Image database management systems design considerations algorithms and architecture
Nes, N.J.

Citation for published version (APA):
Nes, N. J. (2000). Image database management systems design considerations algorithms and architecture

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Chapter 4

The Image Retrieval Algebra

4.1 Introduction

With the advent of large image databases becoming readily available for inspection and browsing, it becomes mandatory to improve image database query support beyond the classical textual annotation and domain specific solutions[117]. An ideal image DBMS provides a data model to describe the image domain features, a general technique to segment images into meaningful units, and provides a query language to study domain specific algorithms with respect to their precision and recall capabilities. However, it is still largely unknown how to construct such a generic image database system.

The early image retrieval systems, such as QBIC[41] and VisualSEEk[102], have demonstrated some success in supporting domain-independent queries using global image properties, such as dominant angle and color histograms. The prototypical query posed to the system is \((Q_1) \) "find me images similar to this one". The user should supply such image or a sketch, leading to techniques called query by visual example (QBE). The system searches for all "similar" images based on pre-calculated features and built-in similarity measures.

This query evaluation technique is bound to fail in the long run for several reasons. First, it assumes that the user has a correct sample of the envisioned sample set. It presupposes that the envisioned target image is stored in the database, and that progressing from a random sample set will lead to it quickly. This assumption does not hold when the databases becomes large, such as envisioned for the Acoi image database\(^1\). Using (Random) sample sets to steer the query process becomes confusing, because they likely lack an evident color, texture and shape relationship with the semantic domain of interest.

\(^1\)Acoi is the experimental base for the national project on multi-media indexing and search (AMIS). More information about Acoi can be on the web site: http://www.cwi.nl/acoi

63
Image databases are rarely used to answer query $Q_1$. Instead, the user formulates a query ($Q_2$): “find me an image that contains (part of) the one selected” where the containment relationship is expressed as a user controlled metric over selected features or directly ($Q_3$) :“find me an image that contains specific features using my own metric”.

Secondly, global image properties alone are not sufficient to prune false hits, spatial information about object locality is also needed. For example, in a large image database one could be interested to locate all images that contain part of the Coca-cola logo. This query could be formulated by clipping part of a sample Coca-cola logo to derive its reddish (R) and white (W) color and to formulate a (SQL-like) query of the form:

```sql
select display(img) 
from image_segment s1,s2, image img  
where distance(s1.avghue, R) < 0.2  
and distance(s2.avghue, W) < 0.2  
and s1.area overlaps s2.area  
and s1 in img 
sort by distance(s1.avghue, R),  
        distance(s2.avghue, W)
```

This query uses two primitive parameterized metric functions. The function distance calculates a distance in the hue color space and overlaps determines segment containment. The former is defined as part of the color data type and the latter for the segment data type. In principle, the DBMS should support overloading and refinement of this function by the user.

The big challenge for image database designers is to identify the minimal set of features, topological operators, and indexing structures to accommodate such image retrieval queries. In particular, those (indexed) features where their derivation from the source image is time consuming, but still can be pre-calculated and kept at reasonable storage cost. Features may be viewpoint, scale, rotation, and translation invariant, but need not be, see Section 2.1.3. These problems becomes even more acute when the envisioned database is to contain over a million images. Observe also that SQL is a declarative language, which should be translated into an execution algebra. This lead to the requirement of a supportive Image Algebra satisfying the following global requirements.

Navigational queries Image retrieval applications have a strong navigational behavior. A user guides the search for a collection of interest by repeatedly rephrasing the query posed to the system. Usually it starts with a randomly selected image set taken from the database. The first real query posed by a user is to select all images similar to an element of this sample set. By selecting a new image from the result obtained, the user presumably navigates to the collection of interest.
4.2. IMAGE RETRIEVAL BY CONTENT

Extensional relational framework Many researchers are looking for new similarity measures to compare and rank images. Therefore, it should be easy to extend the algebra with new data types, operators, and algorithms. This way code-reuse can be guaranteed.

Proximity queries Features are derived from the image data. Since the image data is inherently imprecise, so will the feature data. Therefore, queries based on feature spaces should be supported by proximity queries, probabilistic reasoning, and a toolkit of similarity measures. This way the user has precise control over the query model which is needed to advance research.

Computationally Complete The algebra should be computationally complete. We want image analysis researchers to start using database techniques. Therefore, we should at least support the operations necessary. Besides that it should be extensible using third generation languages and allow rapid prototyping using scripting languages.

4.2 Image Retrieval by Content

The early attempts for image retrieval systems used primarily keyword annotations[18, 19, 20]. Image retrieval is simplified by formulation in terms of keywords. The annotation is mainly manual, although some automatic approaches exist. Examples like [119, 102, 43] use words found in the surrounding of the image.

Experience with keyword based retrieval systems has been accumulated in the area of information retrieval for several decades[113]. The wide spread use of WEB search engines illustrate their limited effectiveness. Although successful in bibliographical information retrieval, keyword annotation for image retrieval suffers from major problems. The first problem is its lack of scalability. Manually annotation of 1000 images may still be reasonable, but databases with of over 100,000 images to annotate, becomes practically impossible. At best a rough classification is done. Secondly, each person will describe an image by a different set of keywords. This is a result of the person’s perception of the information found on the image. Therefore, using only keywords to describe images for retrieval purposes becomes impossible for image databases in mind.

There are two solutions to the problem. The first is to broaden the group of annotaters, which leads to social indexing. The second is to improve manipulation of content, which will be our focus. This solution is simple, just stick to the information found in the image, i.e. use the image content. Although this sounds trivial, its realization is not. Deciding what content
CHAPTER 4. THE IMAGE RETRIEVAL ALGEBRA

to use and how to compare these image contents is still an open research issue.

Retrieval methods based on color features, such as color histograms, are a promising track [46, 41, 106]. Color is a powerful retrieval feature. However, these retrieval algorithms largely ignore spatial information in the matching process. At best a query can be specified in terms of color percentages or the user has to outline objects as part of entering the image into the database. Then color histograms for the (sub-) objects can be used in the retrieval process. Although the index structures will be large for these methods, both cases lead to a high percentage of false hits.

To understand the requirements of image retrieval systems we implemented a prototype system. The system is designed to answer best match for complete image queries, based on color and spatial information. Our query interface is based on the query-by-example paradigm, and the system returns a list of best matches in order of significance.

We also use spatial features, since it adds significant information to the content description. It simply makes a lot of difference were a color appears in an image. For example having blue on top often indicate air.

4.2.1 Multi-Level Signature

Query by features calls for a both color and spatial features. In this section we describe an index scheme, which combines both color and spatial features. The indexing scheme proposed, called the MLS (Multi-Level Signature), is based on a recursive splitting of the image. For each sub-image we calculate the color feature. The color feature used is the average color in the sub-image. Concatenation of the color features leads to a signature that characterizes an image at various levels of detail.

Using spatial information directs us at considering space dividing methods, such as multi-level grids. In this study we focus on two methods to the MLS, called quad-tree and prime-factor split. The quad-tree split is based on the traditional quad-tree index structure[92]. The algorithm for the quad-tree splitting process is shown in pseudo code in figure 4.1.

This algorithm first calculates the color feature, i.e. the average color, for the image and stores this in the MLS. If we haven’t met the stop criteria, the image is recursively divided into four equal adjacent parts. For each part this process is repeated (see Figure 4.2). Therefore, the MLS, keeps information on various levels of details. Each level describes the image color content, deeper levels keep more details.

A potential problem of the quad tree splitting is that it ignores the object boundaries. In general, objects of relevance will not nicely fit a cell. When an object inside the image lays in the center of the image, its color features will contribute to all four parts, so it will mix with the rest of the colors in those parts and, therefore, have a limited effect on the selection. Objects
4.2. IMAGE RETRIEVAL BY CONTENT

```c
mls(MLS sig, Image i) {
    color_feature(sig, i);
    if (stop_condition(sig))
        return;
    quad_split(i, r00, r01, r10, r11);
    mls(sig, r00); mls(sig, r10);
    mls(sig, r01); mls(sig, r11);
}
```

Figure 4.1: Quad-split pseudo code

![Quad-split illustration](image)

Figure 4.2: The quad splitting process

which extend over the borders of the image part may have less influence to the MLS. Therefore, we came up with a slight variation on the quad-tree splitting process, the prime-factor method. The prime-factor splitting process splits each time the original image in \( p^2 \) parts, where \( p \) is a prime-factor. See figure 4.3. for a graphical example of a prime-factor split. The effect of objects crossing grid boundaries is reduced, since the prime-factor split makes sure that grid boundaries are always on a different place for each level. Grid elements at lower levels are not fully contained in a cell at a higher level. They combine information from parts of upper layer cells.

![Prime splitting illustration](image)

Figure 4.3: The prime splitting process

4.2.2 Data Model for MLS Image Database

The data produced in the splitting process is stored in BATs managed by Monet. This required extension of the system with an atomic type `Image`. Its implementation provides the operational primitives to handle image processing in a structured way; orthogonal to the other data types. See Section 3 for details of this module.
Figure 4.4: The relational data model used for storing the multi level signature.

The BATs for our retrieval system (MLS) are \textit{MLS.source}, \textit{MLS.color}, \textit{MLS.spatial}, \textit{MLS.image} and \textit{MLS.icon}, (see Figure 4.4). \textit{MLS.image} contains the actual image data. In \textit{MLS.icon} an icon of the image is stored. Showing icons instead of the whole image reduces retrieval cost (less data needs to be transferred) and image display.

\textit{MLS.source} contains the relationship between image fragments and their source. For each image fragment a pair of object identifiers (cell, image) is stored to indicating this relationship. Although we could have used a single data type to represent both spatial and color features together, using some form of pyramid structure, we decided upon separation. This is done to easily extent the system with different features calculated over the image parts, such as texture, shape and alternative color, and spatial features.

\textit{MLS.color} contains the color feature, i.e. the average color, for each image-fragment.

In our prototype implementation we use the prime color component of the HSV-color model[65]. The reason is to be robust against light reflections[40]. Furthermore, the hue closely resembles the human perception of related colors, which improves image retrieval from an ergonomic perspective.

\textit{MLS.spatial} contains the spatial description of each image fragment. It describes the spatial information obtained by the splitting process. We use a simple spatial representation, i.e. a box. To keep the description scale invariant we used normalized positions, i.e. box(0,0,1,0,1,0) describes the whole image, and box(0.5,0.5,0.5,0.5) describes the first bottom right image fragment of the quad split.

\textbf{4.2.3 Stop Condition}

Image splitting continues up to the point that further splitting does not produce significant new information about the spatial color distribution. This requires a flexible and user-controlled stop condition. For example, splitting stops when one of the following conditions occurs:

1. Stop when the split level equals some predefined \( \alpha \).
4.2. **IMAGE RETRIEVAL BY CONTENT**

2. Stop when the average color of corresponding parts on level n and n+1 differs less than some $\delta$.

3. Stop when there are less than $\gamma$ colors remaining in a sub image.

4. Stop when the MLS vector occupies too much space.

5. Stop when the area of the image fragments is less than $\beta^2$ pixels.

The first condition is independent of the image content. The second condition is intended to signal smoothness, but it suffers from large outliers. The third variant would be of use if the image contains a large number of small details with different color distributions. In the worst case the index becomes larger than the image itself, due to overhead of the index data structures. The fifth condition uses a multi level resolution principle. We choose the third option, since it is less sensitive to outliers and combine it with the first and last. This way, the index storage size will never exceed the image size.

4.2.4 Querying the image database

The selection process is initiated when the user specifies a query image, which should be a representative sample of the desired answer set. The process will split the query image recursively and uses the signatures obtained to exploit the index. The spatial information is used to assure that candidate images in the database have the same spatial relationships amongst them as the query image.

We will look at the selection process of quad split and calculate algorithm in detail. From the image database the candidate (sub) images $CIs$, are selected based on an equal bounding box. Equal bounding boxes are required, since we are only looking for similar images, not for images with a similar sub-image. From this set of candidates images are selected which have an average color within a given range from QIs average color. Using the source relation the original images belonging to the selected (sub) images are found.

QI is then split into 4 parts as described before. For all parts the average color and bounding box are calculated. They are used to reduce the candidate image set. Again the (sub) images with equal bounding boxes are selected from the set of candidate images. Only those images which have for all parts a similar average color as QIs parts are selected.

The selection process continues until a small enough answer set is reached. The selected images are than ranked, based on a similarity measure taking both spatial and color properties into account. The similarity measures known from literature, Histogram intersection[106] and Histogram distance [41] do not use spatial information.
Instead, we use a similarity measure, called the *Multi-level signature similarity measure*, which computes the weighted distance between the signatures of the query image and the selected images on the split level on which the images were retrieved. The similarity measure requires a non-expensive computation. Formally, at each level $\lambda$ the similarity between the $QI$ and $CI$ is calculated as

$$\delta(QI, CI) = 1 - \sqrt[n]{\left(\sum_{i=0}^{n} \frac{C_{QI_i} - C_{CI_i}}{cb}\right)^2}$$

Where $C_i$ is the average color of the sub image with $i$ as its spatial representation. The $n$ is the number of spatial descriptions at level $\lambda$. The function is normalized using the maximum color difference $cb$ found in the database.

![Monet Image Retrieval System](image)

**Figure 4.5:** The prototype image retrieval system.
4.2.5 Prototype and Experiment

An early prototype image retrieval system was implemented in 1997. It uses the Monet version 3.0 database kernel and the image extensions explained in Chapter 3. A graphical user interface was build using Tcl/Tk[85]. Queries are specified by selecting an example image from a set of image taken randomly from the image database. Figure 4.5 shows a screenshot of the system. The result of this query are images ordered on their similarity.

An indepth evaluating of image similarity measures and retrieval quality was beyond the scope of this thesis. Our focus was to provide a layer of database functionality to be used by image analysis researchers to pursue this task. To illustrate that the MLS description is a valuable addition to the existing set of image descriptions, we conducted a small (non-exhaustive) experiment. The experiment is conducted with a database of video frames, coming from multiple video sequences. Our approach returns all the frames of the same scene followed by images that have a significant less similarity value. Even when they are distributed over multiple shots.

![Graph showing Scalability Experiment results.](image)

Figure 4.6: The results of the Scalability Experiment.

Scalability experiments were conducted using database sizes from 100 to 3000 images to evaluate the query-by-example processing time. The results can be seen in Figure 4.6. It confirmed that the processing time is linear in the number of images, which is achieved because larger databases will require images to be compared on more levels. Although linear is adequate for small sized databases for larger databases better use of the index structures is required.

The retrieval performance of the prime-split algorithm is about the 20 to 50 percent better. This indicates less calculations are required to answer the queries, i.e. smaller trees need to be compared.
4.2.6 Conclusion

Monet could be used -extended, queried- for the task a hand
From this exercise we can derive requirements for the Image Retrieval Algebra:

Multiple Image Descriptions  Although the MLS is a valuable addition
to the set of image descriptions, it becomes clear that alternative descriptions
are needed. Therefore, one or more abstract image descriptions are needed.

Partial Image Queries  The Image Retrieval Algebra should support
global image queries, but also partial image queries (Point based retrieval
does not require an expensive index)

Index Structures  A retrieval algebra requires proper support by Index
structures.

4.3 Segment Image Indexing

The early experience with the MLS image description showed its advantages,
but its applicability is limited to support localization of images according
to query $Q_1$. It confirmed that the average color of successively partitioned
regions provide a good handle to steer the query process.

In this section we evaluate the viability of a segment based approach,
both the segmentation process and storage implications are considered hav-
ing in mind a image database of 1 million images. Segment based image
retrieval would accommodate queries of type, "find me images similar to
(this and) this segment" ($Q_2$). Again we take a grid base approach, know-
ing that proper image segmentation is an unsolved process in image analysis
research.

Segments are found using both a split and a merge image indexing algo-
rithm. Two kinds of algorithms are considered; a top-down method based
on recursive splitting, called S-split, and a bottom-up method based on suc-
cessive merging, called S-merge. The former recursively splits an image into
smaller segments until their feature vectors dissimilarities fall below a cut-
off point. The image objects thus considered are all rectangular in shape.
The latter uses a bottom-up strategy, i.e., rectangular regions are merged
to form segments as long as their feature vectors are closely related. This
leads to more general image objects. The effect of this approach compares
with R-trees in GIS, which have proven effect for spatial filtering.
4.3. SEGMENT IMAGE INDEXING

4.3.1 Segment Indexing

The key challenge is to develop an efficient algorithm to locate the segments of interest for a given image. No attempt is made to detect or infer hidden faces. Neither do we consider a search for optimal segmentation schemes common in image recognition research. We conceive the index primarily as a filter for applications dealing with image retrieval.

The algorithm S-split finds the collection of discriminating segments by recursively splitting the image into two sub-images. Splitting is attempted both horizontally and vertically. Sub-images are chosen such that their dissimilarity in average Hue is maximal. This improves the selectivity of the individual segments.

The recursive process is controlled by several stop criteria as follows.

- Let $I_i$ be an image split into two segments $I_{i,1}$ and $I_{i,2}$, Then the new segments are added to the `img_segment` index provided their average Hue differs from $I_i$ more than a given minimal threshold $H_{threshold}$. This guards against storing redundant information into the database.

- The size of the resulting two segments should both be larger than some threshold. This guards against border effects and too small segments.

- The maximal number of segments per image is limited by a system parameter, $H_{objects}$. This guards against repeatedly splitting images up to the pixel level. Instead, we assume that a limited number of segments (possibly dependent on the image size) is often sufficient.

The worst case complexity of this algorithm is $O(d \times n^2)$ with $n$ the maximum image width or height and $d$ is equal to $H_{objects}$.

Usually, $d$ will be less than $H_{objects}$ because of the first stop condition. The algorithm S-merge attempts to merge segments into larger units. The algorithm starts by dividing the original image into equal sized segments using a grid layout. Each grid element is a candidate segment for inclusion in the `img_segment` index. The minimal grid size considered is $H_{grid}$ pixels.

Subsequently, we repeatedly attempt to combine segments into larger units as follows. Let $I_i$ and $I_j$ be two segments, then they are merged into a single segment $I_k$ if the following criteria hold.

- The average Hue of both segments $I_i$ and $I_j$ differ at most by a given constant $H_{threshold}$.

- Both segments share at least one edge. Otherwise far apart segment will be merged.

- The merge is locally optimal. Only merge the closest neighbor both in spatial and in color distance.
A spatial join operation finds all pairs in the 8-connected neighborhood. Then, in a recursive process, the similarity measures for all pairs are calculated using the average hue. Using $H_{\text{threshold}}$ the merge candidates are selected. One segment can have 8 possible merge candidates. We can not simply merge all candidates, because then the similarity measures between the segments merged together could be much larger than $H_{\text{threshold}}$. Only one pair could be merged per iteration. We select the candidate with maximum similarity. If there are more candidates with equal similarity we select one at random. Once the pairs are selected we can update the histograms. The image features are derived from the enlarged image $I_k$ using the properties of its constituents.

This process continues until no more segments can be merged. The worst case complexity of this algorithm is $O(n^2 \ln(n))$, with $n$ the number of segments to start with.

There are large differences between the two algorithms considered. The S-split algorithm uses a simple segment representation, since splitting a rectangle always results in two new rectangles. Conversely, S-merge needs a polygon to follow the object boundaries.

An advantage of S-merge is that it enables reuse of the hue average. The nature of splitting does not allow us for such reuse of intermediate results at all. At each stage we have to inspect all pixels. A potential disadvantage of S-merge would be the large number of polygons to start with. The split algorithm starts with only one rectangle. In Section 4.3.3 we study the performance cost to gain a better understanding of the scalability.

Both algorithms are based on the same similarity function. They merely differ in its interpretation. The similarity function calculates the weighted distance between the features in that segment. The similarity for a single feature of two segments is calculated using the following function: $S(R_1, R_2) = (\frac{F_{R_1} - F_{R_2}}{\text{weight}})^2$, with segments $R_1$ and $R_2$ and primary feature $F$.

The collection segments following from the S-split/S-merge phase are used to calculate the secondary segment features. These features are inserted into the described BATs.

### 4.3.2 The Query Primitives

Query formulation is based on a single sample, i.e. query_by_example. This command returns a ranked list of images similar to the sample image. The command first calculates the collection of no overlapping segments from the given example image using the S-split/S-merge algorithms. For each segment the a set of features are calculated. These features are used to select similar segments, using a special similarity join operation.

For all selected images the total similarity is calculated. This is the sum of the similarity measures of all the supporting segments divided by the number of segments in the example image collection of segments.
4.3. SEGMENT IMAGE INDEXING

The query processing is facilitated by the primitives shown in Table 4.1. The first group controls the global or segment features to be used, such as control over invariance to certain transformations. For example to search invariant of rotation the user should not use the dominant texture angle.

The second group controls how much segment features may differ to still be classified as "similar". The similarity join operation uses this primitive feature to find the similar segments. This join operation finds all pairs x,y where the similarity of the features for x and y is within the specified minimum.

The last group controls which query type should be used. Also both query by example types can be selected. The primitives for text based retrieval are not given here.

<table>
<thead>
<tr>
<th>Query Primitive</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>use_avg_hue</td>
<td>use the average hue</td>
</tr>
<tr>
<td>use_dom_angle_freq</td>
<td>use the frequency of the dominant texture angle</td>
</tr>
<tr>
<td>use_dom_angle</td>
<td>use the dominant texture angle</td>
</tr>
<tr>
<td>use_histogram</td>
<td>use the global hue histogram feature</td>
</tr>
<tr>
<td>use_area</td>
<td>use the area of the segments</td>
</tr>
<tr>
<td>use_neighboor_dist</td>
<td>use the distance between the closest neighbor</td>
</tr>
<tr>
<td>avg_hue_diff</td>
<td>max. difference between hue values</td>
</tr>
<tr>
<td>dom_angle_freq_diff</td>
<td>max. difference between angle frequencies</td>
</tr>
<tr>
<td>dom_angle</td>
<td>max. difference between angles</td>
</tr>
<tr>
<td>histogram_diff</td>
<td>max. difference between histograms</td>
</tr>
<tr>
<td>area_diff</td>
<td>max. difference between the areas</td>
</tr>
<tr>
<td>neighbor_dist</td>
<td>max. difference between neighbor distances</td>
</tr>
<tr>
<td>sub_image_quering</td>
<td>Query type B</td>
</tr>
<tr>
<td>image_queriing</td>
<td>Query type A</td>
</tr>
</tbody>
</table>

Table 4.1: The Query Primitives

4.3.3 Experimental results

We conducted several experiments to show that the envisioned database of one million images could use a technique, like Region Image Indexing, to support partial image queries. Construction of this database requires a step-wise approach, because its construction is both CPU and storage intensive. Therefore, we conducted two kinds of initial experiments. First, we determine the resource requirements for the indexing algorithms on a small footprint 500-image database. Second, a web-robot is used to take a sample to assess scalability.

The 500-image database is a standard database for image analysis research at the University of Amsterdam. As such it provides a reference point for the
algorithms in terms of precision later on. We fed a sample of 100 256x256 sized images to both S-split and S-merge to determine the average number of segments in an image. This depends on the algorithmic parameters $H_{\text{threshold}}$, $H_{\text{objects}}$ and $H_{\text{grid}}$.

Figure 4.7 illustrates that S-merge should start with a reasonable grid size, i.e. very small grid sizes gives to many regions. Figure 4.8 illustrates that $H_{\text{objects}}$ should be set to 32, since splitting deeper will generally not result in more segments, due to the image size. It also illustrates that S-split finds more segments.

![Figure 4.7: The number of segments using the S-merge algorithm](image)

Based on these experiments we can predict the storage and processing requirements for the complete database. The segment administration consumes 18 bytes in the current implementation. This should be multiplied by $\min\{H_{\text{objects}}, S\}$ where $S$ is the actual number of segments determined by the algorithm. With a starting grid size of a single pixel the average number of segments found by S-merge is less than 200, i.e. approximately 4Kb to store the segment features and index structures. S-split leads to many more objects and requires about 9Kb per image. This leads to an index size of about 4 Gbyte for a database of 1M images.

In addition, we need space for the global features, e.g. URL, keywords and key-phrases, and secondary features, e.g. histogram and texture. To assess the size and to confirm the index resource requirements, we used the web-robot to obtain the first sample of about 1K GIF images from the NL domain. Table 4.2 shows the BAT sizes of this 1000 image large database. It indicates that far less than 200 segments are found per Internet image. This
4.3. **SEGMENT IMAGE INDEXING**

Figure 4.8: The number of segments using the S-split algorithm

4.3.4.3. **SEGMENT IMAGE INDEXING**

means that our 500-sample is an upperbound for the storage requirements. The table also shows the number of keywords and the distribution of multimedia objects. The storage requirements of the icon is 7.5 K and about 2.5 for the remaining features, leading to a total of about 15 Gbyte.

<table>
<thead>
<tr>
<th># BAT name</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>mmo_url</td>
<td>1002</td>
</tr>
<tr>
<td>mmo_name</td>
<td>1002</td>
</tr>
<tr>
<td>txt_keyword</td>
<td>17340</td>
</tr>
<tr>
<td>txt_phrase</td>
<td>612</td>
</tr>
<tr>
<td>img_segment</td>
<td>9042</td>
</tr>
<tr>
<td>ir_hue</td>
<td>9042</td>
</tr>
<tr>
<td>ir_texture</td>
<td>9042</td>
</tr>
<tr>
<td>ir_area</td>
<td>9042</td>
</tr>
</tbody>
</table>

Table 4.2: Bat sizes

The final question to consider at this stage is whether creation of the Region Image Indexing database won’t take forever. To quantify this, we ran a small experiment on 100 images to determine the wall-clock for the complete process. The Figures 4.9 and 4.10 show the timing results for the S-merge and S-split using different threshold values.

S-split and S-merge dominate the insertion cost, e.g. with a grid size of 4x4 pixels S-merge takes less than 3 seconds. Since localization and down-
load of the candidate images can take place in the background in parallel; this enables downloading of 30K images per day per CPU.

4.3.4 Conclusions

In this section we have introduced the necessary data structures and operators to build a image database system aimed at supporting embedded image querying. We have experimentally demonstrated that a bottom-up index construction outperforms a top-down approach in terms of storage requirements and performance. The storage overhead for the segment feature index of an image is about 4 Kbytes.

From this exercise we can also derive requirements for the Image Retrieval Algebra:

**Extensible with new Region/Segment features** The set of Region and Segment features will only grow. Therefore, the image algebra should be extensible with new region and segment features.

**Query Primitives for Segment construction and Retrieval** To support multiple segmentation algorithms primitives are needed for segment construction and retrieval.

**Index Structures for Region/Segments** To make efficient use of regions and segments index structures are required.
4.3.5 Image Retrieval Algebra

The image retrieval problem is a special case of the general problem of object recognition. When objects can be automatically recognized, and condensed into semantic object descriptors, the image retrieval problem can be solved using conventional database technology. Unfortunately, object recognition is solved for limited domains only. This calls for an image feature database model and a query algebra in which a user can express domain specific knowledge to recognize the objects of interest.

Such query algebra has the following requirements:

1. The algebra should support navigational queries and query refinement.
2. The algebra should be data independent.
3. The algebra should be based on an extensional relational framework.
4. The algebra should support proximity queries and the computational approach should be configurable by the user.
5. The algebra should be computationally complete to satisfy the wide community of (none-database) image users.

Research on image retrieval algebras has so far been rather limited. The running image retrieval systems support query by example[41] or by sketch [101], only. For example, the interface of the QBIC system lets the user choose for retrieval based on keywords or image features. These systems
have a canned query for which only a few parameters can be adjusted. It does not provide a functional or algebraic abstraction to enable the user to formulate a specific request. In the WebSeek Internet demo the user can adjust a color histogram of a sample image to specify the more important colors. However, this interface allows no user defined metric on colors.

Only Photobook [87] supports user defined similarity metric functions through dynamically loadable C-libraries. Although this approach is a step forward, it is still far from a concise algebraic framework that has boosted database systems in the administrative domain. In section 4.4 we introduce the components of such an algebra.

### 4.3.6 Logical Image Data Model

The logical data model needed for an image retrieval systems is shown in Figure 4.11. Requirements for a logical data model are: be able to capture the raw data and provide hooks to reason about semantic objects.

The top of the data model captures the image. In abstract terms, an image is a mapping from a set of pixel positions \((X)\) to a set of pixel value \((F)\). In traditional systems the constraints implicit in the data model is that all possible pixel values in a 2-D region are part of an image. As explained in Chapter 3 we use the BAT to store these mappings.

Pixel values and their pixel positions are the raw data of the images. Pixel positions can be grouped together to form regions. Each region is a two dimensional fully connected space, i.e. no holes. Each pixel position can only be part of one region.

The next level consists of segments. Segments are simply a set of regions. Since many image segmentation algorithms exist, all with their own strengths and weaknesses, regions could be assigned to many segments. These segmentation algorithms could use both the spatial and range values of the pixels of the underlying regions. A segment can contain holes, since a set of regions with similar features, for example similar color and texture, could enclose other segments with completely different features.

Segments can be merged to form objects. Each segment can end up in many objects. Object represent semantic entities, such as cars and persons. For example a car is made up out metal, glass and rubber, which all have a different features.

This shows that a logical image data model requires topological and spatial operators and abstract data types.

### 4.3.7 Physical Segment Representation

The bulk of the storage deals with region and segment representation. Large image databases require a segment representation, which is compact without
data loss. Many different approaches exist. All have proven to be useful in a specific context, but none is globally perfect.

The chain code as described by Freeman [44] encodes the contour of a segment using the 8-connected neighborhood directions. Chain codes are used in edge, curve and corner finding algorithms [70]. It is not useful for segment feature extraction, since it only represents part of the boundary of an area, no interior structure is seen. The complexity is $O(p)$ for both storage and performance, where $p$ is the perimeter of the segment.

Many boundary representations exist [61], e.g. polygons and functional shape descriptors. Functional shape descriptors use a function to approximate the segment boundary. Fourier, fractal and wavelet analysis have been proposed for this [22, 71, 95]. Although these representations can have low storage requirements, i.e. each boundary could be represented using a few parameters, they are of limited use aside from shape representation. Recalculation of the segments interior from polygons is very hard and from
functional descriptions generally impossible.

A representation to also describe the interior of the segment is run length encoding using (position, length) pairs in the scan direction [48]. This simple jet compact representation captures details description of the segments outline. Diagonal shaped segments are handled poorly by this coding schema.

The pyramid structures [109, 108] represent an segment using multiple levels of detail. They are used in image segmentation and object recognition [109, 89]. These structures are in carnations of to the quad tree [92]. The quad tree is a hierarchical representation, which divides segments recursively into four elements. The quad tree has been used to represent binary images efficiently. The tree needs only to store those segments which have a different color than its parent nodes. The complexity of this structure per segment is \(O(p + n)\), where the segment is located in a \(2^n \times 2^n\) image and \(p\) is again the perimeter of the segment. Quad trees can be stored efficiently using a pointerless representation.

Since none of the structures above solve the segments representation problem, there is a strong need for an extensible framework. It would permit domain specific representations to be integrated into a database kernel, such that scalable image databases and their querying becomes feasible.

To explore this route we use a minimalistic approach, i.e. regions are described by rectangular grids and segments by sets of regions. In line with Section 4.3, the underlying DBMS can deal with them in an efficient manner.

Database Scheme

The core of the database schema is illustrated in Table 4.3.

The first BAT group illustrate the administration of multi-media objects located on the Web. Observe that their URL is sufficient to gain access upon need. The second BAT group contains features obtained from the source, i.e. information part of the image representation format.

The final group contains features to support region-based querying. Features are used for the image segmentation process. For each obtained segment a set of features can be calculated. The img_region BAT enumerates the regions in each image. The remaining BATs represent features derived to support image querying.

### 4.4 Algebraic Primitives

Analysis of the requirements encountered in image retrieval application and the techniques applied in prototype image systems, such as [41, 101, 46], indicate the need for algebraic operators listed in Table 4.4. The parameter \(i\) denotes an image, \(p\) a pixel, \(r\) a region, \(s\) a segment and \(o\) an object. Most functions are overloaded for many types. \(T_0\) indicates that the function is
defined to work on the types: pixel, region, segment and object. $T_1$ indicates the function are defined for all types in $T_0$ and on images.

The first group provides access to the basic image features, such as pixels, regions, segments and objects. Their value is either redundantly stored as materialized view or calculated upon need. The Point, Color, Vector and Histogram datatypes are sufficient extensions to the base types supported by the database management system to accommodate the features encountered in practice so far.

The second group defines topological relationships. This set is taken from [25], because there is no fundamental difference between spatial information derived from images and spatial information derived from geographic information systems.

The third group addresses the prime algorithmic steps encountered in algorithms developed in the Image processing community. They have been generalized from the instance-at-a-time behavior to the more convenient set-at-a-time behavior in the database context. This group differs from traditional relational algebra in stressing the need for $θ$-like joins and predicates described by complex mathematical formulae.

A image join ($F_{\text{join}}$) combines region pairs maximizing a match function, $f(rs, rs) \rightarrow \text{float}$. The pairs found merge into a single segment. The
metric join \((M\text{-join})\) finds all pairs for which the distance is less than the given maximum \(m\). The distance is calculated using a given metric function, \(d(rs, rs) \rightarrow \text{float}\). The last function in this group, called predicate join \((P\text{-join})\), is a normal join which merges regions for which the predicate \(p\) holds. An example of such an expression is the predicate "similar", which holds if regions \(r_1\) and \(r_2\) touch and the average colors are no more than 0.1 apart in the domain of the color space. A functional description is:

\[
\text{similar}(r_1, r_2) := \text{touch}(r_1, r_2) \text{ and } d(r_1, r_2) < 0.1
\]

The next group of primitives is needed for selection. The image find \((F\text{-find})\) returns the region which best matches the given region, according to function \(f(rs, rs)\). The metric select \((M\text{-select})\) returns a set of regions at most at distance \(m\), using the given metric \(d(rs, rs)\) function. The predicate select \((P\text{-select})\) selects all regions from the input set for which the predicate is valid.

The last group can be used to sort region sets. We have encountered many algorithms with a need for a partial order. \(P\text{-sort}\) derives a partial order amongst objects. Each entry may come with a weight, which can be used by the metric sort \((M\text{-sort})\). This sort operation is based on a distance metric between all regions in the set and a given region. The \(N\text{-sort}\) uses a function to map regions onto the domain \(N\).

After the partial order the \(\text{Top}\) returns the top \(n\) objects of the ordered table. The \(\text{Slice}\) primitive will slice a part out of such an ordered table. The \(\text{Sample}\) primitive returns a random sample from the input set.

### 4.5 Acoi Image Retrieval Benchmark

To show the maturity of the algebra we can now formulate a functional benchmark for image retrieval problems. Many such benchmarks have steered progress in DBMS development in a variety of application areas. Examples in transaction processing are the TP series developed by the transaction processing community[55] and in geographic information systems the SEQUOIA 2000 storage benchmark. We are not aware of similar widespread benchmarks for image retrieval.

The construction of such a public benchmark would benefit both the database and image processing community. Its primary purpose is to demonstrate function and to support research in image processing and analysis in a database context. Based on our experimentation in both fields, we derived the following characteristics from the algorithms used in the image processing domain.

- **Large Data Objects** The algorithms use large data objects (>40k). Both in terms of base storage (pixels), but also the data derived incurs
large space overhead.

- **Complex Data Types** The algorithms often use specialized complex data types. No distinction is made between logical and physical models. Derived data is often stored in special data structures.

- **Fuzzy data** The computational model used is based on heuristics and fuzzy data often embedded in application code or a probabilistic model. This fuzzy data should be accompanied by some form of fuzzy logic.

**The Acoi Benchmark Data** The database for the benchmark consists of two Image sets, one of 1K images and one of 1M images. The images are retrieved randomly from the Internet using a Web robot. The set contains all kinds of images, i.e. binary and gray scale, small and large but mostly color images.

**The Acoi Benchmark Queries** Based on the characteristics encountered in the image processing community, a set of 6 distinctive queries for the benchmark has been identified, as shown in Table 4.6.

Query 1 loads the database from external storage. This means storing images in database format and calculation of derived data. Since the benchmark involves both global and local image features this query may also segment the images and pre-calculate local image features.

Query 2 is an example of global feature extraction as used in QBIC. This query extracts a normalized color histogram. We only use the Hue component of the HSB color model. The histogram has a fixed number of 64 bins. In query 3 these histograms are used to retrieve histograms within a given distance and the related images. The histogram $h$ should have 16 none-zero bins and 48 zero. The none-zero bins should be distributed homogeneous over the histogram. The query Q3a sorts the resulting set for inspection.

Query 4 finds the nearest neighboring regions in an image. Near is defined here using a user-defined function $f$. This function should be chosen so that neighbors touch and that the colors are as close as possible.

Query 5 segments an input image. Segmentation can also be done with specialized image processing functions, but to show the expressive power of the algebra we also include it here in its bare form. Finally Q6 searches for all images in the database which have similar segments as the example image. The resulting list of images is sorted in query 6a.

**The Benchmark Evaluation** To compare the results of various implementations of the benchmark we used the following simple overall evaluation scheme. The performance of the Acoi Benchmark against different implementation strategies can be compared using the equation $score =$
\[
\frac{(Q_1 + Q_2) \times DBsize + Q_3 + Q_{3a} + Q_4 + Q_5 + Q_6 + Q_{6a}}{8},
\]
where \( Q_x \) are the execution times. This way moving a lot of pre-calculation to the DB-load query will not improve performance unless the information stored has low storage overhead and is expensive to recalculate on the fly.

4.6 Initial Performance Assessment

The benchmark has been implemented in Monet using its extensible features. The DB-load query loads the images using the image import statement into the Acoi_Images set. We only load the images in the system. No pre-calculation has been performed.

The color histogram query \((Q_2)\) can be expressed in the Acoi algebra as follows:

\[
\text{var } Q_2 := [\text{normalized_color_histogram}](\text{Acoi}_\text{Images});
\]

The brackets will perform the operation \text{normalized_color_histogram} on all images in the Acoi_Images set. It returns a set of a histograms. \( Q_3 \) uses a \text{M_select} with the \( L^2 \) metric. The sorting of \( Q_{3a} \) can be done using the \text{M_sort} primitive. Query \( Q_4 \) is implemented in the Acoi algebra using a \text{F_join} with the function \( f(r_1, r_2) \) defined as follows:

\[
f(r_1, R_2) :=
\begin{align*}
\text{dist} & \left( r_1.color(), r_2.color() \right) \text{ if } r_1\text{.touch}(r_2) \\
\text{max_dist}
\end{align*}
\]

Queries 5 and 6 are implemented by longer pieces of Monet code. The segmentation of query \( Q_5 \) uses an iterative process. This process can make use of the \text{F_join} primitive to find the best touching regions based on the color distance, see [79] for full details.

Query \( Q_6 \) can be solved using a series of \text{M_select} calls. For each segment in the example image we should select all Images with similar segments, where similar is defined using the metric given. The intersection of the selected images is the result of query 6. This can be sorted using the \text{M_sort} primitive.

**The Benchmark Results** We run these queries using the small Acoi database of 1K images. The small benchmark fits in main memory of a large workstation. The database size is approximately 1G. We used a Sparc Ultra II with 128 MB of main memory running the Solaris operating system, to perform the benchmark on. Using the Acoi algebra we were able to implement the benchmark with little effort.

The initial breakdown of the results can be found in Table 4.6, which leads to the overall benchmark score is 1.158.

In the result we can see that the DB-load query takes more than 80 percent of the overall benchmark result. This unexpected result stems from
heavy swapping of the virtual memory management system. Main memory runs out quickly, so swapping will influence the performance. Based on our early experimentation with multi-Giga-byte databases this problem can be resolved with some careful loading scripts.

We found that the results of queries Q4 and Q5 were low. The non-optimized current implementation of F.join was responsible for the low performance. To improve it we moved the spatial constrains out of the F.join. This allows us to find candidate pairs based on the spatial relation between regions quickly. This way we improved the performance of the queries Q4 from 5 to 1 second and Q5 from 21 to 1.2 seconds using a few minutes of programming. A similar step in a traditional image programming environment would have meant partly re-coding several pages of c/c++ code.

4.7 Conclusions

In this chapter we introduced an algebraic framework to express queries on images, pixels, regions, segments and objects. We illustrated the expressive power of the Acoi algebra using a representative set of queries in the image retrieval domain. The algebra allows for user-defined metric functions and similarity functions, which can be used to join, select and sort regions. The algebra is extensible with new region properties to accommodate end user driven image analysis in a database context.

We have implemented the algebra within an extensible DBMS and developed a functional benchmark to assess its performance. In the near future we expect further improvement using extensibility in search methods and index structures to improve the performance of the algebra. As soon as the full Acoi database is ready we will perform the benchmark on the set of 1M images.
### Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>area(T_i)</code></td>
<td>float</td>
</tr>
<tr>
<td><code>perimeter(T_i)</code></td>
<td>float</td>
</tr>
<tr>
<td><code>center(T_i)</code></td>
<td>point</td>
</tr>
<tr>
<td><code>avg_color(T_i)</code></td>
<td>color</td>
</tr>
<tr>
<td><code>color_hist(T_i)</code></td>
<td>Histogram</td>
</tr>
<tr>
<td><code>texture(T_i)</code></td>
<td>vector</td>
</tr>
<tr>
<td><code>moment(T_i)</code></td>
<td>float</td>
</tr>
</tbody>
</table>

### Topological operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>touch(T_0, T_0)</code></td>
<td>boolean</td>
</tr>
<tr>
<td><code>inside(T_0, T_0)</code></td>
<td>boolean</td>
</tr>
<tr>
<td><code>cross(T_0, T_0)</code></td>
<td>boolean</td>
</tr>
<tr>
<td><code>overlap(T_0, T_0)</code></td>
<td>boolean</td>
</tr>
<tr>
<td><code>disjoint(T_0, T_0)</code></td>
<td>boolean</td>
</tr>
</tbody>
</table>

### Join operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>F_join_f(T_0, T_0)</code></td>
<td>{T_0}</td>
</tr>
<tr>
<td><code>M_join_d(T_0, T_0, m)</code></td>
<td>{T_0}</td>
</tr>
<tr>
<td><code>P_join_p(T_0, T_0)</code></td>
<td>{T_0}</td>
</tr>
</tbody>
</table>

### Selection operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>F_find_f(T_1, T_1)</code></td>
<td>T_1</td>
</tr>
<tr>
<td><code>M_select_d(T_1, T_1, m)</code></td>
<td>{T_1}</td>
</tr>
<tr>
<td><code>P_select_p(T_1, T_1)</code></td>
<td>{T_1}</td>
</tr>
</tbody>
</table>

### Ranking and Sample operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>P_sort({T_1})</code></td>
<td>{T_1}</td>
</tr>
<tr>
<td><code>M_sort_d(T_1, T_1)</code></td>
<td>{T_1}</td>
</tr>
<tr>
<td><code>N_sort({T_1})</code></td>
<td>{T_1}</td>
</tr>
<tr>
<td><code>Top({T_1}, int)</code></td>
<td>{T_1}</td>
</tr>
<tr>
<td><code>Slice({T_1}, int, int)</code></td>
<td>{T_1}</td>
</tr>
<tr>
<td><code>Sample({T_1}, int)</code></td>
<td>{T_1}</td>
</tr>
</tbody>
</table>

Table 4.4: The Image Retrieval Algebra

<table>
<thead>
<tr>
<th>Join operations</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>F_join_f(L, R)</code> -&gt; {T_0}</td>
<td>{lr</td>
</tr>
<tr>
<td><code>M_join_d(L, R)</code> -&gt; {T_0}</td>
<td>{lr</td>
</tr>
<tr>
<td><code>P_join_p(L, R)</code> -&gt; {T_0}</td>
<td>{lr</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Selection operations</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>F_find_f(L, r)</code> -&gt; T_1</td>
<td>l ∈ L, \forall l' ∈ L ∧ f(l', r) &gt; f(l, r)</td>
</tr>
<tr>
<td><code>M_select_d(L, r)</code> -&gt; T_1</td>
<td>{l</td>
</tr>
<tr>
<td><code>P_select_p(L, r)</code> -&gt; T_1</td>
<td>{l</td>
</tr>
</tbody>
</table>

Table 4.5: Signatures of the Join and Selection operations
## 4.7. CONCLUSIONS

<table>
<thead>
<tr>
<th>nr</th>
<th>query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>DB-load</td>
</tr>
<tr>
<td>Q2</td>
<td>{h</td>
</tr>
<tr>
<td>Q3</td>
<td>{i</td>
</tr>
<tr>
<td>Q3a</td>
<td>sort Q3</td>
</tr>
<tr>
<td>Q4</td>
<td>{n_1n_2</td>
</tr>
<tr>
<td>Q5</td>
<td>{rs</td>
</tr>
<tr>
<td>Q6</td>
<td>{i</td>
</tr>
<tr>
<td>Q6a</td>
<td>sort Q6</td>
</tr>
</tbody>
</table>

| table 4.6: Benchmark Queries |

<table>
<thead>
<tr>
<th>Query</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>2865</td>
</tr>
<tr>
<td>Q2</td>
<td>598</td>
</tr>
<tr>
<td>Q3</td>
<td>1.5</td>
</tr>
<tr>
<td>Q3a</td>
<td>0.3</td>
</tr>
<tr>
<td>Q4</td>
<td>1.0</td>
</tr>
<tr>
<td>Q5</td>
<td>1.2</td>
</tr>
<tr>
<td>Q6</td>
<td>1.5</td>
</tr>
<tr>
<td>Q6a</td>
<td>0.3</td>
</tr>
</tbody>
</table>

| table 4.7: The Acoi Benchmark Results |