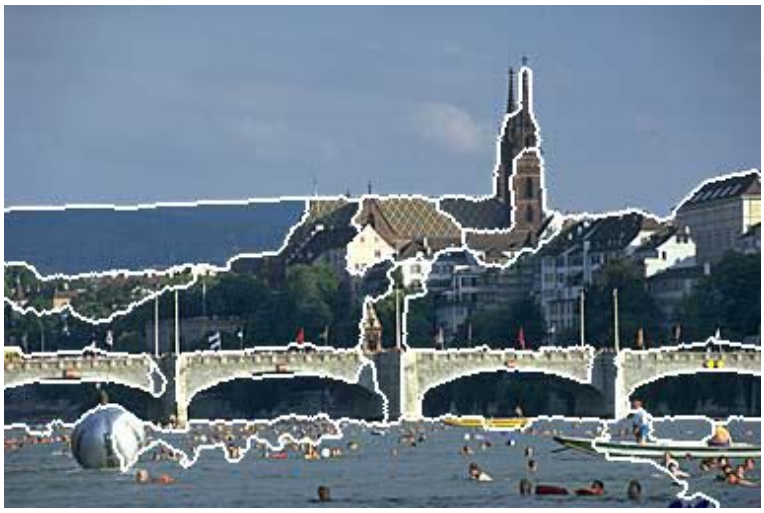
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Prof. Dr. H.-J. Schek
Institute for Information Systems
Database Research Group
http://www.dbs.ethz.ch

Diploma Thesis

# Region Based Image Similarity Search



Silvan Andreas Saxer

Summer Semester 2002

Prof. Dr. H.-J. Schek
Supervisors: M. Mlivoncic, Dr. R. Weber

# Abstract

There exists a number of image similarity search systems that do retrieval on the whole image content. In this thesis an efficient and flexible implementation of a content based image retrieval system that works on fractions of an image, the so called regions, is discussed.

The thesis presents two correct and fast algorithms that find similar images to arbitrary complex queries. Using a filtering step these algorithms perform significantly better than other correct algorithms.

In this work we describe the implementation of a region based image retrieval system in ISIS, the multimedia data retrieval framework of ETH Zürich. It integrates the search algorithms with region and feature extraction.

Experiments have shown that the system may return answers to even complex queries in an acceptable time frame and that the use of regions can improve the query results compared to the former system that only supported searching on the whole image or on static regions.

# Acknowledgements

My thank goes to all people that were involved in my diploma thesis. First of all to my assistant Michael Mlivoncic who had implemented a prototype of the current system and supported me on the implementation step, and notably extended my limited C++ skills. He also gave me great support in improving the text of this work and the presentation of the thesis. Another thank goes to Roger Weber who acted as project leader. He was always concerned about the work going in the right direction. He also supported me with his great knowledge of the VA-File and C++ programming. Not to forget the splendid Java-like C++ environment that he had developed. It made my life a lot easier (I did not have to struggle with Microsoft Foundation Classes). He also gave me highly valuable feedback on the manuscript.

Christoph Schuler helped me with the definition of processes and the building of the demo. Thanks to his commitment, the demo was ready nearly one week before the final presentation. Thomas Moscibroda supported me by integrating the region based features into his web-framework and defining processes to execute region based image retrieval. Marco Schmidt helped with infrastructure belongings. Another thank goes to Claudio Vaccani who read through the manuscript and gave me valuable feedback.

Thanks go to my brother Roland who answered my mathematical questions and gave me feedback on the presentation. Last but not least I would like to thank my parents for standing by my side for all my life and supporting me both financially and mentally during my studies at ETH.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Multimedia Information Retrieval

In the last couple of years more and more information has been published in computer readable formats. In the meanwhile much of the information in older books, journals and newspapers have been digitized and made computer readable. Big archives of film, music, images, satellite pictures, books, newspapers and magazines have been made accessible for computer users, some of them are open to the public, e.g. the Corbis image library [COR] or the archive of the Neue Zürcher Zeitung [Zei].

One of the enabling technologies for these huge multimedia libraries is the internet. A typical web page is a multimedia document, it contains images, text, sometimes video or sounds. A study of BrightPlanet assumes that the internet contains about 7500 TByte of data including private and dynamically created pages [Ber01]. This huge number of potentially available multimedia data has the drawback that we often don't find the information we need, since we do not know where to look for it. The huge amount of information tends to become a data graveyard, information that is there but is never used, because no one finds it.

Computer science research has early addressed this problem. First approaches concentrated on text retrieval since in the early days of the computer, textual information has been the most widely used media type. When the internet and digital image libraries became more and more popular, the computer scientists tried to extend the text retrieval to images. This was done by indexing manually entered textual annotations of the images. This human interaction has proven to be inefficient and slow. Therefore automatic annotation algorithms have been proposed [Sax01].

### 1.1.1 Content Based Image Retrieval

When the manual annotation of images became more and more unfeasible, new approaches became popular that use the raw image data and statistics and transformations thereof to perform similarity search. The user of such systems either

provides an image that is similar to the one he is looking for or draws a sketch of it. The system then compares the content of the query image with the content of all images in its index and returns the images that are most similar to the query image regarding some features as texture, color or shape. Most of these systems provide a means of refining the query (so called relevance feedback) in order to get better results in an iterative process.

### 1.1.2 Region Based Image Similarity Search

One of the main drawbacks of standard content based image retrieval systems is that they calculate and compare features for the image as a whole. Most pictures don't contain just one object in it. There are typically a number of regions in the image containing different objects, like a house, the sun, water or a tree. Newer approaches take this into account by segmenting the image into homogeneous regions. The search process explicitly compares the similarity between the regions of the query image and the regions of the indexed images.

A key prerequisite for a good region based image retrieval system is a robust segmentation algorithm. A segmentation algorithm takes an input image and clusters pixels of this image that seem to be similar with respect to some feature (e.g. color, texture or shape). The result of this clustering phase is a division of the image into 5 to 30 regions that each corresponds to an object or subject in the image.

## 1.2 ISIS: Interactive Similarity Search

In several projects in the past, the database research group at ETH Zürich has developed a prototype system called ISIS (Interactive SImilarity Search) that supports interactive image similarity search on a PC cluster [SSW02]. The retrieval part of the system is strongly based on the VA-File, a simple and efficient indexing and searching data structure that is based on approximations. In order to improve the retrieval effectiveness, complex similarity queries consisting of a number of reference images, different feature types, textual attributes and predicates are supported. The system uses relevance feedback for incremental query refinement.

### 1.2.1 The Retrieval Part of the System

Although the search on textual fragments - as used in most commercially available multimedia retrieval engines - is very efficient, the retrieval quality is seldom satisfactory. On the other hand, image retrieval systems that use numerous content descriptors (textual and visual) provide accurate results, but only on a relatively small data set.

The system developed at the database research group of ETH Zürich aims at enabling multimedia retrieval in huge image databases with more than one million of images using the best descriptors available. The searching and indexing component is based on the so-called vector approximation file (VA-File), which efficiently

performs a nearest neighbor search to identify similar images for a given set of sample images.

(a) (b)

Figure 1.1: Two image of the same dog having a completely different background.

### 1.2.2 Motivation for this Work

The current system provides fast search on the image features extracted from whole images or images with predefined, static regions. Unfortunately, this approach is often not specific enough to describe the content of the images. E.g., take an image that contains a dog jumping over a fence. If we were only searching for dogs, the current system would hardly find the same dog standing on a rock (cf. figure 1.1). The two images only share some color-textural features and the dog is not at the same position in both images.

The system developed in this thesis first segments images into regions that correspond to the objects in it. The search engine then looks for for images which contain regions similar to the ones selected in the query image. In our dog example the images are divided into regions of the dog and regions of the background. The user can explicitly specify the regions of the dog to define his query. The system returns images that contain this dog ignoring the background regions. Due to the more precise queries and the finer granular regions, the system developed in this thesis significantly improves the overall retrieval quality (precision and recall) compared to the former system.

# Chapter 2

# Image Similarity Search

The search for similar images in large-scale image databases has been an active research area in the last couple of years. A very promising approach is content based image retrieval (CBIR). In such systems, images are typically represented by approximations of their content. Typical approximations consist of statistics and fourier or wavelet transformations of the raw image data. This so called *feature extraction* aims at extracting information that is semantically meaningful but needs a small amount of storage. A detailed description of feature extraction can be found in section 2.1.

The information gained by feature extraction is used to measure the similarity between two images. All images are represented by a point in the high dimensional feature space. Each extent of the feature corresponds to one dimension of the feature space. A metric is defined to calculate the actual similarity between two of these points. An overview of common metrics is given in section 2.2.

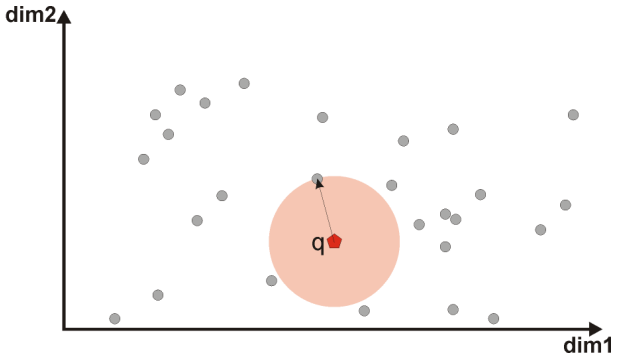



Figure 2.1: The nearest neighbor in two dimensions.

In this basic model, the search for images similar to a query image $q$ results in finding the $k$ nearest neighbors of $q$ (cf. figure 2.1). The model can be extended to support more complex queries that can consist of more than one query image and more than one feature type. *Distance combining functions* are introduced to

combine the distances of the sub-queries into an overall distance. The extended query model is explained in section 2.3.

For fast retrieval, an indexing structure based on the query model is developed. In section 2.4 we present the VA-File, the sequential file based indexing structure that is used in ISIS.

## 2.1 Feature Extraction

Feature extraction is a means of extracting compact but semantically valuable information from images. This information is used as a signature for the image. Similar images should have similar signatures. If we look at the Capitol shown in figure 2.2, the white color and the texture of the building are characteristic properties. In a similar way the sky can be described by its blue color. Further more we can take the size of the objects on the image into account.

Figure 2.2: The capitol in Washington D.C.

We often divide the extractable features into primary features, i.e. the raw image data and secondary features, i.e. features derived from this data. Secondary features may be categorized into primitive features, i.e. features computed from the raw image data, logical features, i.e. the identity or name of an object, and abstract features, i.e. the environment in which or the cause why the image was made. In this work we will focus on primitive features, since the extraction is relatively simple and fast. The extracted features are usually stored in a $n$-dimensional vector, where $n$ depends on the type of the feature.

In the following subsections commonly used features are described. The extraction algorithms for most of these features have been integrated into the ISIS system.

### 2.1.1 Color

Color has proven to be a very discriminant feature for object recognition and image similarity search on photographic images. Often, color histograms are used to describe the dominant colors of an image. A color histogram consists of a fixed number $n$ of reference colors $r_1..r_n$ that are usually equally distributed over the

color space. For each pixel in the image the perceptually nearest $r_i$ is found and the statistic on this reference color is updated (cf. figure 2.3).

Figure 2.3: The example of a color histogram.

If we use histograms, we often have to deal with very high dimensional vectors to get a good description of the image's content. Another problem is that for histograms relatively complicated similarity measures have to be used, since the similarity of two reference colors has to be considered. Therefore so called color moments are preferable for image retrieval. In the $L^*a^*b^*$ color space for example, the following moments are used: $variance(L^*)$, $variance(a^*)$, $variance(b^*)$, $covariance(L^*, a^*)$, $covariance(L^*, b^*)$ and $covariance(a^*, b^*)$, $mean(L^*)$, $mean(a^*)$, $mean(b^*)$ [Web01].

### 2.1.2 Texture

Texture describes the direction and granularity of the structuring elements of a region, e.g. its lines. Examples of simple textures are shown in figure 2.4. Simple texture extraction algorithms compare the image region with predefined texture patterns. More sophisticated algorithms use fourier or wavelet transformations to extract textural features. In the case of fast fourier transformations, Gabor filters are used to extract specific directions of the texture in the image (cf. figure 2.5).

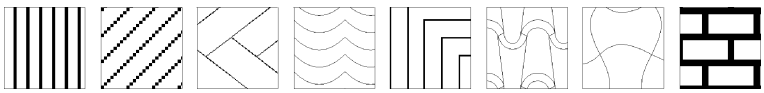


Figure 2.4: Simple textures as they can be found in almost every paint program.

### 2.1.3 Shape

Shape features are mainly used for technical images where the number of colors is small and the shape of the objects in the image is relatively well defined. One approach looks at the directions in the image. The directions are found by extracting lines of the image using an edge detector. Another approach does statistical evaluation on the size of regions using underlying templates (cf. figure 2.6).

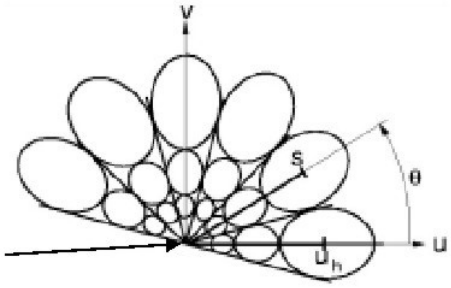



Figure 2.5: Gabor filters are used to extract texture features.

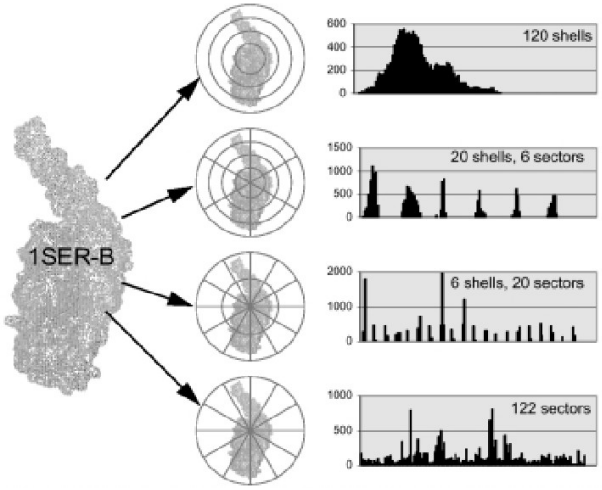



Figure 2.6: Shape histograms using underlying templates [BKK97].

## 2.2 Similarity Measure

The similarity between two images (represented by their feature values) is defined by a *similarity measure* $\sigma(q, p_i) \in [0, 1]$. Value 0 means no similarity and value 1 identity. The similarity is often derived from a *distance measure* $\delta(q, p_i)$. A high value corresponds to a small similarity value and a small distance results in a high similarity. In practice, a *correspondence function* $h$ is used that goes from $h(0) = 1$ to $h(\infty) = 0$ and $h'(x) \leq 0$. The similarity value is calculated as $\sigma(q, p_i) = h(\delta(q, p_i))$.

A widely used distance measure (*metric*) for uncorrelated features is the $L_p$-Norm. The formulas for three $L_p$-Norms are shown in table 2.1. For correlated features we use a quadratic function $\delta(q, p_i) = (q - p_i)^T A(q - p_i)$. The dependency on the axes is taken into account by the matrix $A$.

| | | |
|---|---|---|
| $L_1$-Norm | Manhattan-Norm | $\delta(q, p_i) = \sum_{j=0}^{d} \lvert q_j - p_{ij} \rvert$ |
| $L_2$-Norm | Euclidean-Norm | $\delta(q, p_i) = \sqrt{\sum_{j=0}^{d} (q_j - p_{ij})^2}$ |
| $L_\infty$-Norm | Maximum-Norm | $\delta(q, p_i) = \max_{j=0}^{d} \lvert q_j - p_{ij} \rvert$ |

Table 2.1: $L_p$-Norms

## 2.3 Query Model

With the extracted features and a similarity measure on them we have the basics for similarity search. In the previous sections we have looked at the query as being one point, i.e., one query object. In general this doesn't need to be the case. If we have only one query image the semantics of our query is very limited. In this section we discuss the extension of the simple query model with one query point to a query model that consists of multiple query images and a number of different features. A detailed explanation of ISIS's query model is given in [BMSW01].

**One-Feature One-Object Query (Atomic Query)** The *Atomic Query* is the simplest type of query. It consists of one query image and one feature. To answer the query we have to implement a nearest neighbor search in the feature space of one feature.

**Multiple-Features One-Object Query (Single Query)** A *Single Query* consists of one query image and a number of features. A big number of current content based image retrieval systems statically combine a number of features [LWW00, BCP01]. In our model we aim at allowing the flexible combination of features. A simple approach would be to precalculate all combinations of features. Obviously, this approach would increase the size of our index dramatically. A more practicable solution is to combine the features at query execution time. This means that all feature vectors are read in parallel. The combined vector is constructed in main memory. To have a homogenous feature value space, it is mandatory that the feature values are normalized. In our implementation gauss normalization is used: $\hat{x}_j = \frac{\hat{x}_j}{\sigma_j} - \mu_j$, $\sigma_j$ being the standard derivation and $\mu_j$ being the mean value of the feature vector's $j$-th component. In the case of quadratic functions principal component analysis is used to transform the space. In the transformed space the euclidian metric can be used for distance calculations. The final distance is calculated using a distance combining function. The definition of four distance combining function is given in table 2.2.

**One-Feature Multiple-Objects Query (Multi Query)** A query with only one feature, but more than one query image is called *Multi Query*. The implementation is straightforward: Determine the distance between the current data point and each reference object and combine them using a distance combining function.

| | |
|---|---|
| Fuzzy Standard AND | $\delta(a,b) = \max(a,b)$ |
| Fuzzy Standard OR | $\delta(a,b) = \min(a,b)$ |
| Average | $\delta(a_1,...,a_n) = \frac{1}{n}\sum_{i=1}^{n} a_i$ |
| Weighted Average | $\delta(w_1,...,w_n,a_1,...,a_n) = \sum_{i=1}^{n} w_i * a_i, \sum_{i=1}^{n} w_i \stackrel{!}{=} 1$ |

Table 2.2: Distance combining functions

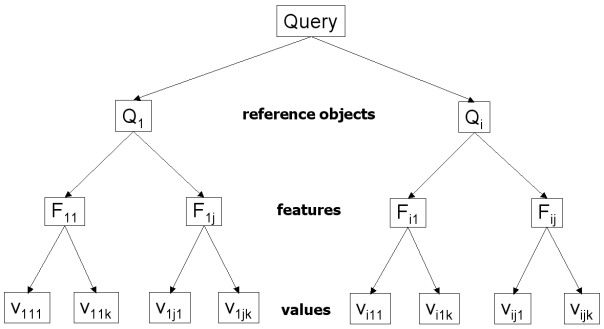



Figure 2.7: Schema of the general query model.

**Multiple-Features Multiple-Objects Query (Complex Query)** The most general case, called *Complex Query*, is illustrated in figure 2.7. It consists of several reference images. For each one a number of features can be used to describe its content. This sort of query can be modelled as a group of single queries. It is common that the same feature type is used for more than one query image. In this case we have to make sure that the feature values are read only once and are shared between the sub-queries.

## 2.4 Indexing Structures

To support fast similarity search and arbitrary complex queries, we need an appropriate indexing structure. Many CBIR systems use the $R-Tree$ [Gut84] and its extensions ($R^+-Tree$ [SRF87], $R^*-Tree$ [BKSS90], $M-Tree$ [CPZ97],...). Even though these tree based approaches are efficient in low dimensional vector spaces, it has been shown that they are outperformed by a sequential scan through the database if the dimensionality of the vector space exceeds about 10 [WSB98].

A new approach, the so called Vector Approximation File (VA-File), has recently been developed at ETH Zürich. It outperforms the tree based indexing structures by explicitly taking into account that random access to a hard disk is far slower than sequential access. It starts with a full scan through a file of approximated vectors to find candidate nearest neighbors. The correct values of the candidate's feature values are fetched in a second step. With these values the correct nearest neighbor can be found. This method reduces both the amount of data scanned and the random accesses to the disk. There are two algorithms that can be performed

on the VA-File each one optimizing either IO or CPU costs.

**Algorithm 2.1** *Nearest Neighbor Search Simple Search Algorithm. (Optimizes CPU costs)*

1. *Let $\mathcal{A}$ be the list of vector approximations and $\mathcal{V}$ the list of vectors, $\forall i$ let $a_i \in \mathcal{A}$ be the approximation of the vector $v_i \in \mathcal{V}$, let $\mathcal{C} = \{\}$ be the candidate set, let $q$ be the query vector.*
2. ***For each** $a_i \in \mathcal{A}$*

   *Get $a_i$ from disk*

   ***If** $bound_{lower}(a_i) <$ k-smallest distance*

   *Get $v_i$ from disk*

   ***If** $dist(v_i, q) <$ k-smallest distance, add $v_i$ to $\mathcal{C}$*
3. *Return the content of $\mathcal{C}$*

**Algorithm 2.2** *Nearest Neighbor Search Nearest Optimal Algorithm. (Optimizes IO costs)*

***Sequential Access Phase***

1. *Let $\mathcal{A}$ be the list of vector approximations and $\mathcal{V}$ the list of vectors, $\forall i$ let $a_i \in \mathcal{A}$ be the approximation of the vector $v_i \in \mathcal{V}$, let $k$ be the number of nearest neighbors and $q$ be the query vector, let $\mathcal{B} = \{\}$ be the set of the $k$-lowest upper bounds, let $\mathcal{C} = \{\}$ be the candidate set.*
2. ***For each** $a_i \in \mathcal{A}$*

   *Get $a_i$ from disk*

   *Calculate the upper bound $u_i$ and the lower bound $l_i$ of the distance between $q$ and $a_i$.*

   *Insert $u_i$ into $\mathcal{B}$ and update $\mathcal{B}$ so that it contains the $k$-lowest upper bounds.*

   ***If** $l_i \leq \max(\mathcal{B})$, add object $i$ to $\mathcal{C}$.*

***Random Access Phase***

1. *Let $\mathcal{W}$ be the set of the $k$-best distances.*
2. *Sort $\mathcal{C}$ in increasing order of the lower bound of $c_i$*
3. ***While** $bound_{lower}(c_i \in \mathcal{C}) <$ k-lowest calculated distance*

   *Get $v_i$ from disk*

   *Calculate the distance $d$ between $v_i$ and $q$.*

   ***If** $d < \max(\mathcal{W})$, add it to $\mathcal{W}$.*
4. *Return the content of $\mathcal{W}$*

# Chapter 3

# Region Based Image Similarity Search

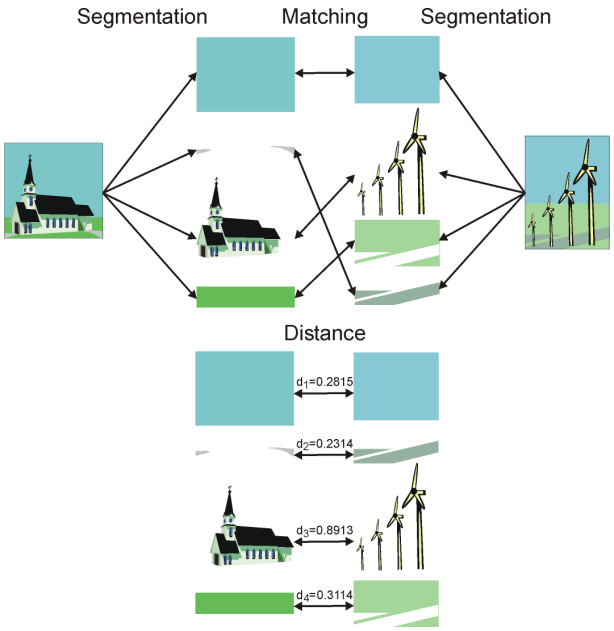
In traditional content based image retrieval, we take a signature for the whole image (cf. chapter 2). In most cases, an image does not only contain one object. If we take signatures for each of these objects, we can describe the image more accurately. A simple approach is to divide the image into predefined regions, e.g., one region in the image center and four background regions (cf. figure 3.1). The drawback of this approach is that the partitioning of the image and the assignment of the regions is static. An image that is not suited for the partitioning cannot be found easily. Newer approaches therefore partition the image dynamically. They use a segmentation algorithm to partition the image into homogenous regions. Segmentation algorithms are discussed in section 3.1. In the ISIS system we currently use the JSEG segmentation algorithm that is described in section 3.1.2.

Figure 3.1: Image with static regions

For each of the regions one or more features are extracted that are taken as signature for that region. We can use the features that we have discussed in section 2.1. Additionally, features that only make sense with segmented images can be extracted. The region based features that have been implemented into the ISIS system are presented in section 3.2.

During the query phase, the assignment of the query image's regions and the regions of the images in the database must be performed. Evidently, regions that are similar should be matched. To calculate the similarity between regions, we can use

one of the metrics defined in section 2.2. To find the "best" assignment, the distances for all possible combinations of regions of the query image and the database image must be calculated. This results in a so called *distance matrix*. A visualization of the matching and distance calculation steps is given in figure 3.2. Algorithms to find a matching are discussed in section 3.3. For the ISIS system we have developed two fast algorithms to find a matching. We present them in section 3.5.




$$distance = w_1 \cdot d_1 + w_2 \cdot d_2 + w_3 \cdot d_3 + w_4 \cdot d_4$$

Figure 3.2: The calculation of the distance using regions.

The query model that was introduced in section 2.3 can be adapted to region based image similarity search. An additional step that calculates the assignment of the regions has to be introduced. The updated query model is presented in section 3.4.

## 3.1 Image Segmentation

The decomposition of an image into regions is the first step in region based image retrieval. For this purpose image segmentation algorithms have been developed which try to segment images into regions that correspond to objects or subjects on the image. More formally, a segmentation algorithm defines an injective function

$$f_{seg} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}, (x, y) \mapsto f_{seg}(x, y) = r.$$

The function assigns to every point $P(x, y)$ of the image the region $r$ it is contained in.

Even though humans can easily divide an image into objects, there are currently no algorithms that can do this job correctly in all cases and as good as humans would do. Computer vision research assumes that humans know the objects that they see and do the segmentation according to this knowledge. Unfortunately there are no algorithms so far, that can do object recognition without having segmented the image previously.

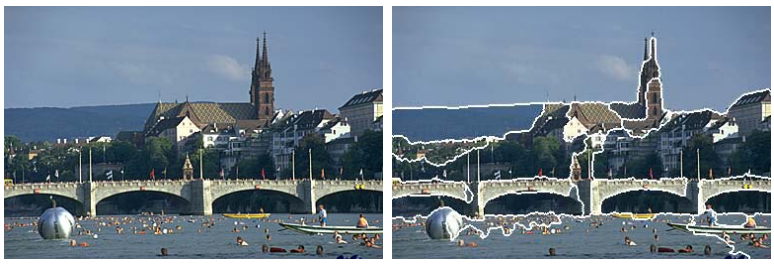


Figure 3.3: The skyline of Basel and the output of the JSEG segmentation algorithm

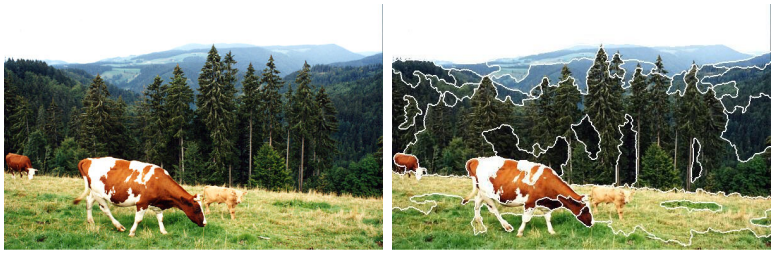


Figure 3.4: Cows in the German black forest and the output of the JSEG segmentation algorithm

In figure 3.3 you can see the skyline of Basel. A human would probably segment the image into the Rhine, the Minster, some villas, a bridge, the Jura hills and the sky. Even though the automatic segmentation finds some of the objects that humans do, it still does not segment the image correctly. The bridge is divided into seven regions and the second tower of the Minster is associated to the region of the sky, just to mention two segmentation errors. If we look at figure 3.4, we can see similar artifacts: The cow in the front is divided into several regions and on the other hand, the cow in the background is in the same region as the grass around it.

There have been proposed several segmentation algorithms. Simple algorithms just use the color of the image pixels and do a bottom up clustering (cf. section 3.1.1). More advanced segmentation algorithms combine the bottom-up approach with some higher level top-down methods that look at the image as a whole (cf. sections 3.1.2, 3.1.3).

### 3.1.1 The k-means Algorithm

This algorithm clusters pixels that have similar feature values. We start with k randomly chosen feature values $m_1 ... m_k$ and assign each pixel to the value $m_i$ ($i = 1..k$) that is most similar to it. After this first step, we redefine $m_i$ as the mean value of the pixels assigned to the former $m_i$, i.e., we calculate the centroid of the pixels assigned to the old $m_i$. This clustering - reassigning $m_i$ phases are repeated until a steady state has been established or a maximum number of iteration steps has been reached.

This algorithm uses only low level features (color, texture, wavelet coefficients). There is no top down view on the image. It does not deal well with smooth color changes of the image, e.g. a sunset over the sea. One possible advantage is that the pixels of one region do not need to be adjacent to each other as depicted in figure 3.5.

Figure 3.5: Example of a segmentation by the k-means algorithm [LWW00]

### 3.1.2 The JSEG Algortihm

JSEG is an algorithm proposed by Deng and Manjunath [DM01] that is based on color quantization and spatial segmentation. The first step quantizes the thousands of colors to about 10 to 20. An algorithm that takes human perception into account is used. Each of the quantized colors is assigned a label. A new image is created in which the image pixel colors are replaced by their corresponding color class labels.

Deng and Manjunath define a novel measure called J-value that is based on statistics on the color classes. Calculating the J-value on a local area of the image can indicate whether that area is at region boundaries or within a region. The higher the J-value the more likely it is that the corresponding pixel is near a region boundary. A small local window in which the J-value is calculated can detect intensity and color edges whereas a large window is useful to detect texture boundaries (cf. figure 3.6 c) and d)). JSEG uses multiple scales (window sizes) to segment an image.

With the use of J-images, the algorithm can find points in the image that are likely to be the region centers. From these ”seed” pixels, a region growing is performed in order to roughly segment the image. After this growing phase follows a merging phase where regions of similar color histograms are merged into one region.

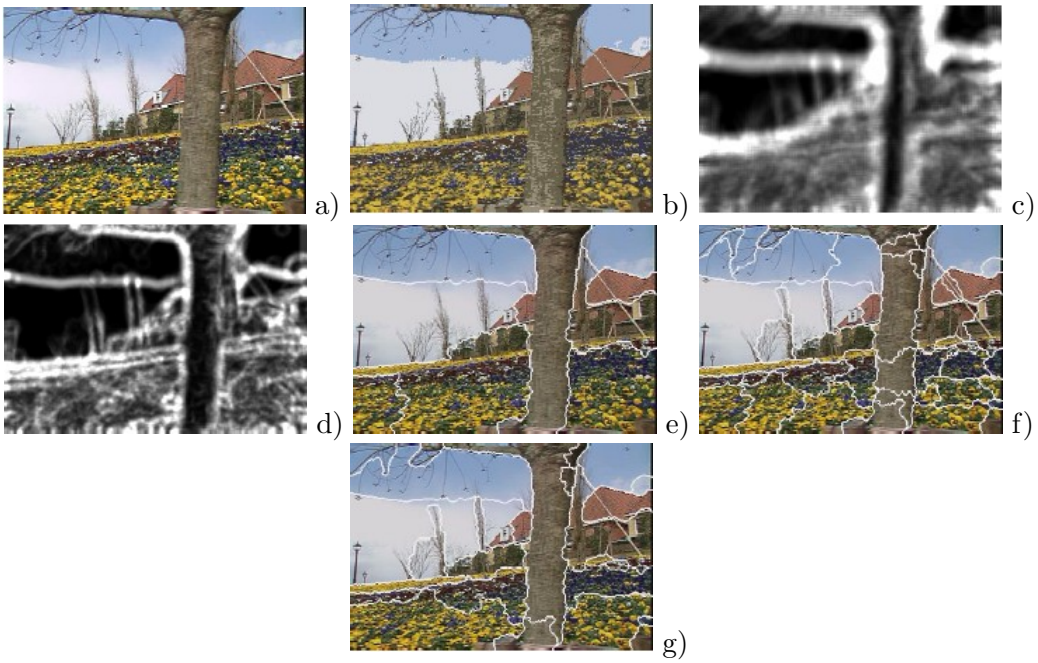


Figure 3.6: Eight steps in the segmentation process of the JSEG algorithm: (a) The original image. (b) Result of color quantization with 13 colors. (c) The J-image at scale 3. (d) The J-image at scale 2. (e) Result after segmentation at scale 3. (f) Result after segmentation at scale 2. (g) The final result after merging. Figure taken from [DM01]

### 3.1.3 Algorithm of Felzenszwalb and Huttenlocher

This algorithm uses a graph theoretic approach, defining every pixel as a node of the graph and trying to partition the graph into regions [FH98]. The partitioning is done in such a way that the resulting segmentation is neither over- nor under-segmented according to the definitions given by the authors. Finally, for each pair of neighboring regions the variation between regions will be larger than the variation within regions. This principle ensures that local and global properties of the image are used and that the segmentation has the desired property of being neither over- nor under-segmented. Figure 3.7 shows an example of a segmentation performed by this algorithm.

## 3.2 Region Based Features

For each region of an image one or more features are extracted. Common features are color distribution and texture (cf. section 2.1). Besides the features that are useful for traditional content based image retrieval systems, region shape and spatial features can be extracted. It is sometimes useful to include spatial constraints in

Figure 3.7: Example of a segmentation performed by the algorithm of Felzenszwalb and Huttenlocher [FH98]

the query, e.g., if you are searching for a house in the left half of the image and a car in the right. In the current implementation of the ISIS system two new features are implemented according to the suggestions made in [SC96].

**Area:** Area is given by the number of pixels of a region divided by the total number of pixels of the image. We therefore have one dimension $d_r^a$ for each region $r$ with

$$d_r^a = \frac{1}{height \cdot width} \cdot \sum_{x=1}^{width} \sum_{y=1}^{height} A(x, y, r)$$

$$A(x, y, r) = \begin{cases} 1, & f_{seg}(x, y) = r; \\ 0, & \text{else.} \end{cases}$$

It is recommended to use the Manhattan norm for distance calculations:

$$\delta_{r_1, r_2} = |d_{r_1}^a - d_{r_2}^a|$$

.

**Location:** Location is defined as the center of gravity of the pixels of each region. We therefore have two dimensions $d_{1,r}^l$, $d_{2,r}^l$ for each region $r$ with

$$d_{1,r}^l = \frac{1}{width \cdot area(r)} \cdot \sum_{x=1}^{width} \sum_{y=1}^{height} L_1(x, y, r)$$

$$L_1(x, y, r) = \begin{cases} x, & f_{seg}(x, y) = r; \\ 0, & \text{else.} \end{cases}$$

and

$$d_{2,r}^l = \frac{1}{height \cdot area(r)} \cdot \sum_{x=1}^{width} \sum_{y=1}^{height} L_2(x, y, r)$$

$$L_2(x, y, r) = \begin{cases} y, & f_{seg}(x, y) = r; \\ 0, & \text{else.} \end{cases}$$

It is recommended to use the euclidian norm for distance calculations:

$$\delta_{r_1,r_2} = \sqrt{\sum_{j=1}^{2}(d_{j,r_1} - d_{j,r2})^2}$$

.

The variables *height* and *width* used above are the image's height and width, respectively. The function $f_{seg}(x, y)$ returns the region of pixel $P(x, y)$ (cf. section 3.1).

## 3.3 Similarity Measure

In region based image retrieval, similarity is calculated as a weighted sum of the similarity between the assigned regions. In order to find a matching, the distances between regions must be computed. This is done using a metric (cf. section 2.2) and results in a $m \times n$ *distance matrix*, $m$ being the number of regions of the query image and $n$ being the number of regions of the database image.

When we have computed the distance matrix, an assignment of the regions must be made. Generally there are four possibilities to do that:

1. We assign each region of the query image to exactly one region of the database image (1:1)
2. We assign each region of the query image to one or more regions of the database image (1:N)
3. We assign one or more regions of the query image to every region of the database image (N:1)
4. We assign one or more regions of the query image to one or more regions of the database image (N:N)

Possibility 2 and 3 are not sensible since they lead to an asymmetric matching. Algorithms of category 4 return good results if the segmentation of the images is not exact, i.e., if the same object is segmented differently in two images (cf. figure 3.8). The problem is to find an algorithm that yields a correct solution, i.e., an algorithm that minimizes the overall distance. By now, no algorithm has been proposed for region based image retrieval that computes a correct N:N assignment. An incorrect algorithm that nevertheless returns good results is integrated region matching (IRM). It is described in section 3.3.1.

In ISIS a perfect assignment of type 1 is used. Finding a perfect assignment is a well studied subject in combinatorial optimization. It is known under the name Assignment Problem (AP). In fact it is the calculation of a minimum weight perfect matching in a bipartite graph. The vertices of the bipartite graph correspond to the regions of each image and the similarity distance between them are the edge weights. To solve the optimization, the Hungarian Algorithm can be used. A formal

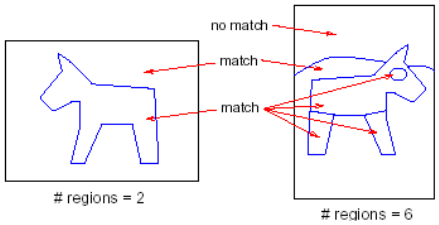



Figure 3.8: Advantage of N:N matching. Figure taken from [LWW00]

definition of the Assignment Problem and the Hungarian Algorithm are given in section 3.3.2.

Unfortunately, the Hungarian Algorithm has a complexity of $O(n^3)$, $n$ being the number of regions. Since we have to calculate an assignment for every image in the database, we have to expect high computational costs. To reduce the number of assignment calculations we have developed two algorithms that work with bounds that are of lower computational complexity. Only for candidate images that are likely to belong to the final result set the correct matching is computed. The algorithms and the calculation of the bounds are presented in section 3.5.

### 3.3.1 IRM: Integrated Region Matching

Since segmentation is often not perfect, the usage of a 1:1 assignment seems to be too strict. If we consider an image of a house where the house is segmented into two regions and one where it is one region, we can only assign one of the two house regions. The assignment solution called IRM, Integrated Region Matching, proposed by Li, Wang and Wiederhold [LWW00], addresses this problem by allowing to match more than one region. Despite the problem that IRM does not guarantee to return the optimal solution, IRM is not able to distinguish between an image with two houses of the same style and an image with one such house. The algorithm can be summarized as follows:

**Algorithm 3.1** ***IRM: Integrated Region Matching***

1. *Let $p_i$ be the significance of region $i$ in image 1 and $p'_j$ be the significance of region $j$ in image 2. Let $s_{i,j}$ be the significance credit of the matching between region $i$ of image 1 and region $j$ of image 2.*
2. *Set $\mathcal{L} = \{\}$, denote $\mathcal{M} = \{(i,j) : i = 1,...,m; j = 1,...,n\}$.*
3. *Choose the minimum $d_{i,j}$ for $(i,j) \in \mathcal{M} - \mathcal{L}$. Label the corresponding $(i,j)$ as $(i',j')$.*
4. $\min(p_{i'}, p'_{j'}) \rightarrow s_{i',j'}$.
5. *If $p_{i'} < p'_{j'}$, set $s_{i',j} = 0, j \neq j'$; otherwise, set $s_{i,j'} = 0, i \neq i'$.*

6. $p_{i'} - \min(p_{i'}, p'_{j'}) \rightarrow p_{i'}$.
7. $p_{j'} - \min(p_{i'}, p'_{j'}) \rightarrow p_{j'}$.
8. $\mathcal{L} + \{(i', j')\} \rightarrow \mathcal{L}$.
9. *If* $\sum_{i=1}^{m} p_i > 0$ *and* $\sum_{j=1}^{n} p'_j > 0$, *go to Step 3; otherwise, stop.*

For this algorithm to run correctly, $p_i$ and $p'_j$ must be set previously. There are two propositions. The first proposition, it is called *uniform scheme*, is to set $p_i = 1/m$ where $m$ is the number of regions. The second proposition that is preferred by the authors of [LWW00], is called *area percentage scheme*: $p_i$ is set to the percentage of the image covered by region $i$. This choice of $p_i$ is less sensitive to inaccurate segmentation than the uniform scheme.

### 3.3.2 The Hungarian Algorithm

The correct solution for solving the 1:1 assignment is the calculation of a minimum weight perfect matching. We can formalize the problem as follows:

**Definition 3.1** ***The Assignment Problem.***
*Given a weighted bipartite graph* $G(X, Y, X \times Y)$ *with weights* $w(x_i, y_j)$. *Without loss of generality we assume that* $|X| \leq |Y|$. *Find for each node* $x_i \in X$ *a distinct node* $y_j \in Y$ *so that the total distance*

$$d = \sum_{i=1}^{|X|} w(x_i, y_j)$$

*is minimized.*

**Definition 3.2** *The resulting set of pairs* $\{(i, j)|x_i$ *is assigned to* $y_j\}$ *is called the* ***minimum weight perfect matching in a bipartite graph*** *or* ***assignment.***

**Definition 3.3** *The weights* $w(x_i, y_j)$ *are defined by a* $|X| \times |Y|$ *matrix* $\mathcal{D}$ *that is called* ***distance matrix.***

The best known algorithm to solve the assignment problem is the so called "Hungarian Method", published by Harold W. Kuhn [Kuh55]. It is based on mathematical theories developed by two Hungarians, D. Konig and E. Egevary. It is based on two observations:

**Observation 3.1** *Each node* $x_i$ *must be assigned to one and only one node* $y_j$ *and vice versa.*

**Observation 3.2** *In the distance matrix* $\mathcal{D}$ *a constant* $c$ *can be added or subtracted from all cost values in a column or all cost values in a row without having any effect on the minimum weight perfect matching.*

Having made these observation we can look at the algorithm of Kuhn.

**Algorithm 3.2** ***Hungarian Algorithm***

1. *Make the distance matrix quadratic by adding rows or columns that have zero cost.*
2. *Find the opportunity distance matrix*
3. *Test for an optimal assignment. If an optimal assignment can be made, make it and stop.*
4. *Revise the opportunity distance matrix and return to step 3*

**Finding the opportunity distance matrix** We start with the distance matrix $\mathcal{D}$ defined in definition 3.3 and subtract the smallest number in each row from every row of the matrix and the smallest number in each column from every column. This can be done due to our observation 3.2. By this step we ensure that there is one zero in each row and column. The name "opportunity distance matrix" comes from the fact that each position with a zero has the opportunity to be member of the assignment.

**Testing for an optimal assignment** To test if an optimal assignment can be made, we draw the minimum number $n$ of horizontal or vertical straight lines on the opportunity distance matrix to cover all the zeros. Each line should cover as much zeros as possible. If $n = \max(|X|, |Y|)$, an optimal assignment can be made.

**Revising the opportunity distance matrix** Find the smallest number $s$ of the actual opportunity distance matrix that is not covered by a straight line. Subtract $s$ from all elements of the matrix not covered by a straight line and add this number to each element lying at the intersection of any two lines.

**Making the final assignment** To make the final assignment, first consider rows and colums that have only one zero entry. Then go on and find an assignment for all other rows and columns.

A very efficient implementation of the hungarian algorithm has been published by Donald Knuth [Knu93]. Examples and a more detailed description of the algorithm can be found in [Heb01].

## 3.4 Query Model

The query model for region based queries is similar to the query model for simple content based image retrieval systems. The same types of queries occur. The main difference is that we have to do the assignment of the regions.

### 3.4.1 Atomic Similarity Query

In queries with one query point and one feature, we scan through all database images. The assignment is calculated for the distance matrix given by the metric that is used for the feature.

### 3.4.2 Single Query

A single query consists of one query image but multiple features. Since we want to do the matching using all features and we want to support mixed queries that not only contain region aware features but also legacy static region based features, we have to distinguish four cases:

1. We have only static region based features
2. We have a number of static region based features and one dynamic region based feature
3. We have only dynamic region based features
4. We have $n$ static region based features and $m$ dynamic region based features, $n > 0, m > 1$

In case 1 we can use the algorithms described in section 2.3 and do not have to calculate a matching. Case 2 is quite similar, we first have to calculated the distance for the object with the region based feature and can use the same algorithm as in case 1. This holds because we don't have to calculate a matching over all features.

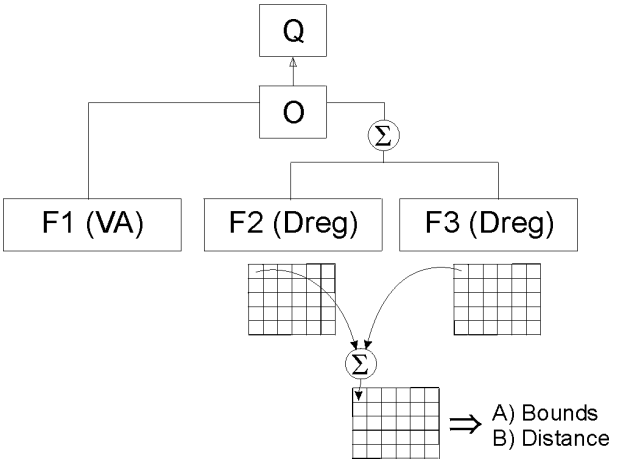



Figure 3.9: The distance evaluation scheme for single queries

Conceptually equivalent to case 1 is case 3. It just consists of a number of region based objects. The main difference is that we have to calculate a matching over all features. First we calculate the distance matrices for all features. Then

we combine the distances in every cell of the matrix using a distance combining function provided by the user (section 2.3). The assignment is then calculated on this combined matrix. The process is shown graphically in figure 3.9.

The most complex and most general is case 4. Since we have to calculate a matching using all dynamic region based features, the calculation is done in two steps. We first calculate the distance for all static region based features and as last step add the distance for all dynamic region based features as a whole. This is allowed since the distance combining functions are associative (see definition in [BMSW01]). Nevertheless, distance combining functions do not need to be commutative and therefore we have to use two different instances of the same distance combining function, one that combines the dynamic region based features and one that combines the static region based features and as last entry the already combined dynamic regions based features. This procedure is shown in figure 3.9.

To make it a bit clearer how to choose the two distance combining functions, let's take the non commutative *weighted average* distance combining function (cf. table 2.2). If we want to weight feature 1 (static region based) with 0.5 and feature 2 (dynamic region based) with 0.3 and feature 3 (dynamic region based) with 0.2 we have to set the overall distance combining function to $(0.5, 0.5)$ and the region based distance combining function to $(0.6, 0.4)$.

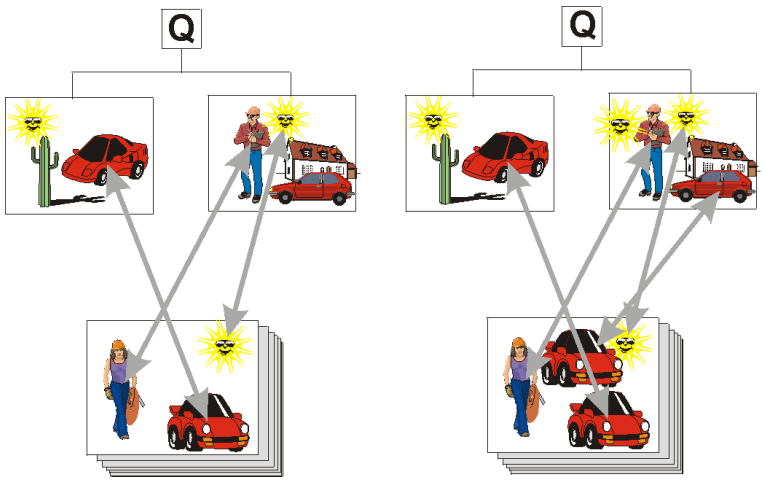



Figure 3.10: Making a matching using the regions of two reference images.

### 3.4.3 Multi Query

A multi query is a query on a number of images and one feature. In principle there are two possibilities:

1. Calculate the matching for every image individually and combine these distances using a distance combining function.

2. Merge all selected regions from all images into one ”pseudo image” and calculate the matching on this image.

If we look at the example given in figure 3.10 we can see that solution 2 is not intuitive in most cases, especially if we use a perfect matching. Let's take for example an image with a red car as our query image $q_1$ and an image with another red car as query image $q_2$. If we have a database image with exactly one red car we can match it with one of the query cars. If the database contained another image with two red cars, the algorithm can match both cars. By having matched more elements, it results in a better overall distance even though we might have just searched for an image with one car. The implementation of solution 2 is quite simple: just combine all selected regions to a ”pseudo image” and run an atomic query with this ”pseudo image” as query point.

If we take solution 1, the search can be done more intuitively, i.e. by using a `AND` distance combining function we would find images that have regions which are similar to the regions selected in all of the reference images. This would lead us to solutions similar to that achieved by solution 2. Nevertheless we would be flexible enough to choose another distance combining function that matches images that are similar to only one of the reference images.

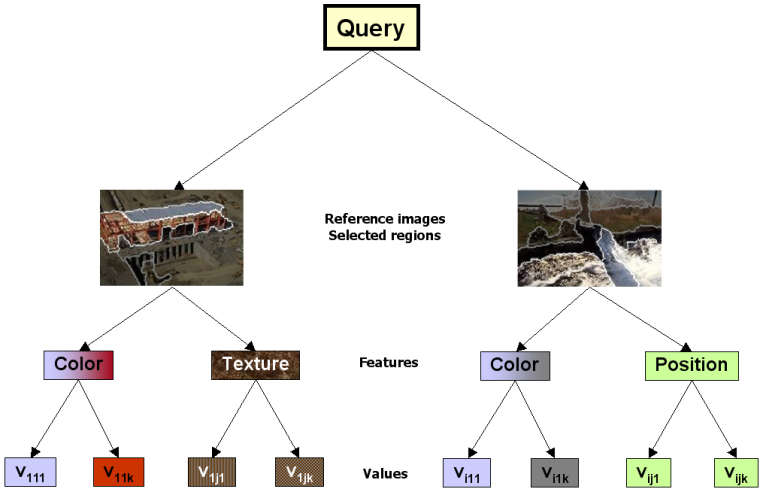



Figure 3.11: Region based complex query

### 3.4.4 Complex Query

The most general case when we have several features and several reference images can be handled in analogy to the case without dynamic regions (see section 2.3). A complex query can be constructed as a set of single queries. Of course we have to take our new version of single queries. Since it is likely that one or more features are used for more than one reference image, the iteration over these features must

be performed only once. Nevertheless the distance matrix calculation must be performed independently for each query image. A schematic depiction of a complex query is given in figure 3.11.

## 3.5 Indexing Structure and Search Algorithms

The index that we use in ISIS is a sequential file. We therefore need a sequential algorithms to perform the image similarity search. A simple algorithm called `ERASE` (Exact Region Assignment SEquential) has been proposed by Bartolini, Ciaccia and Patella in [BCP01]. It scans through the whole image database and calculates the minimum weight perfect matching for every image. In the end the best images are returned. The algorithm is formalized below.

**Algorithm 3.3** *ERASE*
***For each*** *image in the database*

1. *Calculate the distance matrix*
2. *Find a matching*
3. *Calculate the distance of this matching*
4. *keep the best k distances*

*Return the best k images*

It is easy to see that this algorithm has at least two properties:

**Observation 3.3** *It scans through the whole database*

**Observation 3.4** *We have to calculate a matching for each database image*

Looking at observation 3.3, the fact that we have to scan the whole database seems to be a bad property of this algorithm and our sequential data structure. Weber, Schek and Blott showed in [WSB98] that other indexing structures, mainly MBR-based indexing structures like the R-Tree [Gut84], are outperformed by a simple scan through the whole data set if the dimensionality of the data points exceeds about 10. Since we use high dimensional feature descriptors, most of which having dimensionality highly above 10 (e.g. 256 for color histograms), a sequential scan is at least as fast as using a sophisticated indexing structure.

One additional advantage is the easy parallelize-ability and distribute-ability of sequential indexing structures. Subranges of the whole index can be treated independently by many computers (in the distributed case) or many parallel disks (in the parallelization case). A more detailed investigation of the problem shown for the VA-File can be found in [Web97].

Since it seems to be inevitable to scan through the whole database, the two algorithms presented in this thesis aim at minimizing the number of matchings

computations to improve the performance of the query (cf. observation 3.4). Both algorithms use a two phase approach where first an approximation for the matching is built, resulting in a lower and an upper bound of the final distance. Only for candidates that are likely to belong to the result, the exact distance is calculated using the Hungarian Algorithm implemented according to Donald Knuth (cf. section 3.3.2).

### 3.5.1 Two Phase Region Search

The first algorithm presented here is called `ToPhReSe` (Two Phase Region Search). It minimizes the number of matching calculations but needs additional random accesses to the disk and additional distance matrix calculations. It is similar to the Nearest Neighbor Search Nearest Optimal Algorithm described in section 2.2.

**Algorithm 3.4** *ToPhReSe*

***Sequential Access Phase***

1. *Let $\mathcal{V}$ be the list of vectors, let $k$ be the number of nearest neighbors and $q$ be the query vector, let $\mathcal{B} = \{\}$ be the set of the $k$-lowest upper bounds, let $\mathcal{C} = \{\}$ be the candidate set.*
2. ***For each** $v_i \in \mathcal{V}$*

   *Get $v_i$ from disk*

   *Compute the distance matrix between $q$ and $v_i$*

   *Calculate the upper bound $u_i$ and the lower bound $l_i$ of a minimum weight perfect matching using the distance matrix.*

   ***If** $l_i < \max(\mathcal{B})$, add $(i, l_i)$ to $\mathcal{C}$ and $(i, u_i)$ to $\mathcal{B}$.*

***Random Access Phase***

1. *Let $\mathcal{W}$ be the set of the $k$-best distances.*
2. *Sort $\mathcal{C}$ in increasing order of the lower bound of $c_i$*
3. ***While** $bound_{lower}(c_i \in \mathcal{C}) <$ $k$-lowest calculated distance*

   *Get $v_i$ from disk*

   *Compute the distance matrix*

   *Calculate the minimum weight perfect matching using the Hungarian Algorithm*

   *Calculate the distance $d$ between $v_i$ and $q$.*

   ***If** $d < \max(\mathcal{W})$, add $d$ to $\mathcal{W}$.*
4. *Return the content of $\mathcal{W}$*

We have also built a version of this algorithm that caches the distance matrix in memory. A quantitative analysis of the two versions of the algorithm is given in section 5.1.

### 3.5.2 One Phase Region Search

The second algorithm is called `OnPhReSe` (One Phase Region Search). It reduces the number of times the Hungarian Algorithm is invoked, but it directly calculates the minimum weight perfect matching if the vector is a candidate. This has the advantage that we don't have to re-fetch the vector in a second phase which reduces IO costs. On the other hand this has the disadvantage that we have to invoke the Hungarian Algorithm more often.

**Algorithm 3.5** *`OnPhReSe`*

1. *Let $\mathcal{V}$ be the list of vectors, let $\mathcal{C} = \{\}$ be the candidate set, let $q$ be the query vector.*
2. ***For each*** $v_i \in \mathcal{V}$

   *Get $v_i$ from disk*

   *Compute the distance matrix*

   *Calculate the lower bound $l_i$ of a minimum weight perfect matching using the distance matrix.*

   ***If*** $l_i <$ *k-smallest distance*

   *Calculate the minimum weight perfect matching using the Hungarian Algorithm (distance $d_i$)*

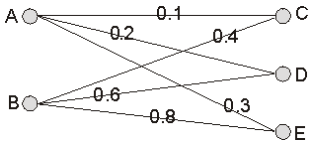
   *Add $v_i$ and $d_i$ to $\mathcal{C}$.*
3. *Return the content of $\mathcal{C}$*

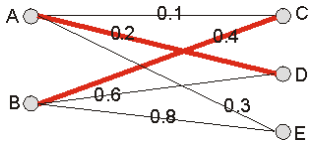
A quantitative analysis of this algorithm is given in section 5.1.

### 3.5.3 Bounds Calculation

The search algorithms demand easy computable bounds of the minimum weight perfect matching. In this section we present two algorithms that calculate an upper and a lower bound, respectively. Throughout this section we will refer to the example given in figure 3.12.

**Lower Bound** The lower bound needn't be a matching. It should be smaller or equal to the final minimum weight perfect matching. In our implementation the lower bound is calculated as the sum of the smallest $\min(n_1, n_2)$ column minima in the distance matrix as formalized in algorithm 3.6, $n_1$ and $n_2$ being the number of regions of the query and database image, respectively. An example is given in figure 3.13. The algorithm has an overall complexity of $O(n^2)$ since the distance matrix computation in step 1 is $O(n^2)$, step 2 is $O(n^2)$, step 3 $O(n \log(n))$ and step 4 $O(n)$.




| | A | B |
|---|---|---|
| **C** | 0.1 | 0.4 |
| **D** | 0.2 | 0.6 |
| **E** | 0.3 | 0.8 |

$$A \to D \wedge B \to C \equiv 0.2 + 0.4 = 0.6$$

Figure 3.12: Example of a weighted bipartite graph, its distance matrix and its minimum weight perfect matching.

**Algorithm 3.6** *Lower Bound*

1. *Let $\mathcal{D}$ be the distance matrix, let $n_1$ be the number of regions of the database image, let $n_2$ be the number of regions of the query image. Let $m$ be the array of the column minima.*

2. ***For each** column $i$ of $\mathcal{D}$ find its minimum entry $m_i$*

3. ***Sort** the array $m$ in increasing order*

4. *lower bound $= \frac{1}{\tau} \cdot \sum_{i=1}^{\min(n_1,n2)} m_i$, $\tau = \tau(n_1, n_2)$ being a normalization value.*

**Theorem 3.1** *The lower bound calculated by the algorithm 3.6 run on a distance matrix $\mathcal{D}$ is a lower bound of the minimum weight perfect matching calculated on $\mathcal{D}$.*

**Proof of Theorem 3.1** *Proof by contradiction.*

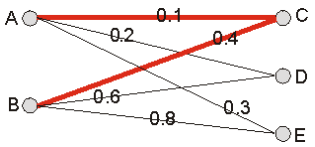
1. *Both the minimum weight perfect matching and the lower bound of algorithm 3.6 consists of $\min(n_1, n2)$ distances.*

2. *Assume, the lower bound of algorithm 3.6 is larger than the minimum weight perfect matching.*

3. *$\Rightarrow$ At least one taken distance $m_i$ is larger than the taken distance $m_i^*$ of the minimum weight perfect matching.*

4. *Contradiction: Since we take the minimum in each column for $m_i$, it is impossible for $m_i^*$ to be smaller $\forall i$.*

■




$$A \to C \wedge B \to C \equiv 0.1 + 0.4 = 0.5$$

Figure 3.13: The lower bound of the matching in figure 3.12.

**Upper Bound** The upper bound is calculated as a good matching that is not the minimum weight perfect matching. It should be computationally simple to calculate this matching. In our implementation we scan the distance matrix column by column and match the smallest distance value of a row that has not yet been assigned. The result could be improved by starting at more than one position and finally taking the smallest distance.

The calculation is formalized in algorithm 3.7. Its overall complexity is $O(n^2)$ since the distance matrix calculation in step 1 is $O(n^2)$, step 3 is $O(n^2)$ and step 4 has a complexity of $O(n)$.

**Algorithm 3.7** *Upper Bound*

1. *Let $\mathcal{D}$ be the distance matrix, let $n_1$ be the number of regions of the database image, let $n_2$ be the number of regions of the query image, let $m$ be the array of matched rows (initialized to **false** $\forall$ entries $m_i$).*
2. *Without loss of generality we assume that we have more rows than columns.*
3. ***For each** column $i$ of $\mathcal{D}$ find the minimum entry $d_{i,j}$ where $m_j$ is **false**. Set $m_j$ to **true**, save $j_i = j$.*
4. *upper bound $= \frac{1}{\tau} \cdot \sum_{i=1}^{\min(n_1,n2)} d_{i,j_i}$, $\tau = \tau(n_1, n_2)$ being a normalization value.*

**Theorem 3.2** *The upper bound calculated by the algorithm 3.7 run on a distance matrix $\mathcal{D}$ is an upper bound of the minimum weight perfect matching calculated on $\mathcal{D}$.*

**Proof of Theorem 3.2** *Informal proof.*

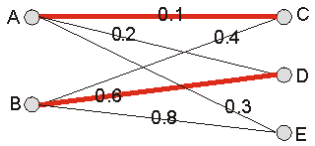
1. *Since we have more rows than columns, it is given that we can match every column with one row. Since we only match rows that have a $m_j =$ **false** and mark all matched rows with a $m_j =$ **true** ($j$ is the row number), we make sure that every row is matched with at most one column. $\Rightarrow$ the algorithm calculates a perfect matching.*

2. *Since a perfect matching has to match exactly* $\min(n_1, n_2)$ *columns with rows, both the minimum weight perfect matching and the perfect matching of algorithm 3.7 consists of the same number of distances.*

3. *Both matchings consist of the same number of distances and the weight of the minimum weight perfect matching is minimum by definition.* $\Rightarrow$ *the distance calculated in algorithm 3.7 is greater or equal to the distance of the minimum weight perfect matching.*

4. $\Rightarrow$ *algorithm 3.7 calculates an upper bound of the minimum weight perfect matching.*

■




$$A \to C \wedge B \to D \equiv 0.1 + 0.6 = 0.7$$

Figure 3.14: The upper bound of the matching in figure 3.12.

# Chapter 4

# Implementation

## 4.1 Component Framework and Libraries

In earlier work, the database research group at ETH Zürich has developed a component framework for multimedia data retrieval. The fundamental layer is a message oriented middleware called OSIRIS (Open Service Infrastructure for Reliable and Integrated proccess Support) that interconnects several autonomous components (cf. figure 4.1). Processes define the flow of messages between the components (cf. section 4.2). For this thesis some of the components had to be updated or added. We describe each of them shortly in the following paragraphs. A more detailed description of the implementation is given in the subsequent sections.

**REE - Region Extraction Engine** Initially there was no component to segment images. Therefore a new component had to be added that extracts regions of a given image. The component additionally provides segmentation data for the java applet that we use to select regions on the frontend. A detailed description is given in section 4.3.

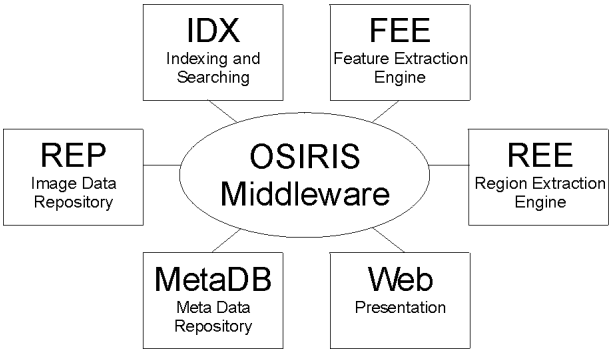



Figure 4.1: Overview on the component framework

**FEE - Feature Extraction Engine** The feature extraction component only provided the extraction of features for images with predefined, static regions. The component has been extended to be able to extract features on the regions of a segmented image. We have also integrated new feature types that take the spatial extent and spatial position of regions into account. We describe the new feature types in section 3.2. More detailed information on the implementation of the FEE component can be found in section 4.4.

**Web - Presentation** The presentation component had to be updated to support region based queries. A java applet has been developed for the purpose of selecting image's regions on the user interface (cf. figure 4.2). The presentation related developments are described in section 4.5.

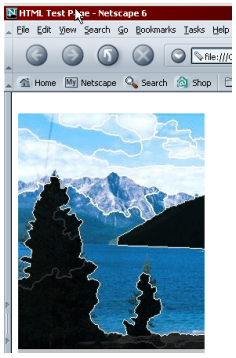



Figure 4.2: The region selection java applet

**IDX - Indexing and Searching** Indexing and searching were based on the VA-File for vectors and SigFile for signatures. The component had to be extended to additionally support region based querying. Due to the flexible implementation, all sort of simple and complex queries can be used with or without dynamic regions. In section 4.6, we describe the component in more detail. An explanation of the indexing structure is given in section 4.7. We discuss the implementation of complex queries in section 4.8.

**MetaDB - Meta Data Repository** The Meta Data Repository stores all information in the system. It had to be updated to save the segmentation of images and the extracted features.

The implementation of the components is based on a number of C++ libraries. Some of them had to be newly created others had to be updated with new functionality.

**Segmentation Library** For supporting image segmentation, a new library has been developed that encapsulates the functionality of segmentation algorithms. This includes the segmentation of images and the creation of a nice representation of the segmented image, e.g. by displaying lines at the region boundaries. For interaction within the component framework, a compressed representation of the segmentation function (which pixel belongs to which region, later referred to as "region map") is used.

**Image Library** The image library had to be slightly updated in order to support dynamic regions (given as a region map) extracted outside the library. Until now, the region map for the statically defined regions was built inside the library. The creation of the region map is now done in the REE component.

**Index Library** The searching and indexing mechanism is implemented in the Index library. Since the searching and indexing process had to be extended to support dynamic regions, new implementations had to be added to this library: A file structure, similar to the VA-File [WSB98], had to be implemented. It is called `DynRegFile` and stores the feature vectors for each region of an image separately. Nevertheless, we can access each image's features with the same position information as in the VA-File. Therefore, we can easily mix up VA-Files and `DynRegFile`s in the search process.

Search algorithms have been implemented that support region based retrieval. They are based on perfect matching (cf. section 3.3) to find the optimal matching between regions. Two search strategies - presented in section 3.5 - have been developed that minimize the number of perfect matching calculation.

## 4.2 Processes

The interaction between components is modelled using processes (workflows). A process is graphically defined using a modelling tool called IvyFrame. To support region based image retrieval a number of processes have been implemented. One process fetches images from the repository an sends them to the REE component. The component extracts the regions and returns a region map. The region map is saved in the MetaDB. In the meanwhile the process sends the image and the region map to the feature extraction component. This component extracts the features and stores them into the MetaDB. Another process is used to run the query. It gets the feature information of the query point from the MetaDB and sends the query to the IDX component that finds and returns a list of similar images. This list is then used to build the answer HTML document using the XSLT component.

| Input message | Output message |
|---|---|
| `ExtractRegions` | `ResultRegions` |
| Name of the segmentation algorithm | The region map |
| Type of the image | The number of regions |
| Data of the image | All input message |
| `BuildRegionImage` | `ResultRegionImage` |
| Region map | The image with edges (jpeg-format) |
| Name of the segmentation algorithm | The applet region map |
| Number of regions | All input message |
| Size of the result image | |
| Type of the input image | |
| Data of the input image | |

Table 4.1: The messages of the REE component

## 4.3 The Region Extraction Component

The region extraction component encapsulates the segmentation of images. It provides two methods: *Extract regions* takes an image and runs the segmentation algorithm on it. It returns the segmentation and the number of regions. *Build region image* takes an image and its segmentation and creates a new image on which the segmentation boundaries are shown as white lines. It additionally creates code to initialize the region selection java applet (described in section 4.5.2). A summary of the messages of the REE component is given in table 4.1.

### 4.3.1 Segmentation

In general, the segmentation can be performed by any segmentation algorithm. In the current implementation we use JSEG, which is further described in section 3.1.2. It builds a good segmentation of images in reasonable time without the need of manual fine tuning. To process an image of 200x133 pixels, our REE component needs less than 2 seconds on a Pentium 4 with 1,8 GHz. Since we had access to the algorithm source code, it has been directly included.

As mentioned before, other algorithms can easily be integrated thanks to a wrapping class hierarchy that encapsulates the implementation details of the algorithm and provides the easy to use interface shown in figure 4.3. A concrete segmentation algorithm has to provide a constructor that takes an image, performs the segmentation and initializes the instance variables of the `ImageSegmentation` class. The `GetImageWithEdges` method can be overwritten in order to provide a individual presentation of the segmentation result. In the current implementation of `ImageSegmentation` this method produces an image with white lines at the region boundaries.


Figure 4.3: UML diagram of the `ImageSegmentation` class

### 4.3.2 The Region Map

The segmentation produced by the segmentation algorithm is stored in a pseudo image in which we store the region number instead of the color for each pixel. Internally, this pseudo image is implemented as an array of `unsigned char`. Since this array maps the pixel positions $(x, y)$ to the respective region number, we calls this structure the *Region Map*.

Even though the region map is very efficient to test the region membership of a point, it is not compact enough to be exchanged over the network and stored in the database. We therefore use a compressed format of the region map externally. The format is shown in figure 4.4. It starts with `"REE1.0"` as identifier followed by a list of (`int`,`unsigned char`) pairs. The `int` value contains the number of occurrences of the region defined in the `unsigned char` value. The end of the region map is marked by a flag (`int`).


Figure 4.4: The `REE1.0` format

The code to initialize the region selection applet (cf. section 4.5.2) is quite similar but encoded as a string to avoid special characters in the HTML document. The encoding is just a comma separated list of numbers. The order and semantic of the numbers is the same as in the previously described format. Since a string has a defined length, we don't explicitly use an end flag.

In figure 4.5 an example picture and its transformations are shown: The region map, the `REE1.0` format and the format used for the region selection applet.

### 4.3.3 Image with Edges

To visualize the regions of an image, we have implemented an algorithms that draws the region boundaries into the image. We call the resulting image *image with edges*. In the current implementation the region boundaries are directly painted into the image by a simple algorithm that linearly scans through the region map and writes

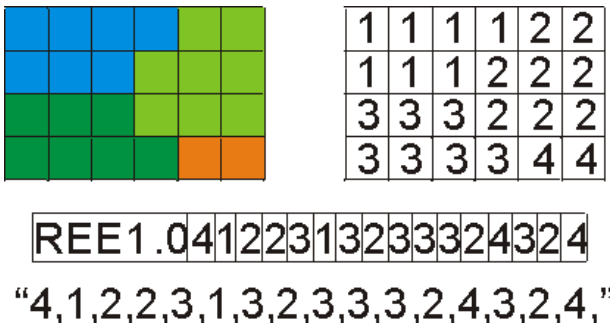



Figure 4.5: Example of the region map and its compressed formats.

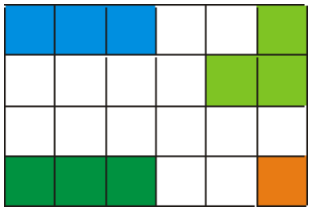


Figure 4.6: The image of figure 4.5 with edges.

white pixels whenever the actual pixel is in another region than the previous or next one. The algorithm has to scan the map both horizontally and vertically. If we take the example of figure 4.5, the fourth and fifth pixel of the first row, pixel one to four of the second row, all pixels of the third row and the fourth and fifth pixel of the last row would be painted white (figure 4.6).

## 4.4 The Feature Extraction Component

The existing feature extraction component had to be extended to support dynamic regions. The existing feature extraction algorithms that have been designed for static regions were reused without change. The feature extractor component FEE is now able to take a region map in `REE1.0` format (cf. section 4.3.2) as an optional parameter. If this parameter is set, the extraction is performed on the regions of the image. There are two different messages depending on the user's choice to extract one or more features. A summary of their parameters is listed in table 4.2.

### 4.4.1 Feature Data Formats

The extracted feature data is stored in two different ways. In the case of static or no regions the extracted features are stored linearly in an array of real numbers. This data structure is called `floatvec`. In the case of dynamic regions we have to store the number of regions internally. Therefore a slightly different format is used called `dynfloatvec`. It is also an array of real numbers but the first entry contains

| Input message | Output message |
|---|---|
| `ExtractFeature` | `ResultFeature` |
| Name of the feature | Type of the feature |
| Type of the input image | Dimensionality of the feature |
| Data of the input image | Value of the extracted feature |
| Region map (only for dynamic regions) | All input data |
| Number of regions (only for dynamic regions) | Image information |
| Priority (optional) | Statistics |
| `ExtractMultipleFeatures` | `ResultMultipleFeatures` |
| Number of features | Number of features |
| Names of the features | Extracted flag (yes/no) |
| Type of the input object (image) | Types of the features |
| Data of the input object (image) | Dimensionality of the features |
| Region map (only for dynamic regions) | Values of the features |
| Number of regions (only for dynamic regions) | All input data |
| Priority | Object information |
| | Statistics |

Table 4.2: The messages of the FEE component

the number of regions. The feature values then follow directly, region by region. An example of a `floatvec` array with four static regions and three dimensions and an example of a `dynfloatvec` array with six dynamic regions and three dimensions is shown in figure 4.7.

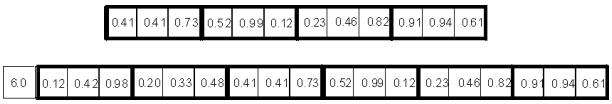



Figure 4.7: A `floatvec` with four static regions and a `dynfloatvec` with six regions, both having three dimensions.

## 4.5 The Presentation Component

The presentation component has mainly been developed by Thomas Moscibroda [Mos02]. It provides an easy to use interface for image similarity search. We describe the extensions made for region based image retrieval in section 4.5.1. For the query interface interface we have developed a java applet that is used to select the query regions. Details on the usage and implementation of the applet are given in section 4.5.2.

### 4.5.1 The Web Interface

The interface of ISIS is currently focused on the old similarity search on static regions. It supports all sort of queries, i.e., atomic-, single-, multi- and complex-queries for static regions (cf. section 2.3). Unfortunately, only atomic region based queries are currently supported on the frontend.

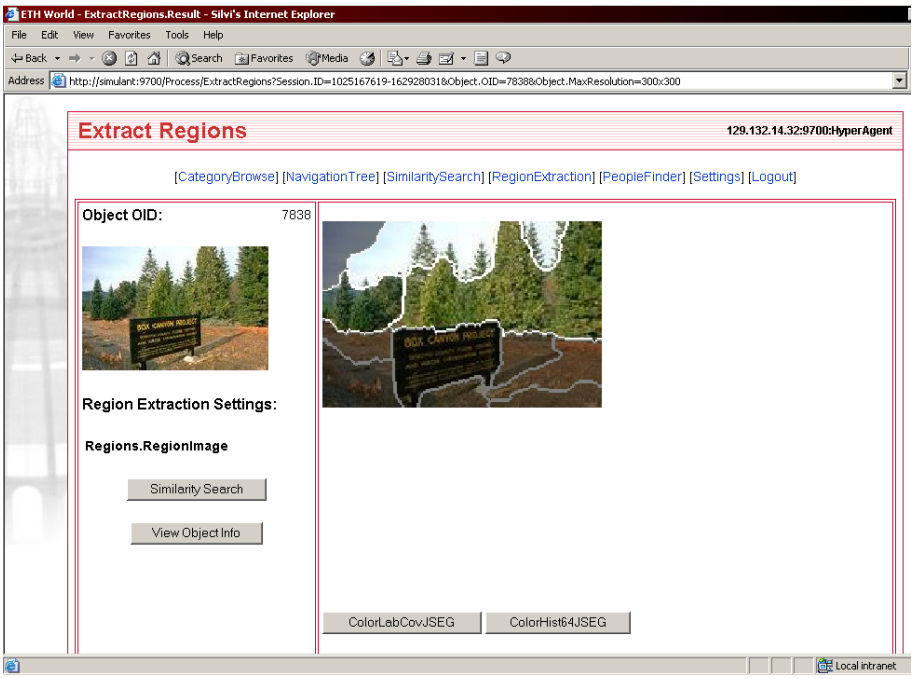



Figure 4.8: The web interface for region selection

In figure 4.8 you can see a screenshot of the query page. The user can click on the image with edges and choose one or more regions by clicking on them. A double click on the image selects all regions. This image is in fact the java applet described in section 4.5.2. After choosing the regions, you can either click on the `ColorLabCovJSEG` button or on the `ColorHist64JSEG button` depending on which feature you want to use for the search. The search is then performed and a ranked list of result images is returned and displayed in the same manner as it was done with the former similarity search without dynamic regions (cf. figure 4.9).

### 4.5.2 The Region Selection Applet

The region selection applet is a java applet based on the java development kit version 1.1.x. This API is implemented in most of the web browsers. Applets are embedded programs in HTML pages and are started and initialized using a special HTML tag `<APPLET>`. As shown in figure 4.10, two parameters are needed to initialize the region selection applet. The parameter `imageName` takes the `CODEBASE` relative URL

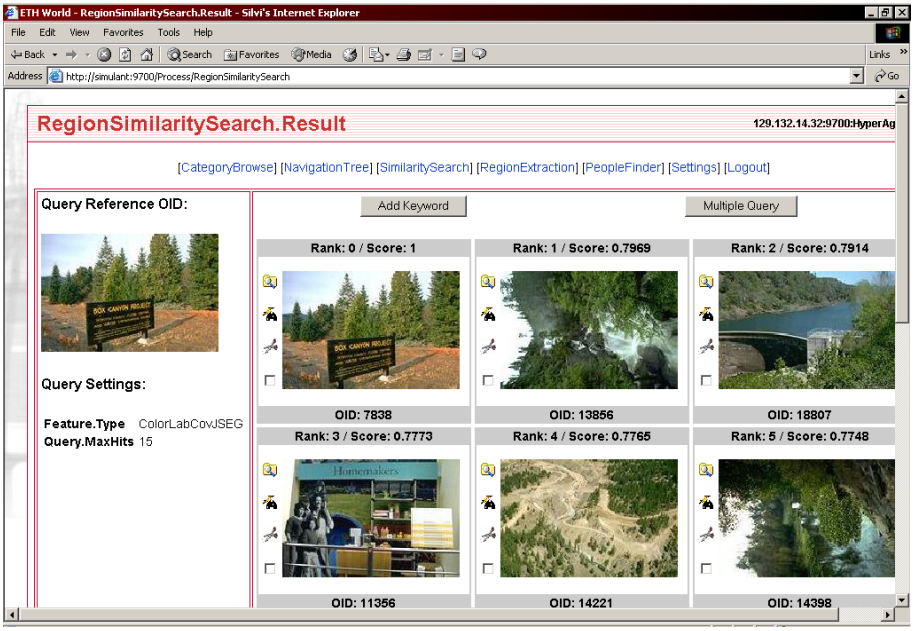



Figure 4.9: The result list

of the segmented image. The second parameter takes an applet specific region map as described in section 4.3.2.

```
<APPLET CODE = "ch.ethz.dbs.regions.RegionSelectionApplet"
        CODEBASE = "regions/classes/."  WIDTH = 180 HEIGHT = 185
        ALIGN = middle NAME = "RegionSelectionApplet" >
  <PARAM NAME = "imageName" VALUE = "seg_darth.jpg">
  <PARAM NAME = "regionMap" VALUE = "37,13,42,10,48,9,23,...">
</APPLET>
```

Figure 4.10: Initialization code for the region selection applet

In the initialization step of the applet, the image is read and displayed and the region map is decompressed. Each time a user clicks on the image, the applet determines on which region the click has been performed and either adds or removes the region from the list of already selected regions.

### 4.5.3 Accessing Applet Information from HTML

It is often necessary to get information from applets. In our case we want to find out which regions have been selected by the user. This can be done using a javascript function in the HTML document. The applet must be named and then javascript

```
<SCRIPT>
   function getSelectedRegions() {
     document.myForm.SelectedRegions.value
          = document.RegionSelectionApplet.getSelectedRegions();
     return true;
   }
</SCRIPT>

<APPLET (...)
  NAME = "RegionSelectionApplet" >
  (...)
</APPLET>

<FORM METHOD="GET" ACTION="query.jsp" NAME="myForm"
      onSubmit="return getSelectedRegions()">
  <INPUT TYPE="hidden" VALUE="xxx" NAME="SelectedRegions">
  <INPUT TYPE="submit" VALUE="submit">
</FORM>
```

Figure 4.11: Getting data from an applet

can access the applet using this name (in figure 4.11 it is `RegionSelectionApplet`). Every public method of the applet can be accessed in this way. To set a hidden value in a HTML form, we can use the `onSubmit` action. The called method (in figure 4.11 it is `getSelectedRegions`) can then set this hidden value. Using this mechanism, arbitrary interaction between the HTML document and the applet can be implemented.

## 4.6 The Index Engine Component

The index engine component (IDX) encapsulates two main tasks. First, it provides the management of the dynamic regions file which is the indexing data structure used for searching on dynamic regions. The dynamic regions file is further described in section 4.7. Second, it encapsulates the execution of similarity search on dynamic regions. The messages of the IDX component are summarized in table 4.3.

### 4.6.1 Management of the Dynamic Region File

The management of the dynamic region file through the IDX component is split into three messages. The *BuildIndexFiles* message builds new and empty files to store index data. Given the name of one or more features and their respective number of dimensions, the component creates for each feature both, a new VA-File and a new

| Input message | Output message |
|---|---|
| `BuildIndexFiles`<br>Number of features<br>Size of the overflow file<br>Name of feature i<br>Dimension of feature i<br>Default norm of feature i | `IndexFilesBuilt` |
| `UpdateFeatures`<br>OID<br>Position<br>Number of features<br>Name of feature i<br>Type of feature i<br>Value of feature i | `FeaturesUpdated`<br>OID<br>Position |
| `DeleteFeatures`<br>Position<br>Number of features<br>Name of feature i | `FeatureDeleted`<br>Position |
| `Query`<br>Desired number of results<br>Minimum score<br>Distance combining<br>and correspondence function<br>Query plan<br>Query hints<br>Reference images and their features and selected regions | `ResultList`<br>Actual number of results<br>OID, distance,<br>similarity and rank<br>of result image i |

Table 4.3: The messages of the IDX component.

`DynRegFile`. This message must be called each time a new feature type is added to the index.

For adding and updating feature values, a message called *UpdateFeatures* exists. It takes the information for one or more features (feature name, data type, data, object identifier) and inserts it at a given position into the index file. Depending on the data type of the feature, it is either inserted in a `DynRegFile` (data type `dynfloatvec`) or a VA-File (data type `floatvec`). If the position in the file is already taken, the data at this position will be overwritten without notice.

To delete entries in all index files, you can use the *DeleteFeatures* message. It clears an entry at a given position in one or more features. Using this message the entry is cleared in both the `DynRegFile` and the VA-File.

### 4.6.2 Querying

The most complex message of the IDX component is its *Query* message. Using this message, queries on the previously constructed VA-Files and `DynRegFile`s can be performed. In the message the number of result images, the minimum score each image must achieve, the distance combining function and the correspondence function can be set. If these values are not set, the component uses the defaults defined in the configuration. To specialize the query, a search method (refer to table 4.4 for a list of valid values for this parameters) and the search range can be set. The query can be further influenced using so called *query hints*. Using query hints the user can define the exact behavior of the underlying search algorithm. The currently available hints are listed in table 4.5.

| | |
|---|---|
| `SCAN` | Linear scan through the index |
| `NOA` | Two phase region search (`ToPhReSe`) |
| `SSA` | One phase region search (`OnPhReSe`) |

Table 4.4: Available search methods in the *Query* message

| **Algorithm** | **Hint name** | **Hint type** |
|---|---|---|
| NOA | UseElevator | `boolean` |
| NOA | CommuteUppBndSeperate | `boolean` |
| NOA | UseBothBounds | boolean |
| all with dynamic regions | MatchingAlgorithm | `string` |
| NOA with dynamic regions | CopyDists | `boolean` |
| all with dynamic regions | UsePenalty | `boolean` |

Table 4.5: List of query hints

The definition of the query point(s) of the similarity search is done by providing one or more reference images. For each of them, one or more features that are either `DynRegFile` or VA-File based can be supplied. Of course the index for these features must be available to the component, i.e., they must have been built earlier using the messages described in section 4.6.1.

To determine whether a feature is VA-File based or `DynRegFile` based, for each feature a feature type that is either `floatvec` or `dynfloatvec` must be given. If the query should be performed only on some regions of an image, the identifiers for the selected regions must be delivered with the message. The component then converts the provided feature vector to make sure that only the selected regions are included in the vector (cf. figure 4.12). For a more fine-grained definition of the query point, feature weights, distance combining functions, dimension weights and metrics can be defined as desired.

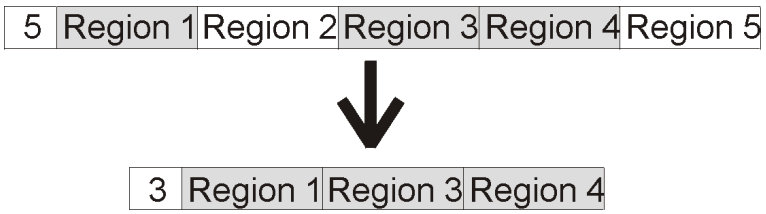



Figure 4.12: Conversion of the `dynfloatvec` data if only some regions are selected

## 4.7 The Dynamic Regions File

To support fast searching, an index of precalculated feature values is needed. For static region and whole image features the ISIS system uses a sequential file called VA-File (described in sections 1.2 and 2.4). For compatibility reasons and due to its suitability for parallel execution, we have developed a similar file, called `DynRegFile`, that can store the feature values of a variable number of regions for each image entry.

In contrast to the VA-File, we always have to scan through the real data and cannot use approximations, since we have to find a minimum weight perfect matching to assign the regions of the query to the regions of the database image (cf. section 3). Since finding a matching is computationally intensive, we have to minimize the number of runs of the matching algorithm. Using vector approximations would require to calculate the matching for the lower bound, the upper bound and the final solution, an approximated calculation of the bounds using a simpler algorithm would either be incorrect or return too many candidates for the random access phase. We therefore always scan through the real data and use the algorithms described in section 3.5 to further minimize the number of matching calculations.

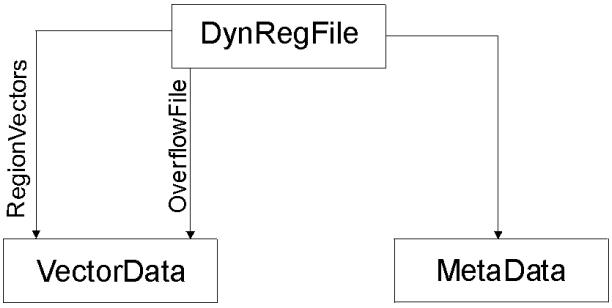



Figure 4.13: UML diagram of the `DynRegFile`.

### 4.7.1 Organization of the File

The `DynRegFile` is based on two classes developed earlier for the VA-File: `MetaData` and `VectorData`. The `VectorData` class manages the storage of equal length vectors

and their object identifier (OID) in a sequential *vector file*. The `MetaData` class stores information about the distribution and correlation between the dimensions of the vectors stored in an associated `VectorData` class. The `MetaData` instance of each `DynRegFile` is associated to the `VectorData` instance that holds the region vectors. To support updates without having to reorganize the file each time, we use an overflow file of small, fixed size that should always remain in main memory. If the overflow file is full, the information in this file is copied into the main region vectors file (an UML diagram is given in figure 4.13).

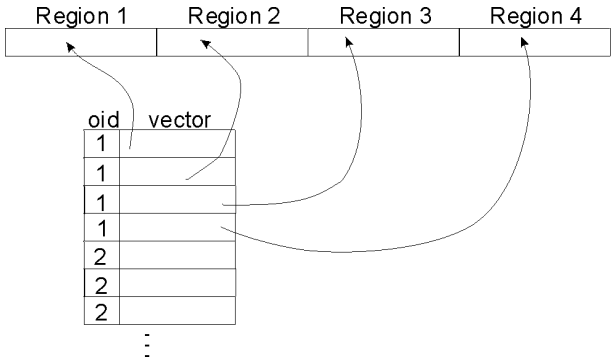



Figure 4.14: The consecutive dynamic regions are mapped into one vector

The features of each region are stored as consecutive vectors in the underlying vector file. All of them have the same object identifier (OID). When the user requests a certain region vector, it is recreated "on the fly" by gluing together all consecutive vectors with equal OID. In figure 4.14, a request for OID 1 is shown. To have the same region numbering as in the region map, we have to assure that the $i$-th vector with OID $j$ contains the data for the $i$-th region of the region vector with OID $j$.

### 4.7.2 The "Lazy" Position Cache

If we have complex queries (cf. section 3.4), it is necessary that the feature vectors of all images are at the same position in every file. Above all, if we mix VA-File based features with region based features, our search algorithms need to be able to access objects at the same position in every index file. Since in the `DynRegFile` each image needs more than one position (one for each region), we have to provide a way to convert from our *internal position* to the *external position* and vice versa.

The solution built in the `DynRegFile` is to use a global conversion cache. In this cache we have a mapping from external positions to the real positions in the vector file. This cache is referred to as *"lazy" position cache* since it is not initialized until a scan through the `DynRegFile` is performed. Earlier, the cache is marked as invalid. If someone wants to access a vector directly by its external position without first having scanned through the file, the cache is built at this time through an implicit scan. If there are changes in the file, the cache may become invalid. Since our search

algorithms always first scan through the whole file, we can be sure that the ”lazy” building of the position cache is preferable in most cases.

### 4.7.3 The Overflow File

The overflow file is organized as a memory mapped file, i.e. it is virtually held in memory, physically sometimes swapped out on disk (managed by the operation system). In our case we want to ensure a relatively small size of this file so that it will always be mapped to memory and act as a buffer. The overflow file has a fix maximum number of entries. Whenever the overflow file is full, a total reorganization of the `DynRegFile` is performed (cf. section 4.7.5). The file is then empty and can be refilled.

### 4.7.4 Methods of the DynRegFile

It follows a description of the methods used to manage the `DynRegFile`. We describe what they do and how they are implemented.

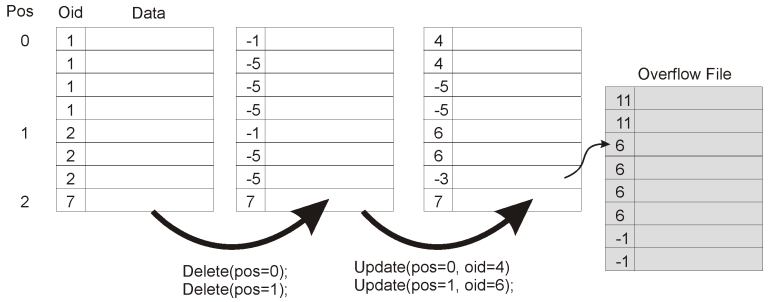



Figure 4.15: Delete and update operations on the dynamic regions file.

**Create** The `Create` method builds the vector file, the overflow file and the meta data file and initialized them. It also sets the maximum number of entries in the overflow file.

**Add** The `Add` method adds a given region vector at the end of the file. It does not check if there are free entries that could be filled earlier in the file. The region vector is split into a vector for each region and each of these vectors are stored in the vector file in consecutive order. All of these vectors are labelled with the same OID. We call them a *vector group*.

**Delete** The `Delete` operation is performed by invalidating the respective vectors. This is done by deleting the first entry in the underlying vector file, resulting in a vector that is empty and labelled with `OID_UNKNOWN` (-1). All OIDs of the vectors of the same image are set to `OID_FREEREGION` (-5). By doing so, we can easily

distinguish the beginning of a new vector group from the deleted vectors of a vector group. The effect of delete operations is shown in figure 4.15.

**Update** The `Update` method first deletes the entry that is updated using the `Delete` method. After that, the region vector is split into its parts as with the `Add` method. These parts are filled into the now free slots in the vector file.

If the number of regions of the new image is equal to the number of regions of the old image, we just replace the old values. If it is smaller, the unused vector slots will remain unchanged (set to `OID_FREEREGION`, cf. figure 4.15). If the number of regions of the new image is bigger than the number of regions in the old image, obviously not all vectors fit into the old slots. In this case, we fill all slots until the last free slot. This slot is filled with a reference to the overflow file with an OID of `OID_OVERFLOW` (-3). The first entry of this vector's data field is set to the next vector's position in the overflow file. Then the overflow file is filled with the remaining vectors of the group starting at this position (cf. figure 4.15). If the overflow file exceeds its maximum size during this update, a reorganization of the whole file is triggered by the `Update` method (the reorganization is described in chapter 4.7.5).

**GetVector** The `GetVector` method must first translate the external position to the internal position using the "lazy" position cache described in section 4.7.2. After having found the right position in the file, the data is read, vector by vector until the next vector has another OID. Of course we eventually have to switch to the overflow file. Having read all vectors, we glue them together into one big region vector that also includes the number of regions used.

### 4.7.5 Reorganize

The reorganization completely rebuilds the whole file. It first closes the working file, renames it and re-opens it under this temporary name. For this to work, all iterators on the file must have been closed, i.e., all references have to be deleted. We therefore test if this is the case. Otherwise there are open file handlers and the renaming of the files is denied by the operating system. In the next step, the underlying meta- and vector files are recreated and refilled using an iterator on the old file.

### 4.7.6 The Iterator

The iterator is used for the scan through the `DynRegFile`. It has to be aware of the special structure of the `DynRegFile`. The implementation is similar to the one of the `GetVector` method discussed in section 4.7.4. If we are at the end of the file, the iteration is stopped. As described in section 4.7.2, with each step in the iteration process, the mapping of external positions to internal positions is updated in order to support random accesses after a scan.

## 4.8 Region Based Complex Queries

### 4.8.1 The Class Hierarchy for Similarity Queries

The query implementation is encapsulated in three main classes: `QueryExecution`, `Query` and `DistanceFilter`. Externally only the `Query` has to be instanced. It is initialized with an index file (subclass of `Index`, e.g., class `DynRegFile`) and query specific information like the query points, weights, etc.

On execution time, the query is specified with a collection of hints (defined as subclasses of the `ExecutionHint` class) that determine the search algorithm and other execution strategies. Using these hints, an instance of `QueryExecution` and an instance of `DistanceFilter` are created. The `QueryExecution`'s `Execute` method is then called that invokes methods on the `DistanceFilter` according to a concrete search strategy. One set of methods of the `DistanceFilter` implements the iteration on the index, the other set of methods encapsulates the bounds and distance calculation.

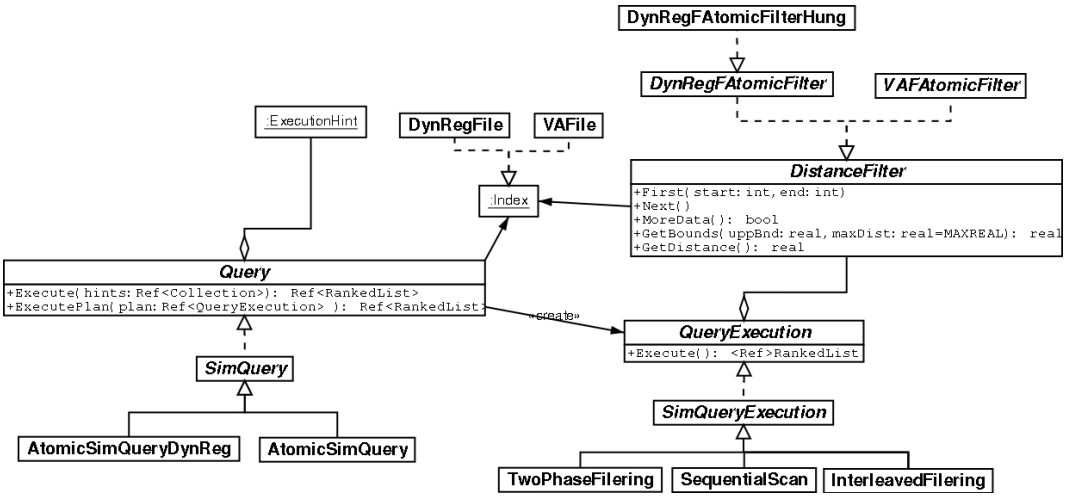



Figure 4.16: Excerpt from the class diagram

### 4.8.2 Implementation of Complex Queries

**Atomic Similarity Query** The atomic similarity query is spread through three classes: `AtomicSimQueryDynReg`, the abstract class `DynRegFAtomicFilter` and its implementing subclass `DynRegFAtomicFilterHung`.

The `AtomicSimQueryDynReg` class is mainly responsible for the creation of the query plan and the initialization handling. The initialization is done by providing the feature values of the query point as an array of real numbers and the number of regions. Optionally, weights for each dimension can be set. Inside the `AtomicSimQueryDynReg`'s `Execute` method the provided hints are used to construct

the right `QueryExecution` object and the right `DistanceFilter`. Currently, only a distance filter using the Hungarian Algorithm is available.

The handling and iteration over the index data structure is encapsulated in the `DynRegFAtomicFilter` class. Furthermore, the calculation of the distance matrix (cf. section 3.5) is implemented in this class. In the search process we can take advantage of a maximum distance that is provided by the search algorithms in the `QueryExecution` subclasses. In the case of the Hungarian Algorithm we can abort the distance calculation, if the lower bound of the minimum weight perfect matching is above this maximum distance.

The Hungarian Algorithm and the calculation of its bounds (as described in section 3.5.3) are encapsulated in the `DynRegFAtomicFilterHung` class. In a future version, another subclass could be added to integrate other matching algorithms into the search framework (e.g., IRM, cf. section 3.3.1).

**Single Query** Single queries are implemented in the `CombinedFilter` class. It is a container of atomic queries. The results of the atomic queries are combined using a distance combining function (class `DistancingFunction`). For region based features, the `CombinedFilterDynReg` class is used to calculate a matching over a number of features. Currently only an implementation for the Hungarian Algorithm exists (`CombinedFilterDynRegHung` class). All sorts of single query can be constructed using `CombinedFilter` and `CombinedFilterDynReg`.

**Multi Query** The multi query is implemented as a container of atomic queries. The total distance is calculated using a `DistancingFunction`.

**Complex Query** The complex query is implemented as a container of single queries. The total distance is calculated using a `DistancingFunction`.

# Chapter 5

# Results

## 5.1 Efficiency

The measurement has been performed on an Intel Pentium 4 with 1.8 GHz, 256 MB RAM, and a hard disk that has a mean data throughput of 50 MB/s. The operating has been Microsoft Windows 2000 Server with Service Pack 2 installed.

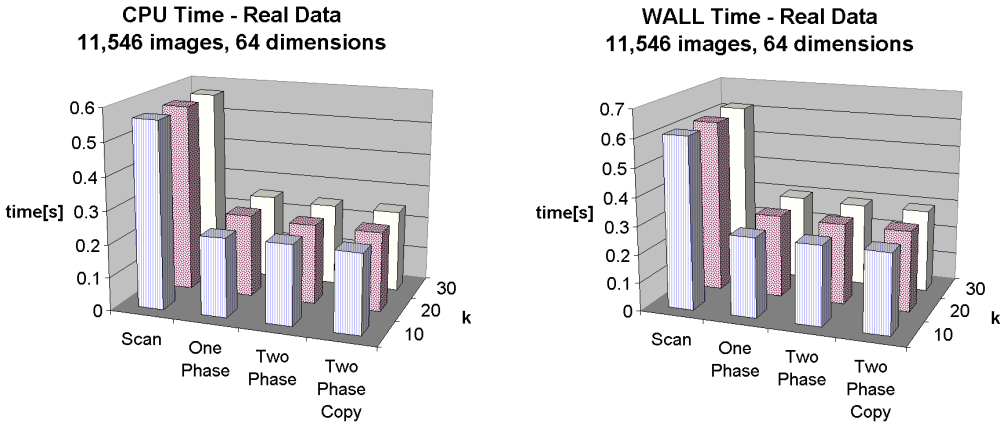



Figure 5.1: Used time, real data

### 5.1.1 Real Data

The measurements in this section have been performed on a data set of 11,546 general purpose images (CHARIOT collection). We have used a 64-bin color histogram feature. The regions have been extracted using the JSEG algorithm (about 10 regions per image). The evaluation has been performed on 100 images taken from the data set (all regions selected).

The CPU time and the total execution time (WALL) are shown in figure 5.1. In this evaluation, our algorithms are more than 2 times faster compared to a sequential scan. They perform equally well, i.e., their execution time differs only in

a few milliseconds. We can see that the number of result images has small influence on the used time. Since we always scan the whole data set, the additional amount of candidates seems not to influence the search time dramatically.

### 5.1.2 Random Data

Due to the limited availability of real data, a more detailed investigation on the algorithms has been performed using synthetical data. The data has been generated randomly. Each entry in the `DynRegFile` has an equal number of regions. The evaluation has been performed on 100 images taken from the data set (all regions selected).

In figure 5.2 we can see that the algorithms perform very well. They reduce the used time (both CPU and WALL) by a factor of 2 to 3. The larger the data set the better the algorithms perform. We can further see that the one phase algorithm (`OnPhReSe`) outperforms the two phase algorithms (`ToPhReSe`), notably for larger data sets. It seems that the additional load of the second phase cannot be gained back by the smaller number of Hungarian Algorithm runs. Both versions of the two phase algorithm (with and without keeping the distance matrix in memory) perform almost equally.

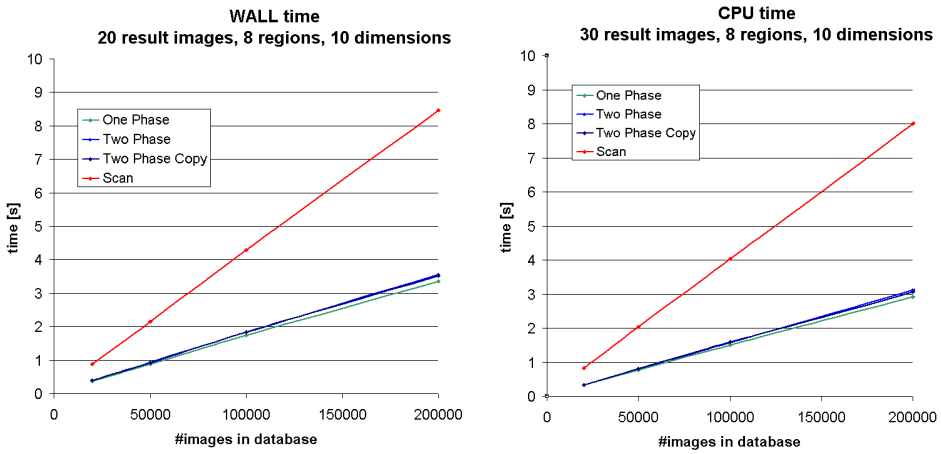



Figure 5.2: Used time, random data

In figure 5.3 the bounds are evaluated (logarithmic axes). The selectivity is extraordinary good. It even increases with bigger data sets.

Figure 5.4 shows the influence of the number of regions. We can see that the used time increases more than linearly for all algorithms. This is plausible since the complexity of the Hungarian Algorithm is $O(n^3)$. The one phase algorithm (`OnPhReSe`) performs best. One reason for this behavior is that the number of candidates in the first phase of `ToPhReSe` increases faster than the number of candidates in the one phase algorithm.

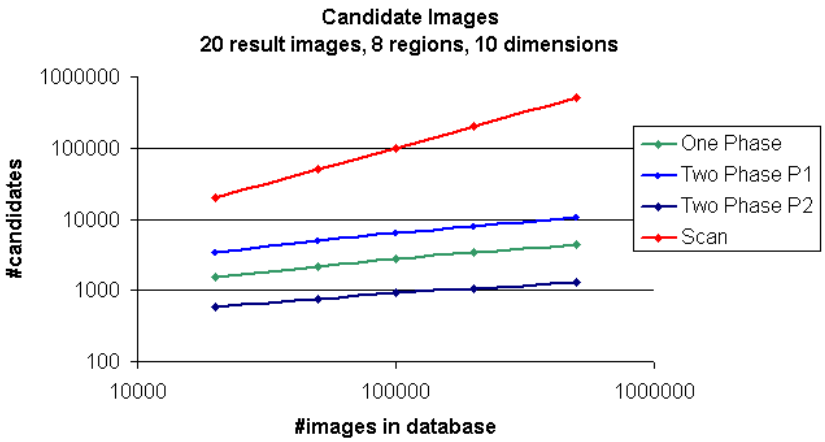



Figure 5.3: Number of candidate images

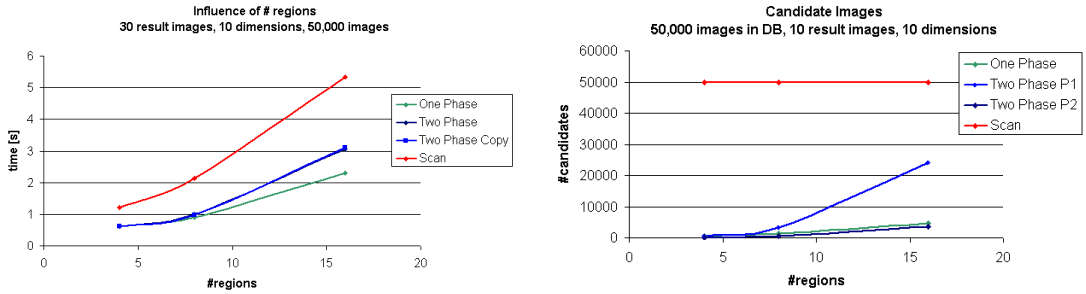



Figure 5.4: Number of regions

All algorithms show a similar behavior with regard to the increase of dimensionality (cf. figure 5.5). This is a good property compared to tree-based approaches that usually perform even worse than a scan with increasing dimensionality.

## 5.2 Effectiveness

The system's effectiveness has been tested on a medium size image database of 11,546 general purpose images. The results are compared to the former CHARIOT system that used five static regions. Only one feature ($L^*a^*b^*$ color covariance) and one reference image have been used for this evaluation. The query set consisted of 37 randomly chosen images.

There were only four images on which the region based retrieval yielded a smaller number of relevant images (11%). In 10 cases using all regions of the image led to a worse result than using static regions (27%). It depends on the image how many regions have to be chosen to yield the best results.

The experiments showed that the algorithms perform extraordinary well if only

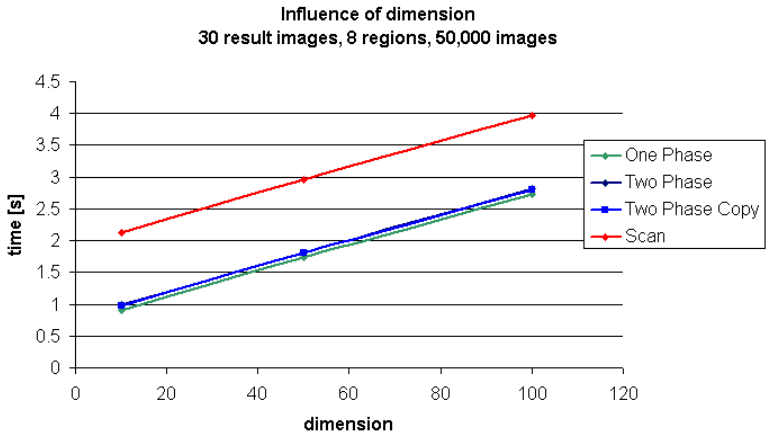



Figure 5.5: Number of dimensions

a portion of the image is relevant to the query. E.g., if we have an image with an oasis in the desert and want to find a lake, we can just select the water to find images in which water plays a role. The more the objects in the image differ, the better is the retrieval quality. The user can therefore define his or her query more precisely due to the more accurate segmentation.

One bad thing is inaccurate segmentation. First of all, when the same object is segmented differently in distinct images. This occurs very often with images of animals, notably when the image is take from a very near distance. JSEG seems not to handle textured areas well enough. Perhaps the use of another segmentation algorithm or a manual adjustment of JSEG's parameters could decrease this problem. One example of inaccurate segmentation is given in figure 5.6. In one image the deer is part of the region of the roadside (b) and in the other one it is part of the street (c). The main color of these regions are completely different. For that reason, the regions are not matched.

(a) (b) (c)

Figure 5.6: Example of different segmentation

We have found out that using all regions of the image sometimes leads to worse results than using static regions. Additionally, the more regions you choose the slower the matching algorithm is. The main problem with too many regions is that it's hard for the algorithm to match all regions. Notably if an object is split in

two segments in one image when it's in one region in another image. It's therefore often difficult to have all regions matched if the search is done on an over-segmented image. In most cases, deselecting regions that are similar to already selected regions leads to a better result.

We show the details of this evaluation in appendix C. Further examples can be found in appendix B.

## 5.3 Summary

The system described in this thesis mostly returns better results than search with static regions. It performs very well in finding parts of an image in other images. The drawback is the increase in computation time compared to static regions, first because we don't use approximations, second because the dynamic combination of the regions into one big vector is relatively expensive and third because we have to calculate a matching of the regions.

# Chapter 6

# Related Work

## 6.1 Blobworld

Blobworld is a system developed at UC Berkeley that treats images as an ensemble of so called "blobs" that correspond to regions of similar color and texture [CTB+99]. An example of an image and its blobworld representation is given in figure 6.1.

Figure 6.1: Image of a tiger and its blobworld representation

The query may be performed on one or more of the blobs of one or more query images and is based on a 218 color histogram vector and two texture descriptors. At query time, each query blob is associated to its best matching blob in the database image. An overall score is calculated using some fuzzy-logic operators on the scores of the matched blobs. Finally, the images are ranked on their overall score. The system uses a R*-Tree [BKSS90] for indexing. To reduce storage and computation costs, distance is calculated on an approximation of the real color histogram that has been constructed using singular value decomposition.

| | |
|---|---|
| Name | Blobworld |
| Institution | UC Berkeley |
| Paper | [CTB+99] |
| Demo URL | http://elib.cs.berkeley.edu/photos/blobworld/start.html |
| Segmentation | Blobworld |
| Features | Color, texture |
| Matching | Best match |
| Search paradigm | One region |
| Specials | Features user adjustable |
| DB size | 35,000 |

## 6.2 VisualSEEk

VisualSEEK of Columbia University is a content based image retrieval system that also supports queries with spatial arrangements of color regions [SC96]. It indexes the color regions of images using so called color sets. Their size and spatial location (absolute and relative) is later used in the search process. The system uses a quantized HSV color space to partition the image into regions of 116 similar colors. In contrast to color histograms (see section 2.1.1), a color set does not save the relative amount of colors. This reduces the computational costs of the final distance calculation between two color sets.

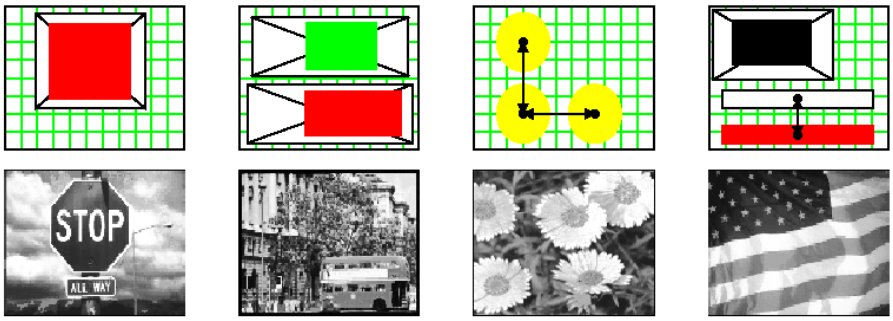



Figure 6.2: Color-spatial queries within VisualSEEK [SC96].

Spatial information is stored as the position of the centroid of each region and the bounding box of each region. A spatial quad-tree [Sam84] is used to index the region centroids and a R-Tree [Gut84] is used to index the bounding boxes. The final similarity measure uses color similarity, relative location, area difference and spatial extent (width and height) between regions. The matching is done by a simple algorithm that is based on the addition of the weighted scores of the best region matches. Examples of color-spatial queries are shown in figure 6.2.

| | |
|---|---|
| Name | VisualSEEk |
| Institution | Columbia University |
| Paper | [SC96] |
| Demo URL | http://www.ctr.columbia.edu/VisualSEEk/ |
| Segmentation | n/a |
| Features | Color sets, shape |
| Matching | Weighted scores |
| Search paradigm | Spatial constraints |
| Specials | Spatial queries |
| DB size | n/a |

## 6.3 Windsurf

The Windsurf system (Wavelet-Based Indexing of Images Using Region Fragmentation) has been developed at University of Bologna, Italy [BCP01]. It uses the k-means algorithm (with a validity function used to chose k) on color and wavelet features to segment images. The querying is done on color and texture features (derived from a Haar wavelet transform) of the k centroids of each image.

Region similarity is determined using a 37-D vector that includes size, wavelet features of the centroid and four $3 \times 3$ covariance matrices. The assignment which region in the query image is matched to which region in the database image is defined to be a minimum weight perfect matching, i.e. each region of the query image is assigned to exactly one region of each database image.

For the final search, two algorithms, a sequential and a index based, are proposed that ensure the matching to be a minimum weight perfect matching. The first algorithm called `ERASE` (Exact Region Assignment SEquential) linearly scans through all images in the database and uses the Hungarian Algorithm (cf. section 3.3.2) to determine the minimum weight perfect matching between the regions of the query image and the regions of the database images. The second, index based algorithm works on a M-Tree [CPZ97], a R-Tree [Gut84] like indexing structure. The algorithm is called $\mathcal{A}_0^{WS}$ and extends Fagin's combining algorithm $\mathcal{A}_0$ [Fag96] to ensure that the assignment of regions is a matching.

| | |
|---|---|
| Name | Windsurf |
| Institution | University of Bologna |
| Paper | [BCP01] |
| Demo URL | no demo |
| Segmentation | k-means |
| Features | Color, Texture |
| Matching | Perfect matching |
| Search paradigm | n/a |
| Specials | Algorithms |
| DB size | 2,000 |

## 6.4 SIMPLIcity

The SIMPLIcity system developed at Stanford university uses the k-means algorithm on blocks of 4x4 pixels to segment the image [WLW01]. The k is chosen adaptively. The search is done using L,U,V color features and the energy of high frequency bands of the Daubechies-4-Wavelet transform as texture features. An automatic procedure can identify non-textured images, from which shape features are extracted additionally.

The matching of the regions is done by an algorithm called IRM (integrated region matching). It supports the matching of more than one region to another in order to be tolerant to inaccurate segmentation (cf. section 3.3.1). Experiments have shown the system to be robust to image alterations such as intensity variation, sharpness variation, color distortions, shape distortions, cropping, shifting, and rotation.

Recently, IRM has been extended to a new measure called FIRM (fuzzily integrated region matching) [LWW01]. It increases the insensitivity to inaccurate segmentation. Each pixel of the image is not strictly assigned to one region, it has a certain probability to belong to a region. The probability is based on the pixel's similarity to the region centroid. From this point of view, each region is a multidimensional fuzzy set of pixels. An image is then a class of fuzzy sets. The similarity measure is finally defined as the overall similarity between two classes of fuzzy sets.

| | |
|---|---|
| Name | SIMPLIcity |
| Institution | Stanford University, Penn State University |
| Paper | [WLW01] |
| Demo URL | http://wang14.ist.psu.edu/cgi-bin/zwang/regionsearch_show.cgi |
| Segmentation | k-means |
| Features | Color, Texture |
| Matching | IRM |
| Search paradigm | Full image |
| Specials | Semantic categories |
| DB size | 200,000 |

## 6.5 NeTra

The NeTra system of UC Santa Barbara uses the so called edge flow algorithm for image segmentation [MM97]. For the newer system called NeTra II the JSEG algorithm has been developed. The search is performed on color, shape and texture features. As shape features, the contour of the regions and a fourier-based shape description are extracted. The texture feature extraction is based on Gabor decomposition.

The indexing of texture and shape features is done using a balanced version of the SS-tree. Color, shape and texture of each region are indexed separately. The

combination of the results is performed by taking the intersection of the result lists and ranking according to a weighted similarity measure. NeTra uses an implicit ordering of the image features to prune the search space.

The NeTra system additionally supports spatial queries. The spatial information is saved as metadata of each region. The coordinate of the region centroid and the minimum bounding box are used to determine the position and extent of the regions. A quad-tree [Sam84] is used to index the centroids. The bounding boxes are indexed by an R-tree [Gut84]. The spatial query is defined using two rectangles. The inner rectangle defines the part of the image that must be covered by the result region. The outer rectangle must cover the whole region.

| | |
|---|---|
| Name | NeTra |
| Institution | UC Santa Barbara |
| Paper | [MM97] |
| Demo URL | http://vivaldi.ece.ecsb.edu/Netra |
| Segmentation | JSEG |
| Features | Color, texture, shape |
| Matching | One to many |
| Search paradigm | One region |
| Specials | Segmentation, Spatial query |
| DB size | 2,600 |

## 6.6 Other Systems

| | |
|---|---|
| Name | n/a |
| Institution | Microsoft Research China |
| Paper | [JZL+02] |
| Demo URL | no demo |
| Segmentation | JSEG |
| Features | Color moments |
| Matching | SLRI (Self-Learned Region Importance) |
| Search paradigm | n.n. |
| Specials | Relevance Feedback |
| DB size | 8,600 |

| | |
|---|---|
| Name | n/a |
| Institution | Indian Institute of Technology |
| Paper | [PGB01] |
| Demo URL | no demo |
| Segmentation | color quantization |
| Features | Color sets, shape |
| Matching | Threshold |
| Search paradigm | Full image |
| Specials | |
| DB size | 320 |

| | |
|---|---|
| Name | n/a |
| Institution | Hiroshima City University |
| Paper | [SIHK02] |
| Demo URL | no demo |
| Segmentation | Texture clustering |
| Features | Texture |
| Matching | Nearest Neighbor |
| Search paradigm | One Region |
| Specials | |
| DB size | 1,750 |

# Chapter 7

# Conclusions and Future Work

In this thesis, a flexible and fast region based image retrieval system has been developed that is able to cope with huge image databases. The experiments have shown a remarkable increase of retrieval quality compared to the former system that only supported static regions. Due to the extendable implementation, newer (and probably better) segmentation and matching algorithms can be integrated into the system. A detailed analysis of a set of segmentation and matching algorithms could lead to a more powerful system with respect to retrieval quality. Since this is a first time implementation of region based image similarity search in ISIS there is a wide area of future development tasks.

## 7.1 Segmentation and Feature Extraction

Besides the use of other algorithms, the current algorithms could be extended to support overlapping regions. It is often not clear where the actual boundaries of a region are and if two regions should be merged into one or not. By supporting overlapping regions, we could provide both a coarse and a fine grain segmentation for the same image. In figure 7.1 this is shown for a picture with two pipelines. Four possible overlapping segmentations are shown: only a part of one pipeline (yellow), one pipeline (red), both pipelines (blue) and both pipelines including the basement (green). The straightforward solution for our system would be to extend the JSEG algorithm. Suematsu et al. have recently implemented another approach following the same direction [SIHK02]. They segment the image hierarchically and introduce a sort of containment semantic.

Similarly, the idea of fuzzy segmentation as proposed in [LWW01] could be implemented. In this case we postulate that every point of the image belongs to every region but only with a certain probability. The resulting feature value for each region is calculated by taking these probabilities as weights for each pixel.

At the query frontend we could imagine to support the interactive segmentation of the image. The user could draw the extent of the regions he is interested in directly on the picture and in this way could interactively select what he really wants. The

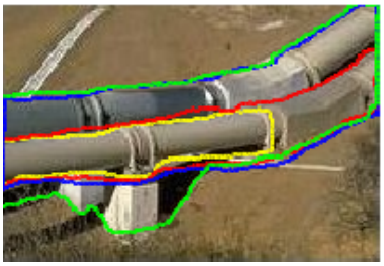


Figure 7.1: Four overlapped segmentations of the pipelines.

features of the selected parts of the image would then be directly extracted by the FEE component and used for querying the database. It would not be certain that the query image itself (if it is present in the database) has a high rank, e.g., if it is segmented in a completely different way.

The feature extraction component could be extended to support more shape features. In textured images shape features usually aren't semantically important. But since in the case of regions based image retrieval we have a (hopefully good enough) segmentation that really detects object boundaries, the shape of the regions could be interesting. E.g., it is sometimes desired to match a region with a red ball (shape $\approx$ circle) with a region with a blue ball (also shape $\approx$ circle) rather than a region with a red car (shape $\approx$ rectangle). One simple shape feature would be to extract the bounding box of each region. A more complex extraction algorithm could extract three or more "interest points" (i.e. points where the differentiation of the region's contour line is near to zero) for each region. The better two sets of "interest points" match, the more similar the shape of two regions is. More shape features for region based image retrieval are described in [PGB01].

Figure 7.2: Modelling the relative positions in an image

Another extension in this area would be to take the relative positions into account. So called spatial constraints could be set (e.g. as a special sort of predicates). We could then explicitly distinguish between an image with a red region on the left and a green region on the right and an image with a red region on top and a green region on the left bottom. In the image in figure 7.2, we could for example explicitly model that the forest is above the snowfield, that the boy stands left of the old man and that the head is above the body. A special query language similar to

the suggestions made in [SC96] could be developed that can give extended query definement abilities to the user.

## 7.2 Indexing and Searching

In the indexing and searching component, special support for spatial constraints could be implemented that directly affect the matching between the regions. Using these constraints, the Hungarian Algorithm could probably stop earlier and therefore the overall speed of the system could be improved.

Another approach to decrease the complexity of the calculation of a minimum weight perfect matching would be to do all calculations directly in the feature space. Currently, we first calculate all distances between the region features and then calculate the matching. It might be faster to find a matching in the feature space. Performing a k-nearest neighbor search for each query region followed by a combining phase to find the actual matching would be the straight forward algorithm to solve the problem, but perhaps some additional optimizations could be introduced.

Since we seldom have a perfect segmentation, an adaptive merging of regions during the query step could be introduced. In this case two or more regions would be adaptively merged if the resulting matching distance is smaller. It is certainly not trivial to find an efficient algorithm that can detect such cases and optimize them. An approximation to this behavior would be the precalculation of the feature value for glued regions and simply running the hungarian algorithm on all regions and combinations of regions. Using this approximation, it could occur that different regions of the query image are assigned to the same region of the database image. To visualize this idee you can look at figure 7.3. The query (a) consists of two regions: a man and an US-flag. Unfortunately the image in our database (b) is over-segmented and has four regions for the flag and two for the man. Since this leads to a completely different color distribution of the regions it would be unlikely that this database image would be considered relevant. But if the search algorithm would adaptively merge the four regions of the flag into one region and the two regions of the man, the image would certainly be returned with a high rank.

(a) (b) (c)

Figure 7.3: Adaptive merging. (a) The query, (b) the over-segmented image in the database, (c) the database image after an adaptive merge.

It would be interesting to know how important the calculation of a minimum weight perfect matching is in practice. A precision- and recall analysis of the current implementation compared to always taking the lower or upper bound matching

could give an answer to this question. If the results are comparable good, the use of vector approximation could be considered since the optimization of CPU costs would not be a main issue anymore.

In the current system the relevance feedback mechanisms do not care about the regions. We could extend the relevance feedback methods to find out which region is more important and how the importance of a feature is in one region and in another. A first approach in this direction is described in [JZL+02].

## 7.3 Improvements of the Code

In the current implementation of the `DynRegFile`, a big amount of memory allocation and copy operations make the search process unnecessarily slow. It would therefore be of great benefit if the support of dynamic regions would be handled at a lower system level, i.e., if the region vectors would be stored natively in a specially designed region vector file instead of using the legacy vector file class.

# Bibliography


[BCP01] Ilaria Bartolini, Paolo Ciaccia, and Marco Patella. Windsurf: a region-based image retrieval system. Technical report, University of Bologna, July 2001.

[Ber01] Micheal K. Bergman. The deep web: Surfacing hidden value. *The Journal of Electronic Publishing*, 7(1), August 2001.

[BKK97] S. Berchtold, D. A. Keim, and H.-P. Kriegel. Section Coding: Ein Verfahren zur Ähnlichkeitssuche in CAD-Datenbanken. *Datenbanksysteme in Büro, Technik und Wissenschaft*, pages 152–172, March 1997.

[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 Hector Garcia-Molina and H. V. Jagadish, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 322–331, Atlantic City, NJ, USA, May 1990. ACM Press.

[BMSW01] Klemens Böhm, Michael Mlivoncic, Hans-Jörg Schek, and Roger Weber. Fast evaluation techniques for complex similarity queries. In *Proceedings of the 27th VLDB Conference*, Roma, Italy, 2001.

[COR] CORBIS. Corbis image library. http://www.corbis.com.

[CPZ97] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, Greece, 1997.

[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 *Third International Conference on Visual Information Systems*. Springer, 1999.

[DM01] Yining Deng and B.S. Manjunath. Unsupervised segmentation of color-texture regions in images and video. In *PAMI*, 2001.

[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.

[FH98] P. Felzenszwalb and D. Huttenlocher. Image segmentation using local variation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 98–104, June 1998.

[Gut84] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *ACM SIGMOD International Conference on Management of Data*, pages 47–57, June 1984.

[Heb01] John Hebborn. *Decision Maths 2*, chapter 2. Allocation (assignment) problems, pages 29–45. Heinemann Modular Maths for Edexcel AS and A-level. Heinemann, 2001.

[JZL+02] Feng Jing, Bo Zhang, Fuzong Lin, Wei-Ying Ma, and Hong-Jiang Zhang. A novel region-based image retrieval method using relevance feedback. http://woodworm.cs.uml.edu/~rprice/ep/fengjing/, 2002.

[Knu93] Donald E. Knuth. *The Stanford Graph Base*. ACM press, New York, 1993.

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

[LWW00] Jia Li, James Ze Wang, and Gio Wiederhold. IRM: integrated region matching for image retrieval. In *ACM Multimedia*, pages 147–156, 2000.

[LWW01] Jia Li, James Z. Wang, and Gio Wiederhold. FIRM: Fuzzily integrated region matching for content-based image retrieval. Technical report, Pennsylvania State University, 2001.

[MM97] W.Y. Ma and B.S. Manjunath. NeTra: A toolbox for navigating large image databases. In *IEEE International Conference on Image Processing*, volume I, pages 568–571, Santa Barbara, California, October 1997.

[Mos02] Thomas Moscibroda. Session management und frontend für ISIS. Semesterarbeit, ETH Zürich, 2002.

[PGB01] B.G. Prasad, S.K. Gupta, and K.K. Biswas. Color and shape index for region-based image retrieval. In C. Arcelli et al, editor, *IWVF4*, LNCS 2059, pages 716–725. Springer, 2001.

[Sam84] H. Samet. The quadrtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260, 1984.

[Sax01] Silvan Saxer. Text extraction from HTML pages II (in german). Semesterarbeit, October 2001.

[SC96] John R. Smith and Shih-Fu Chang. VisualSEEk: A fully automated content-based image query system. In *ACM Multimedia*, pages 87–98, 1996.

[SIHK02] Nobuo Suematsu, Yoshihiro Ishida, Akira Hayashi, and Toshihiko Kanbara. Region-based image retrieval using wavelet transform. http://www.cipprs.org/vi2002/pdf/s1-2.pdf, 2002.

[SRF87] T. Sellis, N. Roussopoulos, and C. Faloustos. The R+-tree: A dynamic index for multi-dimensional objects. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 507–518, Brighton, England, 1987.

[SSW02] H.-J. Schek, H. Schuldt, and R. Weber. Hyperdatabases - infrastructure for the information space. In *Advances in Visual Database Systems - Proceedings of the 6th IFIP 2.6 Working Conference on Visual Database Systems (VDB'02)*, Brisbane, Australia, May 2002.

[Web97] Roger Weber. Parallel VA-file. Technical Report 25, ESPRIT project HERMES (no. 9141), October 1997. Available at http://www-dbs.ethz.ch/∼weber/paper/HTR25.ps.

[Web01] Roger Weber. Multimedia retrieval. Lecture Notes, 2001.

[WLW01] J.Z. Wang, J. Li, and G. Wiederhold. SIMPLIcity: Semantics- sensitive integrated matching for picture libraries. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23, 2001.

[WSB98] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proc. 24th Int. Conf. Very Large Data Bases, VLDB*, pages 194–205, 24–27 1998.

[Zei] Neue Zürcher Zeitung. NZZ online archive. http://www.gbi.de/nzz/.

# Appendix A

# Aufgabenstellung

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Prof. Dr. H.-J. Schek
Institute for Information Systems
Database Research Group
http://www.dbs.ethz.ch

## Diplomarbeit: Regionenbasierte Bildähnlichkeitssuche

---

| | |
|---|---|
| Professor | Prof. Dr. H.-J. Schek |
| Student | Silvan Saxer |
| Betreuung | Michael Mlivoncic, IFW C45.1, Tel.: 632 75 53 |
| Beginn | 18. März 2002 |
| Abgabe | 17. Juli 2002 |

### Motivation

In der Datenbankgruppe der ETH wurde in den letzten Jahren ein Prototyp zur Bildsuche entwickelt. Dabei wird die Ähnlichkeit der Bilder untereinander aufgrund inhaltlicher Merkmale (z.B. Farbe) ermittelt. Bislang wurde jedes Bild in seiner Gesamtheit bzw. in einer festgelegten Anzahl statischer Regionen analysiert. Beispielsweise würde ein Bild mit Wald in der linken Bildhälfte und einem See in der rechten Bildhälfte nur eine geringe Ähnlichkeit zu einem seitenverkehrten Pendant haben. Als Lösung bietet es sich an, die Bilder in dynamische Regionen zu unterteilen, die ungefähr den Objekten im Bild entsprechen. Um die Ähnlichkeit zweier Bilder zu bestimmen, sucht man nach möglichst guten Übereinstimmungen zwischen den Regionen der Bilder bezüglich der gegebenen Merkmale.

## Aufgabenstellung

Ziel dieser Diplomarbeit ist es, den Prototyp zur Bildsuche um ein regionenbasiertes Suchverfahren zu erweitern. Die Aufgabenstellung umfasst im Einzelnen die folgenden Tätigkeiten:

- Einarbeitung
  Einlesen in die Literatur. Welche Verfahren zur regionenbasierten Ähnlichkeitssuche existieren bereits? Wo liegen die Gemeinsamkeiten und Unterschiede?

- Extraktion von Bildregionen
  Ein Algorithmus zur Segmentierung von Bildern soll in eine Systemkomponente eingebettet werden, die in Zusammenarbeit mit den vorhandenen Merkmalsextraktoren das Zerlegen eines Bildes in seine Regionen und eine anschliessende Extraktion der Merkmale pro Region erlaubt.

- Erweiterung der Suchfunktionalität
  Die Bildähnlichkeitssuche soll so erweitert werden, das sie auf den Merkmalsdaten der dynamischen Regionen durchgeführt werden kann. Bislang hatten diese Daten in Abhängigkeit vom Merkmalstyp für jedes Bild eine einheitliche Länge. Die Datenstrukturen müssen dahingehend erweitert werden, dass sie den effizienten Zugriff auf die Datensätze variabler Länge erlauben. Weiterhin muss das Sucheverfahren selbst erweitert werden, damit die beste Übereinstimmung zwischen den Regionen zweier Bilder ermittelt werden kann.

- Einfaches Frontend zur Regionensuche
  Zur Demonstration der neuen Fähigkeiten des Systems soll das Frontend so erweitert werden, dass der Benutzer Regionen eines Bildes spezifizieren kann, nach denen er suchen möchte.

Bei dieser Arbeit geht es die Umsetzung eines vielversprechenden Ansatzes zur Bildsuche. Es ist anzunehmen, dass der erweiterte Prototyp Suchresultate höherer Qualität liefern wird, als bislang. Sofern es im Rahmen der Arbeit möglich sein wird, sollen erste Experimente die Leistungsfähigkeit des Systems untersuchen.

# Appendix B

# Examples

## B.1 Bird Example

The task in this example was to find a bird walking on grass. We have selected the region of the body of the bird and the grass region that is least blurred. The result is shown in figure B.1. Compared to the result of the query with five static regions we have found two more bird images. One image (OID 14581) cannot be found by an algorithm that uses static regions since it is in landscape format. In the other image, the bird is at the image's border and hence not in the same region as the other images (OID 14583).


Figure B.1: Region based query using two regions

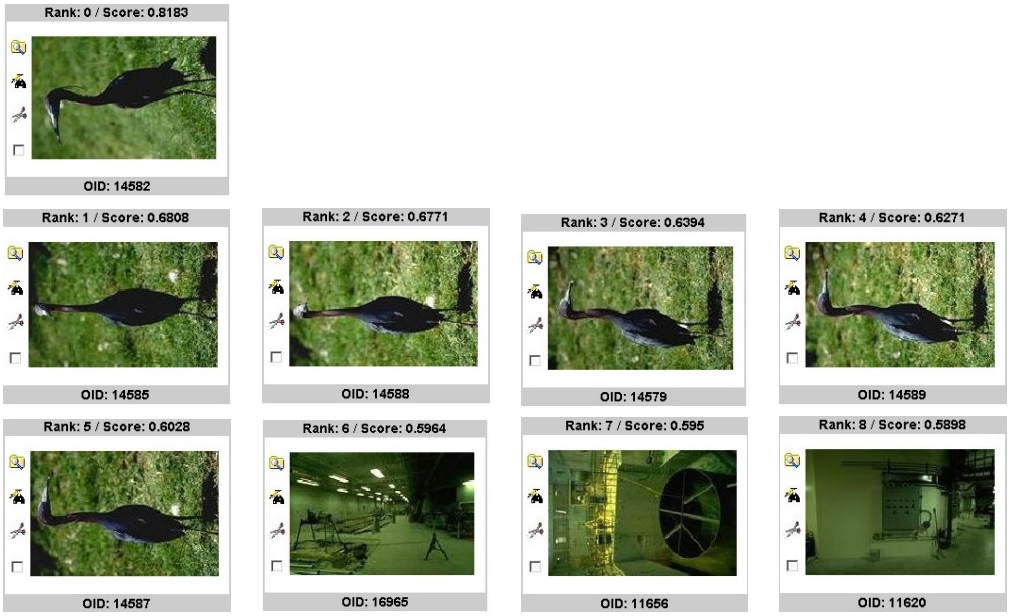



Figure B.2: Region based query using five static regions

## B.2 Building Example

This example demonstrates how the algorithm can find similar images even though the perspective of the photograph is totally different. We are searching for a building as highlighted in figure B.3 (a). The system returned two images that both contain the selected building (cf. figure B.3). In image (c) it is just photographed from a longer distance (with many surrounding regions) and in image (d) from a nearer distance. The other images returned do not contain this building and are therefore not shown in the figure.


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

Figure B.3: Finding a building.

## B.3 Worker Example

In this example we look for a person in the database that has a red shirt and gray pants. In contrast to the query on five static regions, we find two other pictures on

which this person is working. The query and its results are shown in figure B.4.


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

Figure B.4: Finding a person.

## B.4 Water Example

This example shows one benefit of dynamic regions in contrast to static regions: The orientation of the picture is not important. In contrast to the result to a query with five static regions, the system described in this thesis also finds an image that is taken from another perspective. An example for this is shown in figure B.5. With region based image retrieval we do not only find images (a), (c) and (d) but also image (b) where the water flows from right to left (instead of left to right as in the other images).


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

Figure B.5: With region based image retrieval we also find mirrored images.

# Appendix C

# Evaluation

In this section the exact results of the evaluation on 38 queries are shown. The images have been chosen randomly and their content has been described. After that the queries have been performed. The selection of regions has been tried out for different constellation and the best has been taken. The results are listed in table C.1. The numbers stand for the number of relevant images in the top ten results.

| OID | Description | (1) | (2) | (3) | (4) | (5) |
|---|---|---|---|---|---|---|
| 19178 | yellow flowers | 1 | 2 | 2 | 2 | 2 |
| 17350 | bird | 9 | 5 | 5 | 8 | 10 |
| 14315 | sunset | 1 | 1 | 2 | 2 | 1 |
| 7841 | bridge over river | 1 | 1 | 2 | 2 | 1 |
| 15078 | a group of persons | 2 | 7 | 4 | 7 | 8 |
| 15036 | sign in the desert | 1 | 1 | 2 | 2 | 2 |
| 10007 | dishwasher | 1 | 1 | 1 | 1 | 1 |
| 10646 | man with snow | 2 | 3 | 2 | 5 | 4 |
| 11844 | wild cat | 3 | 1 | 2 | 2 | 1 |
| 16912 | pipeline | 1 | 1 | 2 | 2 | 3 |
| 10696 | fish in aquarium (swarm not counted) | 3 | 5 | 3 | 3 | 5 |
| 14345 | diver | 1 | 1 | 1 | 1 | 1 |
| 14300 | white rock | 1 | 1 | 1 | 1 | 1 |
| 19094 | white car | 1 | 1 | 1 | 1 | 2 |
| 18230 | mechanic | 2 | 1 | 1 | 2 | 2 |
| 8451 | researcher | 2 | 2 | 1 | 3 | 3 |
| 10384 | river | 3 | 3 | 5 | 4 | 6 |
| 19264 | highway | 2 | 1 | 1 | 1 | 1 |
| 14448 | construction site and building workers | 2 | 3 | 4 | 3 | 3 |
| 13418 | red scaffold | 9 | 2 | 3 | 5 | 5 |
| 9807 | people under a sun shade | 1 | 1 | 1 | 1 | 1 |
| 7843 | something | 4 | 3 | 0 | 4 | 3 |
| 13909 | edge of the forest | 1 | 3 | 2 | 3 | 4 |
| 8916 | gravel rowing boat | 1 | 1 | 1 | 1 | 1 |
| 10605 | factory | 5 | 5 | 1 | 2 | 3 |
| 16744 | deer crossing the road | 2 | 1 | 1 | 1 | 1 |
| 12179 | blue truck | 4 | 4 | 1 | 2 | 3 |
| 12892 | ambulance | 1 | 1 | 1 | 1 | 1 |
| 10823 | skyscrapers | 2 | 1 | 1 | 2 | 2 |
| 15074 | presentation | 3 | 7 | 4 | 5 | 5 |
| 19057 | dog | 1 | 1 | 1 | 1 | 1 |
| 19167 | inundation | 1 | 1 | 1 | 1 | 1 |
| 17276 | barrage | 1 | 1 | 1 | 2 | 2 |
| 8241 | green machine | 1 | 1 | 1 | 1 | 1 |
| 8878 | lake with green surroundings | 6 | 3 | 8 | 9 | 9 |
| 9659 | owl | 6 | 3 | 4 | 5 | 6 |
| 9653 | owl with black background | 2 | 2 | 5 | 5 | 2 |

Table C.1: Effectiveness evaluation. (1) 5 static regions, (2) All regions, (3) 1 region, (4) 2 regions, (5) 5 regions selected.