM QBIC system that uses color histograms to represent images. As an example, consider Figure 3.21: The query image depicts the USA flag (note that in the data set there are only nine images containing USA flags and other six images representing flags of different countries); results for QBIC (QBIC1 - QBIC5) contain images semantically uncorrelated to the query image (e.g. images (QBIC1) and (QBIC4), representing an aircraft, and image (QBIC5), people). As for the results of WINDSURF (WS1 - WS5), all

Figure 3.18: Results for the "mountains" query.

images contain the USA flag.

A second experiment shows the superior effectiveness of WINDSURF not only for "difficult" queries but also for gray-scale images. Figure 3.22 shows the results for a gray-scale image representing an airship of WWI having only eight similar images in the data set: For QBIC, only two of the eight images are included in the result, also considering $k = 10$. WINDSURF, on the other hand, retrieves four images, considering the first five results, and all eight images considering ten results. Moreover, QBIC's results present also two color images; that confirms its limit to deal with gray-scale images and the better effectiveness of WINDSURF.

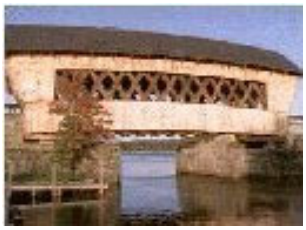
(query)

(SO1)

(SO2)

(query)

(WS1)



(WS2)

Figure 3.19: Results for the "bridge" query.

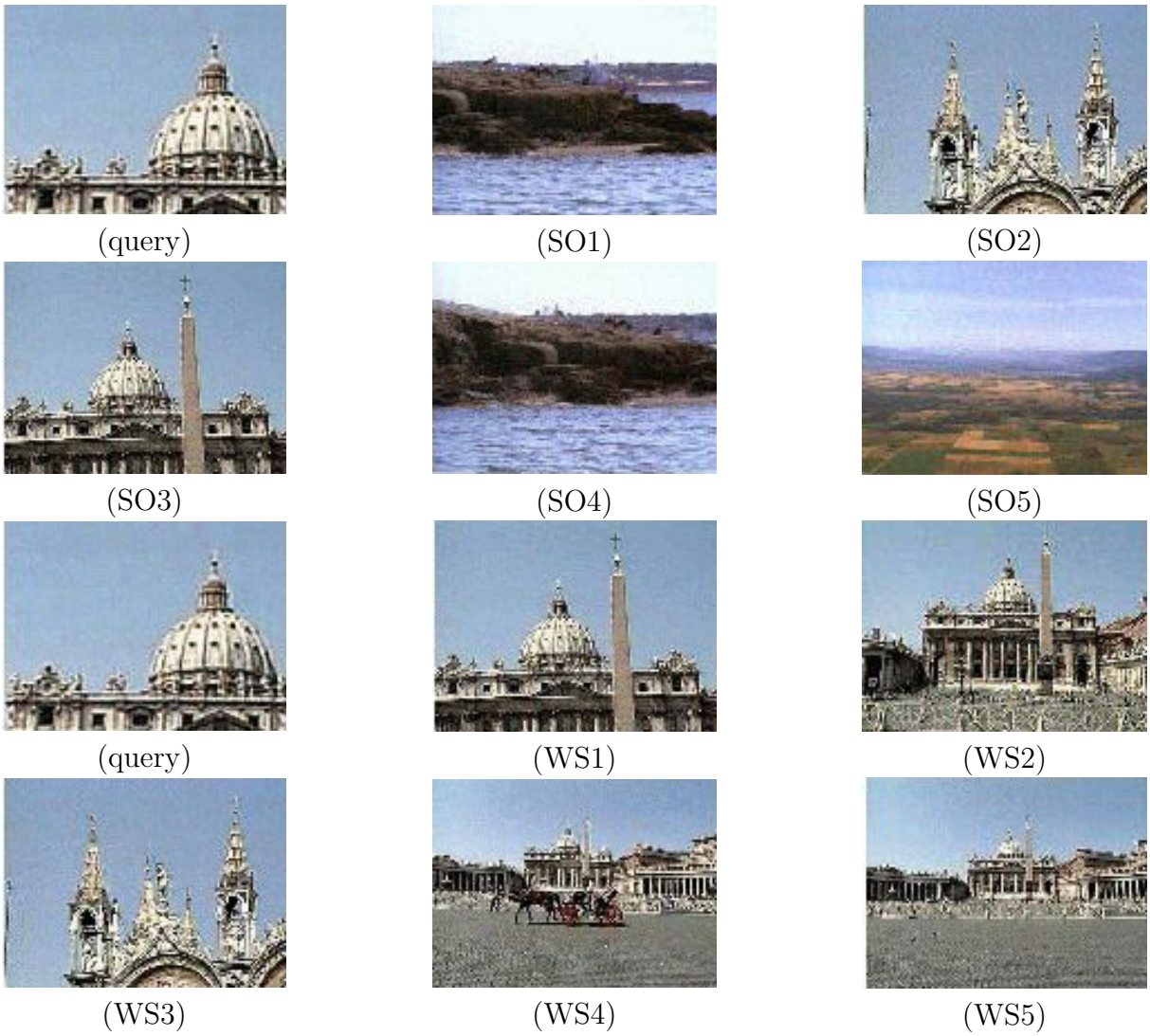


Figure 3.20: Results for the "dome" query.

Figure 3.21: Results for the "flag" query.

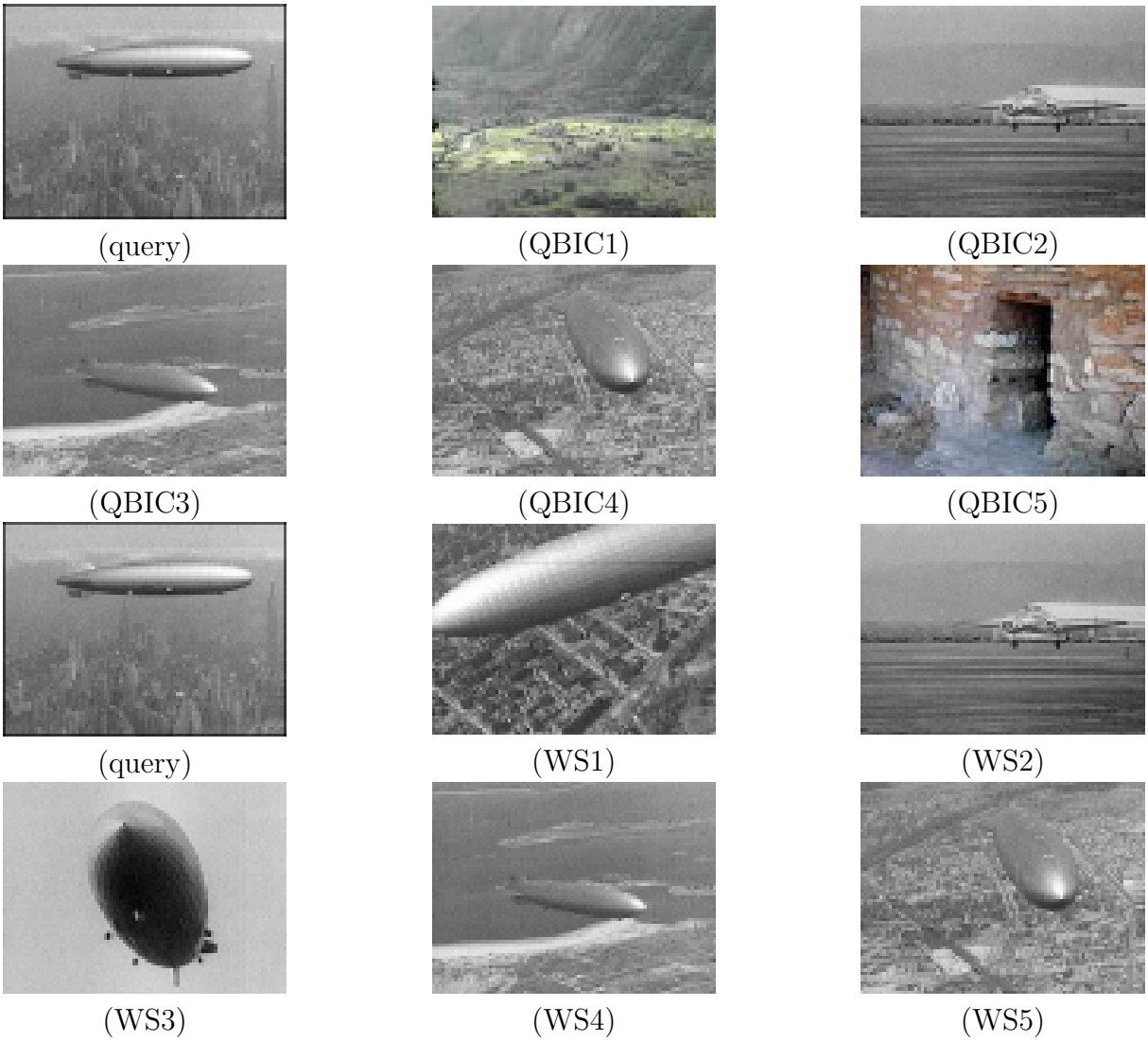


Figure 3.22: Results for the "airship" query.

Chapter 4

FeedbackBypass

In Section 2.3 we observe that, although relevance feedback has been recognized as a method for improving interactive retrieval effectiveness, its applicability suffers two major annoyances:

1. Depending on the query, numerous iterations might occur before an acceptable result is found, thus convergence might be slow.

2. Once the feedback loop of a query is terminated, no information about the particular query is retained for re-use in further processing. Rather, for successive queries, the feedback process is started anew.

To overcome such limitations, we have presented FeedbackBypass [BCW00, BCW01a, BCW01b], a new approach to interactive similarity query processing, which complements the role of current relevance feedback engines. Note that, for the moment, we have taken into account only image retrieval systems that rely on global features extraction, but we plan to apply the FeedbackBypass approach also to WINDSURF and more in general, to any region-based image retrieval system.

4.1 Basic Principles

FeedbackBypass is based on the idea that, by properly storing and maintaining the information on query parameters gathered from past feedback loops, it is possible to either "bypass" the feedback loop completely for already-seen queries, or to "predict" near-optimal parameters for new queries. In both cases, as an overall effect, the number of feedback and DB search iterations is greatly reduced, thus resulting in a significant speed-up of the interactive search process.

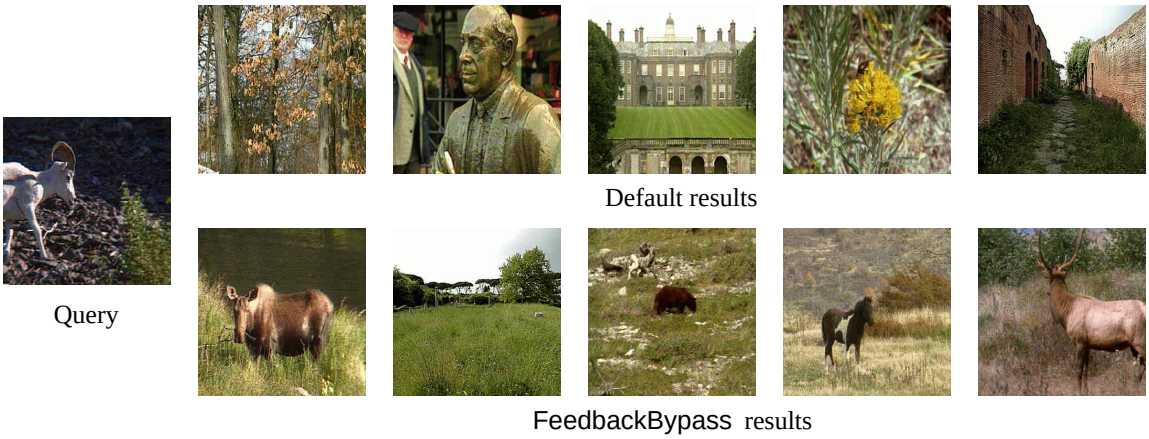


Figure 4.1: FeedbackBypass in action. The upper line shows the 5 best matches computed using default parameters for the query image on the left. The bottom line shows the results obtained for the same query when the parameters suggested by FeedbackBypass are used.

Figure 4.1 shows a query image together with the 5 best results obtained from searching with default parameters a data set of about 10,000 color images. No result image belongs to the same semantic category of the query image, which is "Mammal" (see Section 4.5 for a description of image categories). The bottom line of the figure shows the 5 best matches obtained for the same query when FeedbackBypass has been switched-on, and the system uses the predicted query parameters. This leads to have 4 relevant images (i.e. 4 mammals) in the 5 top positions of the result.

The implementation of FeedbackBypass is based on a novel wavelet-based data structure, called *Simplex Tree*, whose storage overhead is linear in the dimensionality of the query space, thus making even sophisticated query spaces amenable. Its resource requirements are *independent* of the number of processed queries but only depend on the complexity of the query parameter function, which guarantees proper scalability and performance levels. Furthermore, storage requirements can be easily traded-off for the accuracy of the prediction. FeedbackBypass can be well combined with all state-of-the-art relevance feedback techniques working in high-dimensional vector spaces.

4.2 The FeedbackBypass Approach

This Section explains in detail how FeedbackBypass can be smoothly integrated with query and relevance feedback models commonly used for similarity search.

The basic idea of our approach is to "bypass", or at least to reduce, the loop iterations to be performed by an interactive similarity retrieval system by trying to "guess" what

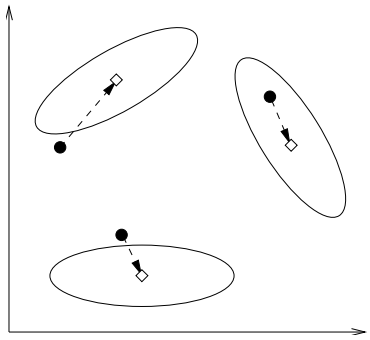


Figure 4.2: The optimal query mapping for 3 sample query points, assuming a quadratic distance.

the user is actually looking for, based only on the initial query he/she submits to the system.

If we abstract from the specific differences arising between existing feedback systems and concentrate on what all such systems share, two important observations can be done:

1. All systems assume that the user has in mind an "optimal" query point as well as an "optimal" distance function for that query.

2. Each time a new distance function is computed, this is taken from a *parameterized class* of functions (e.g. the class of weighted Euclidean distances), by appropriately setting the values of the class parameters.

This general state of things can be synthetically represented as a mapping:

$$\mathbf{q} \mapsto (\mathbf{q_{opt}}, d_{opt}) \equiv (\mathbf{\Delta_{opt}}, \mathbf{W_{opt}}) \tag{4.1}$$

which assigns to the initial query point $\mathbf{q}$ an optimal query point, $\mathbf{q_{opt}}$, and an optimal distance function, d_{opt}. The equivalence just highlights that d_{opt} is the distance function obtained when the parameters are set to $\mathbf{W_{opt}}$, and that $\mathbf{q_{opt}}$ can be obtained from the initial query point by adding to it the "optimal offset" $\mathbf{\Delta_{opt}} = \mathbf{q_{opt}} - \mathbf{q}$. In the following we refer to the pair $(\mathbf{\Delta_{opt}}, \mathbf{W_{opt}})$ as the *optimal query parameters*, OQPs, of query $\mathbf{q}$. Figure 4.2 provides an intuitive graphical representation of the above mapping for three 2-dimensional query points.

FeedbackBypass is based on the observation that, as more and more query points are added, an "optimal" *query mapping*, M_{opt}, from query points to query points and distance functions, will take shape, and that "learning" such mapping can indeed lead to "bypass" the feedback loop.

Let $\mathcal{Q} \subseteq \Re^D$ be the domain of query points and let $\mathcal{W} \subseteq \Re^P$ be the set of possible parameter choices, where each $\mathbf{W} \in \mathcal{W}$ corresponds to a distance function in the considered class and P is the number of independent parameters that characterize a distance function. Then, the problem faced by FeedbackBypass can be precisely formulated as follows:

Problem 4.1
Given the $\mathcal{Q}$ query domain and a class of distance functions with set of parameters $\mathcal{W}$, "learn" the query mapping $M_{opt} : \mathcal{Q} \rightarrow \Re^D \times \mathcal{W}$ which associates to each query point $\mathbf{q} \in \mathcal{Q}$ the optimal query parameters $(\mathbf{\Delta_{opt}}, \mathbf{W_{opt}}) = M_{opt}(\mathbf{q})$.

In other terms, the problem to be faced is that of learning the optimal way to map (query) points of $\Re^D$ into points of $\Re^{D+P}$. It should be remarked that when query points are normalized, the dimensionality of both the input (feature) and the output space of M_{opt} can be reduced by 1.

Of course, statistical techniques for *dimensionality reduction* could be applied to lower the dimensionality of both the input and the output space. We do not consider dimensionality reduction in this thesis, and leave it as an interesting follow-up of our research.

Example 4.1
Assume that objects are color images, which are represented by using a 32-bins color histogram, and that similarity is measured by the weighted Euclidean distance. Since the sum of the color bins is constant (it equals the number of pixels in the image) and one of the weights of the distance function can be set to a constant value, say 1, without altering at all the retrieval process, it turns out that M_{opt} is a function from $\Re^{31}$ to $\Re^{31+31}$.

Figure 4.3 shows the basic architecture of a generic interactive retrieval system enriched with FeedbackBypass, with the flow of interactions being summarized in Figure 4.4 using a C++ like notation. Upon receiving the initial user query $\mathbf{q}$, the system forwards $\mathbf{q}$ to FeedbackBypass by invoking its `Mopt` method, which returns the predicted OQPs $(\mathbf{\Delta_{opt}}, \mathbf{W_{opt}})$ for $\mathbf{q}$. Then, the usual query processing-user evaluation-feedback computation loop can take place. When the loop ends, the new OQPs are passed to FeedbackBypass by invoking its `Insert` method, to be stored as new optimal parameters for $\mathbf{q}$. Clearly, this insertion step can be skipped at all if no feedback information has been provided by the user.

4.2.1 Requirements

The method we seek for learning M_{opt} from sample queries has to satisfy a set of somewhat contrasting requirements, which are summarized as follows:

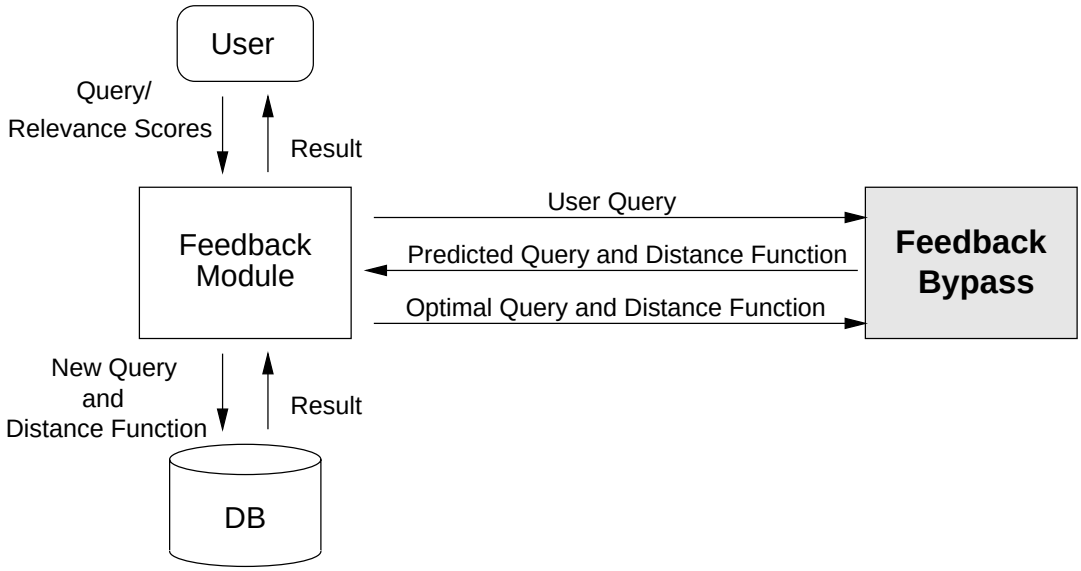


Figure 4.3: An interactive retrieval system enriched with the FeedbackBypass module.

Limited Storage Overhead. Since the number of possible queries to be posed to the system is huge and will grow over time, it is not conceivable to just do some "query book-keeping", i.e. storing the values of M_{opt} for all already-seen queries. The method we seek should have a complexity *independent* of the number of queries and only a low (e.g. linear) complexity in the dimensionalities of the feature and the output spaces.

Prediction. The method should also be able to provide reasonable "guesses" for new queries. It is also requested that the quality of this approximation has to increase over time, as more and more queries and user-feedback information are processed.

Dynamicity. Since we consider an interactive retrieval scenario, it is absolutely necessary that the method is able to efficiently handle updates, i.e. incorporate additional data without rebuilding the approximation of M_{opt} from scratch.

We have been able to achieve a satisfactory trade-off, thus meeting all above requirements, by implementing FeedbackBypass using a wavelet-based data structure, which we call the *Simplex Tree*.

4.3 Wavelets, Lifting, and Interpolation

The process of *learning* a function can be understood as *approximating* the function. From the rich mathematical toolkit of approximation theory, we chose to go with wavelets

```
 //data structure for optimal query parameters (OQPs)
class Oqp {
 Vector Delta(D);
 Vector W(P);
}
 // get the user query
Vector &q = getUserQuery();
 // obtain OQPs from FeedbackBypass
Oqp &v = FeedbackBypass::Mopt(q);
Oqp &vPred = v.copy();
 // main feedback loop
while(feedbackLoop) {
  // compute results for q using OQPs
 Vector results[] = queryEvaluate(q, v);
  // get relevance scores for results
 Score scores[] = getUserFeedback(results)
  // compute new OQPs given the scores
 newValues(q, v, scores);
}
 // in case feedback information has been provided
if(vPred != v)
  // insert new OQPs for query q
 FeedbackBypass::Insert(q, v);
```

Figure 4.4: Basic interactions between an interactive retrieval system and FeedbackBypass.

constructed by a technique called *Lifting*. In this Section, we briefly outline the principles but refer the interested reader, for example, to [Swe96, SS96].

The lifting schema, introduced by Sweldens [Swe96], is a highly effective yet simple technique to derive a wavelet decomposition for a given data set.

Lifting consists of three steps: *Split*, *predict*, and *update*, which are repeatedly applied to the data set. Before we go into detail, it may be helpful to give the reader an intuition of the process. Essentially, the idea behind lifting is to gradually remove data from the original signal and to replace it with information that allows to reconstruct the original data. This removal process is recursively repeated: At the end of each such iteration we obtain a coarser approximation of the original data and information necessary to revert the last approximation step. Finally, after the recursive application of this schema terminated we arrived at the coarsest approximation possible (e.g. one data point) but have also the information how to reconstruct the original data step by step. This proceeding, formally speaking, implements the Wavelet Transform.

For simplicity, let us assume the original data, usually referred to as signal, is given as pairs (x_i, y_i). Moreover, let x_i be equidistant (see Figure 4.5 (a)). The three steps in detail are as follows:

1. In the *split* step, the original data set $Y = \{y_1, y_2, \ldots, y_n\}$ is divided into two subsets Y_1 and Y_2. Although there are no further requirements as to how to choose the subsets, let's assume $Y_1 = \{y_1, \ldots, y_{2k+1}\}$ and $Y_2 = \{y_2, \ldots, y_{2k+2}\}$, see also Figure 4.5 (b). We then remove Y_2 from the data set.

2. In the next step, we *predict* each of the removed points in Y_2 by interpolating on l of the remaining neighbors in Y_1 (in the case of linear interpolation, we simply use the line segment $(x_m, y_m), (x_{m+2}, y_{m+2})$ to approximate (x_{m+1}, y_{m+1})). Let $(\hat{x}_{m+1}, \hat{y}_{m+1})$ be the result of the interpolation. In the case where the interpolation coincides with the original data point, we obviously did not loose any information. However, in general, the points do not coincide. To make up for the loss of information, we determine the difference δ between the predicted and the actual value and store it in place of the original data. At the end of this step we can encode the signal using only Y_1 and $\Delta = \{\delta_1, \ldots, \delta_{k+1}\}$ (cf. Figure 4.5 (c)). The number l of neighbors considered during the interpolation determines the degree of the polynomial function: $l = 2$ corresponds to linear, $l = 4$ to cubic interpolation etc. Special care has to be taken to the fringes of the data set though [Swe96]. In case of $l = 2$, we obtain the well-known Haar-Wavelet (see e.g. [Kai94]).

3. In the *update* step, the remaining original data is adjusted to preserve the total energy of the signal. To see what we mean by this, consider the following example where $y_{2i+1} = 0$ and $y_{2i+2} = 1$. The average value of the signal is 0.5. However after removing all values y_{2i+2}, the signal's average drops to 0, no matter the difference data we stored. Like the predict step, the update step can be performed locally by taking only a data point and its removed neighbor into account.

Conversely, lifting can be used to interpolate signals where data points are added successively. To do so, we simply need to reverse step 1 and 2. There is no need to apply step 3, the update step, as we do not know the total energy of the signal in advance; as a result, the approximation may change fundamentally as the data set changes, in other words, shape and quality of the approximation are evolving with the data set [SS96]. Like with the original lifting technique, we may use polynoms of any degree.

When detailing the three steps above, we assumed the data to be equidistant; this assumption is valid in classical areas of application like signal or image processing. How-

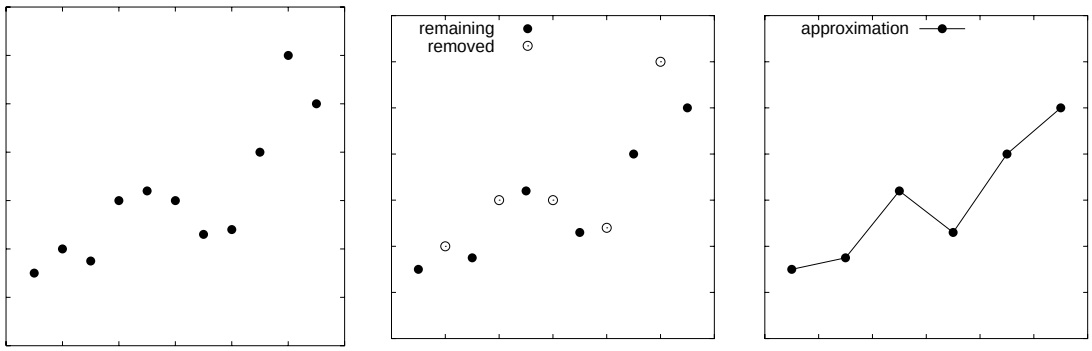


Figure 4.5: Lifting: (a) original data set, (b) split and removing of data set, and (c) predicted data by interpolation.

ever, in the case of interpolation of a growing data set, the x_i cannot be equidistant due to the fact that we introduce more and more points as we go. Thus, the interpolation has to take into account that intervals between data points may vary. In case of linear interpolation, we obtain of the *unbalanced* Haar-Wavelet.

4.4 The Simplex Tree

The Simplex Tree forms the core of our approach. It organizes the query domain $\mathcal{Q}$ as a set of non-overlapping multi-dimensional intervals on which the approximation for M_{opt} can be defined.

Recall that we want to approximate the optimal query mapping $M_{opt} : \mathcal{Q} \rightarrow \Re^D \times \mathcal{W}$, where $\mathcal{Q} \subseteq \Re^D$ and $\Re^D \times \mathcal{W} \subseteq \Re^N$, with $N = D + P$ (see Problem 4.1), given a small but evolving sample of data points, namely queries for which feedback data is available.

Of the various techniques that mathematical approximation theory provides, we have chosen wavelets to approximate the query mapping. Unlike other transforms, such as the Fourier Transform, wavelets model a target function as composition of functions with a limited support. Therefore, modifying the wavelet at a later point in time entails only local recomputations but no re-organization of the representation as a whole.[1] In the following, we make use of this locality and develop the approximation of the optimal query mapping step by step by local recomputation around newly added feedback points. We will use the well-known Haar-Wavelet in the following.

[1]For a comprehensive overview of wavelets and multi-resolution analysis in particular, see, for example, [Kai94, Swe96]

In order to define wavelets in $\mathcal{Q} \subseteq \Re^D$, we first need to organize this high-dimensional vector space as a collection of intervals $\mathcal{S} = \{S_h\}$ such that their union covers the whole query domain, that is, $\mathcal{Q} \subseteq \bigcup_h S_h$. The delimiters of the intervals managed by the Simplex Tree are taken from the sets of points for which user feedback has been provided. Let us denote with $\mathbf{s}$ one of such delimiters, i.e. a query point *stored* in the Simplex Tree. For each $\mathbf{s}$ we maintain in the Simplex Tree also its N-dimensional vector of OQPs, $M_{opt}(\mathbf{s})$. Given $\mathcal{S}$ and a new query point $\mathbf{q}$, the wavelet-based prediction of the OQPs for $\mathbf{q}$ is then obtained, as explained in more detail below, from the OQPs of the stored points that delimit the (unique) interval that contains $\mathbf{q}$.

4.4.1 Multi-Dimensional Triangulation

Given an initial set of query points for which feedback data is available, we define suitable intervals on which we can base our wavelet by *triangulating* the set. In general, a triangulation is a decomposition into simplices, i.e. intervals spanned by $D + 1$ points (that is, triangles in $\Re^2$, tetrahedrons in $\Re^3$, and so forth). Figure 4.6 shows an example for $D = 2$.

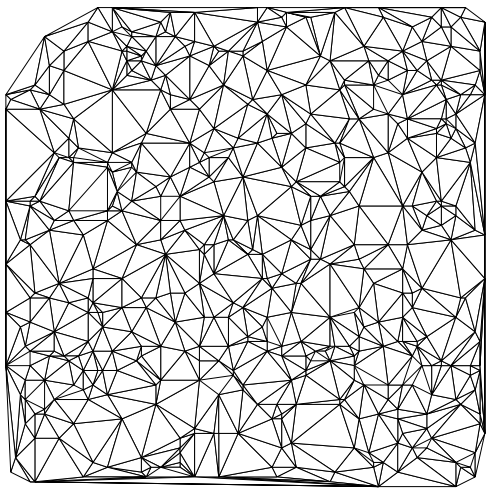


Figure 4.6: Example of triangulation for $D = 2$.

Triangulations are one of the fundamental problems in computational geometry and very efficient techniques to find "good" triangulations are known for low dimensional spaces [Meh94, PS85]. Computing triangulations like the Delaunay triangulation, which minimizes the lengths of edges of the simplices, is computational expensive and too time consuming for dimensions higher than 10.

Instead, to keep the computational effort low, we use an *incremental* triangulation technique as we go forward and *split* for every new point its enclosing simplex. More

formally, let $S = \{\mathbf{s}_1, \ldots, \mathbf{s}_{D+1}\}$ be the set of points spanning the simplex that encloses the new to-be-stored query point $\mathbf{q}$. Then,

$$S_h = \{\mathbf{s}_j | j \neq h\} \cup \{\mathbf{q}\}, \qquad 1 \leq h \leq D+1$$

is a decomposition of S into $D+1$ simplices.[2] Figure 4.7 shows examples for splits in two and three dimensions, respectively.

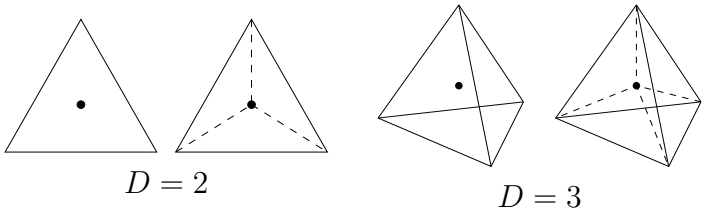


Figure 4.7: Splitting of 2- and 3-dimensional simplices.

Note that splitting a simplex can be done in $O(1)$ time for a fixed dimension, and that the the number of simplices scales linearly with the number of stored query points. Obviously, we can only split a simplex if the new point is inside the simplex itself. To this end we need to define an initial simplex, denoted S_0, such that $\mathcal{Q} \subseteq S_0$, i.e. S_0 covers the entire query domain.

The specific details on how S_0 can be defined depend on the data set at hand. For instance, if $\mathcal{Q} = [0,1]^D$, setting $S_0 = \{(0,0,\ldots,0),(D,0,\ldots,0),\ldots,(0,0,\ldots,D)\}$ guarantees that $\mathcal{Q} \subseteq S_0$, as it can be easily verified. On the other hand, when the data set consist of normalized histograms (i.e. the sum over the bins equals 1), by dropping the value of one bin (e.g. the last one) leads to a query domain $\mathcal{Q}$ which already is a simplex, namely $S_0 = \{(0,0,\ldots,0),(1,0,\ldots,0),\ldots,(0,0,\ldots,1)\}$.

4.4.2 The Data Structure

The Simplex Tree is an index structure that can be characterized as follows:

- each node is a simplex S defined by $D+1$ points;
- every inner node S has pointers to $D+1$ children S_h, which partition S and are pairwise disjoint, i.e. $S = \bigcup_h S_h$ and $S_{h_1} \cap S_{h_2} = \emptyset \ \ \forall h_1, h_2$;
- every leaf node stores for each of its $D+1$ points $\mathbf{s}_j$ the corresponding OQPs, $M_{opt}(\mathbf{s}_j)$;

[2] For simplicity, we use the same notation to denote both a simplex, i.e. an interval of $\Re^D$, and the set of its $D+1$ vertices.

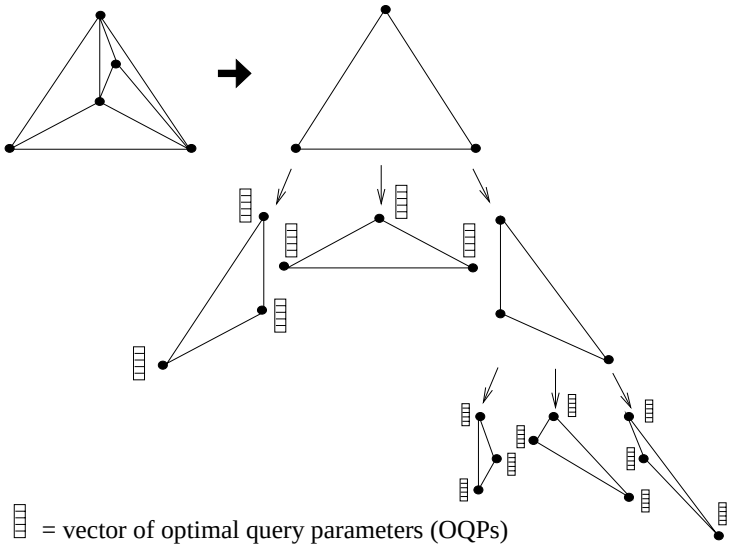


Figure 4.8: The structure of the Simplex Tree ($D = 2$).

- S_0, the root, covers the entire query domain $\mathcal{Q}$.

Figure 4.8 shows the Simplex Tree corresponding to a 2-dimensional triangulation.

The operations necessary to maintain the index are sketched in Figure 4.9. Below, the individual parts are discussed in more detail.

Lookups. Given a new query point, we need to determine in which simplex the new query point is contained. Starting with the root node we traverse the tree descending at each inner node into the child node which contains the new point. The case where the query point is not properly contained in one of the child simplices but it is an element of the delimiting hyperplanes of several simplices can be solved by considering such hyperplanes as simplices of dimension $D' < D$.

We do not re-organize the tree in case it gets unbalanced due to the distribution of the data. Hence, the depth of the tree is $O(n)$ in the worst case, n being the number of stored query points, and $O(\log_{D+1} n)$ in the best case. We will assess the average behaviour experimentally in Section 4.5.

Interpolation. To interpolate off the Simplex Tree, i.e. define a wavelet representation of the mapping, first observe that for each point $\mathbf{s}$ in the Simplex Tree the value of $M_{opt}(\mathbf{s}) = (m_1(\mathbf{s}), m_2(\mathbf{s}), \ldots, m_N(\mathbf{s}))$ is stored. Thus, given a query point $\mathbf{q}$ for which an approximation of $M_{opt}(\mathbf{q})$ is sought, we can solve the problems of approximating each of the N $m_i(\mathbf{q})$ values independently of each other.

```
 // initially called with the root simplex
Simplex &SimplexTree::Lookup(Simplex &S, Vector &q) {
   // when in leaf node, we know we found it
  if (S.IsLeaf()) return S;
   // otherwise check each child
  for (int h = 0; h < D + 1; h++)
    if (S.child[h]->Contains(q))
       // descend into h-th child
      return Lookup(S.Child[h],q);
}
Oqp &SimplexTree::Predict(Point &q) {
   // get the enclosing simplex
  Simplex &S = Lookup(q);
   // interpolate in point q using the points of S
  return Wavelet::Interpolate(S,q);
}
void SimplexTree::Insert(Point &q, Oqp &v) {
   // get enclosing simplex
  Simplex &S = Lookup(q);
   // get predicted values in this point
  Oqp &vPredict = Predict(q);
   // if predicted and actual OQPs differ
   // by more than 'epsilon' insert the point
  if (v.Difference(vPredict) > epsilon)
    for (int h = 0; h < D + 1; h++){
       // get the h-th corner of the simplex
      Vector &pCorner = GetCorner(h);
       // create a simplex using the points of this
       // simplex but exclude pCorner and add q instead
      Simplex &SSon = S.CreateSimplex(pCorner, q);
       // add the new simplex as child
      S.AddChild(SSon);}
}
```

Figure 4.9: Basic functionality of the Simplex Tree.

Using an unbalanced Haar-Wavelet for approximating $v_i = m_i(\mathbf{q})$ means to perform a linear interpolation in $\mathbf{q}$ given the values ${v_i}^{\mathbf{s}_j} = m_i(\mathbf{s}_j)$ of the $D+1$ points defining the simplex $S = \{\mathbf{s}_1, \ldots, \mathbf{s}_{D+1}\}$ enclosing $\mathbf{q}$. Since S is a D-dimensional linear subspace, solving for the unknown $\widehat{v_i}$ Equation 4.2 will yield the desired approximation of $v_i = m_i(\mathbf{q})$. Note that for a given data set, the complexity of interpolation is $O(1)$, since neither D nor P change.

$$\begin{vmatrix} q_1 - s_{1,1} & \ldots & q_D - s_{1,D} & \widehat{v_i} - {v_i}^{\mathbf{s}_1} \\ s_{2,1} - s_{1,1} & \ldots & s_{2,D} - s_{1,D} & {v_i}^{\mathbf{s}_2} - {v_i}^{\mathbf{s}_1} \\ \ldots & & \ldots & \ldots \\ s_{D+1,1} - s_{1,1} & \ldots & s_{D+1,D} - s_{1,D} & {v_i}^{\mathbf{s}_{D+1}} - {v_i}^{\mathbf{s}_1} \end{vmatrix} = 0 \tag{4.2}$$

Figure 4.10 shows the resulting approximation (by linear approximation) for a synthetic example. The approximation was done using the triangulation of Figure 4.6.

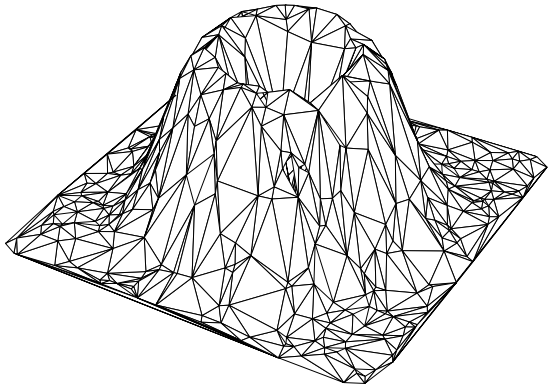


Figure 4.10: Example of approximation for $D = 2$.

Inserts. As opposed to typical spatial index structures, the Simplex Tree is not an index whose aim is to store points to be searched. Instead, it stores points to organize the feature space into simplices. As a consequence, not every point needs to be inserted, since it is sufficient to insert only those points that can improve the quality of the approximation in a *significant* way. These are the points for which

$$\max_i |m_i(\mathbf{q}) - \widehat{v}_i| > \epsilon$$

holds, for a given threshold ϵ. In other words, if all the predictions $\widehat{v}_i$'s are already almost equal to the corresponding $m_i(\mathbf{q})$'s there is no need to store $\mathbf{q}$ in the Simplex Tree. The particular choice of the threshold ϵ determines the quality of the approximation: For low thresholds the approximation is more accurate whereas high thresholds cause more slack. More important, however, is the character of the optimal query mapping. If M_{opt} is composed of low frequencies, only very few query points are stored, whereas for a query mapping composed of high frequencies, more query points are needed to reach approximations of suitable quality. As a limit case, when the OQPs always coincide with the default ones, no point at all is inserted in the Simplex Tree. Consequently, the resource requirements of the Simplex Tree *do not* depend on the number of queries for which feedback is provided but on the intrinsic complexity of the optimal query mapping and on the insert threshold.

Example 4.2

Figure 4.11 demonstrates the evolution of the approximation of a 2-dimensional parameter function. The individual plots show the approximation after 50, 250, 500, and 1000 incremental updates. The figures underline the two major advantages of our method:

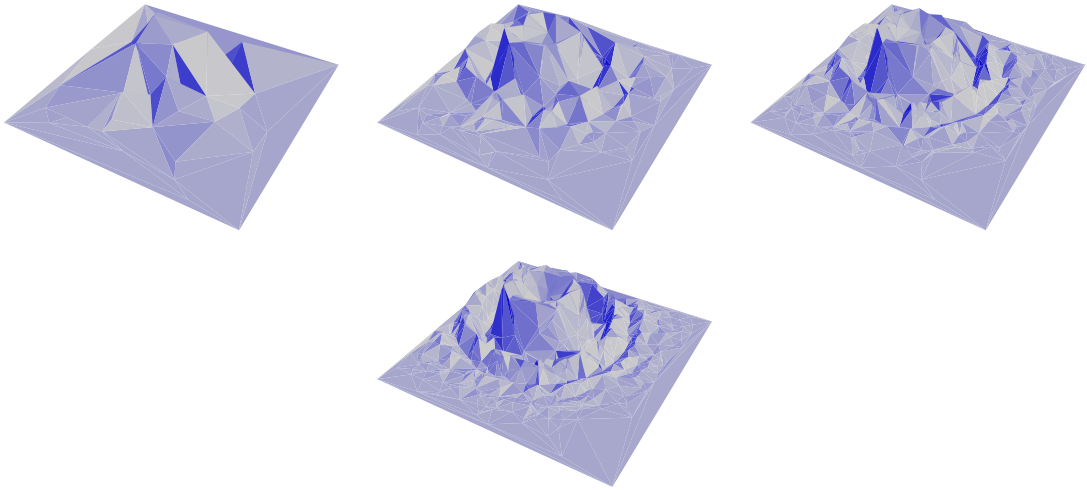


Figure 4.11: Approximation of a parameter function after incorporating 50, 250, 500, and 1000 feedback points.

Firstly, the approximation converges very quickly towards the actual parameter function. Secondly, the approximation is self-maintaining in the sense that only data points whose contribution is significant are included into the function,i.e. in areas where the parameter function has low frequency (foreground) only few points are added. Oversampling in the form of storing data points which do not contribute significantly to the shape of the function is automatically prevented.

4.5 Experimental Results

We have implemented FeedbackBypass in C++ under Linux, and tested its performance in order to answer the following basic questions:

- Which are the actual prediction capabilities of FeedbackBypass? How much feedback information does FeedbackBypass need to perform reasonably well? How long does it take to learn the optimal query mapping?
- How much do the predictions of FeedbackBypass depend on the specific data set? Alternatively, is FeedbackBypass robust to changes in the type of queries to be learned?
- How much do we gain, in terms of efficiency, by "skipping", or shortening, the feedback loop?

Figure 4.12: Sample images from the "Fish" category.

For evaluation purposes we used the IMSI data set consisting of about 10,000 color images [IMS]. Each image is already annotated with a *category* (such as "birds", "monuments", etc.). From each image, represented in the HSV color space, we extracted a 32-bins color histogram, by dividing the hue channel H into 8 ranges and the saturation channel S into 4 ranges.[3] To compare histograms we use the class of weighted Euclidean distances, with the (unweighted) Euclidean distance being the default function. We implemented both query point movement and re-weighting feedback strategies, as described in Section 2.3.1, which means that M_{opt} is a function from $\Re^{31}$ to $\Re^{62}$ (see also Example 4.1).

The setup for the experiments was as follows. From the whole data set we selected 2,491 images belonging to 7 categories: Bird (318 images), Fish (129), Mammal (834), Blossom (189), TreeLeaf (575), Bridge (148), and Monument (298). This subset of images was then used to randomly sample queries, whereas images in other classes were just used to add further noise to the retrieval process. For each query image, any image in the same category was considered a "good" match whereas all other images were considered "bad" matches, *regardless of their color similarity*. This leads to hard conceptual queries, which however well represent what users might want to ask to an image retrieval system. Since, within each category, images largely differ as to color content, any query based on a color distance cannot be expected to find more than a fraction of relevant images to be close in color space. For instance, all the 4 images shown in Figure 4.12 belong to the "Fish" category: Only the 2nd image ("shark") has a dominant blue color, whereas others have strong components of yellow, gray, and orange, respectively. A similar evaluation procedure was also adopted in [RH00].

To measure the effectiveness of FeedbackBypass we consider classical *precision* and *recall* metrics [Sal89], averaged over the set of processed queries. For a given number k of retrieved objects, precision (Pr) is the number of retrieved relevant objects over k, and recall (Re) is the number of retrieved relevant objects over the total number of relevant objects (in our case, the number of images in the category of the query). The formal

[3] See also `http://kdd.ics.uci.edu/databases/CorelFeatures/CorelFeatures.data.html`.

definition is:

$$Pr = \frac{|relevant \bigcap retrieved|}{|retrieved|} \tag{4.3}$$

$$Re = \frac{|relevant \bigcap retrieved|}{|relevant|} \tag{4.4}$$

where$|E|$ represents, in this context, the cardinality of the set E. A typical example of precision-recall graph is shown in Figure 4.14 (c).

In our experiments we used a typical value of $k = 50$ and, in any case, k never exceeded 80. This is because we consider that a real user will hardly provide feedback information for larger result sets. As a consequence, since the number of retrieved good matches is limited above by k (and in practice stays well below the k limit), the use of distance functions more complex than weighted Euclidean, such as Mahalanobis, was not considered. Indeed, as observed in [RH00], improvement due to feedback information is possible only when the number of good matches is not much less than the number of parameters of the distance function to be learned, which is 31 in our case but would be $31 \times 32/2 = 496$ for the Mahalanobis distance.

The results we show refer to three different scenarios:

- Default: This is the strategy currently used by *all* interactive retrieval systems, which starts the search by using the user query point and the default distance function (i.e. the Euclidean one in our case);

- FeedbackBypass, for which precision and recall*always* refer to "new" (i.e. never seen before) queries for which the optimal query point and the optimal distance function, as predicted by the FeedbackBypass module, are used in place of the user query and the default Euclidean distance;

- AlreadySeen: This is mainly used for reference purpose, and corresponds to the case where the FeedbackBypass module delivers predictions for already seen queries, for which the predicted parameters indeed coincide with the optimal ones. It can be argued that the more the results from FeedbackBypass and AlreadySeen are similar, the more FeedbackBypass is approaching the intrinsic limit established by the use of a given class of distance functions and of specific relevance feedback strategies.

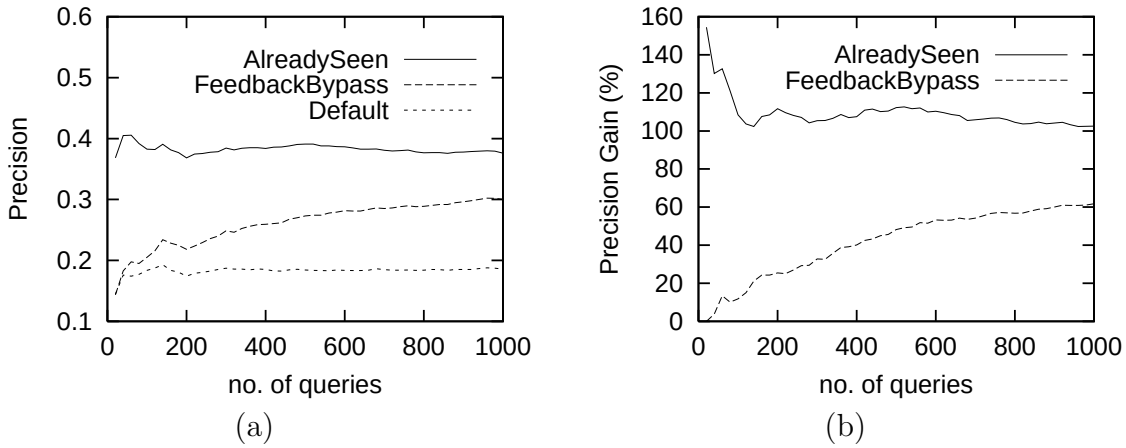
For each query, after measuring precision and recall for the first round of k results, we automatically run (using the category information associated to each image) the feedback loop until it converges to a stable situation, i.e. when no changes are observed anymore in the result list. The corresponding query parameters are then sent to FeedbackBypass for insertion.

4.5.1 Speed of Learning

Figure 4.13 (a) shows average precision as a function of the number of processed queries. For this graph the number of retrieved objects was set to $k = 50$. It is evident that the performance of FeedbackBypass monotonically increases with the number of queries, whereas that of Default and AlreadySeen do not depend on the number of executed queries, and that the difference between FeedbackBypass and the Default strategy is already significant after the first few hundred queries. This is also emphasized in Figure 4.13 (b), where we show values of the *precision gain*, PrGain, defined as:

$$\text{PrGain}(\textsf{FeedbackBypass}) = \left(\frac{\text{Pr}(\textsf{FeedbackBypass})}{\text{Pr}(\textsf{Default})} - 1 \right) \times 100$$

and similarly for the AlreadySeen case. The number of good matches doubles for already seen queries, and increase by 60% for queries never seen before.



(a) (b)

Figure 4.13: Precision results: (a) absolute values; (b) gains with respect to the DEFAULT strategy.

Figures 4.14 (a), (b), and (c) show, respectively, the values of average precision, recall, and precision vs. recall after 1000 queries, when k varies between 10 and 80. The graphs confirm that our method is able to provide accurate predictions even when the number of retrieved objects per query, k, is low. This can also be appreciated in Figures 4.15 (a) and (b), where precision and recall curves for $k = 20, 50$, and 80 are plotted versus the number of queries.

In the previous experiments we have considered the same value of k both to train the system and to evaluate it. However, it is also important to understand if training FeedbackBypass with larger values of k can be better than training FeedbackBypass with

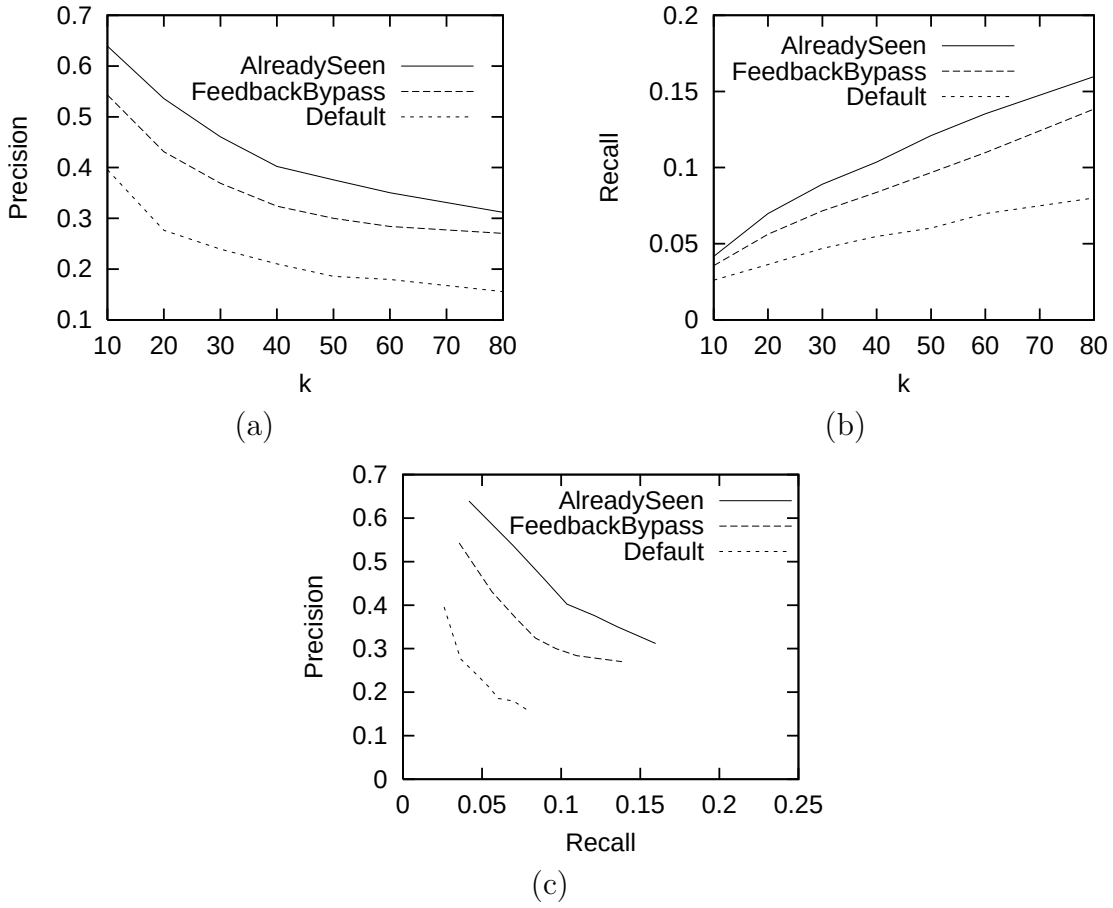
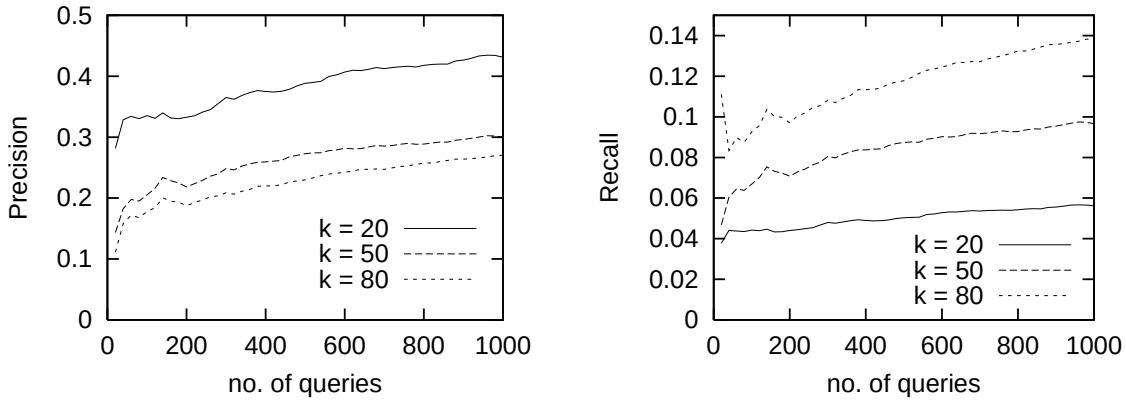


Figure 4.14: Precision (a), recall (b), and precision vs. recall curves (c) after 1000 queries.

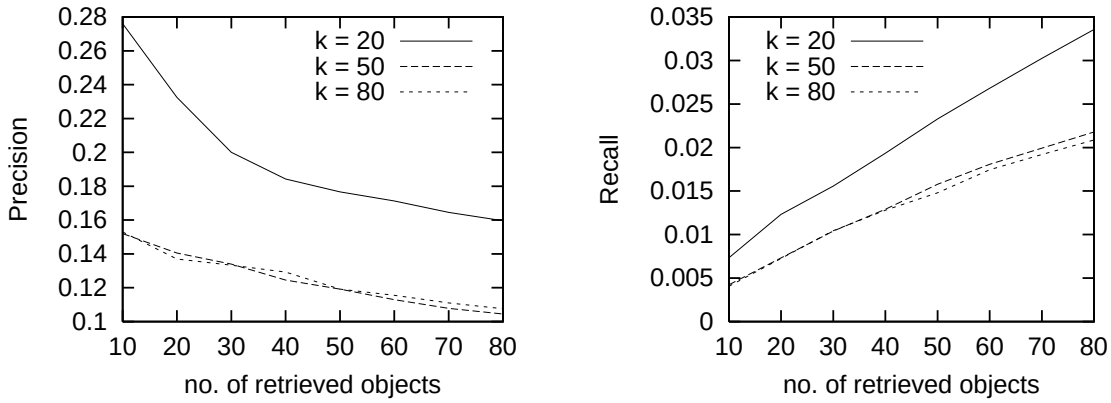
less information. Clearly, precision results shown in Figure 4.15 (a) are of little use to this purpose, since they are obtained with a different number of retrieved objects for each curve. Thus, we have compared several versions of FeedbackBypass, each trained with a specific k value, when they are used to answer queries requesting the same number of objects from each version. The basic conclusion that can be drawn from the results shown in Figure 4.16 is that, even if larger values of k (e.g. $k = 80$) allow to increase the retrieval effectiveness (see values of recall in Figure 4.15 (b)), using smaller k values in the training phase can be beneficial. From Figure 4.16 (b) we see that the graph for $k = 20$ starts from a smaller recall value but, when considering more objects, grows faster than the other lines, thus reaching higher results. Probably this depend on the fact that similar images are in first positions; thus, in this case, FeedbackBypass learns without noise. We have to remember that, like stressed in Figure 4.12, it can happen that in the same image category there are images that are completely different from a color point of view with



(a) (b)

Figure 4.15: Precision (a) and recall (b) of FeedbackBypass for different values of k.

respect to the query.



(a) (b)

Figure 4.16: Precision (a) and recall (b) of FeedbackBypass for different values of k as a function of the number of retrieved objects.

4.5.2 Robustness

We now turn to consider how much the performance of FeedbackBypass depends on the specific queries for which predictions are required. For this experiment we separately measured precision for the 7 query categories. Looking at precision results (see Figure 4.17 (a)) it can be observed that FeedbackBypass is able to provide useful predictions in all cases where there is a significant difference between the Default and the AlreadySeen

cases. Indeed, such a difference is a clear indication that feedback information actually leads to improve the results. This is particularly evident for the largest query category ("Mammal"). On the other hand, when feedback only slightly improves the quality of the results (see the "TreeLeaf" category, denoted simply as "Leaf" in the figure), predictions for new queries do not provide benefits, as it could have been expected. This general behavior is only violated for the "Fish" category, where it seems that no improvement can be obtained from FeedbackBypass on new queries, even if performance of AlreadySeen is particularly good. However, since "Fish" is the smallest category (129 images), it can be argued that the number of sampled queries is still not enough to well approximate the optimal query mapping for that category. Similar results are observed in Figure 4.17 (b) for the recall metric.

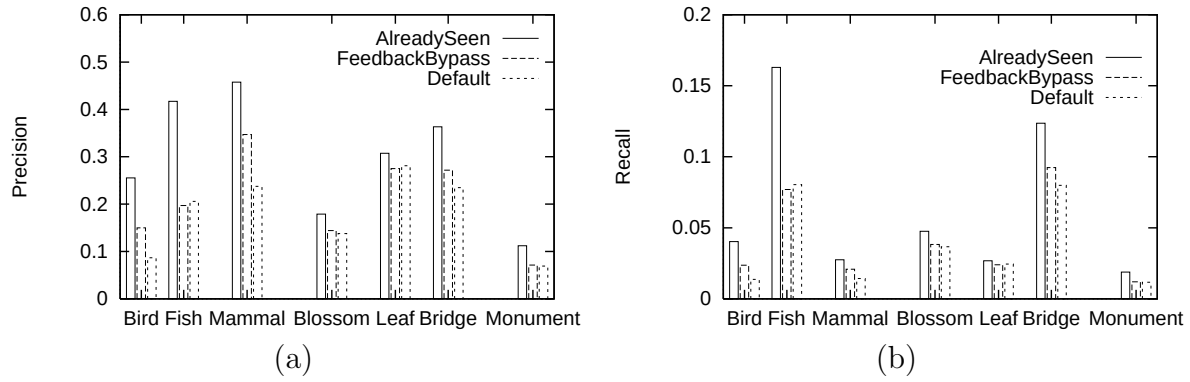


Figure 4.17: Precision (a) and recall (b) for the 7 query categories.

4.5.3 Efficiency

An important aspect that we analyze here is how much we can gain by using FeedbackBypass in terms of *efficiency*. Clearly, the overall performance of an interactive retrieval system will also depend on the specific access methods that are used to retrieve the stored objects, as well as by the indexed features. In order to provide unbiased results, we consider the following performance metrics:

- The average number of feedback iterations that FeedbackBypass saves with respect to the Default strategy, in order to obtain the same level of precision. Thus, for each query we start the feedback loop either from default or from predicted query parameters, and measure how many iterations are needed before no further improvements are possible. This "Saved-Cycles" measure tells us how many query requests to the underlying system we save, on the average, for each user query.

- The average number of objects that we *do not* have to retrieve for achieving the same level of precision than Default. Note that this "Saved-Objects" metric is simply computed as: Saved-Objects = Saved-Cycles $\times$ k

Figure 4.18 presents results for $k = 20, 50, 80$. In both cases it can be seen that the savings improve over time, and that after 1000 queries they amount to about 2 cycles for $k = 50$, which translates in a net reduction of 100 objects retrieved from the underlying system.

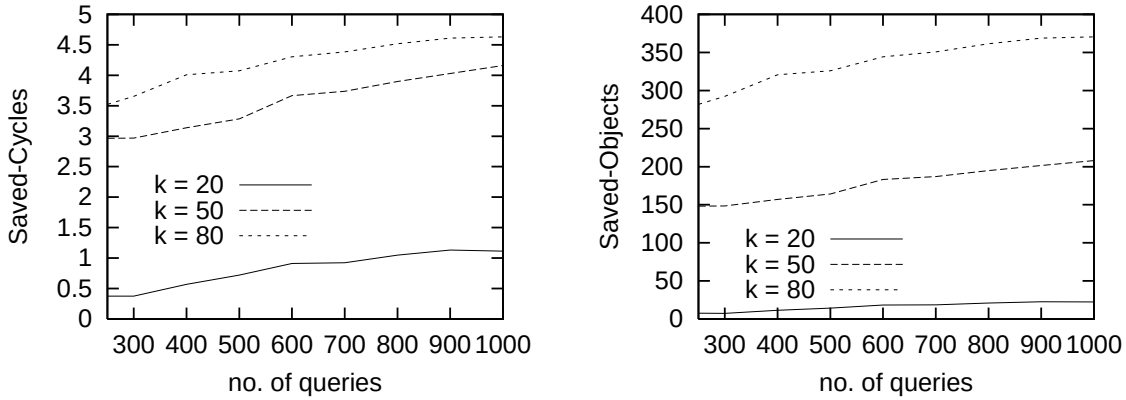


Figure 4.18: Average number of feedback cycles (a) and retrieved objects (b) saved by FeedbackBypass.

Finally, in the last experiment we assess the Simplex Tree as such. Figure 4.19 shows the average number of simplices traversed to reach a leaf node, together with the depth of the tree, i.e. the maximum number of simplices that could be traversed. Both are logarithmically increasing, however, the average number of traversed simplices is significantly lower than the depth of the Simplex Tree, which leads to fast predictions of the optimal query parameters and underlines the efficiency of FeedbackBypass.

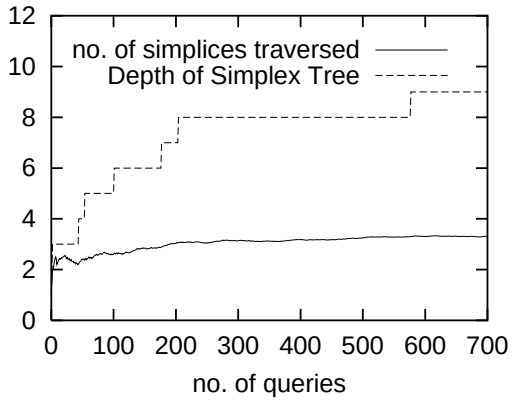


Figure 4.19: Average number of simplices traversed per query and depth of the Simplex Tree.

Chapter 5

Windsurf Extensions

In this Chapter we describe two extensions for the WINDSURF system that allow to further improve the effectiveness of its results.

The first one is a relevance feedback model for region-based image retrieval systems, that we studied for WINDSURF, but that can be applied to every other system that fragments images into regions. It is important to note that, at present time, *no relevance feedback model has been proposed for region-based image retrieval systems.* Let us observe that, starting from what we assumed in Section 3.2, supporting feedback facilities for a CBIR system that fragment images into regions means to apply the $\mathcal{A}_0^{WS^*}$ algorithm N times (N being the number of feedback iterations). Even if $\mathcal{A}_0^{WS^*}$ was proved to be optimal, if repeatedly applied it can lead to very time-consuming feedback loops. Therefore, in this light, FeedbackBypass represents an efficient solution because it is able to cut down the number of iterations, complementing the role of relevance feedback engines.

The second one consists in a new feature that, even if in the original version of the WINDSURF system is implicity extracted, it is not used in the retrieval phase: The shape of the regions. In detail, we describe an approach based on the use of the Discrete Fourier Transform that is scale-, translation-, and rotation-invariant and that, using both the magnitude and the phase information of Fourier coefficients, is able to improve results' effectiveness as compared to those obtained from similar approaches. At this point, it is important to say that the so-extended WINDSURF system is able to perform the search using either color/texture or shape. The user is given the opportunity to decide which feature is more appropriate to his/her needs. Anyway, integration of all features into a single similarity model can be easily obtained, e.g. by using a weighted distance with weights reflecting the relative importance of each feature, much as is currently done in other systems [SC96, MM99, CTB+99].

5.1 A Relevance Feedback Model for Windsurf

From the brief survey of relevance feedback presented in Section 2.3, it is clear that, at present time, no relevance feedback model exists for region-based image retrieval systems. Starting from the state-of-the-art and moving in direction of current research in the area of content-based image retrieval, we present the formalization of the relevance feedback model supported by WINDSURF (see Chapter 3).[1]

Before starting the formalization of the relevance feedback model, we need to point out the main difficulties that we have to deal with in defining the following three separate problems:

Problem 5.1 (Regions Re-weighting Problem)
Given an initial image query $Q = (I_q, k)$ and its result list $Result(Q, d) = \{I_{s_1}, I_{s_2}, ..., I_{s_k}\}$ (obtained using the distance function d) and given the set of relevance scores $Score(Q, d) = \{Score(I_{s_1}), Score(I_{s_2}), ..., Score(I_{s_k})\}$ assigned by the user to each object of $Result(Q, d)$, "learn" the best way to partition each score $Score(I_{s_i})$ among regions belonging to image I_{s_i}.

Problem 5.2 (Features Re-weighting Problem)
Given a solution for Problem 5.1, "learn" the best way to partition the region score among the features extracted from each region $R_{s,j}$.

Problem 5.3 (Query Point Movement and Re-weighting Problem)
Given a solution for Problem 5.2, "learn" the best query mapping which associates to each query point I_q the optimal query parameters for d using the user judgments.

At this point, it is important to remind some concepts, given in Section 3.2.1, concerning the computation of the similarity score between two images I_q and I_s and between each couple of respective regions $R_{q,i}$ and $R_{s,j}$. To this end, in Figure 5.1 we report the different levels at which the WINDSURF system works. In detail, starting from the bottom, the matching among query regions and regions of each data set image establishes which is the similarity score for each pair for matched regions. By way of a scoring function, the optimal match is computed; then, the image score is obtained using the similarity values for pairs of the best matched regions.

Here, we report Equation 3.9 representing the distance used by WINDSURF to compare

[1] Results of the proposed model are not presented in this thesis because we are still completing the implementation phase.

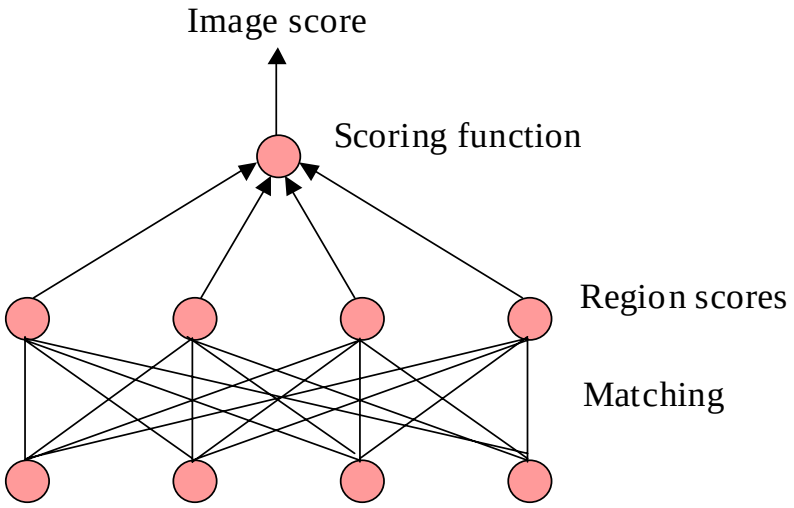


Figure 5.1: Levels of similarity score in WINDSURF.

two regions:

$$d(R_{q,i}, R_{s,j})^2 = \sum_{B \in \mathcal{B}} \gamma_B d_B(R_{q,i}, R_{s,j})^2 + \left(\frac{2}{\frac{size(R_{q,i})}{size(I_q)} + \frac{size(R_{s,j})}{size(I_s)}} \right) \left(\frac{size(R_{q,i})}{size(I_q)} - \frac{size(R_{s,j})}{size(I_s)} \right)^2 \tag{5.1}$$

where the distance $d_B(R_{q,i}, R_{s,j})$ between two regions on the frequency sub-band B is computed by using the *Bhattacharyya* metric (Equation 3.10). The image similarity is then defined as the average similarity between pairs of matched regions (Equation 3.19)

$$I_{sim}(I_q, I_s) = \frac{1}{n_q} \sum_{i=1}^{n_q} h(d(R_{q,i}, \Gamma_s^{opt}(R_{q,i}))) \tag{5.2}$$

where h is the correspondence function mapping distance values into similarity scores, and $\Gamma_s^{opt}(R_{q,i})$ represents the optimal region matching for $R_{q,i}$.

It is intuitive that, while Problem 5.1 is related to the image similarity concept and how this is computed, Problems 5.2 and 5.3 refer to the the region similarity computation. Since the image similarity distance is computed as a simple average, it is clear that the only needed modification is a transformation of Equation 5.2 into a weighted average, with weights given by the solution to Problem 5.1.

5.1.1 Regions Re-Weighting

To solve Problem 5.1 we need to rewrite Equation 5.2 as:

$$I_{sim}(I_q, I_s) = \frac{1}{n_q} \sum_{i=1}^{n_q} \alpha_{R_{q,i}} h(d(R_{q,i}, \Gamma_s^{opt}(R_{q,i}))) \tag{5.3}$$

where each $\alpha_{R_{q,i}}$ represents the weight associated to each region $R_{q,i}$ of the image I_q and its default value is 1. The computation of $\alpha_{R_{q,i}}$ at each cycle of the relevance feedback loop comes from the following observation: Let us suppose, for simplicity, that the number of regions for each image is two, $n_q = 2$. The basic idea of the proposed solution, shown in Figure 5.2, is to compare the result list of the query image, $Result(I_q, k)$, with the two lists $Result(R_{q,1}, k')$ and $Result(R_{q,2}, k')$ obtained using each single region of the query image as query.[2] Note that in the specific example of Figure 5.2, we use $k = 5$ and that $k' \geq k$.

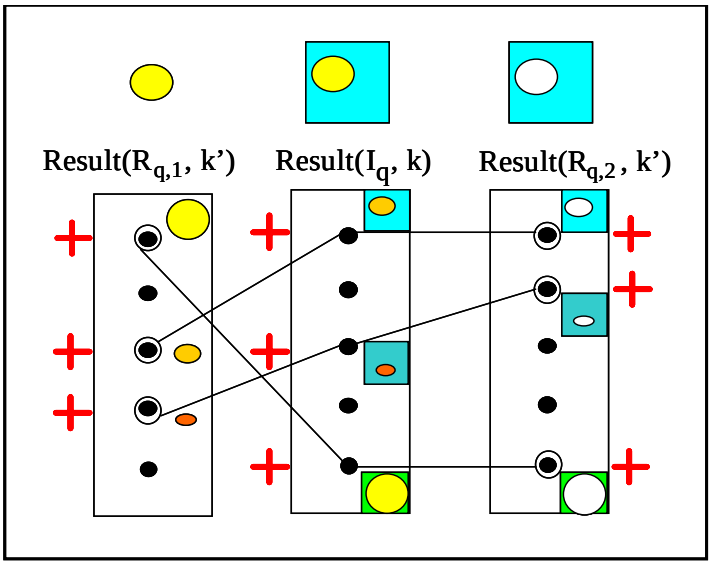


Figure 5.2: Comparison between an image result list and its regions result lists.

In detail, starting from the images of the list $Result(I_q, k)$ (in the center of Figure 5.2), we classify them as "good" (indicated with a cross) and "bad" (without cross) by way of the user judgments (we consider the simple case of binary relevance feedback). We then indicate the cardinality of the set of images with positive feedback (i.e. with the cross) with the variable g. Using only the g good images, we are able to find a good weight for each of the two region $R_{q,1}$ and $R_{q,2}$ looking for the same images in the lists $Result(R_{q,1}, k')$ (the left one in Figure 5.2) and $Result(R_{q,2}, k')$ (the right one in Figure 5.2) and computing the sum of the relative similarity values I_{sim}:

$$\alpha_{R_{q,i}} = \sum_{l=1}^{g} I_{sim}(I_{s,l}) \qquad \forall i = 1, 2 \tag{5.4}$$

[2]For simplicity of readability, in Figure 5.2 the dependance of $Result(I_q, k, d)$ on the distance function d is understood in the notation.

Since, intuitively, the image similarity score is obtained as the sum of similarity scores obtained for its regions, that of computing the weights as a sum seems to be a reasonable, and simple, choice. Note that weights are then normalized to obtain $\sum_{i=1}^{2} \alpha_{R_{q,i}} = 1$.

The above approach can be immediately generalized to the case of $n_q > 2$ regions.

5.1.2 Features Re-Weighting

To solve Problem 5.2 we need to rewrite Equation 3.9 as:

$$d(R_{q,i}, R_{s,j})^2 = \alpha_{CT} \sum_{B \in \mathcal{B}} \gamma_B d_B(R_{q,i}, R_{s,j})^2 + \\ + \alpha_S \left(\frac{2}{\frac{size(R_{q,i})}{size(I_q)} + \frac{size(R_{s,j})}{size(I_s)}} \right) \left(\frac{size(R_{q,i})}{size(I_q)} - \frac{size(R_{s,j})}{size(I_s)} \right)^2 \quad (5.5)$$

where α_{CT} represents the weight for the color and texture features and α_S the weight for the size (the default value for both weights is 1).

We start using the same logic introduced in the previous Section, but now we apply the idea to a single region, instead of an image, and to its correspondent features (Figure 5.3).[3] We note that, at the first search cycle, the distance values of the g images from the region list $Result(R_{q,1}, d)$ (in center of Figure 5.3), corresponding to those g images to which the user has given positive feedback on the image result list, are obtained using Equation 5.5 with default weight values for α_{CT} and α_S.

A good way to have the new weights for the next search cycle is to compute the distance variance σ_{CT}^2 and σ_S^2 on the distance d_{CT}, representing the measure on color and texture (derived from Equation 5.5 by setting $\alpha_S = 0$), and on d_S, the measure of region size (obtained by setting $\alpha_{CT} = 0$), respectively. Each final weight value is then set equal to the inverse of the correspondent variance:

$$\alpha_{CT} = \frac{1}{\sigma_{CT}^2} \qquad \alpha_S = \frac{1}{\sigma_S^2} \quad (5.6)$$

The rationale for this choice is that a feature is more important than the other one if the variance of its distances, computed on the specific feature domain, is smaller, since if a feature has a high distance variance, this means that we are not interested in that feature. Again, the approach can be easily extended to the case where more than two features are present.

[3] Again, for simplicity of readability, in Figure 5.3 the dependance of $Result(R_{q,1}, k, d)$, $Result(R_{q,1}, k', d_{CT})$, and $Result(R_{q,1}, k', d_S)$, on the values of k and of $k' \geq k$ is understood in the notation.

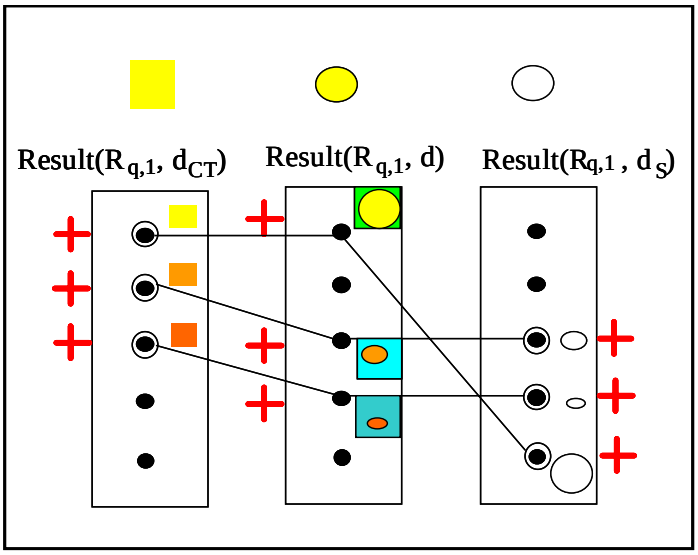


Figure 5.3: Comparison between a region result list and its features result lists.

5.1.3 Query Point Movement and Re-Weighting

The last problem to solve (Problem 5.3) is related (1) to the idea to try to move the query point towards the "good" matches, as evaluated by the user, as well as to move it far from the "bad" results points (Query point movement), and (2) to the observation that user feedback can help identify feature components that are more important than others in determining whether a result is good or bad for the user. Consequently, such components should be given a higher relevance (Re-weighting).

In both cases, the distance function which we have to deal with is the Bhattacharyya metric of Equation 3.10, here reported by putting the attention on its second term only, for simplicity. The term corresponds to the Mahalanobis distance on regions centroid $\mu^B_{R_{q,i}}$ and $\mu^B_{R_{s,j}}$ and uses, as covariance matrix, the average of the regions covariance matrix:

$$d_B(R_{s,j}, R_{q,i})^2 = \left(\mu^B_{R_{s,j}} - \mu^B_{R_{q,i}}\right)^T \times \left(\frac{\mathcal{C}^{3;B}_{R_{s,j}} + \mathcal{C}^{3;B}_{R_{q,i}}}{2}\right)^{-1} \times \left(\mu^B_{R_{s,j}} - \mu^B_{R_{q,i}}\right)$$

Note that the above distance works on regions centroid and not on the complete region feature vectors. So, starting from the assumption that the user has an "ideal" query region in mind (that we represent by way of its centroid, μ') and that distance between regions centroid and this ideal point is a generalized ellipsoid distance, following [ISF98], we formalize the problem as finding μ' and W that minimize the penalty function

$$\min_{W,\mu'} \sum_{l=1}^{g} Score(R_{s_l})(\mu_{R_{s_l}} - \mu')^T W(\mu_{R_{s_l}} - \mu') \tag{5.7}$$

subject to the constraint $det(W) = 1$, to discard the zero matrix from the result.

The new query centroid μ' should be computed such that "positive" regions, i.e. regions having higher similarity scores, are closer to it than other regions. The minimization problem can be solved by way of the *Lagrange multiplier*. In detail, we found that, since the used distance is a quadratic one, we can use positive feedback scores to determine the "optimal" region centroid μ' by way of a weighted average of good results, i.e.:

$$\mu' = \frac{\sum_{l=1}^{g} Score(R_{s_l}) \times \mu_{R_{s_l}}}{\sum_{l=1}^{g} Score(R_{s_l})} \tag{5.8}$$

where R_{s_l} is a region belonging to an image to which a positive feedback has been given, and $\mu_{R_{s_l}}$ is its correspondent centroid. The score of each region, $Score(R_{s_i})$, is defined as the contribution percentage of the region to the overall image similarity.

On the other hand, remembering again that our distance works on regions centroid, we note that the optimal solution presented in [ISF98] is too lossy for our case, because the computation of the W matrix should depend on the regions centroid features only. We found better solution in taking, as the new W, the average of the covariance matrices of all regions with positive feedback, i.e.:

$$W = \frac{1}{g} \sum_{l=1}^{g} \mathcal{C}^{3;B}_{R_{s_l}} = \mathcal{F}_g(\mathcal{C}^{3;B}_{R_{q,i}}) \tag{5.9}$$

Rewriting Equation 3.10 we finally obtained:

$$d_B(R_{s,j}, R_{q,i})^2 = \left(\mu^B_{R_{s,j}} - \mu'\right)^T \times \left(\frac{\mathcal{F}_g(\mathcal{C}^{3;B}_{R_{q,i}}) + \mathcal{C}^{3;B}_{R_{s,j}}}{2}\right)^{-1} \times \left(\mu^B_{R_{s,j}} - \mu'\right) \tag{5.10}$$

5.2 Windsurf with Shape

The WINDSURF system, as described in Chapter 3, only works on color and texture features. Actually, depending on the fragmentation of the images into regions, WINDSURF extracts implicitly also a region shape information that is however not used in the retrieval phase. Starting from this observation and from the fact that combining more features can improve the effectiveness of results, we have decided to enrich our system with a new region feature: The shape. In detail, the shape property can be used together with the color and the texture information, in a way transparent to the user, to increase the goodness of results, or in an interactive way, to allow the formulation of shape-based queries selecting the object of interest directly from the image.

Note that, when designing a shape retrieval technique we have to pay attention to two primary issues:

Shape representation. How can an object shape be represented in terms of its shape properties? What invariance properties should the representation satisfy?

Similarity measure. Given a representation of the shape, how two shapes should be compared (i.e. which is the distance to be used)?

As for the invariance properties, we believe that good shape descriptors should allow translation- and size-invariance, since these are information not directly related to the shape of an object, and that the user should be given the opportunity to decide whether rotation-invariance has to be taken into account or not.

5.2.1 Shape Representation

Among the state-of-the-art of shape representation techniques (see Section 2.1.1), we chose the feature-vector modality and the relative parametric external method, working in the frequency domain. The choice is driven by WINDSURF needs, as the necessity of indexing, the robustness again the noise, and the possibility of satisfy scale-, translation-, and rotation-invariance properties [Del99].

In particular, each WINDSURF region (R_i) groups together the image pixels that are similar for color and texture properties. The global properties that characterize the set of regions belonging each image I are:

1. $\bigcup_i R_i = I$, i.e. the union corresponds to the global image.
2. $\bigcap_i R_i = 0$, i.e. the intersection is empty (regions share no areas).
3. Regions are not connected.

Because of the last point, we apply an algorithm to connect the regions extracted by WINDSURF and put together regions that, taken alone, are too small to be meaningful. The goal is to find regions approximately representing objects present in the image. Then, using the *EdgeOperator* algorithm [GW92], we extract the *boundary* from each object. Finally, following the *Tremaux* algorithm [Ore62], we find boundary points (also called *interest points*) from which we extract the M points with higher curvature representing the vertices of a polygon that approximates the shape of the objects.

Figure 5.4 shows a complete example of the steps needed from WINDSURF to extract the shape information: Starting from the original image on the left, WINDSURF first segments the image by using the the k-means algorithm; then, it connects the regions extracting from each of them the corresponding boundary. Finally, interest points are found. For simplicity, only the interest points related to the region "bird" are reported.

Points representing the vertices of the polygon that approximate the bird are the M most important points, i.e. the M points having the highest curvature.

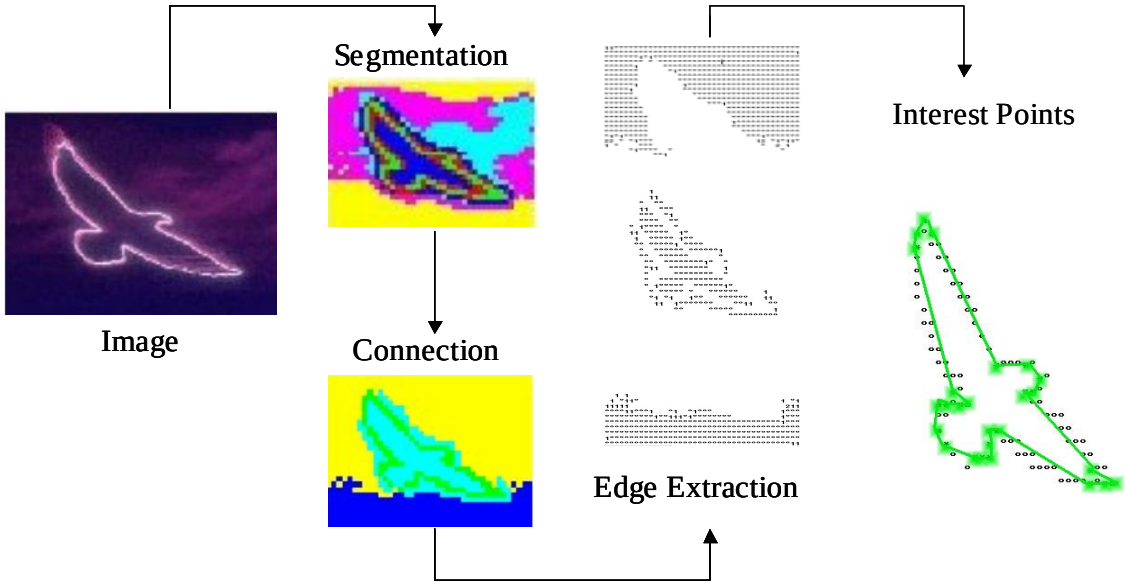


Figure 5.4: Example of shape extraction.

We re-sample the interest points to M samples to ensure that the resulting shape features of all images have the same length. In our experiments, we choose $M = 32$. The set of the M vertices of the polygon can be seen as a discrete signal in the 2-dimensional space. We represent such signal in the complex space and map it to the frequency domain by way of the Discrete Fourier Transform. In detail, we use the *Fast Fourier Transform* (FFT) algorithm [PTVF92] that, if the condition $M = 2^n$ is satisfied, is able to carry the transformation in a very efficient way (i.e. with a complexity of $M \log_2 M$, instead of M^2 [II86]).

Let $z(l) = x(l) + jy(l)$, $l = 0, ..., M-1$, be the complex boundary sequence in the space domain. The boundary can be expressed in the frequency domain applying the Discrete Fourier Transform:

$$Z(m) = \sum_{l=0}^{M-1} z(l) e^{-j\frac{2\pi lm}{M}} = r(m) e^{j\theta(m)} \tag{5.11}$$

Note that $r(m)$ represents the magnitude and $\theta(m)$ the phase of the coefficients.

Each shape descriptor can thus be viewed as a vector of 32 complex coefficients related to the frequencies

$$f_m = \frac{m}{2M\Delta} \tag{5.12}$$

with m in the range $[0, \ldots, M-1]$, and Δ being the sampling interval. To represent such shape information, we use a vector of 64 elements (32 elements for the real parts of the coefficients, and 32 elements for the imaginary part).

At this point, it is important to note that, working in the frequency domain, with sample modifications to the extracted Fourier coefficients, it is possible to ensure the invariance properties.

Scale Invariance

The scale invariance allows to retrieve images from the dataset having similar shape but different size. To ensure this, let $z(l) = r(l)e^{j\theta(l)}$ be the points of a boundary object (denoting with $r(l)$ the module and with $\theta(l)$ the argument of each complex number), and let $z'(l)$ be the points for the same object, but scaled of a factor of β. Being β a scalar value, the previous transformation regards only the modules of the coefficients and not their arguments. Therefore, we can write:

$$z'(l) = \beta r(l)e^{j\theta(l)} = \beta z(l) \tag{5.13}$$

Passing to the frequency domain, the relation between Fourier transforms is:

$$Z'(m) = \sum_{l=0}^{M-1} z'(l)e^{-j\frac{2\pi lm}{M}} = \sum_{l=0}^{M-1} \beta z(l)e^{-j\frac{2\pi lm}{M}} = \beta Z(m) \tag{5.14}$$

Thus, to obtain scale invariance (i.e. to obtain the same coefficients for the two objects) we just need to divide the module of all coefficients for the first module $\neq 0$, maintaining the same argument values.

Translation Invariance

The translation invariance allows to retrieve images having similar shape but different space location. To this end, we need to make an opportune update to the coefficients obtained from the previous step. Again, consider the object boundary $z(l)$ and the boundary $z'(l)$ of the same object spatially translated by $\overline{z}$. Thus, it is:

$$z'(l) = z(l) + \overline{z} \tag{5.15}$$

Switching to the frequency domain, we have:

$$\begin{aligned} Z'(m) = \sum_{l=0}^{M-1} z'(l)e^{-j\frac{2\pi lm}{M}} = \sum_{l=0}^{M-1} (z(l) + \overline{z})e^{-j\frac{2\pi lm}{M}} = \sum_{l=0}^{M-1} z(l)e^{-j\frac{2\pi lm}{M}} + \\ + \sum_{l=0}^{M-1} \overline{z}e^{-j\frac{2\pi lm}{M}} = Z(m) + \overline{z}\sum_{l=0}^{M-1} e^{-j\frac{2\pi lm}{M}} = Z(m) + \overline{z}\chi_0(m) \end{aligned} \tag{5.16}$$

where $\chi_0(m) = 0$, $\forall m, m \neq 0$, and $\chi_0(0) = 1$.

We can conclude that the translation introduces variation only on the coefficient with zero frequency (DC). This means that, to obtain translation invariance, we need to discard the module and the argument related to the DC coefficient.

Rotation Invariance

Suppose that we want to compare an object to another one with the same shape but different orientation. Again, we need to update the coefficients obtained from both the two previous steps. In detail, consider the object boundary $z(l) = r(l)e^{j\theta(l)}$ and the boundary $z'(l)$ of the same object rotated by θ', it is:

$$z'(l) = r(l)e^{j(\theta(l)+\theta')} = r(l)e^{j\theta(l)}e^{j\theta'} = z(l)e^{j\theta'} \tag{5.17}$$

Switching to the frequency domain, we obtain:

$$Z'(m) = \sum_{l=0}^{M-1} z'(l)e^{-j\frac{2\pi lm}{M}} = \sum_{l=0}^{M-1} z(l)e^{j\theta'}e^{-j\frac{2\pi lm}{M}} = Z(m)e^{j\theta'} \tag{5.18}$$

It is clear that object rotation changes only the argument of the coefficients (the modules remaining untouched). Thus, to obtain rotation invariance, we can just subtract to all the arguments the first argument $\neq 0$.

In conclusion, we have demonstrated that it is possible to represent the shape of an object using both the amplitude and the phase information, by opportunely modifying Fourier coefficients to preserve scale-, translation-, and rotation-invariance.[4]

5.2.2 Similarity Measure

Like said in Section 3.2.2, to determine the grade of similarity between two images I_q and I_s, WINDSURF has first to compute the similarity score between each possible pair of regions $(R_{q,i}, R_{s,j})$. Then, by way of the overall scoring function, the optimal matching between all regions is determined, producing an image-level similarity value. Working with shape, WINDSURF follows the same main steps, but modifies the distance measures. In particular, to compute the similarity between a pair of shape objects (each of them represented by a feature vector of M complex modified Fourier coefficients), WINDSURF uses a simple Euclidean distance.

Let $Z(m)$ the Fourier coefficients of a boundary $z(l)$, and $Z'(m)$ be the Fourier coefficients of another boundary $z'(l)$. Moreover, let $Z_d(m)$ be the difference vector between

[4]We solve also the starting point invariance problem by fixing, from the beginning, the point from which start all the *Tremaux*-based interest points extraction.

$Z(m)$ and $Z'(m)$, $Z_d(m) = Z(m) - Z'(m)$, $m = 0, \ldots, M-1$. The Euclidean distance between $Z(m)$ and $Z'(m)$ is defined as:

$$\|Z(m) - Z'(m)\|_2 = \sqrt{\sum_{m=0}^{M-1} |Z_d(m)|^2} \tag{5.19}$$

where $|Z_d(m)|$ represents the module of the complex number $Z_d(m)$.

To prove that the Euclidean distance preserves the invariance properties described in the previous Section, we need to demonstrate that, if $z'(l)$ is obtained from $z(l)$ by scaling it, translating it and/or rotating it, the value $\|Z(m) - Z'(m)\|_2$ is null, i.e. that $|Z_d(m)| = 0, \forall m = 0, \ldots, M-1$.

Let us indicate the module of $Z(m)$ as $R(m) = mod(Z(m))$ and its argument as $\Theta(m) = arg(Z(m))$, $Z(m) = R(m)e^{j\Theta(m)}$. From Section 5.2.1, we can write the modules and the arguments of $Z'(m)$ as follows:

- $R'(0) = \beta(mod(Z(0) + \overline{z}))$
- $R'(m) = \beta R(m) \qquad m \neq 0$
- $\Theta'(0) = arg(Z(0) + \overline{z}) + \theta'$
- $\Theta'(m) = \Theta(m) + \theta' \qquad m \neq 0$

From Section 5.2.1 we know that, in order to preserve the translation invariance, we have to discard the module and the argument of the DC coefficient (i.e. $\mathcal{R}(0)$ and $\Theta(0)$); to obtain the rotation invariance we have to subtract the first argument $\Theta(1)$ to all arguments; finally, to guarantee the scale invariance, we have to divide all modules by the first not null module, that, for simplicity, we suppose to be $R(1)$. In this way, the modified coefficients for $Z(m)$ and $Z'(m)$ become:

- $\overline{R}(m) = \left[0, \frac{R(1)}{R(1)}, \ldots, \frac{R(M)}{R(1)}\right]$
- $\overline{R'}(m) = \left[0, \frac{R'(1)}{R'(1)}, \ldots, \frac{R'(M)}{R'(1)}\right]$
- $\overline{\Theta}(m) = [0, 0, (\Theta(2) - \Theta(1)), \ldots, (\Theta(M) - \Theta(1))]$
- $\overline{\Theta'}(m) = [0, 0, (\Theta'(2) - \Theta'(1)), \ldots, (\Theta'(M) - \Theta'(1))]$

We have now to prove that $|Z_d(m)| = 0$, for all values of m. The cases for $m = 0$ and $m = 1$ are immediately satisfied, whereas for the case for $m > 1$ we obtain:

$$\begin{aligned} Z_d(m) = \frac{R(m)}{R(1)} e^{j(\Theta(m)-\Theta(1))} - \frac{R'(m)}{R'(1)} e^{j(\Theta'(m)-\Theta'(1))} = \\ = \frac{R(m)}{R(1)} e^{j(\Theta(m)-\Theta(1))} - \frac{\beta R(m)}{\beta R(1)} e^{j(\Theta(m)-\Theta(1)+\theta'-\theta')} = \\ = \frac{R(m)}{R(1)} e^{j(\Theta(m)-\Theta(1))} - \frac{R(m)}{R(1)} e^{j(\Theta(m)-\Theta(1))} = 0 \end{aligned} \quad (5.20)$$

5.2.3 Experimental Results

To prove the effectiveness of our approach, among the many shape retrieval systems cited in Section 2.2, we chose to compare WINDSURF with the NeTra system [MM99]. NeTra represents, in fact, a good competitor because it follows the WINDSURF approach to represent the shape information but uses this information in a different way during the comparison phase. In detail, from the feature vector represented by the M complex coefficients, NeTra only uses the magnitude information of the coefficients, discarding the phase component. This, in fact, allows the rotation invariance since, as seen in Section 5.2.1, rotation involves modification of coefficients' arguments only. Scale invariance is achieved by dividing the amplitude of the coefficients by the first not null coefficient. Finally, translation invariance is obtained by discarding the amplitude with zero frequency, i.e. the DC coefficient. An Euclidean distance is used to compare two shape feature vectors. The basic difference to WINDSURF, thus, lies in how rotation invariance is obtained: We keep all the coefficients but one, whereas in NeTra all the arguments of the Fourier Transform are discarded.

Experimental results of the proposed techniques has been performed on a sample of about 3,000 real-images extracted from the IMSI data set [IMS].

Let us note that it is too simplistic to discard the phase information to obtain an invariance property. This means, in fact, to drop exactly half of the entire shape information. From our point of view, phase is important as much as magnitude and, as yet proved in previous Section, also following experimental results confirm that it is possible to use both the magnitude and the phase information preserving all the invariance properties, whereas discarding all coefficients' arguments too much information is lost. In Figure 5.5 we report results obtained from WINDSURF (on the left side) compared to those of the NeTra system (on the right side). Close to each image the segmented image is shown. The query image is represented by the central rectangular region of the top left image, i.e. the blue object (for clarity, a red frame is drawn around the query). Relevant images are

indicated with an arrow. From results, it is clear that the WINDSURF approach is more effective than NeTra. Computing the precision values (using $k = 10$) for both methods we obtain a value of 0.6 for WINDSURF against a 0.2 of NeTra. The superiority of the WINDSURF shape descriptors is also confirmed by other experiments, not shown here for the sake of brevity.

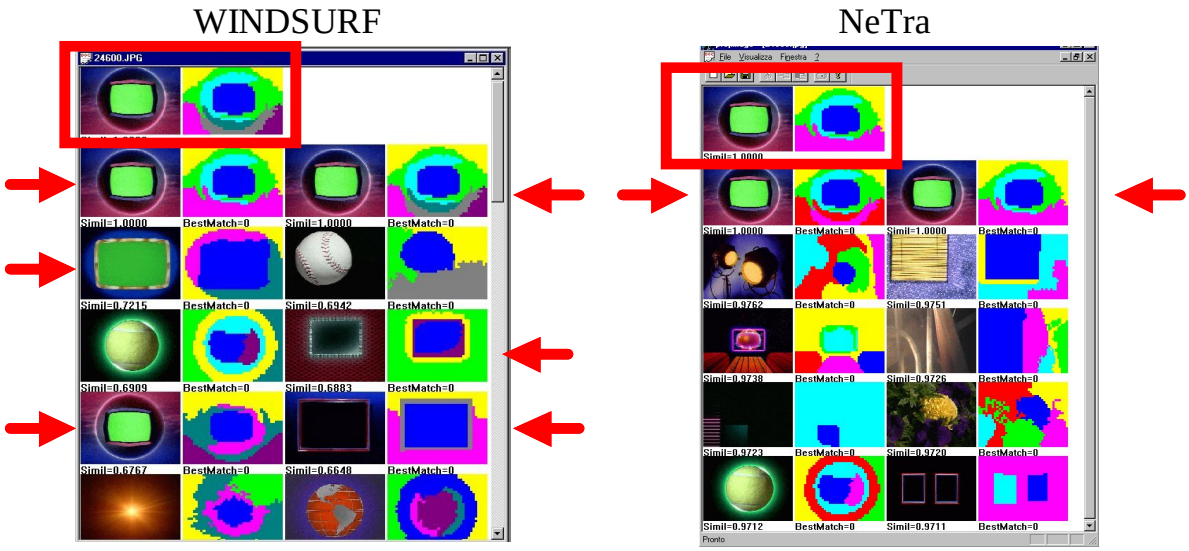


Figure 5.5: Comparison between WINDSURF and the NeTra system.

Finally, in Figure 5.6 we report WINDSURF results using the implemented image similarity modalities. In this case, the user is given the possibility to choose between three different modalities:

- The average function among all the object level score (named "average").
- The maximum function, that establishes which is the object of the image with higher similarity value, setting the image similarity to that value (named "max").
- The possibility for the user to choose a particular object from the image, using the object as image query (indicated as "selected object").

In detail, we can say that the choice of which modality to follow depends both on the query image and on what the user is looking for. If the image is represented by a well-defined object on a background, and the user is interested in finding images that contain the main object, the selected object is the correct modality (see, as an example, the dark square on the white background query image on the left side of Figure 5.6). Considering the same condition but with a query image with noise at edges, we have to observe that

the extraction of the correct boundary of the object becomes more difficult. To deal with this kind of queries, the use of the average function is recommended (an example of this case is represented by the world query image on the dark background of Figure 5.6, reported on the right side). Of course, it can happen that completely different images, from a semantic point of view, are returned (see the two images with a red frame around). Finally, in case the user is not looking for a particular object but only for an image similar to that given in input, the maximum function gives the best results because it considers only the best matched object to compute the overall similarity score (the flag image on the blue background query image, reported in the center of Figure 5.6, represents a typical example of this case). Again, it can happen that no similar images are contained in the results list (see the images with a red frame around).

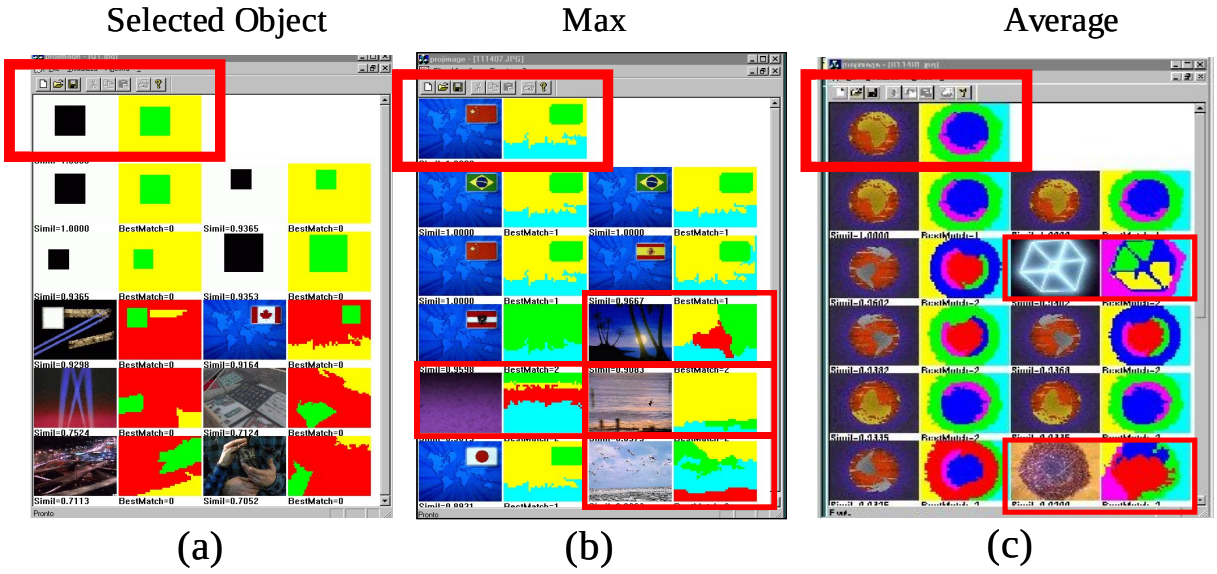


Figure 5.6: WINDSURF results for a selected object (a), for the maximum (b) and the average (c) image similarity measures.

Chapter 6

Conclusions

In this thesis we presented several techniques, concretized in the WINDSURF system and in the FeedbackBypass approach, to improve both the efficiency and the effectiveness of similarity search in image DBs.

The main conclusions we can draw from our work are the following:

- The region-based image retrieval approach is more effective than the usual content-based approach. WINDSURF, by using a wavelet-based clustering approach to segment images into homogeneous regions, is able to better characterize the content of images. Experimental results demonstrate the superior retrieval effectiveness of WINDSURF with respect to both the Stricker and Orengo [SO95] and the IBM QBIC [FSN+95] systems.

- Query processing techniques based on *correct* image matching algorithms are very effective with respect to those based on simplificative heuristics [CTB+99], and their index-based implementations are more efficient as compared to the sequential access modality. We further point out that the WINDSURF index-based algorithm ($\mathcal{A}_0^{WS*}$) is the *first* correct algorithm for region-based image similarity queries.

- We are conscious that WINDSURF needs a more thorough validation in term of effectiveness, e.g. by comparing it with other region-based image retrieval systems. To this end, we are completing the implementation of a comparing platform between WINDSURF and the Blobworld system [CTB+99].

- Interactive similarity search techniques for image DBs share the common idea to exploit user feedback in order to progressively adjust the query parameters and to eventually converge to an "optimal" parameter setting. However, all such methods also share the unpleasant feature to "forget" user preferences across multiple query

sessions. To overcome such problem, an efficient solution has been presented: FeedbackBypass. FeedbackBypass complements the role of relevance feedback engines by storing and maintaining the query parameters using a wavelet-based data structure (the Simplex Tree).

- Experimental results show that FeedbackBypass works well on real high-dimensional data, and that its predictions consistently outperform basic retrieval strategies which start with default query parameters.

- A key feature of FeedbackBypass is its orthogonality to existing feedback models, i.e. FeedbackBypass can be easily incorporated into current retrieval systems regardless of the particular mathematical model underlying the feedback loop. Further, FeedbackBypass is distinguished by its low resource requirements which grow polynomially with the dimensionality of the data set, thus making it applicable to high-dimensional feature spaces.

- As far as we know, no interactive similarity search model for region-based image retrieval systems have been presented. To overcome this limitation, the region-based relevance feedback model of Section 5.1 has been presented, allowing further effectiveness improvement. Note that this formal model has been studied for WINDSURF, but can be applied to every other system that fragments images into regions. Of course, FeedbackBypass can also be applied to the region-based relevance feedback model.

6.1 Future Directions

Throughout this thesis, we pointed out several interesting issues for future research. These include:

- We plan to employ approximate techniques for index access [CP00] for our optimal index-based algorithms ($\mathcal{A}_0^{WS*}$), in order to further reduce the number of distance computations needed to answer a query, and thus to further improve the efficiency.

- Another issue we intend to investigate, still related to the $\mathcal{A}_0^{WS*}$ algorithm, regards the possible parallelization of multiple index scans, along the lines described in [BEKS00].

- Still related to WINDSURF, at present time able to work only on color, texture, and shape properties, we plan to integrate also techniques able to recognize the spatial location of regions.

- Related to FeedbackBypass, we intend to extend the application to other types of multimedia and include a thorough investigation of the relationships existing between the resource requirements and the accuracy of the delivered predictions.
- We then plan to integrate in FeedbackBypass the possibility to manage several Simplex Trees to support multiple user interactive applications (*user profiles* problem).
- For comparison purposes, we are implementing an alternative version of FeedbackBypass based on the use of support vector machines [Vap98]. We also plan to implement a third version based on neural networks.
- We intend to extend FeedbackBypass to general metric (non-vector) spaces.
- Finally, we plan to integrate FeedbackBypass into WINDSURF.

Appendix A

The Wavelet Transform

Wavelet Transform (WT) [Dau92] surged to tremendous popularity during the past decade, to the point of almost replacing the Fourier Transform, at least in terms of the interest shown by researchers if not in a more and more growing number of applications.

The basic idea of the WT is similar to that of Fourier Transform: Approximate a signal through a set of basic mathematical functions [Gra95]. However, wavelet functions are able to give a multi-resolution representation of the signal, since each frequency component can be analyzed with a different resolution and scale, whereas the Fourier Transform divides the time-frequency domain in a homogeneous way (Figure A.1 (a)). This allows the WT to represent discontinuities in the signal by using "short" base functions and, at the same time, to emphasize low frequency components using "wide" base functions (Figure A.1 (b)). To this end, WT doesn't have a fixed set of base functions like the Fourier Transform (that uses only the sine and the cosine functions) but can use an infinite set of possible base functions.

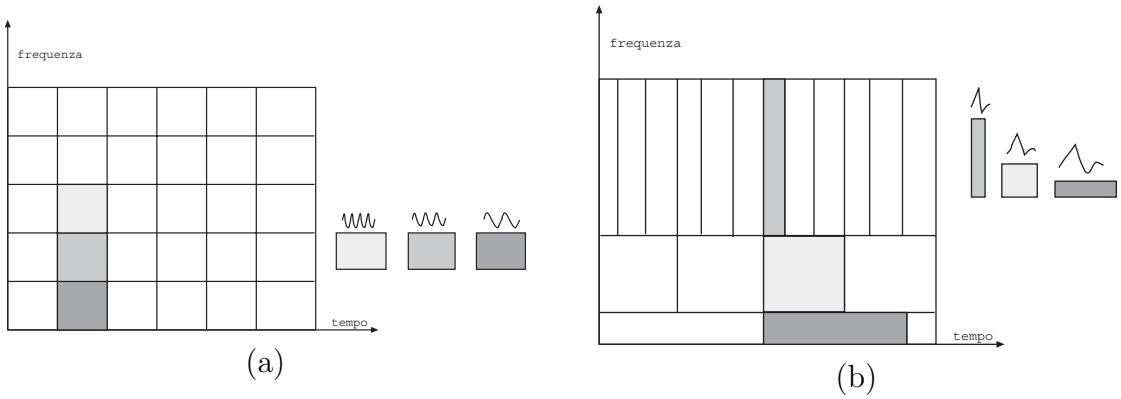


Figure A.1: Time-frequency space for Fourier (a) and Wavelet (b) Transform.

The Continuous WT decomposes a 1-D signal $f(x)$ into a set of *scaling functions* by using a set of wavelet basis functions $\{\psi_{a,b}\}$:

$$(W_a f)(b) = \int f(x)\psi^*_{a,b}(x)\mathrm{d}x \tag{A.1}$$

where each wavelet basis function is obtained from a *mother wavelet* $\psi(x)$ by scaling and shifting:

$$\psi_{a,b}(x) = \frac{1}{\sqrt{a}}\psi\left(\frac{x-b}{a}\right) \tag{A.2}$$

The mother wavelet should only satisfy the zero-average condition, i.e. $\int \psi(x)\mathrm{d}x = 0$. The Discrete WT is obtained by taking $a = 2^n$ and $b \in \mathbb{Z}$ in Equation A.1.

A.1 Haar Wavelet Transform

The oldest, and simplest, example of a mother wavelet is the Haar function, which was first introduced in 1910, and is composed by a pair of rectangular pulses (Figure A.2):

$$\psi(x) = \begin{cases} 1 & 0 \leq x < 1/2 \\ -1 & 1/2 \leq x < 1 \\ 0 & \text{otherwise} \end{cases} \tag{A.3}$$

Consider a discrete signal $x = (x_0, x_1, \ldots, x_{2^L-1})$ having length 2^L. The DWT is computed through the following steps:

1. For each pair of consecutive samples (x_{2i}, x_{2i+1}), $(0 \leq i < 2^{L-1})$, compute $a^1_i = \frac{1}{\sqrt{2}}(x_{2i} + x_{2i+1})$ and $d^1_i = \frac{1}{\sqrt{2}}(x_{2i} - x_{2i+1})$.

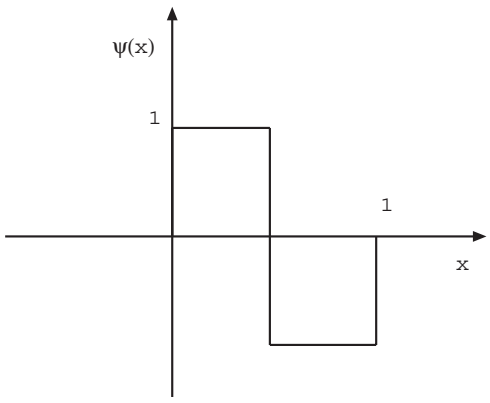


Figure A.2: Haar wavelet.

2. Consider the new signal $(a_0^1, \ldots, a_{2^{L-1}-1}^1)$ and proceed as in step 1., obtaining a_i^2 and d_i^2 $(0 \leq i < 2^{L-2})$.

3. Continue until a single value of a_0^L is obtained.

The Haar Transform of x is given by the set of "difference" values d_i^l $(0 < l \leq L$, $0 < i < 2^{l-1})$, and the "average" value for the last level a_0^L. In the frequency domain, the values a_i^l correspond to the output of a *low pass filter*, thus representing low-frequency information, whereas the d_i^l values correspond to the output of a *high pass filter*, thus representing high-frequency information.

To clarify the process of how a signal is decomposed by means of the Haar Transform, a numeric example follows.

Example A.1
Consider one-dimensional pixel image with the following eight values:

$$I = [5, 3, 6, 2, 3, 3, 8, 4]$$

The Haar WT for the above image can be calculated as follows:

$$a_0^1 = \frac{5+3}{\sqrt{2}} = \frac{8}{\sqrt{2}} \qquad a_1^1 = \frac{6+2}{\sqrt{2}} = \frac{8}{\sqrt{2}} \qquad a_2^1 = \frac{3+3}{\sqrt{2}} = \frac{6}{\sqrt{2}} \qquad a_3^1 = \frac{8+4}{\sqrt{2}} = \frac{12}{\sqrt{2}}$$

$$d_0^1 = \frac{5-3}{\sqrt{2}} = \sqrt{2} \qquad d_1^1 = \frac{6-2}{\sqrt{2}} = 2\sqrt{2} \qquad d_2^1 = \frac{3-3}{\sqrt{2}} = 0 \qquad d_3^1 = \frac{8-4}{\sqrt{2}} = 2\sqrt{2}$$

$$a_0^2 = \frac{\frac{8}{\sqrt{2}} + \frac{8}{\sqrt{2}}}{\sqrt{2}} = 8 \quad a_1^2 = \frac{\frac{6}{\sqrt{2}} + \frac{12}{\sqrt{2}}}{\sqrt{2}} = 9 \quad d_0^2 = \frac{\frac{8}{\sqrt{2}} - \frac{8}{\sqrt{2}}}{\sqrt{2}} = 0 \quad d_1^2 = \frac{\frac{6}{\sqrt{2}} - \frac{12}{\sqrt{2}}}{\sqrt{2}} = -3$$

$$a_0^3 = \frac{8+9}{\sqrt{2}} = \frac{17}{\sqrt{2}} \qquad d_0^3 = \frac{8-9}{\sqrt{2}} = -\frac{1}{\sqrt{2}}$$

The 8 coefficients WT are defined as the single coefficient representing the overall normalized average of the pixel values (a_0^3) followed by the detail coefficients in order of increasing resolution (d_0^3, d_0^2, d_1^2, d_0^1, d_1^1, d_2^1 and d_3^1). Thus, the one-dimensional Haar WT for the original image is given by:

$$W = [\frac{17}{\sqrt{2}}, -\frac{1}{\sqrt{2}}, 0, -3, \sqrt{2}, 2\sqrt{2}, 0, 2\sqrt{2}]$$

Each entry in W is called *wavelet coefficient*. Using the WT of an image, rather than the image itself has several advantages: For example a large number of detail coefficients tend to be very small value. Thus, truncating these small coefficients from the transform introduces only small errors in the reconstructed image, giving a form of "lossy" image compression.

A.2 Two Dimensional Haar Wavelet

In our case, the signal is a 2-D color image, where the "time" domain is the spatial location of pixels and the frequency domain is the color variation between adjacent pixels. In order to build an orthonormal wavelet basis for the 2-dimensional space, one can start from the 1-dimensional domain and compute the product of two 1-dimensional basis functions, that is:

$$\Psi_{j_1,k_1;j_2,k_2}(x_1, x_2) = \psi_{j_1,k_1}(x_1) \cdot \psi_{j_2,k_2}(x_2)$$

If the image to be processed has dimension $N \times M$ (with both N and M power of 2), the first transformation step decomposes the signal into four sub-images of dimension $N/2 \times M/2$, representing the sub-bands in the frequency domain. The obtained sub-images are labelled as LL, LH, HL, HH, where L and H represent low- and high-frequency information, respectively, and the first position refers to the horizontal direction, whereas the second position refers to the vertical direction:

LL: Low-frequency information in both the horizontal and vertical directions.

LH: Low-frequency information in the horizontal direction, high-frequency information in the vertical direction.

HL: High-frequency information in the horizontal direction, low-frequency information in the vertical direction.

HH: High-frequency information in both the horizontal and vertical directions.

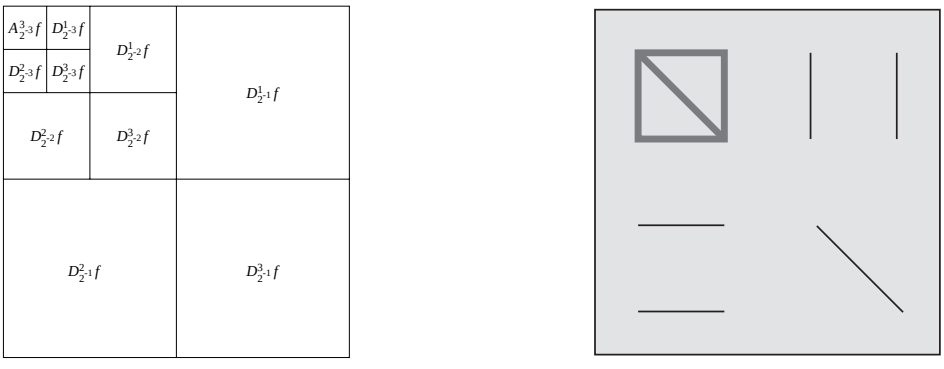


Figure A.3: The sub-images $D^k_{2^{-j}}f, A^d_{2^{-L}}f$ in the "wavelet" image representation (a) and its horizontal, vertical and diagonal information (b).

The second transformation level decomposes the LL sub-image, obtaining four images of dimension $N/4 \times M/4$, and so on. Figure A.3 (a) shows the decomposition of the frequency domain at different scale levels: $A^d_{2^{-L}} f$ contains low-frequency information, whereas $D^1_{2^{-j}} f$, $D^2_{2^{-j}} f$, and $D^3_{2^{-j}} f$ contain horizontal, vertical and diagonal information, respectively (Figure A.3 (b)).

Example A.2

Let us consider the 4×4 image I shown in Figure A.4 and let us use the standard average, instead of the normalized one, for simplicity. The matrix I' in Figure A.4 is the result of first horizontal and vertical pairwise averaging and differencing on the 2×2 boxes of the original image I. The average value in in each box (2.5) is assigned to the 2×2 matrix A, while the detail coefficients are stored in the tree 2×2 boxes of the W matrix. The process is then recursively performed a second time on the averages contained in the A matrix resulting in detail coefficients of 0 and an average value of 2.5, which is stored in the upper-left corner pixel of W.

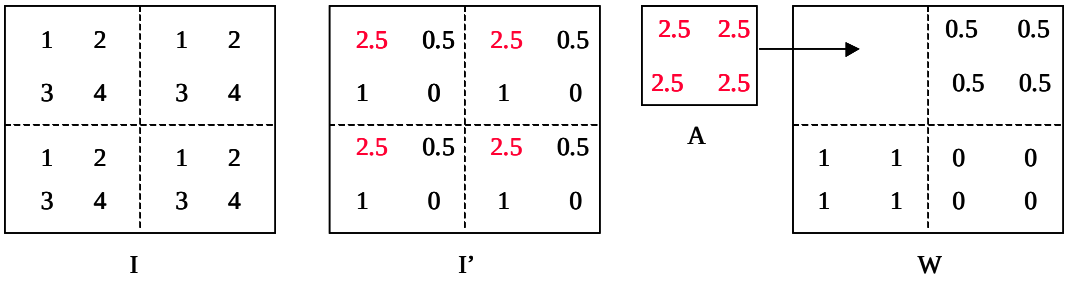


Figure A.4: Computing the Wavelet Transform on a 2-D signal.

Appendix B

Examples of Interactive Similarity Search

In this Chapter we describe some examples of interactive similarity search introducing MoRFEo (Multi-modal Relevance Feedback Environment), a prototype application that we have implemented to study the behavior of the relevance feedback techniques in the content-based image retrieval context. Experiment results on a real image data set demonstrate the large impact that relevance feedback has on the effectiveness of similarity results.

Similarity Environment

To test the MoRFEo application we use the IMSI data set [IMS] consisting of about 10,000 real-color images. Like for the FeedbackBypass experiments, from each image, represented in the HSV color space, we extract a 32-bins color histogram, by dividing the hue channel H into 8 ranges and the saturation channel S into 4 ranges. To compare histograms we use the class of weighted Euclidean distances, taking the Euclidean distance as the default function. MoRFEo implements both query point movement and re-weighting feedback strategies and the user can decide which technique to use during the interactive research (i.e. only query point movement, only re-weighting or both).

System Architecture

The architecture of MoRFEo is sketched in Figure B.1. In detail, the DB component represents the real-image dataset. The Feature Extractor, given an image from the user (by way of the User Interface), extracts the corresponding 32-bin color histogram and stores the feature vector into the DB. Upon receiving a query image, the Retrieval Engine queries the DB. Then, the usual user evaluation and feedback computation loop takes place by way of the Feedback Module. The process continues until the user is satisfied

with results presented by the User Interface.

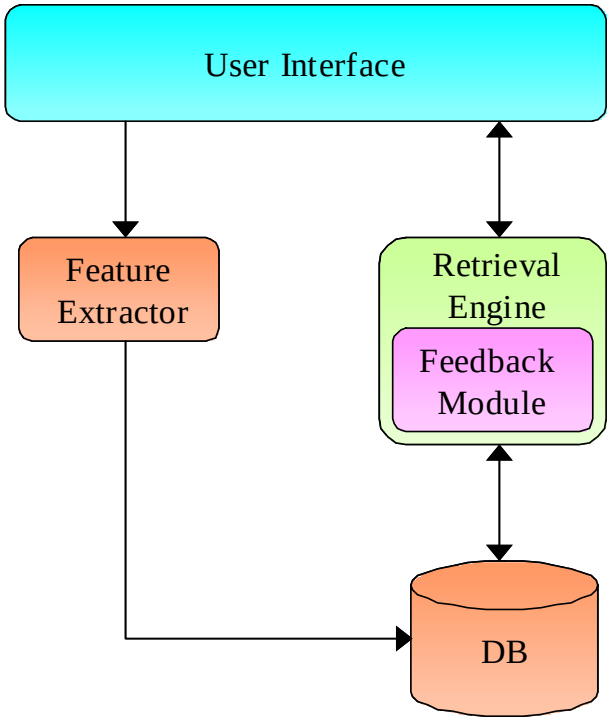


Figure B.1: The MoRFEo system architecture.

Query Specification

The user can issue queries to the system through the User Interface using two different modalities:

1. By choosing a starting image (selecting **File**, then **Open**, and specifying the path of an existing image, see Figure B.2).

2. By choosing a random image of the DB (selecting **Query**, then **Random Query Point**, see Figure B.3).

Having specified the query, the system asks the user for the number of requested result images by way of the results number dialog (Figure B.4).

The Retrieval Engine, then, processes the query, retrieving the result images from the DB and presenting them to the user, sorted in increasing order of distance with respect to the query. Figure B.5 show an example of similarity search using the default distance (i.e. the Euclidean distance).

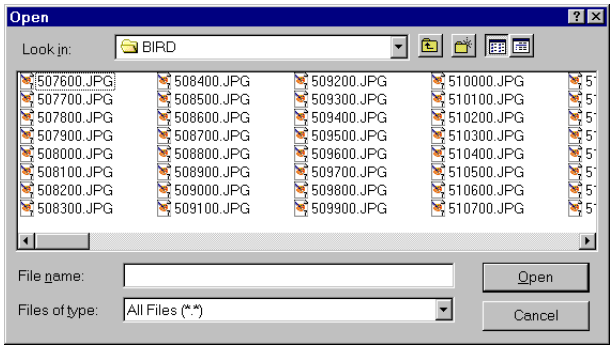


Figure B.2: Choice of an existing query image.

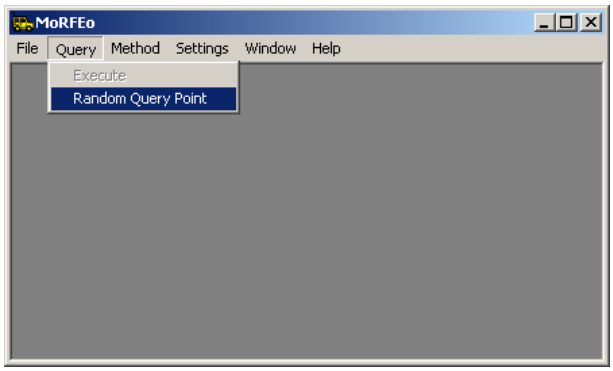


Figure B.3: Choice of a random query image.

Relevance Feedback Technique Specification

Starting from a similarity search default result, the user can give judgments on each returned image. First of all, he/she has to decide which relevance feedback technique to use during the interactive search. In detail, at each cycle of similarity search, through the User Interface, the user can choose three possible options (Figure B.6)):

1. Query Point Movement (selecting **Method**, then **Query Point Movement**).

2. Re-weighting (selecting **Method**, then **Re-weighting**).

3. Query Point Movement plus Re-weighting (selecting **Method**, then **Both**).

After choosing the feedback technique, the user specifies which are the positive examples, i.e. which images are relevant, in our approach, for query image (e.g. in case of query 508200.jpg of Figure B.5, representing a bird, positive examples could be images that contain birds). Positive examples are selected by clicking on the image. The system shows a green frame around each relevant image (shown in Figure B.7).

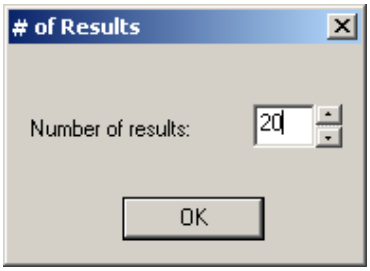


Figure B.4: The results number dialog.

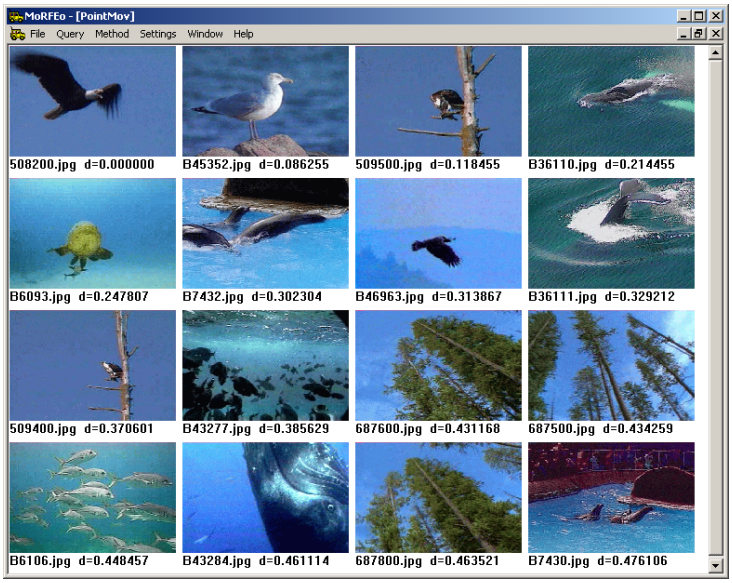


Figure B.5: The default results for the query 508200.jpg.

Figure B.8 show the results obtained, still for the query 508200.jpg, after the first cycle of relevance feedback using the query point movement strategy and selecting, as relevant results, all bird images of Figure B.5.

Like relevant images demonstrate, query point movement allows to move the query point towards the good images. This is confirmed also by Figure B.9, representing the second cycle of search still using only query point movement. In this case we add a new bird image in the set of relevant images.

When no more improvement is obtained using the query point movement technique (i.e. when we are close enough to the optimal query point), it is possible, by using the re-weighting strategy, to re-weigh the default distance, modifying the original Euclidean sphere into an ellipsoid, to further improve the effectiveness of results. Figure B.10 shows images obtained at third cycle of search by applying the re-weighting modality to results of Figure B.9. We can see that all images contain at least a bird.

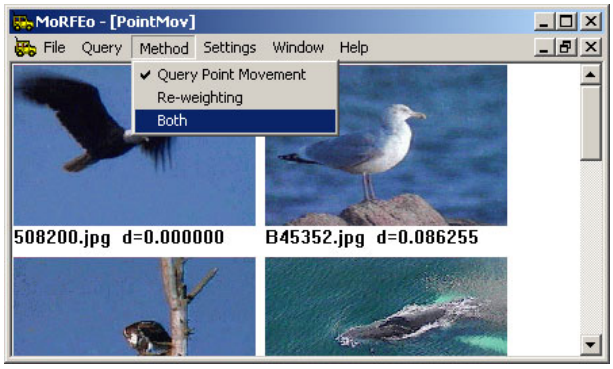


Figure B.6: Choice of the relevance feedback technique.

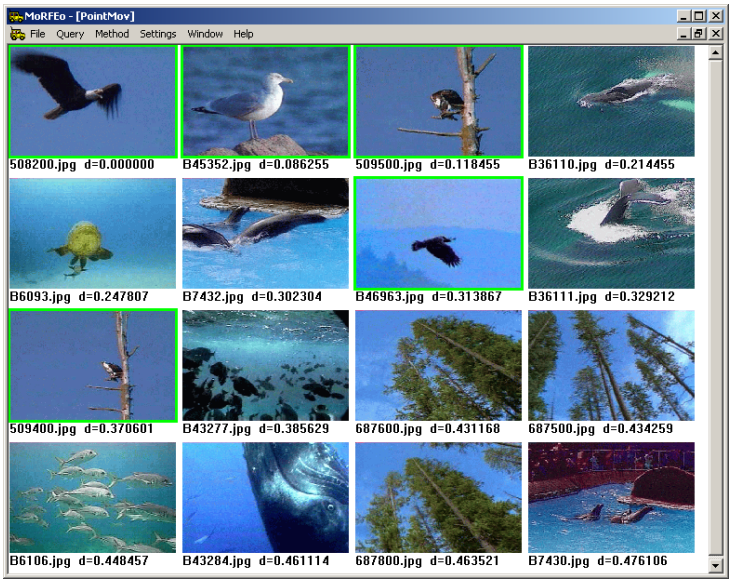


Figure B.7: Positive examples for the query 508200.jpg.

Finally, Figure B.11 shows results obtained from the same query 508200.jpg, starting from the same positive examples of Figure B.8 but using both the query point movement and the re-weighting strategies. In this case, the result list does not contains only images with birds, but also two fish images reported in position 13 and 15.

This example shows that, using the two feedback strategies in an alternate way, it is possible to obtain better results with respect to those obtained when the two techniques are used together, contrary to what commonly assumed in CBIR systems supporting feedback facilities [RHOM98, ISF98, COPM01].

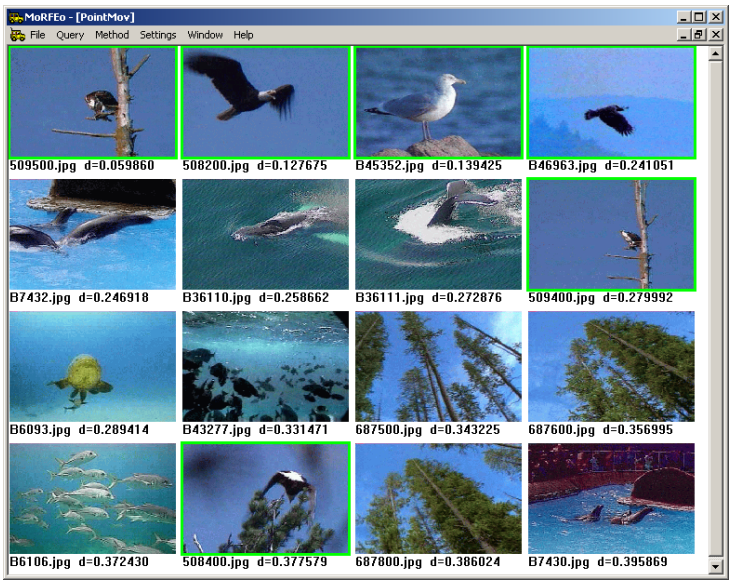


Figure B.8: The first-cycle results for the query 508200.jpg using query point movement.

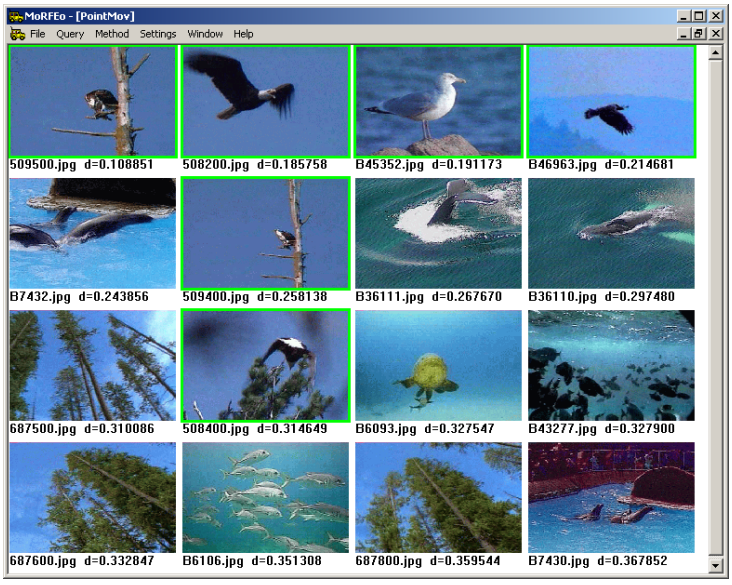


Figure B.9: The second-cycle results for the query 508200.jpg using query point movement.

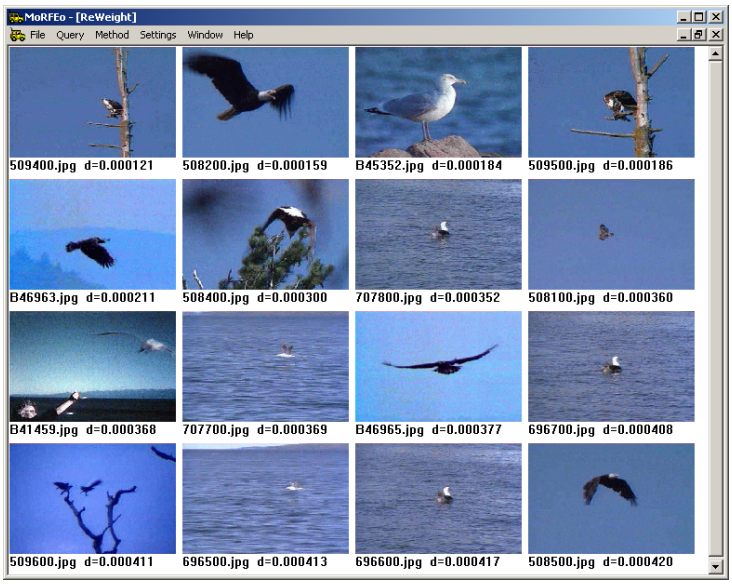


Figure B.10: The third-cycle results for the query 508200.jpg using re-weighting.

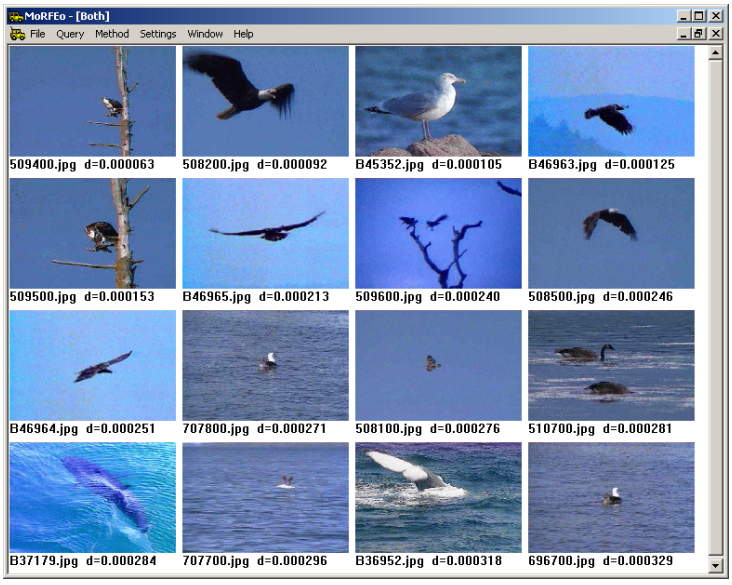


Figure B.11: The results for the query 508200.jpg using the Both strategy.

Bibliography

[ABP99] Stefania Ardizzoni, Ilaria Bartolini, and Marco Patella. Windsurf: Region-based image retrieval using wavelets. In *Proceedings of the 1st International Workshop on Similarity Search (IWOSS'99)*, pages 167–173, Florence, Italy, September 1999.

[Bas89] Michèle Basseville. Distance measure for signal processing and pattern recognition. *European Journal of Signal Processing*, 18(4):349–369, December 1989.

[BCGM98] Serge Belongie, Chad Carson, Hayit Greenspan, and Jitendra Malik. Color- and texture-based image segmentation using EM and its application to content-based image retrieval. In *Proceedings of the 6th International Conference on Computer Vision (ICCV'98)*, Mumbai, India, January 1998.

[BCP00a] Ilaria Bartolini, Paolo Ciaccia, and Marco Patella. A sound algorithm for region-based image retrieval using an index. In *Proceedings of the 4th International Workshop on Query Processing and Multimedia Issue in Distributed Systems (QPMIDS'00)*, pages 930–934, Greenwich, London, UK, September 2000.

[BCP00b] Ilaria Bartolini, Paolo Ciaccia, and Marco Patella. WINDSURF: A region-based image retrieval system. Technical Report CSITE-011-00, CSITE–CNR, 2000. Available at URL `http://www-db.deis.unibo.it/MMDBGroup/TRs.html`.

[BCW00] Ilaria Bartolini, Paolo Ciaccia, and Florian Waas. Using the wavelet transform to learn from user feedback. In *Proceedings of the First DELOS Network of Excellence Workshop on Information Seeking, Searching and Querying in Digital Libraries*, Zurich, Switzerland, December 2000. Online publication `http://www.ercim.org/publication/ws-proceedings/DelNoe01/`.

[BCW01a] Ilaria Bartolini, Paolo Ciaccia, and Florian Waas. FeedbackBypass: A new approach to interactive similarity query processing. Technical Report CSITE-09-01, CSITE–CNR, 2001. Available at URL `http://www-db.deis.unibo.it/MMDBGroup/TRs.html`.

[BCW01b] Ilaria Bartolini, Paolo Ciaccia, and Florian Waas. FeedbackBypass: A new approach to interactive similarity query processing. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB'01)*, pages 201–210, Rome, Italy, September 2001.

[BDV99] Stefano Berretti, Alberto Del Bimbo, and Enrico Vicario. Managing the complexity of match in retrieval by spatial arrangement. In *International Conference on Image Analysis and Processing (ICIAP'99)*, Venezia, Italy, September 1999.

[BEKS00] Bernhard Braunmüller, Martin Ester, Hans-Peter Kriegel, and Jörg Sander. Efficiently supporting multiple similarity queries for mining in metric databases. In *Proceedings of the 16th International Conference on Data Engineering (ICDE 2000)*, pages 256–267, San Diego, CA, March 2000.

[BKK96] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel. The X-tree: An index structure for high-dimensional data. In *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB'96)*, pages 28–39, Mumbai (Bombay), India, September 1996.

[BKSS90] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 322–331, Atlantic City, NJ, May 1990.

[BP00] Ilaria Bartolini and Marco Patella. Correct and efficient evaluation of region-based image search. In *Atti dellOttavo Convegno Nazionale SEBD*, pages 289–302, L'Aquila, Italy, June 2000.

[CK93] Tianhorng Chang and C.-C. Jay Kuo. Texture analysis and classification with tree-structured wavelet transform. *IEEE Transactions on Image Processing*, 2(4):429–441, October 1993.

[COPM01] Kaushik Chakrabarti, Michael Ortega, Kriengkrai Porkaew, and Sharad Mehrotra. Query refinement in similarity retrieval systems. *IEEE Data Engineering Bulletin*, 24(3):3–13, September 2001.

[CP00] Paolo Ciaccia and Marco Patella. PAC nearest neighbor queries: Approximate and controlled search in high-dimensional and metric spaces. In *Proceedings of the 16th International Conference on Data Engineering (ICDE'00)*, pages 244–255, San Diego, CA, March 2000.

[CPZ97] Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB'97)*, pages 426–435, Athens, Greece, August 1997.

[CPZ98] Paolo Ciaccia, Marco Patella, and Pavel Zezula. Processing complex similarity queries with distance-based access methods. In *Proceedings of the 6th International Conference on Extending Database Technology (EDBT'98)*, pages 9–23, Valencia, Spain, March 1998.

[CTB+99] Chad Carson, Megan Thomas, Serge Belongie, Joseph M. Hellerstein, and Jitendra Malik. Blobworld: A system for region-based image indexing and retrieval. In *Proceedings of the 3rd International Conference on Visual Information Systems (VISUAL'99)*, pages 509–516, Amsterdam, The Netherlands, June 1999.

[Dau92] Ingrid Daubechies. *Ten Lectures on Wavelets*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1992.

[Del99] Alberto Del Bimbo. *Visual Information Retrieval*. Morgan Kaufmann, Inc., San Francisco, California, 1999.

[Fag96] Ronald Fagin. Combining fuzzy information from multiple systems. In *Proceedings of the 15th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'96)*, pages 216–226, Montreal, Canada, June 1996.

[Fal96] Christos Faloutsos. *Searching Multimedia Database by Content*. Kluwer Academic Publishers, 1996.

[FEF+94] Christos Faloutsos, Will Equitz, Myron Flickner, Wayne Niblack, Dragutin Petkovic, and Ron Barber. Efficient and effective querying by image content. *Journal of Intelligent Information Systems*, 3(3/4):231–262, July 1994.

[FLN01] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *Proceedings of the 20th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'01)*, Santa Barbara, California, USA, May 2001.

[FSN+95] Myron Flickner, Harpreet Sawhney, Wayne Niblack, Jonathan Ashley, Qian Huang, Byron Dom, Monika Gorkani, Jim Hafner, Denis Lee, Dragutin Petkovic, David Steele, and Peter Yanker. Query by image and video content: The QBIC system. *IEEE Computer*, 28(9):23–32, September 1995. `http://wwwqbic.almaden.ibm.com/`.

[GBK00] Ulrich Güntzer, Wolf-Tilo Balke, and Werner Kießling. Optimizing multi-feature queries for image databases. In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB'00)*, pages 419–428, Cairo, Egypt, September 2000.

[GR95] Venkat N. Gudivada and Vijay V. Raghavan. Content-based image retrieval systems. *IEEE Computer*, 28(9):18–22, September 1995. Guest Editors' Introduction.

[Gra95] Amara Graps. An introduction to wavelet. *IEEE Computational Science Engineering*, 2(2):50–61, June 1995.

[GW92] Rafael C. Gonzales and Richard E. Woods. *Digital Image Processing.* Addison-Wesley, Reading, Ma, 1992.

[HS99] Gísli R. Hjaltason and Hanan Samet. Distance browsing in spatial databases. *ACM Transactions on Database Systems*, 24(2):265–318, June 1999.

[II86] Hiroshi Imai and Masao Iri. Computational-geometric methods for polygonal approximations of a curve. *Computer Vision, Graphics and Image Processing*, 36:31–41, 1986.

[IMS] IMSI MasterPhotos 50,000. IMSI USA. `http://www.imsisoft.com`.

[ISF98] Yoshiharu Ishikawa, Ravishankar Subramanya, and Christos Faloutsos. MindReader: Querying databases through multiple examples. In *Proceedings of*

the 24th International Conference on Very Large Data Bases (VLDB'98), pages 218–227, New York City, NY, August 1998.

[Kai94] Gerald Kaiser. *A Friendly Guide to Wavelets*. Birkhäuser, Boston, 1994.

[Kuh55] Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, 2:83–97, 1955.

[Meh94] Kurt Mehlhorn. *Data Structures and Algorithms*, volume 3: Multi-dimensional Searching and Computational Geometry. Springer-Verlag, Berlin, 1994.

[MM99] Wei-Ying Ma and B. S. Manjunath. NeTra: A toolbox for navigating large image databases. *Multimedia Systems*, 7(3):184–198, May 1999.

[NRS99] Apostol Natsev, Rajeev Rastogi, and Kyuseok Shim. WALRUS: A similarity retrieval algorithm for image databases. In *Proceedings 1999 ACM SIGMOD International Conference on Management of Data*, pages 396–405, Philadelphia, PA, June 1999.

[ORC+98] Michael Ortega, Yong Rui, Kaushik Chakrabarti, Kriengkrai Porkaew, Sharad Mehrotra, and Thomas S. Huang. Supporting ranked boolean similarity queries in MARS. *IEEE Transactions on Knowledge and Data Engineering*, 10(6):905–925, January 1998.

[Ore62] Øystein Ore. *Theory of graphs*. American Mathematical Society, Providence, RI, 1962.

[OS95] Virginia E. Ogle and Michael Stonebraker. Chabot: Retrieval from a relational database of images. *IEEE Computer*, 28(9):40–48, September 1995.

[PC99] Kriengkrai Porkaew and Kaushik Chakrabarti. Query refinement for multimedia similarity retrieval in MARS. In *Proceedings of 7th ACM International Conference on Multimedia*, pages 235–238, Orlando, Florida, September 1999.

[PF95] Ulrich Pfeifer and Norbert Fuhr. Efficient processing of vague queries using a data stream approach. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'95)*, pages 189–197, Seattle, WA, July 1995.

[PPS96] Alex Pentland, Rosalind W. Picard, and Stan Sclaroff. Photobook: Content-based manipulation of image databases. In Borko Furht, editor, *Multimedia Tools and Applications*, chapter 2, pages 43–80. Kluwer Academic Publishers, 1996.

[PS85] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, NY, 1985.

[PTVF92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, Cambridge, NY, 1992.

[RH00] Yong Rui and Thomas S. Huang. Optimizing learning in image retrieval. In *Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition, 2000*, pages 236—243, Hilton Head, SC, June 2000.

[RHOM98] Yong Rui, Thomas S. Huang, Michael Ortega, and Sharad Mehrotra. Relevance feedback: A power tool for interactive content-based image retrieval. *IEEE Transaction on Circuits and Systems for Video Technology*, 8(5):644–655, September 1998.

[RSH96] Yong Rui, Alfred C. She, and Thomas S. Huang. Modified Fourier descriptors for shape representation – a practical approach. In *Proceedings of the 1st International Workshop on Image Databases and Multi Media Search*, Amsterdam, The Netherlands, August 1996.

[Sal89] Gerard Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, Reading, MA, 1989.

[SC96] John R. Smith and Shih-Fu Chang. VisualSEEk: A fully automated content-based image query system. In *Proceedings of the 4th ACM International Conference on Multimedia*, pages 87–98, Boston, MA, November 1996. `http://www.ctr.columbia.edu/visualseek/`.

[SD96] Markus A. Stricker and Alezander Dimai. Color indexing with weak spatial constraints. In *Proceedings of International Conference on Storage and Retrieval for Image and Video Databases (SPIE'96)*, pages 29–40, San Diego, CA, February 1996.

[SJ96] Simone Santini and Ramesh Jain. Similarity queries in image databases. In *Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition (CVPR '96)*, pages 646–651, San Francisco, CA, June 1996.

[SK97] Thomas Seidl and Hans-Peter Kriegel. Efficient user-adaptable similarity search in large multimedia databases. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB'97)*, pages 506–515, Athens, Greece, August 1997.

[Smi97] John R. Smith. *Integrated Spatial and Feature Image Systems: Retrieval, Analysis and Compression*. PhD thesis, Columbia University, 1997.

[SO95] Markus A. Stricker and Markus Orengo. Similarity of color images. In *Storage and Retrieval for Image and Video Databases SPIE*, volume 2420, pages 381–392, San Jose, CA, February 1995.

[SS96] Wim Sweldens and Paul Schröder. Building your own wavelets at home. In *Wavelets in Computer Graphics*, pages 15–87. ACM SIGGRAPH Course notes, 1996.

[Swe96] Wim Sweldens. The lifting scheme: A custom-design construction of biorthogonal wavelets. *Applied Comput. Harmon. Anal.*, 3(2):186–200, 1996.

[SWS+00] Arnold W. M. Smeulders, Marcel Worring, Simone Santini, Amarnath Gupta, and Ramesh Jain. Content-based image retrieval at the end of the early years. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(12):1349–1380, December 2000.

[TCH00] Megan Thomas, Chad Carson, and Joseph M. Hellerstein. Creating a customized access method for Blobworld. In *Proceedings of the 16th International Conference on Data Engineering (ICDE'00)*, page 82, San Diego, CA, March 2000.

[UVJ+97] Geert Uytterhoeven, Filip Van Wulpen, Maarten Jansen, Dirk Roose, and Adhemar Bultheel. WAILI: Wavelets with integer lifting. Technical Report 262, Department of Computer Science, Katholieke Universiteit Leuven, Heverlee, Belgium, July 1997.

[Vap98] Vladimir Vapnik. *Statistical Learning Theory*. Wiley, Chichester, GB, 1998.

[VSLV99] Gert Van de Wouver, Paul Scheunders, Stefan Livens, and Dirk Van Dyck. Wavelet correlation signatures for color texture characterisation. *Pattern Recognition*, 32(3):443–451, March 1999.

[WB00] Roger Weber and Klemens Böhm. Trading quality for time with nearest-neighbor search. In *Proceedings of the 7th International Conference on Extending Database Technology (EDBT'00)*, pages 21–35, Konstanz, Germany, March 2000.

[WFSP00] Leejay Wu, Christos Faloutsos, Katia P. Sycara, and Terry R. Payne. FALCON: Feedback adaptive loop for content-based retrieval. In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB'00)*, pages 297–306, Cairo, Egypt, September 2000.

[WLW01] James Ze Wang, Jia Li, and Gio Wiederhold. SIMPLIcity: Semantics-sensitive Integrated Matching for Picture LIbraries. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(9):947–963, September 2001.

[WWFW97] James Ze Wang, Gio Wiederhold, Oscar Firschein, and Sha Xin Wei. Wavelet-based image indexing techniques with partial sketch retrieval capability. In *Proceedings of the 4th IEEE Forum on Research and Technology Advances in Digital Libraries (ADL'97)*, pages 13–24, Washington, DC, May 1997.

[XB91] Xuanli Lisa Xie and Gerardo Beni. A validity measure for fuzzy clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(8):841–847, August 1991.

[ZSAR98] Pavel Zezula, Pasquale Savino, Giuseppe Amato, and Fausto Rabitti. Approximate similarity retrieval with M-trees. *The VLDB Journal*, 7(4):275–293, 1998.

"Live as if you were to die tomorrow,
Learn as if you were to live forever."

Gandhi