Image Database Management System Design Considerations, Algorithms and Architecture

Niels Nes

Contents

Chapter 1

Introduction

Relational Database Management Systems (RDBMS) are commonly used for many business application for almost three decades. Their user friendly declarative query language and abstract logical view on data organization are the main reasons for this success. This declarative query language releases business application developers from dealing with implementation details. They only have to focus on what information is needed, not on how to get it. The later is the task of a query language compiler using a relational algebra as a sound mathematical basis, and gives many ways of optimization.

Since commonly available hardware makes it possible to capture and store images, a need for databases with image management capabilities arose. Large collections of images can easily be obtained from sources like CD-roms, DVD and the Internet. Searching through these collection for particular images is a complex task which requires a proper image retrieval query language. A research problem still lacking general acceptable solutions.

Image databases can be used for many applications. Simple examples include presentations, games and educational software. We could think of an educational program teaching students a foreign language or taking an exam on the traffic rules. In these applications the image database is used as a persistent image store, which gives the user physical data-independence. The image is physically stored somewhere, but the user/application is not interested in its whereabouts. Instead the image can be retrieved using a logical name. Physical data-independence permits a database administrator to move the image without the application noticing it. The advantage is that the images could be scattered over many disks, distributed over various file systems, located worldwide.

Another application domain which would benefit from image database support is image analysis. Image analysis applications try to determine the semantics of an image. The image analysis process involves, segmentation, of images into objects, clustering objects, searching and data reduction. These

operations are strongly supported by database systems. In the image analysis domain each image is a scene taken from the real world. The transition from a real 'continues' world to a 'digital' world introduces many challenging problems. The image analysis domain could benefit substantially from using an image database management system.

Image analysis applications require that images can be accessed on other methods than logical names. Retrieval of stored images requires access methods based on annotations or the image content. Therefore, new query primitives and search methods are needed. A complicating factor is the lack of accurate data. The data involved, both the image itself and the data derived, are fuzzy since perfect image capturing devices do not exist. Also images are subject to varying interpretations. An image will be interpreted different by any other person, caused by the persons background knowledge.

Image database researchers focus on an important subset of applications; called Image Retrieval by Content. These applications try to retrieve images based on the content of the images. A recurring querying scheme is 'query by example'. Given a sample image the system finds a small set of images with similar content. The critical points here are what is considered similar and what content aspects, called features, should be used.

Similarity is often defined as a mathematical function on the features. These so called similarity measures make calculation of similarity possible. Many different similarity measures have been proposed, examples can be found in [63]. Histogram intersection [106] and weighted Euclidean distance [41] are among the most commonly used measures.

To facilitate image queries a two step process is used. First the images are loaded in the database. Secondly for each image feature values are extracted, a fixed set of features is used. The image queries are expressed in terms of these features and selection is based on predefined similarity measures. Most image retrieval systems provide limited control over the features used and definition of similarity measures.

1.1 RDBMS vs IDBMS

Image database management systems (IDBMS) differ from traditional databases management systems in major ways. The first difference is the data complexity. Transaction records are composed of simple data elements like names and numbers. Images, on the other hand, are complex, large arrays of complex values. Also the derived data in an IDBMS is complex, for example at we the contour of an object in an image can be described using a polygon.

On top of the complex data problem is the problem of large object sizes. To illustrate, image databases handle large amounts of sizeable objects (from a few Kilo bytes to several Mega bytes). The granularity of transactions

ranges from a few hundred to a few Kilo bytes. These large data elements results in new requirements for the IDBMS physical storage mechanism.

Query formulation in image databases is more complex. The image domain brings along a large set of operations. The combination of this set of operations inside the realm of relational operators gives an explosion of possibilities making it difficult to oversee. The interplay between the various operators is difficult to predict.

The semantics of the data stored in an IDBMS is unknown. In a traditional transaction processing application all data values are understood by casual users. When we query the database for all persons with age between 18 and 25 we know exactly what we get back. Contrary an image query is worth a thousand words conveying a multitude of semantics. It may well have various interpretations. These semantics have no mathematical basis. This complicates posing and answering queries. A related issue is that data, especially derived data is fuzzy. Many measurement errors may have altered it. The image may largely differ from the real world scene from which it is taken. Therefore, the database should be able to handle incomplete and noisy data.

The last difference from traditional database systems lays in the way the databases are used. The general use of a database in business applications involves many transactional updates, such as inserting, updating and deleting values. In image database systems updates tend to be much more incremental. Once an image is inserted there is little chance that the image is changed or deleted. Therefore, image database applications are much more query intensive than traditional systems. Knowing these characteristics makes it possible to improve the overall system performance.

1.2 Contributions and Thesis Outline

The main contribution of this dissertation is an exploration of facilities needed for the design and construction of a successful image database system. Portions have been implemented in the context of the Monet DBMS to obtain a first assessment of the choices made. However, a complete IDBMS is beyond the scope of this thesis, for it requires a large, multi person, engineering effort. The facilities needed are organized by chapter as follows.

Chapter 2 introduces the basic requirements for an image database, i.e. image storage and management. We founded the requirements on the theory of the image algebra, which supplies us with a complete set of image types and operations.

In Chapter 3 we introduce the physical representation for our image data type. We map the image processing operations on relational operators making it possible for query optimizers to optimize these operations. Also we show possible optimizations of mapping both in storage and processing

requirements.

To obtain the requirements for image retrieval we looked at new image retrieval methods: the multi-level signature and region image indexing.

One requirement obtained using these experiments is the need for a special image retrieval query language. Therefore, we developed the image retrieval algebra which forms the proper basis for such a language. In Chapter 4 this algebra is introduced. Many new database primitives are introduced and their relevance is shown via a preliminary benchmark definition geared at image retrieval queries.

In Chapter 5 we proved the applicability of image databases in the field of image analysis, using a case study. In this study we looked at line clustering, a basic step in many image analyzing applications. We showed databases are effective; they reduce the programming effort and stimulate code reuse, and they are efficient; our implementation proved significant performance improvement over the earlier attempts in this area.

The inherently fuzzy data as found in this field lead to algorithms which search for similar objects. To handle such queries better we introduce in Chapter 6 a new query predicate, *the fitness join*. Making this key operator explicit at the algebra level provides a handle to optimize its processing. A few directions for optimization are described.

In Chapter 8 we introduce an indexing structure, *the metric index structure*, to optimize this fitness join operation at a minimum of cost overhead. We showed that metric indexing is a low cost index structure outperforming the R-tree[52]. This structure uses the triangular inequality to reduce the number of calculations of a fitness function.

The Chapters mark a route to a successful IDBMS. The individual steps taken have been commented upon by the research community These Chapters were published earlier in other forms:

- The Multi-level Signature image retrieval method was explained in the paper, 'Database support for image retrieval using spatial-color features' [78].
- An earlier version of the Region Image Indexing was explained in the paper, 'Region-based Indexing in an Image Database'[79].
- The paper 'The Acoi Algebra: A Query Algebra for Image Retrieval Systems'[80] introduced the first image retrieval algebra.
- In the paper 'Database support for line clustering'[81] image databases were first introduced for image analyzing tasks.
- Metric indexing was explained in the paper 'Metric Indexing to Improve Distance Joins'[82].
- The fitness joins are explained in 'Fitness joins in the Ballroom'[66].

Other related publications not discussed in this thesis:

- The papers, 'Image Retrieval Using Linear Greyscale Granulometries'[35] and 'Color Image Texture Indexing'[36] explain image retrieval techniques where texture is described using object granularity.

In Chapter 8 we summarize the results of the thesis and look forward for further research possibilities.

Chapter 2

Image Databases

In this chapter we explain what kind of database management system appropriate is to construct and manage image collections. In particular, we introduce a requirement list for assessing a DBMS for deployment in this area. Subsequently we introduce our experimentation platform, the Monet DBMS[8], in more detail. We conclude with a short evaluation of commercial database and image retrieval systems, including Monet, against our requirement list.

2.1 Multi-Media Database Systems

The research field of image database systems can be characterized as a subset of the field of multi-media database systems. Multi-media database management systems are software systems dealing with data both with well- and ill-defined semantics. Data with well-defined semantics is usually called structured data, examples are numbers and formatted records. Data with ill-defined semantics is usually called unstructured data, such as text, audio, image and video. Example application areas for Multi-Media database management systems are Digital Libraries, Video on demand systems and news archives.

The combination of multiple media imposes additional requirements on the DBMS, because they typically can not be dealt with in isolation. The combination of multiple media gives additional valuable information and may therefore be easier to understand. Another important issue is *quality of service*, i.e. the clients of the multi-media database may not be able to handle a fast multi-media stream.

In this thesis, we restrict our research to a single media type, namely still images from the context of database research. Although this greatly reduces the domain, it makes the project manageable within the limited resources given.

We belief that progress obtained in the design of image database man-

agement systems carries over to the subsequent extensions to deal with other media types. The trajectory followed results in a requirements list for image databases. Many of these requirements also hold for multi-media databases at large.

The basic functionality to support Image Database Management Systems centers around the following issues:

- Image Storage
- Image Operations
- Derived Image Data
- Image Semantics
- Image Queries

These issues are elaborated upon in the next sections.

2.1.1 Image Storage

The first requirement for an Image Database Management Systems (*IDBMS*) is, of course, the ability to store images. The image processing community uses many image file formats. Well known examples include: TIFF, GIF, PNG, JPEG, PPM, BMP, PICT and XPM. These formats have been designed for specific applications or as an attempt to set a standard for image data exchange, however, non is general enough to support all image types. To illustrate, many of these file formats, such as GIF, PNG, JPEG, PICT and XPM were designed for viewable images, i.e. mono-chrome, gray scale and color images. This restricted domain makes them not suited for storing images in an IDBMS. For example, satellite images and stereo images can not be stored using these formats. Moreover, the image processing community still lacks a standard image file format, i.e. there does not exist an (extensible) data model to reason about images. As a result, we are challenged to come up with an appropriate image representation to cater for all intended use. For such an image representation scheme, the DBMS should provide an abstract data type (*ADT*) facility which provides for:

1. Extensible representation and algebra
2. Multiple views on components to support segmentation.
3. Cheap (de)compress functions.
4. Easy Input/Output Facilities

Extensible Representation and Algebra

In our search for a flexible image representation scheme, we came across the image algebra[90]. Before we continue with the discussion of our image representation of choice we give a synopsis of its concepts.

The image algebra[90] is a mathematical theory focused on the analysis and transformation of digital images. The main goal was to define a comprehensive and unified theory of image transformations, image analysis and image understanding.

The image algebra is defined as a heterogeneous algebra. Such an algebra is defined as follows:

> **Definition.** An *algebra* $\mathcal{A}$ is a pair $\mathcal{A} = (\mathcal{F}, \mathcal{O})$, where
>
> 1. $\mathcal{F} = \{F_\lambda\}_{\lambda \in \Lambda}$ is a family of non-empty sets of different types of elements and the subscripts λ are members of some indexing set Λ, and
> 2. $\mathcal{O} = \{O_k\}_{k \in K}$ is a set of finitary operations (for some indexing set K), where each $O_k \in \mathcal{O}$ is a mapping of the Cartesian product of some of the F_λ's to another.

The elements F_λ of $\mathcal{F}$ are called the sets of operands of $\mathcal{A}$, and the elements $O_k \in \mathcal{O}$ are called the operators of (or operations on) $\mathcal{A}$.

An Algebra $\mathcal{A}$ is called a *homogeneous* or single valued algebra, if $\mathcal{F}$ contains only one element i.e., $\mathcal{F} = \{F\}$, otherwise $\mathcal{A}$ is called a *heterogeneous* or *many valued* algebra.

The Image algebra[90] defines an image as follows.

> **Definition.** Let F be a homogeneous algebra and X a topological space. An F-valued image on X is any element of F^X. Given an F-Valued image $\mathbf{a} \in F^X$, then $\mathcal{F}$ is called the *set of possible range values* of $\mathbf{a}$ and X the *spatial domain* of $\mathbf{a}$.

It is often convenient to let the "graph" of an image $\mathbf{a} \in F^X$ represent $\mathbf{a}$. The graph of an image is also referred to as the *data structure* representation of the image. Given the data structure representation $a = \{(x, a(x)) : x \in X\}$, then an element $(x, a(x))$ of the data structure is called a *picture element* or *pixel*. The first coordinate $\mathbf{x}$ of a pixel is called the *pixel location* or *image point*, and the second coordinate $\mathbf{a(x)}$ is called the *pixel value* of $\mathbf{a}$ at location $\mathbf{x}$.

Many digital images require the topological space X to be a subspace of $\mathcal{Z}^2$. A sequence of images can be modeled using $X = \mathcal{Z}^3$, with $x \in X$ of the form $x = (x, y, t)$, where the first coordinates (x, y) denote spatial location and where t denotes a time variable.

The value set F can be replaced with $\mathcal{Z}_{2^k}$ or with $(\mathcal{Z}_{2^k}, \mathcal{Z}_{2^l}, \mathcal{Z}_{2^m})$. The first provides us with digital integer-valued images of k-bits. The second provides us with digital vector-valued images.

The image algebra defines a logical image representation, we will follow this definition of an image. To use a logical representation calls for an implementation, i.e. a physical representation. We elaborate more on our physical image representation including performance analysis in Chapter 3

Color Models

Mono-chrome and grayscale images can easily be represented using $F = 0, 1$ and $F = \mathcal{Z}$. More problems arise when we try to model color images. Physically color is a composition of light signals of different wavelengths. These signals are discretized and represented using so called color models. Many such models exist and each has its strong and weak points, see [45]. Examples are RGB, CMY, HSV, HSI, L*a*b*, XYZ, UVW and xyz. The RGB and CMY models are used for output devices, such as displays and printers. XYZ is a color model which is device independent. The L*a*b* and L*u*v* are perceptually uniform, i.e. distances in these spaces correspond to human perceptual differences. The HSV and HSI are intuitive to the user, Hue is the bare color, Saturation is the infection of this hue with other colors, and I is the intensity of the color.

The different models have different applications, RGB and CMY color models are used for output. For many image operations other color models are more appropriate. Therefore, conversion primitives from one color model into another is a pre-requisite for any image analysis system.

An additional requirement for color images is to store the color model information. This can be done directly, using a separate table, or implicitly, using a special pixel value type.

From a semantic point of view there is no reason to differentiate among monochrome, grayscale or color images. They are all images. They can all be stored using (the more general) color images. We therefore like to treat all images equal. Unfortunately the theory for color images is less evolved then for gray and monochrome images. This makes explicit type coercion necessary.

Image Segments

Segments represent interesting parts of an image. Many image processing applications first reduce their problem by looking only at the segment or region of interest within an image. This pre-processing can reduce the resource requirements substantially. This leads to the image database requirement of segment representation and segment construction.

As a consequence images can also be seen as a collection of disjoint segments. Each segment would contain different, but interesting information. This alternative view on images and segments indicates the concepts are closely related. To fully exploit this resemblance the concepts should be

represented using the same logical representation. Image operations should therefore also work on segments without additional work.

Our image representation, a mapping from a spatial domain X to a domain of range values F, fits both concepts of images and segments. Operations to segment an image would therefore easily return new images.

Image Compression

An image database will store lots of images, taking up lots of disk space. Compression techniques to reduce storage can be used. Two alternatives exist, we can compress each image in isolation or we can compress a set of images together. The second will result in better overall reduction, but the first makes access to a single image still reasonably fast. This choice between performance and storage will depend on the applications using the database. Therefore, the database administrator should be able to make this choice. We therefore support compression on the image and table level.

We aim with image set compression on sets of images that contain very similar images. Therefore, we envision of set image compression using a base image and image differences. Each image in the set is stored based on a base image. The differences are computed and stored. Each resulting image can then again be compressed using the single image compression techniques. A 3D wavelet-based approach over the set might be effective.

Input/Output Facilities

Another important requirement for an image ADT are easy input and output facilities. We support input from and output to various image file formats. This makes it possible to access the large number of images currently stored in these formats. Support for output into these types makes it possible to reuse the huge amount of existing software. In our image input/output module we support conversion of our image ADT to JPEG, GIF, TIFF and PPM.

2.1.2 Image Operations

From a database perspective having a single image data type in the query language is ideal. It greatly simplifies query construction and optimization. This single image data type should come with a complete set of operations, such that by combining operations all relevant logical image operations can be performed.

The Image Algebra [90] defines a complete set of operations for all image types. It consists of operation on images and templates. All operations use basic mathematical functions on the spatial domain or range value domain. Therefore, the semantics are properly determined by the semantics of these basic operations.

For some image types considered in practice, however, most of the mathematical operations have no clear semantics. For example mathematical morphological operations on multi channel images, such as color images, have no proper theoretical background yet. Also there is no consensus on the linear filter theory for these images. In these directions progress is made as can be read in [21], [107] and [37].

Although a complete set of operations is known for the individual image types, their results are usually very different. A histogram of a color image is different from a histogram of a gray scale image. This complicates any query language greatly.

Dynamic resolution of the result types would make query optimization impossible. A query optimizer needs to know the result types of all operations to be able to make a complete query graph. With a single image type this is impossible. Therefore, we need to express an image as a complex data type which is uniquely described by the type of its spatial domain and its range domain. This requires a polymorphic image data type.

Operations

Following the image algebra [90] we can classify the operations on images in the following categories.

- Restriction and Extension
- Induced pixel operations
- Reduction Operations
- Spatial Operations
- Template Image Operations
- Template Operations

The image algebra defines two operations, *domain(***a***)* and *range(***a***)*, to extract point sets and value sets from a particular image, **a**. The domain of an image is the set of points expressing the spatial extent of the image. The range of an image are the range values of an image, for example the gray values or colors of the image.

Image Restriction removes pixels from an image. This can be realized through both the spatial domain and range value domain. Given a set of points, an image can be reduced to include only these positions. Or given a set of range values, an image can be restricted to only include these values. The inverse of image restriction is image extension. This operation adds pixels to an image, which it not yet contains. A combination of image restriction and extension could be used for replacing parts of an image, for example the blue screen replacement often seen in video editing.

Induced pixel operations are unary and binary operations on individual image pixels. If the operation γ is a binary operation on F, then γ induces a binary operation γ on F^X defined as follows:

Let $a, b \in F^X$, then

$$a\gamma b = \{(x, c(x)) : c(x) = a(x)\gamma b(x), x \in X\}$$

If the operation θ is a unary operation on F, then θ induces a unary operation, also called θ on F^X defined as follows:

Let $a \in F^X$, then

$$\theta(a) = \{(x, c(x)) : c(x) = \theta a(x), x \in X\}$$

See table 2.1 for a list of unary and binary image operations.

Operation	**Description**
$-a$	image negation
$\neg a$	logical image negation
$sin(a)$	sinus image
$a + b$	image addition
$a - b$	image subtraction
$a * b$	image multiplication
a/b	image division
$a \vee b$	image minimum
$a \wedge b$	image maximum
$a < b$	image smaller than
$a >= b$	image larger equal than

Table 2.1: Example Induce Pixel Operations

When for one of the operands of the binary operation a constant value is used we get a scalar operation. If the operation γ is a binary operation on F, then γ induces a binary scalar operation γ on F^X defined as follows:

Let $k \in F$ and $a \in F^X$, then

$$a\gamma k = \{(x, c(x)) : c(x) = a(x)\gamma k, x \in X\}$$
$$k\gamma a = \{(x, c(x)) : c(x) = k\gamma a(x), x \in X\}$$

The global reduction operations reduces an image into a single complex value. Let operation γ be a binary operation on F, then γ induces a unary operation

$$\Gamma : F^X \rightarrow F$$

called the *global reduce operation*, which is defined as

$$\Gamma a = \Gamma_{x \in X} a(x) = \Gamma_{k=1}^{n} a(x_k) = a(x_1)\gamma a(x_2)\gamma \ldots \gamma a(x_n).$$

Simple examples of reduce functions are addition, multiplication, minimum and maximum of pixels.

Spatial operations transform images based on the point set, which represents the topology of the image. Examples of spatial transforms are image translation, rotation and reflection. Also the family of affine transforms are spatial operations. Let $f : Y \to X$ and $a \in F^X$, then we define the induced image $a \circ f \in F^Y$ by:

$$a \circ f = \{(y, a(f(y))) : y \in Y\}$$

Template image operations transform images based on templates. A *template* is an image whose pixel values are images (functions). Formally defined as follows:

> **Definition.** A *template* t is an F-valued *template* from $\mathbf{Y}$ to $\mathbf{X}$ is a function t: $\mathbf{Y} \to F^{\mathbf{X}}$. Thus, t $\in (F^{\mathbf{X}})^{\mathbf{Y}}$ and t is an $F^{\mathbf{X}}$-valued image on $\mathbf{Y}$.

For notational convenience we define $\mathbf{t}_y \equiv \mathbf{t}(\mathbf{y})\ \forall$ y $\in \mathbf{Y}$. The pixel values $\mathbf{t}_y(x)$ of this image are called the *weights* of the template at point $\mathbf{y}$.

We can divide templates into two categories, the translation variant and invariant templates. A template is called translation invariant when for each triple x,y,z $\in \mathbf{X}$ we have $t_y(x) = t_{y+z}(x + z)$. Many of the translation invariant templates can be defined pictorially. See Figure 2.1.2 for an example pictorial definition of a template.

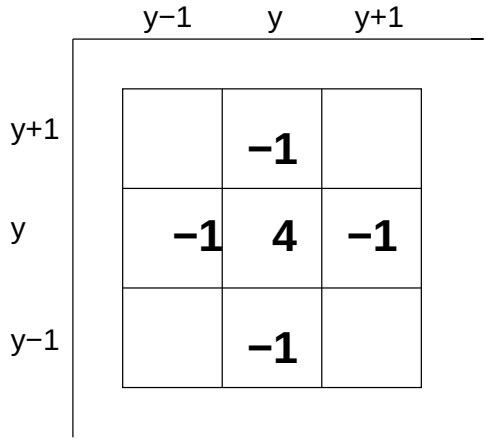


Figure 2.1: Example Template

A template image operation performs an induced pixel operation for each image, t_y, in the template. Each resulting image is reduced using a global image reduce operation. The resulting value will be the pixel value at the

pixel position y in the resulting image. Formally, let template t $\in (G^X)^Y$, image $a \in E^X$, and $a \bigcirc t_y \in F^X$ and $\Gamma(a \bigcirc t_y) \in F$. It follows that the binary operations $\bigcirc$ and γ induce a binary operation

$$\circledcirc : E^X * (G^X)^Y \rightarrow F^Y,$$

where

$$b = a \circledcirc t \in F^Y$$

is defined by

$$b(y) = \Gamma(a \bigcirc t_y) = \Gamma_{x \in X}(a(x) \bigcirc t_y(x))$$
$$= (a(x_1) \bigcirc t_y(x_1)) \gamma (a(x_2) \bigcirc t_y(x_2)) \gamma \ldots \gamma (a(x_n) \bigcirc t_y(x_n)).$$

This is the right product of image a with template t also the left product of a with template t exists.

Example template operations are image convolution and the basic morphological operations, dilation and erosion. In case of the convolution, the original image is multiplied with each template image. Each resulting image is summed to a single value.

A histogram for an image $a(x)$ can be calculated uses template operations. A template, $t \rightarrow (N^Y)^X$, used together with a function,

$$t(a)_x(j) = \begin{cases} 1 & \textbf{if a(x) = j} \\ 0 & \textbf{otherwise} \end{cases}$$

transform an image into a set of images. Using an image reduce operation which sums each image the histogram is calculated.

Templates are just a special kind of image. This assures all image operations are also defined on templates. We can restrict and extend templates. The induced operations on templates map each operation on each pixel, i.e. an image. So template addition maps to image addition for each image in the template and the image addition will map to pixel addition for each value in these images. These template operations make the image algebra such a powerful framework.

Requirements Summary

The requirements involving image primitives are summarized as follows.

1. Requires support for a polymorphic image type.
2. Requires support for the complete set of image operations as specified by the image algebra on this image type.

2.1.3 Features

Another basic requirement for image databases is storing and managing derived data. In the image processing domain derived data types are called *features*. These features are used to describe, interpret or understand the image data. The features derived from the complete image are usually called global features as opposed to the local features calculated at a region, segment or single point within the image.

Over the years, a large collection of image features has been proposed, which can be grouped into a few categories: color, texture, frequency analysis, and shape features. Some examples in each category are given below.

Color Features Average color, dominant color, color histogram, color distribution and color variance.

Texture Features Dominate angle, object granularity.

Frequency Features wavelet and discrete Fourier transforms

Shape Features Circularity, eccentricity, bounding energy, boundary, moment features.

Example single pixel value features include intensity, color and reflectivity. Over a region or segment a histogram of the pixel values can be calculated, this is called a complex feature. From this histogram many features can be derived, like dispersion, mean, variance, mean square value and average energy. A second order histogram, a histogram of all pairs of pixel values, is also used often[28].

Texture is observed in the structural patterns of surfaces of objects such as wood, grain, sand, grass and cloth. Textures are usually described using a repetition of basic texture elements. Natural textures have usually random repetitions and changing texture elements. Artificial textures are often deterministic and periodicy. Often the textures are described using measures for the coarseness of the basic textures, periodic and orientation. Many other texture features exists. Examples can be found in [49, 60] and [61].

Examples of frequency features are based on the discrete Fourier and wavelet transforms. The Discrete Fourier Transform (DFT) results in a decomposition of the image in the frequencies of cosine functions. The frequencies tell us something about the content of the image. A high number of high frequency cosine functions indicates many small changes in the image. Low frequencies would indicate a rather smooth image.

The wavelet transform [31, 104, 114] analyses an image at multiple scales. It recursively decomposes an image using a low and a high band filters. This gives a similar description of the image as a Fourier transform, but with an additional component of locality. The Fourier transform only globally

decomposes a signal in its frequencies, i.e. no locality is preserved. The wavelet transform uses small filters with a limited size so it preserves locality.

Shape features can be divided into two categories, geometrical and moment features. Examples of geometrical features are perimeter, area, max-min radii and eccentricity, corners, roundness, bending energy, holes, Euler number and symmetry. Moment features are e.g. center of mass, orientation, bounding rectangle, best-fit ellipse and eccentricity. Object boundary and skeleton are also interesting features. Shape can also be described using Fourier and wavelet coefficients.

Another important category are the spatial relations within the image. However, they are not often used in prototype image retrieval systems. Example spatial relations are overlap, touch and disjoint.

This list of features is by no means complete, but it gives a good indication of the various features studied in the field. New features are likely to be found to solve specific problems. This requires that the database management system needs an extension mechanism for both data structures and operations on (complex) features.

Although many features exist, a limited number of data structures would suffice to store them. Many features are single values, i.e. no need for extra data structures. Some examples are area, pixel sum, and mean orientation. These can be stored using the database management systems built-in types, such as integer and floating point number.

Multi value features, like histograms of pixel values, vectors of eigenvalues, moment description vectors and segment descriptions, such as polygons, need additional data structures. However, a vector of complex data values would suffice to represent many of them. For polygons and histograms special data structures are needed. In the area of geographic information systems proper representations and index structures for geographic data like polygons exist[13].

The local features are calculated over parts of an image. For instance a single pixel or a segment can be the basis for this calculation. Since an image may have several pixels or segments, these features usually result in feature value sets for the whole image. This complicates calculation but the additional information may also produce better retrieval results[100].

Invariant features

One aspect of features has received great interest from the image processing community, namely their in variances to certain aspects, such as scale, rotation, view point and light source. To illustrate, assume we are searching for a certain scene in our image database. We do not care if the scene is recorded under a white or under a colored light source. In that case we should use features invariant under light sources. This means we have to look at the hue color component only or extract the color shift. But when

interested in sunsets or images taken at indoor dance parties, we definitely want variant features. In case of the sunsets we would like natural light sources, in the later case we look for artificial light sources. A similar story holds for the other variances, for example scale invariance could be useful unless your searching for objects you know the size of.

The negative aspect of invariance in feature space is that it reduces the feature selectivity. Invariance to some aspects makes a feature less specific and, therefore, less selective. Therefore, from a retrieval point of view invariant features are certainly not more important then variant features. Using the appropriate one at the correct time is far more crucial. This means that invariance is a predicate to be expressed at the query time only.

Retrieval requires that the permissible variances can be modeled explicitly. Modeling (in)variances requires the knowledge about the aspects that a feature is variant to. So if a feature is variant to a light source we should record that.

Requirements Summary

The requirements involving image features are summarized here.

1. The IDBMS should support for feature data types
2. The IDBMS should have support for modeling feature (in)variances.
3. Invariance can be expressed in the Query Language.

2.1.4 Image Semantics

Image recognition is research concerned with object recognition, i.e. tries to recognize the objects in an image and attache a description to these objects. Unfortunately, the image recognition problem has not been (and cannot be) solved in general.

However, in certain sub-domains interesting results have been obtained. One striking example is face recognition[86]. When it is possible to recognize the objects, we can construct a semantic description. Such semantic descriptions should be stored in an image database as well, which introduce new interesting problems. We will mention two: the *multiple interpretation problem* and the *accumulated error problem*.

The former stems from the fact that an image has many interpretations. Everybody can have its own interpretation of an image depending on the knowledge and cultural setting of the person. This results in many possibly large semantical descriptions. The consequence is that at query time, such an image database should be able to reason with multiple interpretations.

The second problem, the accumulated errors, results from the fact that images are always derived from inaccurate devices. When recoding an image

using some sensor error signals are added to the original scene. Also because of the digitization errors are introduced. Building semantic descriptions for these images will yield an accumulated error. The database management system should therefore be able to handle errors and error propagation.

Requirements Summary

The image recognition problem is still an open research area. Therefore, the problems related to representing the semantic information will not be considered in this thesis. In the future when semantics are attached to images these problems will come back and will than result in requirements for the image database system. Mostly at level of data modeling and semantic driven querying.

2.1.5 Image Queries

In database systems all relational operations are based on logical predicates being either true or false. This rigid logic perspective works, because the semantics of the data entities in business database systems are known and fixed.

For image database systems this is no longer sufficient. The world of images is a lot more fuzzy. Images can be very similar, but are hardly ever exactly the same. This makes it hard to write boolean predicates, such as image equality. A fuzzy set approach is in order here as an alternative.

Image queries are often navigational and steered by a user. Take for example image retrieval systems, which let a user navigate through the image space. In such systems the user constantly refines his query to navigate to the desired image. The reason for this navigational query approach are two fold. First, the mentioned fuzzy data makes predicates hard to use. The second reason is there is still little knowledge of the applications that uses an image database. This makes it hard to predict what sort of queries are needed, because it is unknown what the interesting data is.

Since the predicate logic expressions are hard to use, they should be replaced by a new technique for comparison. A solution found in many image retrieval systems is based on similarity measures.

> **Definition.** A Similarity Measure, $S(a, b) \rightarrow \mathcal{R}_{[0,1]}$, expresses how similar two objects are.

Many similarity measures have been defined. For vectors different similarity measures exist as for value sets. One such fixed form similarity measure is based on the Minkowski metric.

$$S(a, b) = \frac{\sum_{i=0}^{n} \|a_i - b_i\|}{\sum_{i=0}^{n} max(a_i, b_i)} \tag{2.1}$$

This measure assumes the features in the vector are all unrelated, which is usually not the case. For example is the color of an object preserved by a human dependent on the colors in the area around the object.

The best known similarity measure is histogram intersection, which is formally defined as:

$$S(a,b) = \frac{\sum_{i=0}^{n} min(a_i, b_i)}{\sum_{i=0}^{n} a_i} \tag{2.2}$$

This is a non symmetrical definition, i.e.. the similarity for a,b and b,a are not equal. Therefore, the following symmetrical definition is also used frequently.

$$S(a,b) = \frac{\sum_{i=0}^{n} min(a_i, b_i)}{\sum_{i=0}^{n} max(a_i, b_i)} \tag{2.3}$$

These measures are both used for color histogram similarity calculations. The reason for its popularity is its robustness against cluttered images. and its invariance to scale and rotation. Also the measure is less variant to different view points. The measure also assumes the features in the vector are unrelated, for color histograms this is clearly not the case.

Therefore, another well known measure, i.e. the weighted Euclidean similarity measure is used. This measure comes from the family of squared similarity measures, which is formally defined by:

$$S(a,b) = 1 - \sqrt{\sum_{i=0}^{n} (a_i - b_i)^2} \tag{2.4}$$

And the weighted Euclidean similarity measure is defined by:

$$S(a,b) = 1 - aWb^T \tag{2.5}$$

Where W is an $(n * n)$ matrix which represents the weighting factor for each i, j pair. Using this weighting factors the relations among feature values in the vector can be modeled. This W should be derived from global database properties.

Measuring the similarity between feature sets is a less touched research direction. One known measure is the set intersection measure, as defined by:

$$S(a,b) = \frac{\|a \cap b\|}{\|a \cup b\|} \tag{2.6}$$

The problem of feature set comparison becomes even more difficult when the values in the sets are fuzzy. In that case the set elements need to be compared also using some similarity metric. Then the semantics of the $\cap$ and $\cup$ operators have to be changed too.

A database query will usually involve many different features, which should be compared using different similarity measures. Queries over multiple similarity spaces could be handled in various ways. A simple solution

could be to used traditional boolean predicates. For example selecting image on color and texture requires the color feature similarity value should excide a threshold t_c and the texture feature similarity value should excide a threshold t_t.

A more advanced method could be a combination of the feature values involved[58]. This method has the drawback that the search space explodes, since it combines two feature spaces in one, which makes query optimization difficult.

Another method is based on fuzzy logic operators. Fuzzy logic theory maps the **and** and **or** logical operators to **minimum** and **maximum** operations. For example the query showing in Figure 2.1.5 which selects images similar to an example image *ex*, will calculate the minimum of the color and texture features and the maximum of this with the combined color and texture features.

```
select
  from   images i
 where   i.color = ex.color
   and   i.texture = ex.texture
    or   i.color_texture = ex.color_texture
```

Figure 2.2: Example Query

The introduction of this new query model with similarity measures and with fuzzy logic operators requires new index structures. Such structures are more generally applicable when they are independent of the logic operator or similarity measure used.

Requirements Summary

The main requirement coming from image queries is a new query model. The current binary logic model is not suited for image queries. One suggested query model is the fuzzy logic model. Currently, similarity measures are used mainly.

2.2 DBMS

In this section we introduce our extensible main memory database management system, which is the appropriate system for image based applications.

2.2.1 Extensible

Important requirements we found for an image database management system are

1. There is a strong need for Image and feature data types, and their operations.

2. There is a need for Index structures for efficient queries on images and features.

Therefore, we need an extensible DBMS.

An extensible database management system can be extended with new abstract data types, new commands and new index structures. This makes it possible to add an image data type to the system. An image would be treated the same way as ordinary types, like integers and strings. So the basic algebraic functionality, like the set operations: union, intersection, minus, and symmetrical difference and the relational operators: select, join and anti-join, would work without extra coding.

In addition new image processing commands can be added to such a system. Also new feature data types can be added. The new data structures can be large and expensive to query, therefore new index structures may be added to speed up the retrieval of these structures.

2.2.2 Main Memory

A design issue so far ignored is performance. Image applications are CPU demanding and often time critical. Example applications, like surveillance, authentication and error detection all demand high performance. But also interactive access of an IDBMS calls for a performance wise approach to avoid loosing interest of end users.

Therefore, a database management system for image applications needs to deliver high performance. At the hardware level this can be achieved with better CPUs, memory and, disks. Such a system is no longer io-bound but CPU-bound, giving the system the appreciated performance. The ideal system for this is a main memory DBMS.

To further improve the performance of a database system shared memory multi-processor systems are needed. On these systems parallelism could be exploited to improve the throughput.

2.2.3 Objects versus Sets

As image processing software moves more and more towards object oriented program languages [68, 112, 105] it seems wise to choose an OODBMS for image database applications. The same programming language can then be used for both application and database specific code. The problem with this seemingly ideal case is the mismatch between the imperative programming paradigm and the declarative paradigm of the database. The advantage of only declaring what is needed will be lost in such a situation, because the

imperative programming paradigm of the object oriented requires you to specify how to obtain it.

A possible solution is to use a proper object oriented query language, like OQL, to interact with the database. Although such a combination of an object oriented language, such as C++, and an object oriented query language solves the mismatch, it requires proper programming practice from the application programmers to make full use of the database functionality. Its too easy to fall back to object at a time processing.

A better approach is to identify a minimal, but complete set of primitives for image applications, including both image operations and image query primitives. These combined with an extendable SQL like query language, such as SQL-'99, would leave space for optimization and put no extra burden on the programmers programming skill. It clearly separates database use from application programming tasks.

2.3 Architecture of Monet

Monet is a novel database kernel under development at the CWI and UvA since 1994. Its development is based on experience gained in building PRISMA, a full-fledged parallel main-memory RDBMS running on a 100-node multi-processor, and on market trends in database server technology.

Developments in personal workstation hardware are at a high and continuing pace. Main memories of >>1 GB are now affordable and mass-market CPUs currently can perform over 1000 MIPS. They rely more and more on efficient use of registers and cache, to tackle the ever-increasing disparity[1] between processor power and main memory bus speed. These hardware trends pose new rules to computer software – and to database systems – as to what algorithms are efficient.Another trend has been the evolution of operating system functionality towards micro-kernels, i.e. those that make part of the Operating System functionality accessible to customized applications. Prominent research prototypes are Mach, Chorus and Amoeba, but also commercial systems like Silicon Graphics' Irix and Sun's Solaris increasingly provide hooks for better memory and process management.

Given this background, we applied the following ideas in the design of Monet:

- *binary relation model.* Monet vertically partitions all multi-attribute relationships into Binary Association Tables (BATs), consisting of `[OID,attribute]` pairs.

  This Decomposed Storage Model (DSM) [27] facilitates table evolution, since the attributes of a relation are not stored in one fixed-width relation. Figure 1 shows this model in detail.

[1] In recent years this disparity has been growing with 40% each year

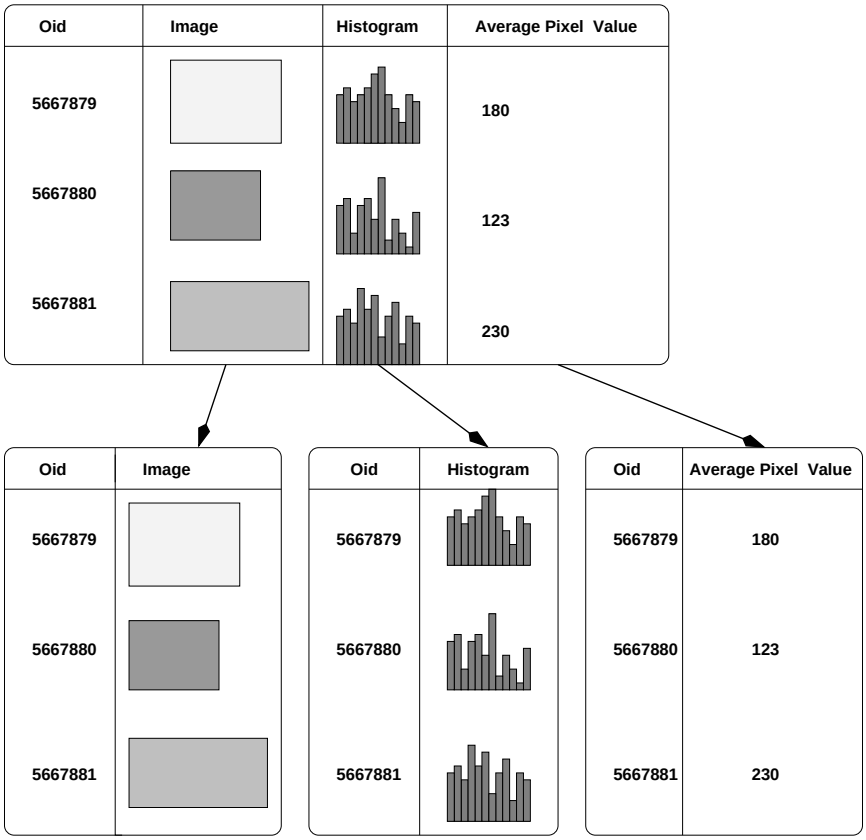


Figure 2.3: Decomposed Storage Model

The price paid for the DSM is small: the slightly bigger storage requirements are compensated by Monet's flexible memory management using heaps. The extra cost for re-assembling multi-attribute tuples before they are returned to an application, is negligible in a main-memory setting, and is clearly outweighed by saving on I/O for queries that do not use all the relations attributes.

Finally, maintaining all attributes in different tables enables Monet to cluster each attribute differently, and to precisely advice the operating system on resource management issues, for each attribute according to its access path characteristics.

- *perform all operations in main memory.* Monet makes aggressive use of main memory by assuming that the database hot-set fits into main memory. All its primitive database operations work on this assumption, no hybrid algorithms are used. For large databases, Monet relies on virtual memory by mapping large files into it. In this way, Monet

avoids introducing code to 'improve' or 'replace' the operating system facilities for memory/buffer management. Instead, it gives advice to the lower level OS-primitives on the intended behavior[2] and lets the MMU do the job in hardware.

Unlike other recent systems that use virtual memory, Monet stores its tables in the same form on disk as in memory (no pointer swizzling), making the memory-mapping technique completely transparent to its main-memory algorithms.

- *extensible algebra.* As has been shown in the Gral system [51], many-sorted algebras have many advantages in database extensibility. Their open nature allows for easy addition of new atomic types, functions on (sets of) those types. Also, an SQL query calculus-to-algebra transformation provides a systematic framework where query optimization and parallelization of even user-extended primitives becomes manageable. Monet's Interface Language (MIL) interpreted language with a C-like syntax, where sets are manipulated using a *BAT-algebra.*

- coarse grained *shared-memory parallelism.* Parallelism is incorporated using parallel blocks and parallel cursors (called "iterators") in the MIL. Unlike mainstream parallel database servers, e.q. PRISMA [2] and Volcano [50], Monet does not use tuple- or segment-pipelining. Instead, the algebraic operators are the units for parallel execution, which simplifies query optimization. Their result is completely materialized before being used in the next phase of the query plan. This approach benefits throughput at a slight expense of response time and memory resources.

2.3.1 Monet Architecture

The architecture of Monet is structured as a frontend/backend system. The current implementation has frontends for the Monet interface language, the object database management groups[17] object definition language (ODL) and for the structured query language (SQL) (see Figure 2.3.1)

The Monet database system is designed to be a extensible in all directions. Meaning, new data types, commands and accelerators can be added. The MIL has a sister language called the Monet Extension Language (MEL), which should be used to specify extension modules. These modules can contain specifications for new atomic types, new instance- or set-primitives and new search accelerators. Implementations have to be supplied in C/C++ compliant object code.

[2]This functionality is achieved with the `mmap()`, `madvise()`,and `mlock()` Unix system calls.

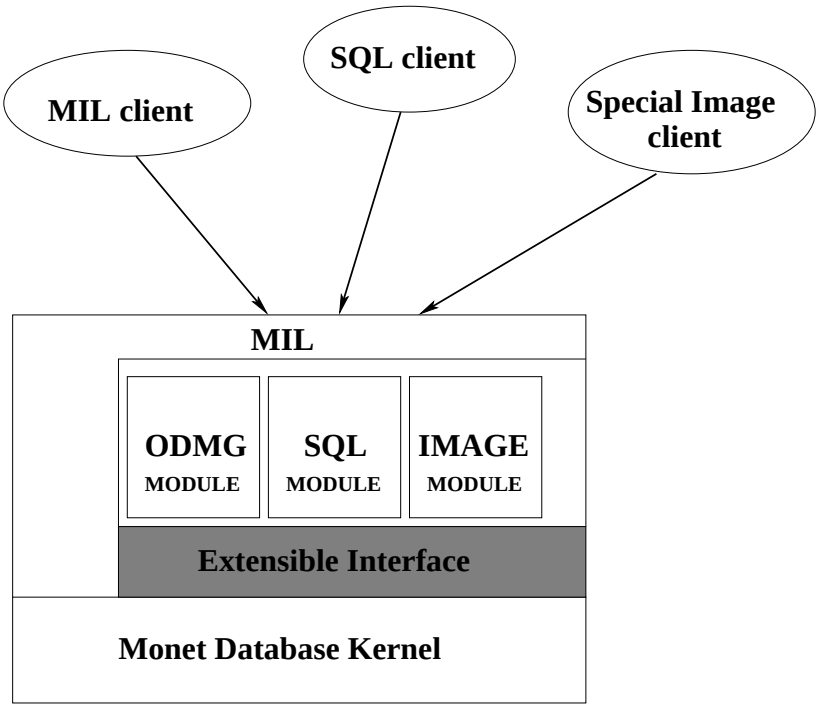


Figure 2.4: Monet Architecture

2.3.2 Monet Interface Language

The Monet Interface Language is a low level BAT-manipulation database query language, extensively described in [9]. For self containment of this thesis, we introduce some part of Monet's instructions set and programming concepts.

Table 2.2 lists the relational operations with there functionality. Monet uses binary tables (BAT) all relational operations are defined on those. The first column of a BAT is called its head column and the second its tail column.

The **select** operation selects all binary units (BUNs) where the tail value of the input BAT is between the lower and upper bounds given. The **semi-join** operation selects all BUNs of the α BAT whose head value also occurs in the β BAT. The **join** operation implements a natural equi-join based on the tail of the α BAT and the head of the β BAT.

The difference operation (**diff**) selects all BUNs which occur in α but not in β BAT. The **union** and **intersect** are the well known set operations. The **unique** operation selects all unique BUNs from a BAT. All these functions work over both the value in the head and tail. i.e. an intersection between two BATs is the intersection between the pairs of head and tail values. They

all have an equivalent version based only on the head value, these functions have the same name prefixed with a k, i.e. **kdiff**, **kunion** and **kintersect**.

RELATIONAL **Operation**	**functionality**
select(BAT α, T low, T high)	$\{ab : ab \in \alpha \wedge low \leq b \leq high\}$
semijoin(BAT α, BAT β)	$\{ab : ab \in \alpha \wedge cd \in \beta \wedge a = c\}$
join(BAT α, BAT β)	$\{ad : ab \in \alpha \wedge cd \in \beta \wedge b = c\}$
SET **Operation**	**functionality**
diff(BAT α, BAT β)	α / β
union(BAT α, BAT β)	$\alpha \cup \beta$
intersect (BAT *alpha*, BAT β)	$\alpha \cap \beta$
unique(BAT α)	$\{ab : \exists ab \in \alpha\}$

Table 2.2: Monet's Relational and Set Operations

Table 2.3 lists the construction and update operations with there functionality. Bats are created using the **new(ht,tt,capacity)** constructor, where ht and tt represent the head and tail column type, respectively. The capacity is an optional parameter to identify the initial table size. A proper guess of the tables size reduces memory fragmentation.

Bats can be updated using the **insert(BAT, h,t)**, **replace(BAT, h,t)**, and **delete(BAT, h, t)** primitives. Insert appends a single pair of atom values to the end of the table. Replace updates all tail values for binary records (BUN) with the given head value. Delete removes all matching pairs. Monet assures BUN's are in consecutive memory, i.e. no holes are allowed between BUN's. Therefore, the BAT scan operations can be kept simple and fully optimized.

Performing a single operation on all tail values of a table can be done using the [] multiplex operator. For example adding the tails of two tables can be done using the statement [+](a,b), this will find pairs with equal heads and add the tail values together. Aggregating groups can be done using the **{}** operator. This operator performs an operation for each group in the given BAT. Groups are identified by unique values in the head column of the given table. For example to sum all groups we can use {sum}(a), where sum is a BAT operation to add all tail values in the BAT. The sum operation gets one parameter, a BAT. In the same way an average, min, and max operators over groups can be defined. Such operator gets the group BAT so it can do any initialization steps it self, i.e. for the sum start at zero, and also any post processing, i.e. for the average dividing the sum by the group count.

MIL is a procedural language. It has well-known flow of control constructs such as, **while** and **if/then/else** statements. Also BAT related iterators such as **batloop**, and **hashloop(a)**, iterates over all BUN's executing a MIL block. New MIL procedures can be introduced using the **proc**

```
1    proc sum (BAT[any,int] b) : int := {
2        var res := 0;
3        b@batloop(){
4            res += $t;
5        }
6        return res;
7    }
```

Figure 2.5: Example mil procedure

keyword. See Figure 2.5 for a simple MIL procedure which sums a BAT. The first line defines the procedure. Sum gets a BAT of integers as parameter, and returns a single integer. In line 2 the **res** variable is initialized. Line 3 to 5 loop over the BAT and add the BUN's tail value ($t) to the **res** variable. Finally in line 6 the sum is returned to the caller.

Operation	**functionality**
new(ht, tt, capacity)	BAT[ht,tt]
insert(BAT α, T h, T t)	$\alpha \cup ht$
delete(BAT α, T h, T t)	$\{ab : ab \in \alpha \wedge a \neq h \wedge b \neq t\}$
replace(BAT α, T h, T t)	$\{at : (at \in \alpha \wedge a \neq h) \vee (ab \in \alpha \wedge a = h)\}$
sum(BAT α)	$a_0 + \ldots + a_n \wedge a_0, \ldots, a_n \in \alpha$
avg(BAT α)	$(a_0 + \ldots + a_n)/n \wedge a_0, \ldots, a_n \in \alpha$
max(BAT α)	$max(a_0, \ldots, a_n) \wedge a_0, \ldots, a_n \in \alpha$
min(BAT α)	$min(a_0, \ldots, a_n) \wedge a_0, \ldots, a_n \in \alpha$
[op](BAT[h,t1] α, BAT[h,t2] β)	$\{at : ab \in \alpha \wedge ac \in \beta \wedge t = op(b, c))\}$
{op}(BAT[h,tt] α)	$\{ab : \beta = \{c : ac \in \alpha\} \wedge b = op(\beta)$ $\}$

Table 2.3: Monet's BAT Update Operations

2.3.3 Monet Extension Language

Monet is an **extensible database system**, and hence MIL is an extensible language. Experts in some application domain can extend the Monet database system to store new kinds of data and define operations on them.

Extending Monet is achieved by writing an *extension module* in the Monet Extension Language (MEL). The MEL is a **specification-only language**, see Figures 2.6, 2.7, 2.8 and 2.9 for an example MEL specifications.

A MEL module describes the types (atoms), index structures (accelera-

tors), and commands to include in the Monet kernel. The module definition also contains help texts on the new primitives, which will be inserted in Monet's online help system. Finally, the module contains references to C functions that implement the primitives.

Mel modules can contain any of the following Monet extensions:

new atomic data types Mil has builtin support for { `boolean, character, integer, oid, pointer, float, double, long, and string` } values. You can add new types, like `date` or `vector` readily (see Figure 2.6).

new algebraic commands or operators Algebraic commands get passed a set of values as parameters, then do some execution, and return one value. Mil allows for overloading of algebraic commands and operators. The extension programmer should specify a type-signature and MIL chooses dynamically which command to use on the basis of the actual parameters (see Figure 2.8).

new search accelerators Some operations on tables need additional – persistent – data structures for efficient execution. Famous examples are R-trees and hash-tables. The accelerator builder should provide the interface operations to create, destroy, and traverse such tables. The database needs to keep these tables consistent under updates; for this reason additional update interface operations, **insert** and **delete**, are required from the extension programmer (see Figure 2.9).

MEL Modules

Modules are the unit of extension in Monet: a module is loaded, or not. When it is loaded, all its new language elements are added to the Monet's interpreter language. When a module is *dropped*, they are removed.

The `.prelude` and `.epilogue` constructs allow for C routines to be called when a module is loaded and dropped **physically**. An example for its use is the initialization of (empty) c structures, or for instance the creation (/destruction) of shared locks. The C routines are parameterless and do not return any value.

The `.load` and `.drop` keywords allow for the specification of MIL code that is to be executed when a module is loaded. These features come especially handy for defining standard MIL procedures (procs) and constants. As opposed to the prelude/epilogue initializations, the load/drop scripts are executed at logical load and drop points: each time a user loads or drops a modules (they are executed in the context of that user). The prelude and epilogue are only execute once at physical module load and unload.

```
.module vectors;
  .atom vector2D[8,4];
    .tostr   = vector2D_tostr;
    .fromstr = vector2D_fromstr;
    .COMP    = vector2D_comp;
    .HASH    = vector2D_hash;
    .null    = vector2D_null;
  .end;

  .atom vector3D[3,1];
    .tostr   = vector3D_tostr;
    .fromstr = vector3D_fromstr;
    .COMP    = vector3D_comp;
    .HASH    = vector3D_hash;
    .null    = vector3D_null;
  .end;
.end vectors;
```

Figure 2.6: Example of the mel ATOM definition

atomic types

An important design issue of Monet's atomic types is that there are basically two classes:

fixed-size atoms Their memory management is simple, because all possible instances have the same size. They are stored directly in the BUN heap (tuple heap) of a BAT. These types are very efficiently implemented in Monet.

variable-size atoms From the builtin-types, `string` is the only variable sized type. Values are stored in a separate heap, the BUN heap contains integer byte-offsets into this heap. Monet ensures in this way that the BUN heap can be implemented as an array of fixed-size elements, even if it contains values of variable-size.

The correct functioning of all Monet's standard operations (like `join` and `select`) is guaranteed by suppling the atom interface for any new ADT, see Figure 2.4 and 2.5.

For instance, a hash-function on a `string` is required to make Monet's hash-join work on columns of `strings`. The heap provides a context for domain dependent optimization. For example, the implementation of the string atom uses a hash-based catalogue in this heap. The catalogue is used to have only a single copy of a string in the heap. MIL will recognize each new atom as a keyword.

Function	description
FromString	constructs the atom from a string
ToString	converts the atom to a string
hash	calculates a hash number for the atom
nequal	tests it two atoms are not equal
comp	compares two atoms, returns smaller, equal or larger
null	get the nil value for this atom type

Table 2.4: Fixed Atom Interface

Function	description
heap	creates and initializes the heap
put	insert an atom in the heap
get	get a copy of an atom from the heap
del	delete an atom from the heap
len	get the size of the atom

Table 2.5: Additional functions for Variable Sized Atom Interface

Redefined Atoms A feature borrowed from Object Orientation is atom overloading. One atom can be implemented using the existing interface functions implemented by a parent atom. For instance an **rgb** atom is implemented as a different interpretation of a vector3D atom, See Figure 2.7. The new type is different from its implementation type on the logical level in MIL, but at the physical level it uses the implementations of its parent type (or actually its root ancestor).

```
.module rgb;
  .atom rgb = vector3D;
  .end;

  .operator (rgb) "+" (rgb) : rgb = rgb_add; "rgb addition"
  .operator (rgb) "-" (rgb) : rgb = rgb_min; "rgb substraction"
  .operator (rgb) "*" (rgb) : rgb = rgb_mul; "rgb multiplication"
.end rgb;
```

Figure 2.7: example of the overloading of Monet ATOMs

2.3.4 new primitives

Apart from atomic types, the MEL extension mechanism also allows for introduction of new execution primitives into MIL.

Commands and operators are much alike. The exact way in which a command implementation in C will get passed its parameters and what it is expected to return will not be discussed here.

The functional part of the MIL language is a set of algebraic commands and binary and unary operators. Commands get passed a set of values as parameters, then do some execution, and return a (single) value. A collection of such commands and operators, where foreach $f(D) \subset D$ holds, is called an algebra. [3]

Figure 2.8 shows an example image module containing some new mil commands and operator.

```
.module image;
  .atom image = BAT;
    .tostr   = imagetostr;
    .fromstr = imagefromstr;
  .end;

  .command width( image )  : int = imagewidth;
   "Get the width of this image"

  .command height( image )  : int = imageheight;
   "Get the height of this image"

  .operator (image) "=" (image) : bit = imageeq;
   "Test if the images are equal"

  .operator (image) "!=" (image): bit = imagene; ""
   "Test if the images are not equal"

  .command readFile( str filename ) : image = readFile;
        "read an image from the file, the extension
         expresses the format in which the image is stored"

  .command writeFile( str filename ) : image = writeFile;
        "write an image to the file, the extension expresses
         the format in which the image should be stored"

.end image;
```

Figure 2.8: The image module

[3] It is actually not correct to call this a true algebra, since BAT-parameters are call-by reference (as opposed to the simple values which are call-by-value), and can hence be modified. This is just a pragmatical choice.

MEL also allows for overloading of the multiplex, [], and group aggregate operations, {}. At run time when MIL resolves the formal call [+] to a physical function it will take the special optimized case for this operator. The default implementation of the [+] operator uses a batloop and calls a + function for each BUN.

2.3.5 New Search Accelerators

A *search accelerator* is defined as 'a data structure associated with a database column kept up to date with changes'. The prime reason for maintaining such data structures is to achieve better speed on common database operations.

Well-known search accelerators for traditional data types are the B-tree and hash-tables. Examples from the GIS application domain are Grid-files and R-trees[52, 4, 83].

For this reason the Monet's extension mechanism allows for addition of new (persistent) search accelerators. A BAT can hold two user-defined search accelerators: one for the head and one for the tail. Monet provides two standard search accelerators oriented towards main-memory relational processing: the `index` binary tree index and the `hash` chained bucket hash-tables.

The metric accelerator which will be introduce in Chapter 8 is shown in Figure 2.9. This example accelerator requires besides the construction (BUILD) and destruction (DESTROY) only insert and delete instructions to keep the accelerator inline with the underlying BAT. It includes the definitions of the vector module using the USE statement.

Monet will recognize each new accelerator as a MIL keyword.

2.4 State of the Art of Image Database Systems

In this section we give a short overview of image database management systems, image retrieval systems and Image indexing techniques described in literature. Each image database management and image retrieval system is first described individually. These sections conclude with a evaluation based on the requirement list. This means we evaluate each IDBMS on extensibility with new data types, commands and index structures. For image retrieval systems we evaluate based on the existance of image operations, the available image file formats, and on the available global and local features.

2.4.1 Commercial Image Databases

Despite the awareness in the database research community that general database technology would be a clear asset to multi-media application domains, limited progress has been made so far. This partly stems from a lack of

```
.MODULE metric;
  .USE vectors;

  .ACCELERATOR metric_acc(Vector);
    .BUILD     = metric_acc_build;
    .DESTROY   = metric_acc_destroy;
    .INSERT    = metric_acc_insert;
    .DELETE    = metric_acc_delete;
  .END;

  .COMMAND metric_acc_select( BAT[oid,Vector], Vector,
     flt max_dist ) : BAT[oid,Vector] = m_select;
        "A distance select (select all within max_dist)
         using the metric_acc"

.END metric;
```

Figure 2.9: Example of a MEL accelerator extension module

application domain knowledge within the database community to isolate the functionality needed, as well as the lack of experience in using database technology in the image research community to focus the effort on query formulation and evaluation instead of dedicated storage and index management.

Moreover, it has only recently become manageable to enhance the database kernels with application domain functionality. For example, research prototype database systems, such as postgress[103], Jasmine [26] and Starburst[72] showed the route towards low-level extensibility of a database kernel. This route is only recently followed by Oracle, DB2 and INFORMIX. It is expected that these facilities will become available in all commercial systems within a few years.

Oracle Data Cartridges[84]

The Oracle 8 universal server supports a form of abstract data type (ADT) extensibility. Oracle calls extension modules "data cartridges". A data cartridge defines new "Object data types" (ODT) with their behavior. The description specifies both the ODT attributes in terms of existing Oracle data types, member functions and procedures on these data types.

The procedures can be written both in PL/SQL, Oracle's extended structured query language, and in C using external shared libraries. The PL/SQL is not well suited for object member function implementations, because this high level interpreted language caries to much weight to achieve the required

high performance. The shared libraries runs in a separate process, which ensures Oracle's server stability under bogus member function implementations. The downside of this separate process model is performance. A call to a separate process is orders of magnitude less efficient than an in process function call. Furthermore, the external libraries can only access the server via Oracles call interface (OCI), which again reduces performance. So both implementation paths are perfomance wise not very promising.

INFORMIX Data Blades[57]

The INFORMIX-Universal Server provides a very advanced extensibility interface, called Data Blades. A Data Blade module may include the following components; new data types, functions, access methods, tables and indexes, and client code.

User defined data types are treated as built in types. The database allows various new data type definitions, the opaque type definition offers maximum flexibility. It allows any data represented in C structures to be natively stored and processed by the server.

The *function component* is a collection of function definitions which operate on any data types, new or built in. These functions extend the processing and aggregation functionality of the database.

The *access method component* enables Data Blade developers to write special index structures . An index structure for the INFORMIX server is defined by a set of methods, open a scan, get next record, insert, delete, replace and close scan.

The *interface component* can be used to export functionality of a Data Blade. So, for instance an image retrieval Data Blade can use a text retrieval Data Blade for keyword search.

A Data Blade developer can store and index data needed for the Data Blade in tables and index structures directly. The client code component contains the code which exports a client user interface for the new data types.

INFORMIX supports an extensibility mechanism which is powerful enough to support an image data type and operations on it. There is no support for polymorphic data types.

DB/2 Universal Database[56]

The DB/2 Universal Database from IBM also supports extensibility, called extenders. These extenders can extent the DB/2 server with user defined types and user defined functions. These user defined types can only be stored in large objects. This way common relational operators on these newly created types are lost. The defined functions can be used from the DB/2 SQL interface.

The supported extensibility of DB/2 is limited. New data types are treated different than built-in types and it is not possible to extend the server with special index structures.

Jasmine OO Database server

Jasmine is a fully object oriented database management system. It supports addition of new classes, inheritance and method overloading. The index is hidden, i.e. no support for advanged index structures is available.

This database has been extended with multi-media classes. The main feature is data independence. Images can be stored in the database, so the physical location is no longer needed.

Image Database Comparison

The differences between the commercial systems and Monet's extensibility are summarized in table 2.6. A + indicates the extensibility is available (- means not available). A ++ means available and has a superior performance.

Database	ADT	Commands	accelerators
Oracle 8	+	+	-
Informix	++	++	++
DB/2	+	+	-
Jasmine	++	++	-
Monet	++	++	++

Table 2.6: Image Database Comparison

As can be seen from this comparing table the Informix and Monet system clearly support the extensibility needed. The Monet system was chosen because of its superior performance which comes from its main memory oriented implementation.

2.4.2 Commercial Image Retrieval Systems

Virage: Visual Information Retrieval Module

The Virage Visual Information Retrieval (VIR) Module extends commercial database management systems, such as Oracle 8, INFORMIX Universal Server, Sybase and Object Store from Object Design, with image storage and management capabilities. In addition developers can use the VIR Image Engine to interact with their own DBMS. The VIR Image Engine capabilities include:

storage Reading and writing multiple image file formats.

thumbnail Automatic thumbnail creation.

content Analysis and comparison of images based on their visual content.

The storage capability provides users shared access to images centrally stored in a DBMS. The VIR Image Engine supports translations between popular image file formats during storage and retrieval, the following image file formats are supported: JPEG, BMP, SGI, PSD, Sintex CT, TIFF, PICT, TGA, MAC, RLE, EPS, PNG and PCX. The list of file formats again illustartes the lack of a standardized image data type.

The VIR Image Engine provides a simple interface for image thumbnail creation. A reference to the original full size image is maintained. The thumbnail creation is offered for performance reasons. The lack of a complete set of image operations makes this special feature necesary.

The visual comparison capability allows users to search for images based on their content. Virage uses four features, i.e. color, color composition, structure and texture to describe an image. These quantitative measures provide easy access based on a similarity metric.

The VIR Image Engine provides no query language enhancements, such as fuzzy or probabilistic reasoning. All queries should be defined as boolean predicates on the features extracted from the images. Querying for similar features involves full scans of the feature tables, because no special index structures is added.

Excalibur Visual Retrieval Ware

Excalibur has build retrieval software, which runs on Jasmine, INFORMIX and Oracle. It uses of the image contents shape, color and texture to index the database. Retrieval is supported by query by example or by sketch.

The Excalibur Visual Retrieval Ware is based on Excalibur's Adaptive Pattern Recognition Processing (APRP) technoligy. APRP acts as a self-organizing system that automatically indexes binary patterns in the digital information, creating a pattern-based memory that is self-optimized for the native content of the data. The bases for this indexing are the shape, color and texture features extracted from the images.

Oracle Visual Image Retrieval Data Cartridge

Oracle also has its own image cartridge, the Oracle8 Visual Image Retrieval Cartridge. This cartridge supports image storage in various image formats. No support for image operations and image query extensions is provided.

2.4.3 Research Image Retrieval Systems

Several image database retrieval projects are underway, see survey [91]. A few snapshot descriptions are illustrative for the approaches taken.

Keyword based image retrieval is supported by the web search engines Yahoo and Alta Vista. They support search for images based on categories and keyword matching. Yahoo manually annotates the images. Alta Vista uses an annotated stock photo archive. No support for image retrieval on image content can be found here.

The QBIC project [41], which later became a commercial product, studies methods to query image databases based on the image content, it is based on IBM's DB/2 Image extenders. The content features include color distribution, texture, and position and shape of edges. The color feature is described by the average RGB and Munsell[74] color coordinates and by a 64 bins color histogram. The texture is summarized by a triplet, i.e. coarseness, contrast and directionality. Shape is described as a combination of area, circularity, eccentricity, major axis orientation and a set of algebraic moments. The similarity measure used are limited too quadratic form distance functions, like the Euclidean distance.

To improve efficiency, the search space is reduced using a lower bound metric on the color histogram Euclidean distance. The average color turns out to be a lower bound for this distance [93]. Therefore, using the average color does not result in missing actual hits, though extra false hits will be introduced.

The VisualSEEk image retrieval system, as described in [101], automatically segments the image into objects with equal color-set content. A color set represents the colors in a segment. The spatial information about these objects is stored. Using both the spatial and color properties the user can query this database. A large database of 12,000 images is used in their web demo.

It has an interesting graphical interface, called SaFe where spatial relations between features can be modeled by sketch. The features used for this search type are color and spatial relations between similar regions.

Recently two other research projects appeared with a system using spatial relations, ExSight[120] and Blobworld[16]. Both systems start by automatically segmenting the images in the database. The user can then specify the queries by selecting segments from sample images and spatially arange them to from a spatial image query. The system searches for all similar images based on these spatial arangements using the segments features. Query results show why an image is returned by displaying the segments used and optionally the features can also be visualized.

The Photobook [87] provides a large amount of image processing functionality useful for content-based image retrieval. An example is the semantics preserving image compression technique, which reduces images to a small set of perceptually-significant coefficients. Using a training set of images, the "eigenimage" vectors are computed. These vectors are used to compress the image content information. The similarity between two images is computed using the distance in this compressed "eigenimage" space. This

has been successful in face recognition.

The approach taken by the PictoSeek [46] is to build histograms of the hue, the dominant hue edges and hue corners. The hue color component is chosen since it is invariant to surface specularities, like shadow and highlights. The similarity measure is color histogram intersection [106], which is less variant to occlusion and less dependent on the view point. Histograms are invariant under a number of transformations. A web demo is available with various databases, the largest contains 10000 images.

To improve retrieval performance various experiments are done with signatures. A signature indates the presense or absense of a color in the image. Using binary operators such as and, or, and x-or quickly a set of images with similar colors can be retrieved.

Image Retrieval System Comparison

The difference between the various (non-)commercial image retrieval systems and Monet's image retrieval system are given in tables, 2.7 and 2.8. Table 2.7 expresses how the image retrieval systems score on the availability of image operations and image input/output routines to various standard image file formats. Scoring is again done with ++ (very good), + (available) and - (not available). The non commercial image retrieval systems have limited documentation about their image ADT.

Retrieval	Operations	Input/Output
Virage	+	++
Excalibur	-	+
Jasmine	++	+
Oracle	-	++
Monet	++	+

Table 2.7: Image Retrieval Systems Comparison of the image ADT

The second table shows the level on which the retrieval takes place, globaly or localy (segments or pixels).

As can be seen from this comparing these tables none of the commercial retrieval systems include all required functionality. The VisualSEEk, Blobworld and ExSight score in the same range as Monet. We will explain more about the Monet's image retrieval system in the chapters 4.2 and 4.

2.4.4 Image indexing techniques

Data structures for image feature indexing have received quite some research attention. The baseline is to replace the search key of an ordinary index structure by a feature vector and to include a proper comparison operator.

Retrieval	global features	local features
Virage	++	-
Excalibur	++	-
Jasmine	-	-
Oracle	-	-
QBIC	++	-
VisualSEEk	++	+
PhotoBook	++	-
PictoSeek	+	-
Blobworld	+	++
ExSight	++	+
Monet	++	++

Table 2.8: Image Retrieval Systems Comparison of feature levels

For example, quad-trees can be easily extended to encode multilevel color histograms, by Lu et.al. [53]. This enables fast similarity searches based on those color histograms.

Signature files, originally developed for textual information retrieval, have been extended by Faloutsos [3]. The trick is to use the important image features as signatures for the images. Fast retrieval can be achieved using bit comparison on the signature files.

Chang et.al.[98] proposed a "2D-string representation" to encode the objects and their spatial relationships. Similarity retrieval of images encoded in 2D strings is mapped to substring matching.

Nabil et.al. [75] use a graph-based encoding of the objects and their spatial relationships. Subsequently, retrieval is turned into a weighted graph-matching problem.

The Fourier transform of a signal yields a frequency decomposition which is rather unsuited to describe local transitions. The wavelet transform [31, 104, 114] is designed to describe signals at different scales. The wavelet coefficients yield a multiresolution decomposition of a signal.

Jacobs et al[59] apply a fast Haar wavelet transform to each color band of the images. The feature vector is composed of the N maximal coefficients of the wavelet transform, only the sign and indices are use not the values. Also the average pixel values of each color channel are used. Using this feature vector they claim to be able to find an image based on an in accurate (low-resolution) version of the image.

Wang et.al.[115, 116] uses the Daubechies 4-layer 2-D fast wavelet transform. They use a hierachical query method. First based on the standard deviation in the 8*8 low frequency bands of the wavelet transform is used to fast reduce the result set. This set is furture reduced using the weighted

distance between the 8*8 low frequency bands. The last step uses the 16*16 low frequency bands to find the best matching images.

In [99] a method for segmentation based on the quad-tree index structure is introduced for texture based image queries. Each image is recusively split in four parts util the distances in the texture-feature space between the parts and its enclosing part exceeds a certain threshold. Image parts are merged when their texture feature distance is less than this threshold. For each resulting segment the texture is calculated. So each image is represented by a set of segment, texture pairs. A user can query this by suppliing an example texture. The texture features are based on the Quadrature Mirror Filter wavelet representation.

In [120, 101, 16] methods based on segmented images are described including the segmentation algorithm. The Blobworld segmentation is done based on color features using the HSV color model and texture features. Safe only uses color to segment and query the images. The segments in Blobworld are described using its centroid and scatter matrix expressing the variance, excentricity and orientation. The queries in Blobworld use fuzzy operators to combine feature values. Use of the **and** operator in a query will take the **minimum** of the two feature values incase of the **or** operator the **maximum** is used.

2.4.5 Requirements

From the existing systems we can deduce a list of requirements. Aside from the basic requirements, i.e. storing and retrieving of images and derived features, these systems need to compare features using similarity measures, many of which exist. Unfortunately none perform perfectly in all cases. So we need to support a large set of these feature comparing measures. Because we use an extensible system later found measures can be easily added to the system.

Most current systems do not allow for partial image queries, i.e. give me all images which contain the following parts. Only the SaFe, ExSight and Blobworld systems, lets users specify a query using combinations of spatial relations and color and texture features.

Chapter 3

Database Assisted Image Processing

The activities in image and database research fields seem opposites of a spectrum. Image processing usually involves *object (image) at a time processing* and database systems use *set at a time processing*. However, taking a closer look, they are more related then one might expect. In this chapter we will demonstrate a database approach for image processing, which will open new ways to optimize image processing algorithms against large image collections.

3.1 Data Structures

Storage consideration has long been driving the design of image processing packages. Packages, such as SCILIMAGE, Horus[112], IUE, Khoros, Matlab and PhotoShop, store images in two dimensional arrays of pixel values. This simplified data representation requires no storage for the spatial component of a pixel; its location is implicit. The implicit spatial component is used throughout the algorithms.

A usual further reduction is obtained by using limited pixel value types. For example, 255 gray levels present in an image can be stored in a single byte pixel value. Although the storage requirement drops, it also creates an overflow problem. Performing a pixel value operation can result in an overflow, i.e. the value does not fit in the byte representation.

The storage considerations for modern image processing packages are less import. With memory prices dropping quickly, current workstations easily hold 256M of memory, which is more than adequate for image processing geared towards a limited set of fully exploded images (1-7MB a piece).

Although the two dimensional array approach has its advantages, it also has some difficienties.

- Optimization decisions are visible to the user
- Can not handle arbitrary shaped images
- Low level application programmers interface (API) only, i.e. only pixel level operations are used.
- Cannot handle other regular or ir-regular grids, for the spatial component.

An important database concept is to have a generally defined type at the logical level and to hide the storage optimizations at the physical level. Such a general definition for the logical level is defined in the Image Algebra, which defines an image as a mapping from a spatial domain X into a range value domain F.

In the Monet DBMS we would map an image to a BAT[X,F]. For such mapping we need atomic types to represent the pixel positions and values. The atoms currently provided in the Monet image database system are shown in Table 3.1 together with the MEL packages supporting these types. These packages can be extended with operators needed for new image operations.

PIXEL **Type** **Mel Package**	**Implementation Type representation**
Single value types	
monochrome	bit
grayscale	byte
real valued	float
2D Vector types	
location	int
gradient	float
3D Vector types	
color	rgb
color	HSI

Table 3.1: Pixel Types

This logical image definition solves the earlier mentioned problems related to the two dimensional array approach. It supports arbitrary shaped images, and can handle different spatial representations. The costs for this flexibility is high, because we potentially loose the storage savings of the implicit spatial component. Later in this Chapter we will come back to this storage overhead.

3.2 Primitives

The data models for images and database relations are closely related, but can we also show that their primitives are closely related ? In this investigation we follow the operation classes as defined by the image algebra (Section 2.1.1).

Image restriction on its range values can be mapped directly on the well known select operations in database systems. For example, a restriction to all pixels with values above a constant k maps to a selection of all records with attribute range value above k. Restrictions on its spatial domain map to a range selection (when the spatial ranges are known). When the restricting spatial set is know a natural-semi-join operation can be used. Let a,b be a $bat(X, F)$ and $k \in F$ then formally the mapping is as follows:

$$a\|_{>k} \Longleftrightarrow a.select(k, nil)$$

$$a|_b \Longleftrightarrow a.semijoin(b)$$

The image extension primitive maps on a combination of the set union and difference operations. An extension of **a** with **b** is formally mapped as follows:

$$a|^b \Longleftrightarrow a.\texttt{union}(b.kdiff(a))$$

The range(a) and domain(a) operations map to the project operation in Monet, projecting the column of interest. They are mapped as follows:

$$range(a) \Longleftrightarrow ['nil' \sim a]$$

$$domain(a) \Longleftrightarrow [a \sim' nil']$$

Induced image operations in Horus[112] map onto combinations of natural joins and scan operations in Monet. A binary image operation can be mapped using a natural join between the two spatial attributes and a scan over the resulting table, performing the binary pixel operation on the range value attribute. In Monet the induced image operation λ between two image **a** and **b** can be concisely expressed as follows:

$$a\lambda b \Longleftrightarrow [\lambda](a, b)$$

Monet already has some global reduction operations, namely : min, max, sum, count and histogram. They perform the obvious reduction operations on BATs. Since Monet has no general BAT aggregation operation, each global reduction operation requires an implementation effort. This can be done both by MIL procedures and C functions. A general interface for such operations can be defined to reduce this effort. This interface specifies three operations: the *init*, *next* and *finalize* operations. The *init* function initializes the reduction operation, e.g. setting variables. The *next* operation is called

for each element in the BAT. The *finalize* operation is called wrap up the result.

Spatial operations are somewhat more complex to map into MIL. The problem with such mapping is that the spatial domain of the result should be a priori known. Then it is similar to the induce image operations. First, a scan is performed to transform the spatial domain of the result into the spatial domain of the original image, calling f for each position. The function f should be defined for the two dimensional space. Finally, a join is required to look up the range values. Some $y \in domain(f)$ may require values outside the domain of a, i.e. $f(y) \notin X$. These will not be present in the induced image.

$$a \circ f \Longleftrightarrow [f](domain(f)).join(a)$$

To check whether the template image operations maps onto database operations, we first need a mapping of a template. A template is defined as an image of images, which maps to a table of tables. A template operation will map to a scan of the template pixels, which are again images. For each template image, an induced image operation is performed. The resulting image, which will after being reduced using an image reduce operation, form the resulting pixel value.

$$t \textcircled{\lambda} a \Longleftrightarrow [template\_op](t, [t \sim const\ a])$$

where

$$\Lambda([\bigcirc](t_y, a)) = template\_op(t_y, a)$$

The [t$\sim$ const a] construction creates a temporary template from the image a, so both operands of template_op have the same table of tables format. The template_op operation performs the real induced image operation and reduction.

To make the mapping of template operations to the BAT algebra operations clear we will explain the mapping of a well known image processing operation, **convolution**. The image **convolution** is a template operation which requires an induced image multiplication between the image and each template image. The resulting images are reduced using a summation image reduction operation. Figure 3.2 shows the implementation of image convolution in MIL.

As shown the image algebra operations map onto the binary relational algebra operations. Since templates are just a special kind of images, their operations also map into the algebra operations.

3.3 Benefits of BAT representation

The mapping solution proposed solves the identified problems with the two dimensional array representation. The true benefits should still be made

```
proc   image_mul( BAT[any,any] im, BAT[any,any] ty ):={
       [*](im,ty);
}
proc   image_sum( BAT[any,any] im ):={
       var res := v.fetch(0);
       vbatloop(1,v.count() - 1){
                    res := [+](res,$t);
       }
       return res;
}
proc   image_convolution( BAT[any,any] image,
                                BAT[any,BAT] template ):={
       var a := [image_mul]([template ~ image], template);
       var r := [image_sum](a);
       return r;
}
```

Figure 3.1: Example Template Operation

clear. We already discussed the advantage that arbitrary image segments are obtained without additional work. In addition, we have the following advantages of using binary relational tables as data representation for images:

- Image Integration at the core of the DBMS.
- Simplification of Data Structures and Code Reuse.
- Query Optimization.
- Database Supported Parallelism .
- Performance and Storage Improvements.

3.3.1 Image Integration

We simulated the Horus image representation and operations with the core of the Monet database system. An important immediate benefit is the availability of index structures which come with the binary tables. These index structures can be used to improve the performance of some image operations drastically. For example, searching the spatial and range domains can be optimized using proper accelerators[8].

Segmentation algorithms cluster pixels to form segments. For example [47] describes a segmentation algorithm based on k-means clustering in color

space. Having an index structure or automated lookup table on the pixel values is beneficial.

3.3.2 Simplification of Data Structures

The binary table is Monet's main data structure. The decision for a single complex structure (storing relatively simple atomic types) has proven to be crucial for its core development and its impressive performance. With the mapping we proofed (again) that this is also a powerful data structure to handle image data types. The single structure can be used as image, but also as data structure to store derived data sets. This means users (i.e. image researchers) only have to understand a single complex data type, which significantly decreases the learning curve.

The reuse of the BAT data structure has another important advantage, namely code reuse. All image algebra operations map on combinations of existing relational operators. There is no additional implementation effort. Introducing new image processing operations can easily be done by supplying the pixel value operations required for it. In many cases a simple MIL procedure suffices.

3.3.3 Query Optimization

The key to fast responds in a DBMS is the query optimizer. By mapping images into tables we can benefit from these techniques, i.e. the query optimizer has all information to choose an optimal query execution plan. Some techniques are introduce shortly.

Translation invariant templates

A possible optimization is to use the properties of a translation invariant template. When we know that a template is translation invariant, we can select one image from the template, for example t_y with $y = (0, 0)$, and use it to represent the template. All other template images t_x can be regenerated using this image and a translation of its pixel values from y to x.

To illustrate, instead of doing an induced image operation followed by a reduction for each position of the template t, we translate the original image a over x for each $x \in X$, where X is the position set of the template representant image. For each resulting image perform a binary scalar induced image operation, where the scalar is the pixel value at position x. The last step is to reduce the set of images using the reduce operation as the operator of an induced image operation. Using this scheme reduces the number of lookups of the pixel values of the template images. The steps are displayed graphically in Figure 3.3.3

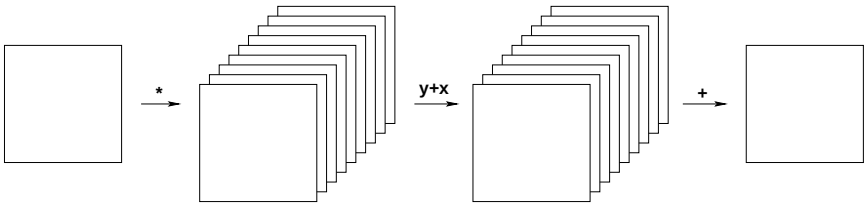


Figure 3.2: Translation invariant convolution

Reuse of intermediate results

Having translation invariant templates and some values v in the image t_y exists multiple times, we can further reduce the number of operations required. For each value in t_y we apply the induced image operation, i.e removing any duplicates. The optimized set of operations is displayed graphically in Figure 3.3.3

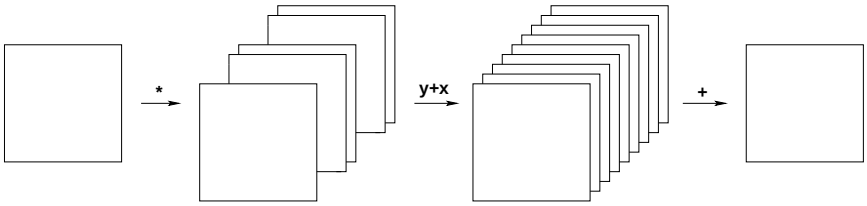


Figure 3.3: convolution reusing intermediate results

For some translation invariant templates, the values of the image t_y are not used. They are set to the unit value, i.e. it only expresses the selection of the image pixel values. For example in a uniform filter the values are not used, only the positions are of interest. The image algebra denotes these as *neighborhood operators*. These operators only require the translation and reduction steps of the template image product. Example neighborhood operations are uniform filter, median filter and dilation and erosion.

Further optimizations, such as "sliding windows", for the uniform image filter can be supported by specialized operations. These operators place a window over the original image, calculate a single result pixel from using the pixels in the window, slide the window, and calculate the next pixel reusing the pixel calculations in the intersection of the two windows. There should be a way to express when a image template operation should use these operations, i.e. a translation invariant template where all template value are equal can use the fast uniform image filter[61] implementation.

Figure 3.3.3 shows the implementation of an optimized translation invariant convolution. The translation invariant template is represented by a

single BAT[Y,F]. First the unique values are discovered. Using these the image multiplication is done. Each resulting image is translated over the given vector. The resulting images are combined (added) using the sum_images operation. The resulting values are divided by the sum of the pixel values in the translation invariant template.

```
proc   image_mul( BAT im, BAT ty ):={
       return [*](im,ty);
}
proc   sum_images( BAT im ):={
       var res := im.fetch(0);
       var rest := im.slice(1,im.count());
       rest@batloop(){
                        res := [+](res,$t);
       }
       return res;
}
proc   image_ti_convolution( BAT[any,any] image,
                     BAT[any,any] template ):={
       var unique_values := template.reverse().kunique();
       unique_values :=
              unique_values.join(unique_values.reverse);
       var mul_ims :=
              [image_mul]([unique_values const im],unique_values);
       var trans_ims := [translate]( mul_ims, template.reverse() );
       var sum_ims := sum_images( trans_ims );
       return sum_ims;
}
```

Figure 3.4: Translation Invariant Convolution

Filter Decomposition

Some translation invariant templates can be decomposed into a set of smaller templates. Such decomposed templates reduce the number of operations needed to calculate a image template product. For example a 3x3 template is decomposed into a 1x3 and a 3x1 template the number of operations required for the template image product reduces from 9 to 6 per pixel. So from $O(n^2)$ to $O(n)$.

A query optimizer is the right place to find out such decompositions and use it to optimize the query plan.

3.3.4 Parallelism

Mapping the image algebra operations onto relational algebra operations opens a road to parallel execution. For relational algebra operations many parallel algorithms exist. For example a template image product can be performed in parallel. The work needed for all the induced image operations can be spread over the pool of processors. Even lower granularity parallelism can be achieved using horizontal decomposition parallelism. A table is horizontally decomposed into multiple smaller fragments. These fragments are distributed over the processors and the work is done there. The resulting fragments are on return gathered to form the result.

An other form of parallelism is single instruction multiple data (SIMD), which can be found in nearly all modern CPU. Example instruction sets are Intel's MMX and SSE, AMD's 3D Now, Motorola's AltiVec, and Sun VIS. All are geared at multi-media applications, but these operations can just as easily be used by database systems. Having the image algebra mapped on the relational algebra it will seamlessly use the SIMD optimized operations. In this thesis, parallelism is not considered.

3.3.5 Performance and Storage Improvements

The mapping of images into Monet's BATs can be implemented with a storage overhead comparable to the two-dimensional array image definition commonly used by the image processing software packages. In this section we also indicate how to further optimize storage requirements.

To understand the solution, we first explain the BATs data structure. Figure 3.3.5 shows a typical BAT structure used in Monet. Each binary table consists of a Binary unit (BUN) heap, to store the head and tail of the relation. Each column has a fixed or variable type and optionally multiple search accelerators. Fixed sized atoms are stored directly in the BUN heap. Variable sized atoms are stored in a separate heap. In the BUN heap the position of the variable atom is stored. A BAT with a head type oid and a tail type chr will require 8 bytes, because integers require 4 bytes alignment on most systems.

Although this storage scheme proved flexible, deployment in the data mining showed another way to reduce the storage requirements. Let us take a look at Figure 1, which shows the decomposition of a relational table into BATs. The head of the BATs contain enumerations of unique object identifiers. This information can be represented by a single object identifier, indicating the first value, and a counter. Leading to virtual object identifiers (voids)[12]. Using the void type reduces the storage requirements drastically, since only one column needs to be stored.

We can use the virtual object identifiers (voids) trick to solve the redundant spatial information. Just using the void type as the head is not

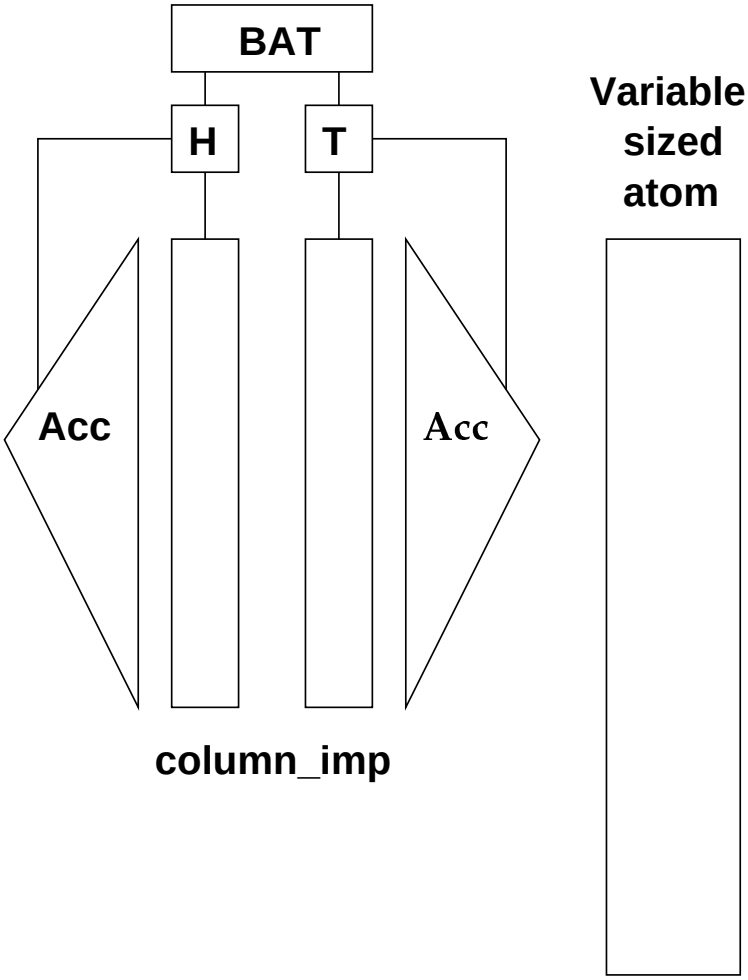


Figure 3.5: BAT data structure

sufficient, because the spatial information would be lost. Therefore, we introduce a separate BAT to store dimensionality information, i.e. the image width and height. Image operations using the spatial component should take care of handling these images based on void BATs. They should lookup the dimensions and generate the implicit spatial component before performing the actual operation.

This way, we achieve a huge storage reduction for the spatial component X of image. Can we also save storage for the F component? A 1024 by 1024 24 bit color image requires, after spatial reduction, still leaves 1024*1024*3 bytes, i.e. 3 MB, which is still huge when considering image databases with over 1M images (i.e. 3 TB databases). Fortunately, the cardinality of the different values in the images is usually much lower than the number of

pixels. This fact is also exploited by image compression schemes, such as found in the JPEG image format. In our Monet implementation a reduction may be possible using an indirection to find the pixel value. Instead of storing all values f directly, a position in a lookup table is stored. [1]

An important consequence of the lookup table is the possibility to defer non-spatial operations on the pixel values to the lookup table, reducing the number of operations dramatically. Example candidate image operations are unary and binary-scalar induced operations.

3.4 Experiments

To demonstrate that this approach is also feasible from a performance point of view, we performed some experiments with one of the most important image operations, the image convolution. As a sanity check, we compare our implementation against the implementations done in Horus.

Horus is a new image process library developed by the university of Amsterdam. It is designed to be a general image processing library, intended for image analyzing tasks, implemented in C++ making heavy use of code templates. The main focus of the library is performance. Therefor, some of the generality of the image algebra is given up in favor of processing speed. The Horus library only looks at translation invariant templates, called kernels. These are the ones used most frequently. Also the image implementation of Horus is focused on square shaped images, it lacks implicit image segment support.

To show whether our optimizations have the desired effects we compare our default convolution of invariant templates with a convolution using the indirected pixel value, i.e, using a lookup tables "ColorMap" representation which requires less multiplication operations.

We compared the execution times of image convolution operations for various image sizes, from 16x16 to 512x512 and various templates. The results for each template are shown in the figures 3.4, 3.4, 3.4 and 3.4. These figures show the results for the Horus convolution and both Monet convolutions with and without pixel value lookup table optimization.

Figure 3.4 shows that the mapping of images to BATs is only approximately 20% more expensive. This is relative small since it is achieved by the existing relational operators. The use of the lookup table directly pays off it gives a performance gain off 20% over Horus.

Figure 3.4 shows that both Monet implementations have approximately the same performance, which is 25% better than Horus. The reason is that the Horus implementation can not make use of the fact that all values in the

[1] A reduction from 3 bytes to 1 byte can be achieved when less than 256 different pixel values exist. Many images coming from the world wide web have this property.

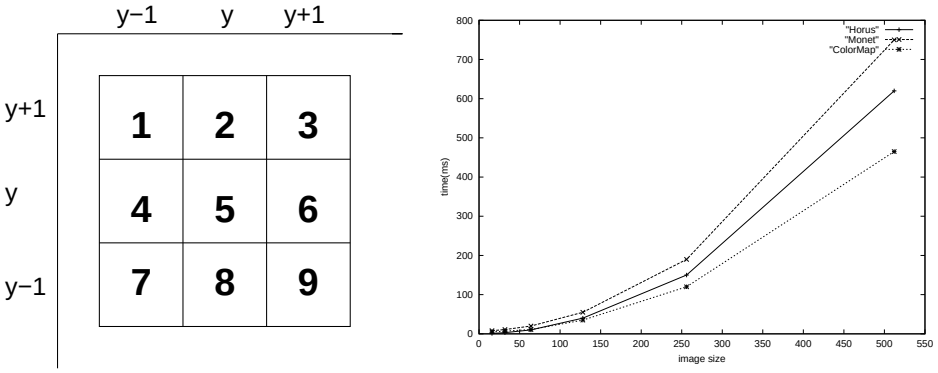


Figure 3.6: Execution times of convolution operation

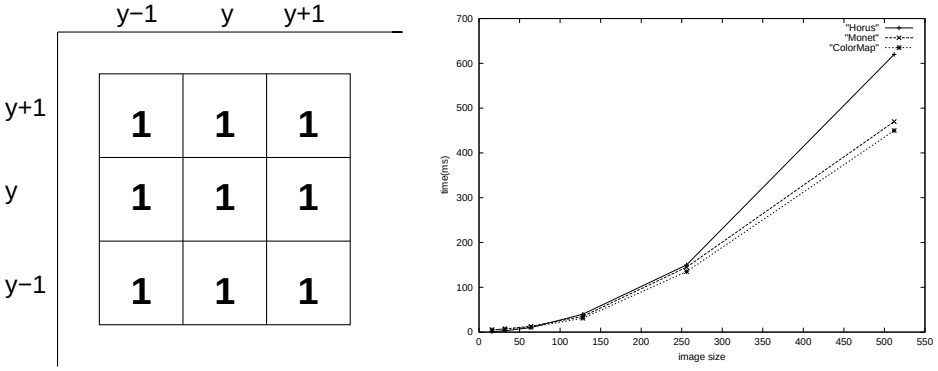


Figure 3.7: Execution times of convolution operation

template are equal. A user could solve in Horus by calling the neighborhood operation, but that shows the optimization to the user.

Figure 3.4 shows that the mapping pays of. The Horus cannot handle arbitrary shaped images and templates and therefore the convolution implementation has to go through the whole 3x3 template values, even though 4 values are 'zero'.

The last figure shows the combination of the previous two optimizations, i.e. reuse the intermediates and no calculations for the 'zero' template values. This gives already about 50% performance increase.

3.5 Requirements

The requirements coming from using binary tables as the data structure for images are:

- DBMS should be extensible with new abstract data types, for pixel

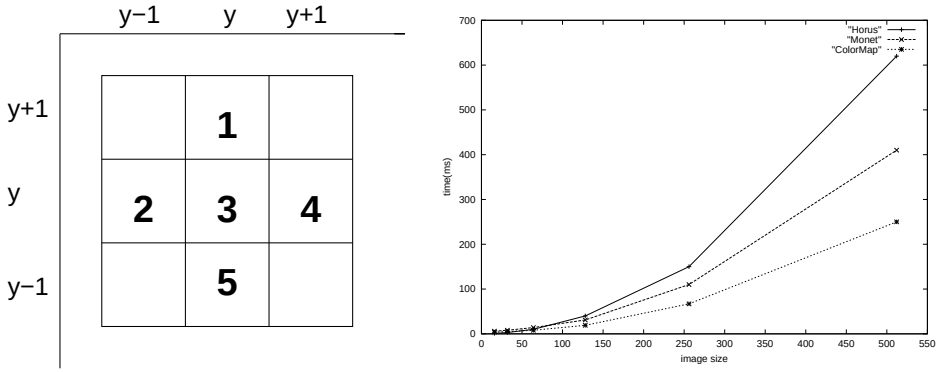


Figure 3.8: Execution times of convolution operation

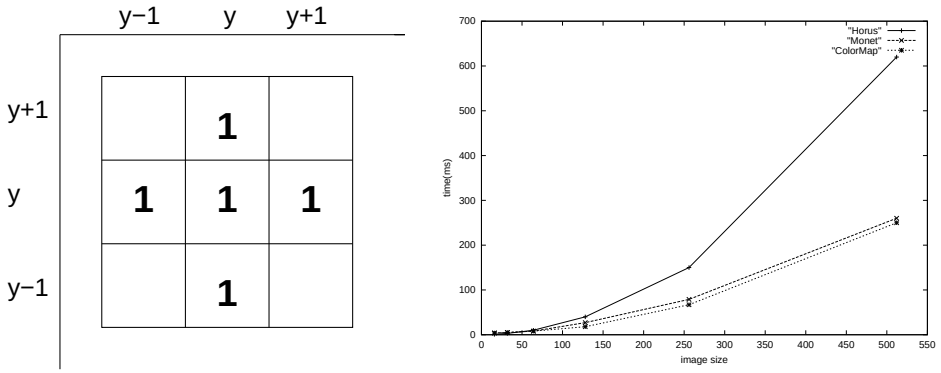


Figure 3.9: Execution times of convolution operation

position and pixel values.

- The DBMS requires a void data type with virtual identifiers.
- The DBMS requires BATs which store values using a lookup table.
- The DBMS should allow for operator overloading.

In our research we found a huge gape between the research communities of image processing and database management systems. A long history of different single object versus set at a time processing has widened the gap. With this chapter we hope to reduce this gap a little, since we know both worlds can benefit substantially from each other.

3.6 Conclusions

In this chapter we showed our mapping of images to BATs, i.e. binary tables. We indicated how a default implementation of the image algebra operations

can be achieved. This also proves the completeness of our approach. Using this representation we indicated many roads towards optimization. We indicated how these optimizations are obtained by the query optimizer to find better query plans.

We showed that sets of images can be compressed with an additional BAT interface. The interface allows for transparent access to BATs with compressed data.

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 on". 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 builtin 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/ãcoi

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:

```
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.

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

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

```
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

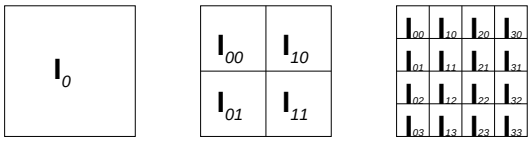


Figure 4.2: The quad splitting process

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.

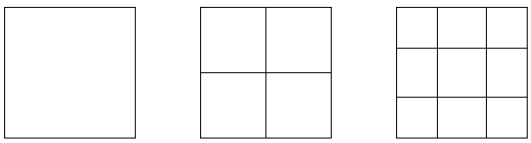


Figure 4.3: The prime splitting process

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.

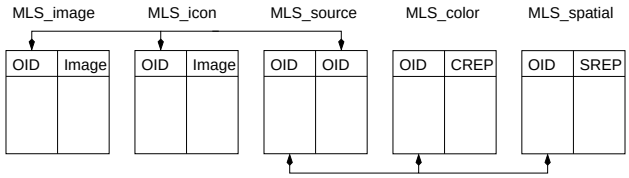


Figure 4.4: The relational data model used for storing the multi level signature.

The BATs for our retrieval system (MLS) are *MLS_source*, *MLS_color*, *MLS_spatial*, *MLS_image* and *MLS_icon*, (see Figure 4.4). *MLS_image* contains the actual image data. In *MLS_icon* an icon of the image is stored. Showing icons instead of the whole image reduces retrieval cost (less data needs to be transfered) and image display.

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.

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[46]. Furthermore, the hue closely resembles the human perception of related colors, which improves image retrieval from an ergonomic perspective.

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.

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 α.

2. Stop when the average color of corresponding parts on level n and n+1 differs less than some δ.
3. Stop when there are less than γ 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 β^2 pixels.

The first condition is independent of the image content. The second condition is intended to signal smoothness, but it suffers from large outlayers. 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 outlayers 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 *CI*s, 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 where retrieved. The similarity measure requires a non-expensive computation. Formally, at each level λ the similarity between the QI and CI is calculated as

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

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 λ. The function is normalized using the maximum color difference cb found in the database.

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.

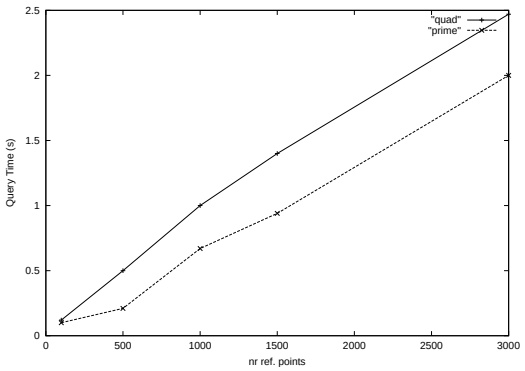


Figure 4.6: The results of the Scalability Experiment.

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 $Q1$. 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 having 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, knowing that proper image segmentation is an unsolved process in image analysis research.

Segments are found using both a split and a merge image indexing algorithm. Two kinds of algorithms are considered; a top-down method based on recursive splitting, called S-split, and a bottom-up method based on successive 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.

4.3.1 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 then 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 $O(d * n^2)$ 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 $H_{threshold}$ 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 $H_{threshold}$. 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: $S(R_1, R_2) = (\frac{(F_{R_1} - F_{R_2})}{weight})^2$, 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.

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.

Query Primitive	comments
use_avg_hue	use the average hue
use_dom_angle_freq	use the frequency of the dominant texture angle
use_dom_angle	use the dominant texture angle
use_histogram	use the global hue histogram feature
use_area	use the area of the segments
use_neighboor_dist	use the distance between the closest neighbor
avg_hue_diff	max. difference between hue values
dom_angle_freq_diff	max. difference between angle frequencies
dom_angle	max. difference between angles
histogram_diff	max. difference between histograms
area_diff	max. difference between the areas
neighboor_dist	max. difference between neighboor distances
sub_image_queriing	Query type B
image_queriing	Query type A

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[110] 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 $H_{threshold}$, $H_{objects}$ and H_{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 $H_{objects}$ 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.

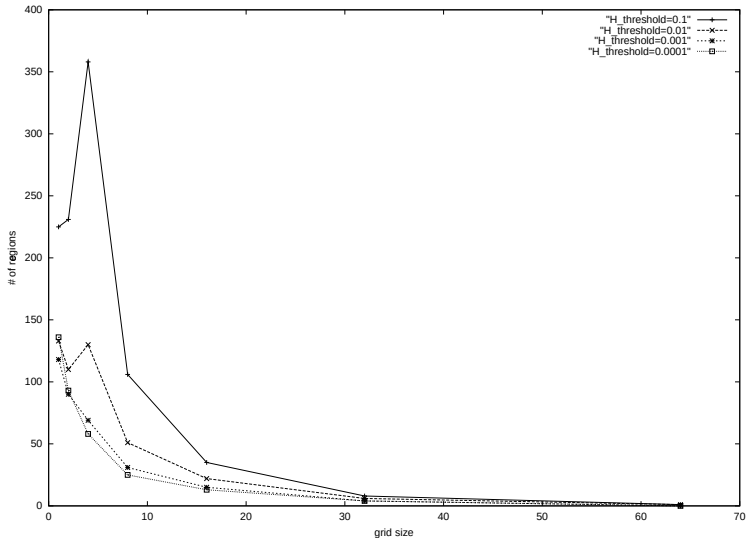


Figure 4.7: The number of segments using the S-merge algorithm

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{ $H_{objects}$, 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

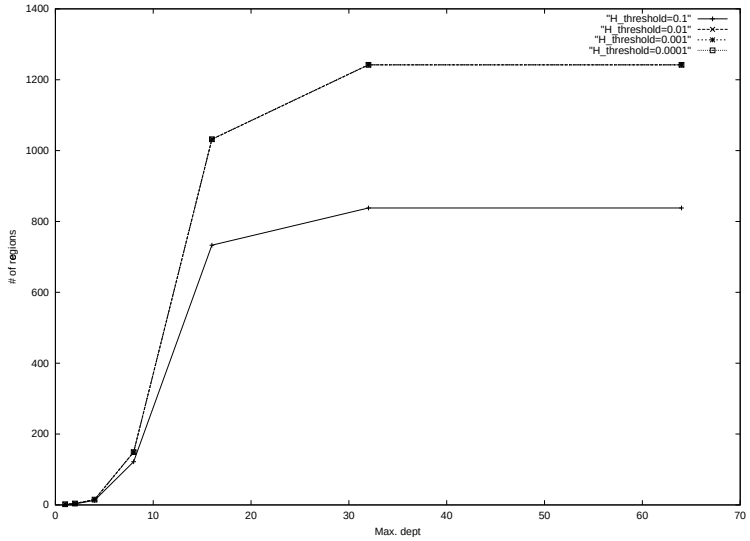


Figure 4.8: The number of segments using the S-split algorithm

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.

# BAT **name**	**count**
mmo_url	1002
mmo_name	1002
txt_keyword	17340
txt_phrase	612
img_segment	9042
ir_hue	9042
ir_texture	9042
ir_area	9042

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-

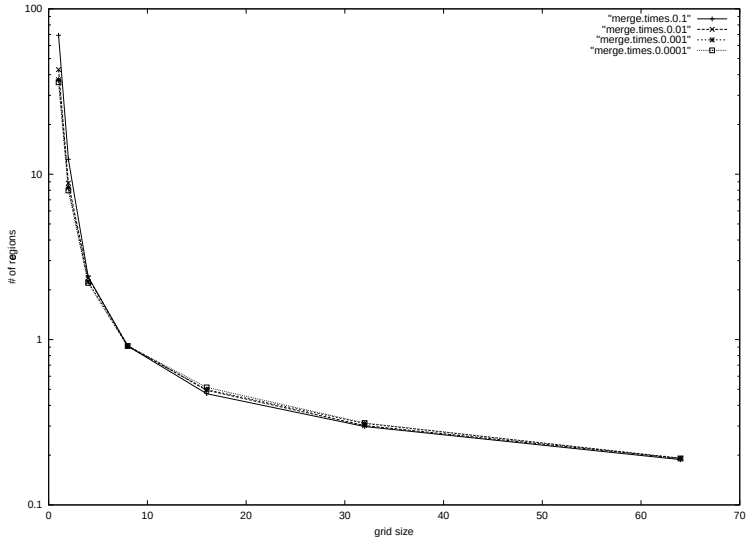


Figure 4.9: Execution times S-merge algorithm

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.

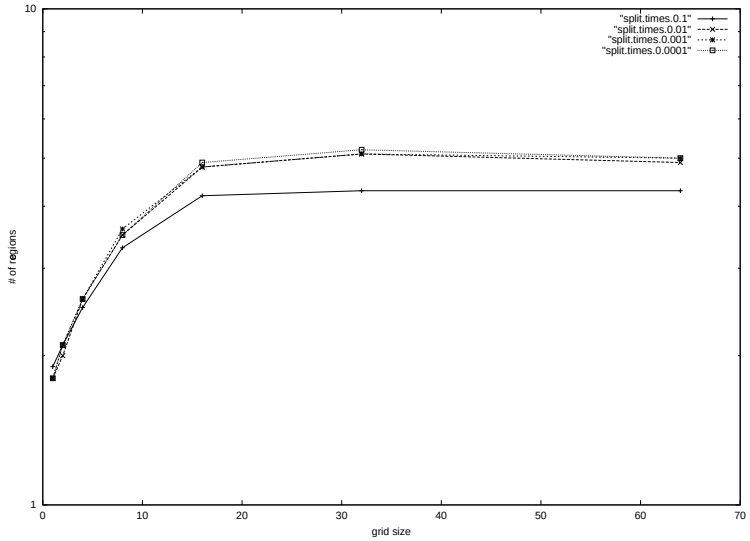


Figure 4.10: Execution times S-split algorithm

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 from 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

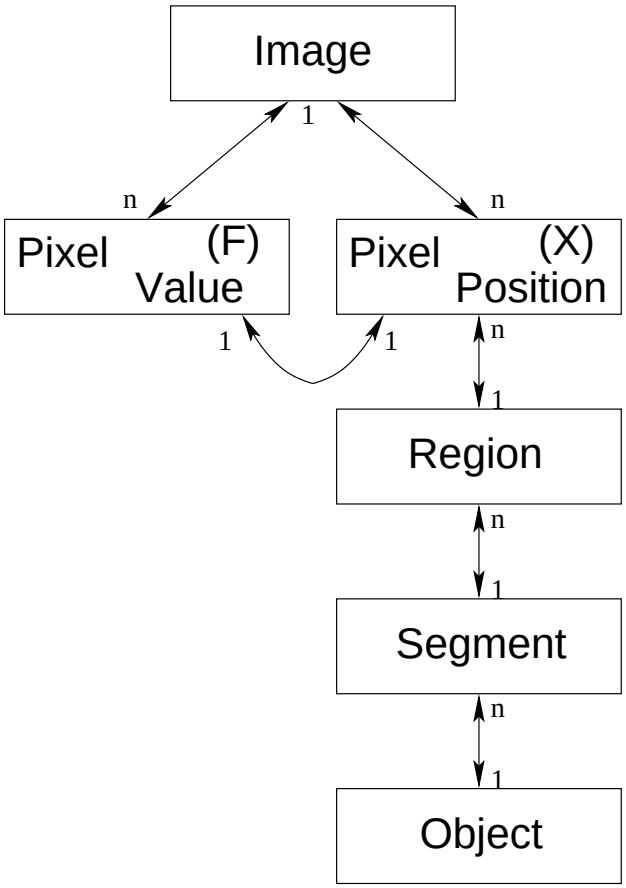


Figure 4.11: Image Retrieval Algebra Data Model

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 $2^n * 2^n$ 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

BAT **name**	**Comments**
mmo_url	resource locator
mmo_name	document base name
mmo_kind	{audio,image,video}
mmo_cntxt	enclosing document
mmo_time	last access time
mmo_censor	{ copyrighted, X-rated }
txt_keyword	keywords for an MMO
txt_phrase	key phrase for an MMO
img_type	{gif,tiff,jpeg,png,bmp,ppm}
img_size	image width x height
img_depth	pixel depth
img_stamp	derived icon
img_icon	user defined icon
img_region	image-region mapping
ir_color	region average hue
ir_domhue	dominant hue
ir_domangle	dominant angle
ir_histogram	region hue histogram
ir_area	region area

Table 4.3: Database schema

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 as materialized vier 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_join) combines region pairs maximizing a match function, $f(rs, rs) \rightarrow float$. The pairs found merge into a single segment. The

metric join (M_join) finds all pairs for which the distance is less than the given maximum m. The distance is calculated using a given metric function, $d(rs, rs) \rightarrow float$. The last function in this group, called *predicate* join (P_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:

similar($r\ r_1, r\ r_2$) :=
$touch(r_1, r_2)$ and
$distance(r_1.avg\_color, r_2.avg\_color) < 0.1$

The next group of primitives is needed for selection. The image find (F_find) returns the region which best matches the given region, according to function $f(rs, rs)$. The metric select (M_select) returns a set of regions at most at distance m, using the given metric $d(rs, rs)$ function. The predicate select (P_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_sort derives a partial order amongst objects. Each entry may come with a weight, which can be used by the *metric* sort (M_sort). This sort operation is based on a distance metric between all regions in the set and a given region. The N_sort uses a function to map regions onto the domain $\mathcal{N}$.

After the partial order the Top returns the top n objects of the ordered table. The *Slice* primitive will slice a part out of such an ordered table. The *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 to 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 $score =$

$((Q_1 + Q_2)/DBsize + Q_3 + Q_{3a} + Q_4 + Q_5 + Q_6 + Q_{6a})/8$, 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 (Q2) can be expressed in the Acoi algebra as follows:

```
var Q2 := [normalized_color_histogram](Acoi_Images);
```

The brackets will perform the operation normalized_color_histogram on all images in the Acoi_Images set. It returns a set of a histograms. Q3 uses a M_select with the L^2 metric. The sorting of Q3a can be done using the M_sort primitive. Query Q4 is implemented in the Acoi algebra using a F_join with the function $f(r_1, r_2)$ defined as follows:

$f(r_1, R_2) :=$
dist($r_1.color()$, $r_2.color()$) if r_1.touch(r_2)
max_dist

Queries 5 and 6 are implemented by longer pieces of Monet code**??**. The segmentation of query Q5 use an iterative process. This process can make use of the F_join primitive to find the best touching regions based on the color distance, see [79] for full details.

Query Q6 can be solved using a series of 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 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		
$area(T_1)$	$\rightarrow$	$float$
$perimeter(T_1)$	$\rightarrow$	$float$
$center(T_1)$	$\rightarrow$	$point$
$avg\_color(T_1)$	$\rightarrow$	$color$
$color\_hist(T_1)$	$\rightarrow$	$Histogram$
$texture(T_1)$	$\rightarrow$	$vector$
$moment(T_1)$	$\rightarrow$	$float$
Topological operations		
$touch(T_0, T_0)$	$\rightarrow$	$boolean$
$inside(T_0, T_0)$	$\rightarrow$	$boolean$
$cross(T_0, T_0)$	$\rightarrow$	$boolean$
$overlap(T_0, T_0)$	$\rightarrow$	$boolean$
$disjoint(T_0, T_0)$	$\rightarrow$	$boolean$
Join operations		
$F\_join_{f(T_0,T_0)}(\{T_0\}, \{T_0\})$	$\rightarrow$	$\{T_0\}$
$M\_join_{d(T_0,T_0),m}(\{T_0\}, \{T_0\})$	$\rightarrow$	$\{T_0\}$
$P\_join_{p(T_0,T_0)}(\{T_0\}, \{T_0\})$	$\rightarrow$	$\{T_0\}$
Selection operations		
$F\_find_{f(T_1,T_1)}(\{T_1\}, T_1)$	$\rightarrow$	T_1
$M\_select_{d(T_1,T_1),m}(\{T_1\}, T_1)$	$\rightarrow$	$\{T_1\}$
$P\_select_{p(T_1,T_1)}(\{T_1\}, T_1)$	$\rightarrow$	$\{T_1\}$
Ranking and Sample operations		
$P\_sort(\{T_1\})$	$\rightarrow$	$\{T_1\}$
$M\_sort_{d(T_1,T_1)}(\{T_1\}, T_1)$	$\rightarrow$	$\{T_1\}$
$N\_sort(\{T_1\})$	$\rightarrow$	$\{T_1\}$
$Top(\{T_1\}, int)$	$\rightarrow$	$\{T_1\}$
$Slice(\{T_1\}, int, int)$	$\rightarrow$	$\{T_1\}$
$Sample(\{T_1\}, int)$	$\rightarrow$	$\{T_1\}$

Table 4.4: The Image Retrieval Algebra

Join operations	result
$F\_join_f(L, R) \rightarrow \{T_0\}$	$\{lr \mid lr \in LR, \nexists l'r' \in LR \wedge f(l', r') > f(l, r)\}$
$M\_join_{d,m}(L, R) \rightarrow \{T_0\}$	$\{lr \mid lr \in LR \wedge d(l, r) < m\}$
$P\_join_p(L, R) \rightarrow \{T_0\}$	$\{lr \mid lr \in LR \wedge p(l, r)\}$
Selection operations	result
$F\_find_f(L, r) \rightarrow T_1$	$l \in L, \nexists l' \in L \wedge f(l', r) > f(l, r)$
$M\_select_{d,m}(L, r) \rightarrow \{T_1\}$	$\{l \mid l \in L \wedge d(l, r) < m\}$
$P\_select_p(L, r) \rightarrow \{T_1\}$	$\{l \mid l \in L \wedge p(l, r)\}$

Table 4.5: Signatures of the Join and Selection operations

nr	**query**
Q1	DB-load
Q2	$\{h \mid i \in Imgs \wedge h = normalized\_color\_histogram(i)\}$
Q3	$\{i \mid i \in Imgs \wedge L^2 distance(normalized\_color\_histogram(i), h) < 0.1\}$
Q3a	sort Q3
Q4	$\{n_1 n_2 \mid n_1 n_2 \in Regs(im) \wedge \nexists n3 \in Regs f(n_1, n_3) > f(n_1, n_2)\}$
Q5	$\{rs \mid rs \subset Regs(i) \wedge \forall r_1 r_2 \in RS :$ $L^2 distance(avg\_color(r_1), avg\_color(r_2)) < 0.1$ $\wedge \exists s_0 \ldots s_n \in rs :$ r touch $s_0 \wedge$ s_i touch $s_{i+1} \wedge$ s_n touch $s\}$
Q6	$\{i \mid \forall s_i \in Q6(i) \exists s_e \in Segs(e)$ $d(s_i, s_e) < min\_dist\}$
Q6a	sort Q6

Table 4.6: Benchmark Queries

Query	Time(s)
Q1	2865
Q2	598
Q3	1.5
Q3a	0.3

Query	Time(s)
Q4	1.0
Q5	1.2
Q6	1.5
Q6a	0.3

Table 4.7: The Acoi Benchmark Results

Chapter 5

Image Analysis: A case study

Image retrieval systems form an interesting, but small subset of the potential application of image databases. Our conjuncture is that image analysis researchers can also benefit from such systems in their day-to-day activities.

Currently, in image processing research, each analysis step is programmed in a third generation programming language. These languages are not known for their ease of programming, code maintenance, and reuse-ability. An object oriented programming style, such as seen in Java and C++, partly solves the maintenance and reuse problems. Java still suffers from performance problems. These problems are countered with proprietary data structures and associated operations to obtain better performance. Likewise, C++ still suffers major compiler compatibility problems when distribution to different platforms is the objective.

Besides the cumbersome programming language, programmers in image analysis often focus on hard recognition problems in isolation. It is not uncommon to be confronted by throw away code, because there is limited tendency to develop code for reuse. As a result, they also lack writing proper documentation and the image analysis community is stuck with software hard to maintain and reuse. Although, exceptions on these rules exist, such as the Horus image library[112], the current software approach hinders progress in this area.

The approach taken in this thesis is to use database technology as a step forward. A database system challenged to support image analysis has to overcome the following problems:

Erroneous Data The initial image data (and all derived data) contains errors due to inaccuracy of the measuring devices. Errors largely come in two flavors: the discretisation error from scanning (devices) and use of inadequate data structures. To illustrate the former, scanning devices have physical constraints on the LED's. To illustrate the latter, one could store a line simply by the end points, which again makes it dependent on the discretisation technique (i.e. grid precision)[51].

Another option is to use a triplet of center point, orientation and length, which is much less dependent on the discretisation technique.

Proximity and Probabilistic reasoning The fuzzy data involved requires mechanism to support proximity-based queries and a probabilistic computational model, This is currently not supported by database management systems. A step into this direction is researched by [32].

Multiple representations Many possible derived data features exist and each one can be represented in several ways, while transformations between them is generally not loss-less. The system should be able to handle all in a uniform way and decide on what to materialize.

Existing Algorithms Many image analysis algorithms exists and one may not expect image researchers to rewrite them into database queries. Instead, these often time consuming algorithms should be callable from the query language directly. To be useful in a database context they should be side effect free, otherwise any optimization is impossible. A criterion hardly ever met.

In this chapter, we propose to add core image processing functionality to the database management system, making it a better tool for image analysis research as well. The approach taken is to identify missing parts using a single representative case: "line clustering". The "line clustering" problem has been chosen, because it is a long standing problem in image analysis research and, therefore, an acceptable solution is known. The outcome of this experiment are operator and database requirements.

The human visionary system seem to recognize straight lines in images easily. Even when the lines are broken into segments, partially visible, and with a small angular distortion. Actually, a human clusters the visible line segments, such that with a certain confidence he or she claims they belong to the same line. This confidence is based on syntactic information, i.e. "most lines are straight" and a priori semantic information "we deal with power lines". The later aspect is a focus for model driven image analysis.

In the rest of this chapter we show, by means of the case study, that an IDBMS simplifies experimentation with the different image analysis algorithms and it takes less time to implement them. However, it is not our intention to solve the line clustering problem at large. Consequently, we don't explore all performance aspects, but focus on the translation of an image analysis problem into a database problem.

5.1 The line clustering problem

Many computer vision and image processing applications involve the seemingly simple problem of line detection. A clustering of segments into lines

that faithfully represent the original image is a pre-requisite for many image understanding algorithms.

Conversion of an image into a set of lines is a two-phase process. In the first phase, the image is converted into an edge map using a segment detection algorithm, such as [15]. The second phase deals with extracting the straight lines from the edge map.

A major problem encountered in practice is the lack of accurateness in the segment extraction algorithms [14, 39]. Segments may be broken, rotated or translated from their actual position in the source image. These short comings of the detection/extraction algorithms show up more when the original image contains different line styles, such as dashed and dotted lines, or the image is cluttered with lines. Some segment extraction algorithms may be better in handling rotations, others in handling width displacements. Since edge detection and segment extraction are long standing topics in computer vision it is unlikely that error free algorithms will be found shortly. The result is that image analysis starts with a large collection of segments that barely resemble the lines in the original image.

The way out of this dilemma is to focus on segment clustering algorithms to derive approximately correct lines. By developing clustering algorithms with a few controlling parameters it becomes possible to automate line detection up to a point that human intervention is reduced to a minimum. The proper parameter settings can be obtained by an expert user in an interactive application, which shows the clustering results for various parameter settings.

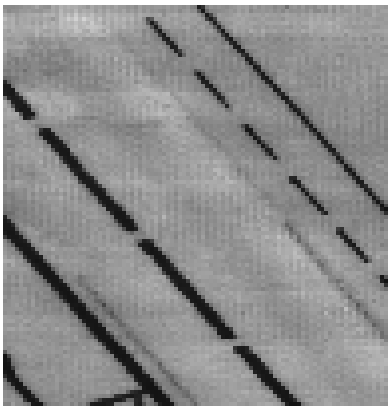


Figure 5.1: Example Image with Line Segments

For practical purposes [40, 76, 64] assume an image with a sparse number of lines, e.g. contours of a single sharp object. The example image, Figure 5.1, is taken from [64] which deals with powerline maps.

5.1.1 Clustering Hierarchy

The line clustering problem can be redefined as clustering the segments obtained from the extraction algorithm to form lines closely resembling the original line in the image. The predominant way to solve this is by constructing a clustering hierarchy, which groups of segments are more likely to belong to the same original line.

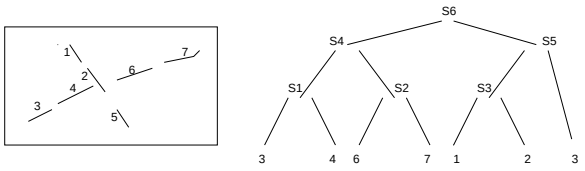


Figure 5.2: Example Clustering Hierarchy

See Figure 5.1.1 for a sample segment set taken from a utility map and the corresponding cluster hierarchy. In such a hierarchy each node represents a hypothetical line s_l, which best fits the underlying segment set. The leaves of the hierarchy contain the initial segments. Each node combines two segment sets into S_l. Note that s_l may, but need not collide with a detected edge.

5.1.2 Clustering Factors

The error classes caused by the detection phase are: *orthogonal distance* $d(s_l, s_i)$, *rotational displacement* $\theta_{s_l} - \theta_{s_i}$, and difference in *line width* between a segment and the hypothetical line. For dashed lines we include the factor *coverage*, i.e. coverage$(s_l, \{s_i\})$, measured as the sum of the segment lengths projected on the hypothetical line s_l, as the ratio to its total length.

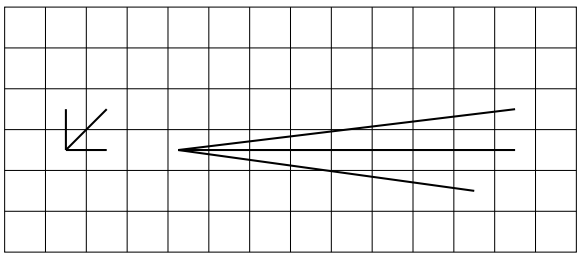


Figure 5.3: Example Rotational Error

The clustering factor *orthogonal distance* $d(s_l, s_i)$ specifies that segments far apart are less likely to belong to the same original line. The same holds for segments with a large angular displacement. This factor depends on

the length of the line segments, since small segments are subject to larger discretisation errors than long segments. Figure 5.1.2 clearly shows the error caused by a single pixel shift. A detector cannot separate a point from a small line.

The width difference factor models that segments of a single original line cannot differ too much in width. The coverage ratio needed to detect dashed lines states that the more of the hypothetical line is covered by the segments the higher the support for that hypothetical line is.

5.1.3 Clustering Function

Segment sets $\{s_i\}$ are constructed using a cluster function $M(\{s_i\})$, which produces a likelihood value in the range [0..1] for a set of segments supporting a single hypothetical line s_l. The hypothetical line s_l for a set of segments $\{s_i\}$ is obtained through a least square fitting algorithm. From [64] we obtained the following cluster function:

$$M(\{s_0, s_1, \ldots, s_n\}) = \sqrt[n]{\lambda(s_l, s_0) * \lambda(s_l, s_1) * \ldots * \lambda(s_l, s_n)} * \frac{L(s_l, \{s_i\})}{l_{s_c}} \tag{5.1}$$

Where the support function $\lambda(s_l, s_i)$ is defined as:

$$\lambda(s_l, s_i) = G_{\sigma_{a_{l(s_i)}}}(\theta_{s_l} - \theta_{s_i}) * G_{\sigma_o}(d(s_l, s_i)) \tag{5.2}$$

And where $G_\sigma(x)$ is defined as the Gaussian distribution:

$$G_\sigma(x) = \frac{1}{\sqrt{\pi\sigma^2}} e^{-(x/\sigma)^2} \tag{5.3}$$

The standard deviation of the rotational displacement θ of s_l and s_i is controlled by $\sigma_{a_{l(s_i)}}$. The permissible deviation is larger for shorter segments. In Equation 5.2 $d(s_l, s_i)$ is the orthogonal distance between s_l and s_i, measured as the shortest distance between the center of s_i and the line s_l. The standard deviation in orthogonal distance is controlled by σ_o.

The last part of Equation 5.1 controls the gap size between segments. $L(s_l, \{s_i\})$ is the sum of the lengths of all segments projected onto the line s_l and l_{s_c} is the total length of covered segment of the line s_l. Larger coverage indicates a higher chance that these segments belong to the same original. These functions are used to calculate clustering values to build a hierarchy of segment sets.

The hierarchy is build bottom up in an iterative process with the initial tree containing the individual segments as leaves. Using a hill-climber algorithm the tree is constructed by repeatedly combining two nodes. Once a clustering set is formed it cannot be split again.

Two Segment sets are clustered if there is no other combination with a higher support value. In particular, at each step, node pairs with a maximum support value are located and merged into a new node. The support function value is always based on the original segments, because even the best fitting algorithm for finding a hypothetical line has inaccuracies. Merging inaccurate hypothetical lines could result in lines that do not represent any of the lines in the original image. For example a circle broken into segments could mistakenly be clustered into a single straight line.

The clustering process continues until all nodes have been merged into sets and the clustering tree is completed by a single root. The hierarchy is then a complete clustering of all the segments into a single hypothetical line. From the clustering hierarchy, it becomes possible to select a clustering according to a user controlled threshold value. This threshold could be found using a process of visual feedback, i.e. an edgemap overlayed with the clustering for a given threshold δ. Clustering of segment sets which are too far apart or have very different orientations are ignored.

The algorithms proposed in [76, 64] calculates the likelihood value for each segment subset. This leads to a combinatorial explosion, because of the large number of segments involved. The algorithm does not scale. Their algorithmic complexity is cubic in the number of segments, (n), considered for clustering $O(n^3)$. Furthermore, the clustering function is expensive, because it recalculates the hypothetical line from the segment set considered.

5.2 Database Optimization

The solution described in the previous section involves user controllable parameters $(\sigma_{a(l(s))}, \sigma_o, \delta)$, and are based on complex calculations. Moreover, the clustering algorithm uses a single hypothetical line at a time processing. Although it gets the job done, it is far from optimal.

Conversion of this approach to a database set-oriented approach seems beneficial. The following optimizations strategies would be applied by a database programmer confronted with the task:

- Domain independent methods
  - Use spatial indices, to reduce the $O(n^2)$ problem with appropriate filters.
  - Factoring out expensive calculation, to reduce the CPU cost.
- Domain specific filtering
  - Use angular distortion to start the search for lines under the assumption that their angular distance is always limited.
  - Use line length to start with long lines, these are considered less erroneous.

- Domain specific algorithms
  - Use a set oriented approach gives a better handle to reduce repeated calculations.
  - Divide & conquer using repetitive splitting into disjoint sets until the optimum is found.

Using the same algorithm we could reduce the calculation of the expensive clustering function by careful selection of candidates for clustering. We can use domain independent indices to speed up this search, when a maximum segment distance is known. For example searching for candidate clustering pairs could be done using a spatial index structure, such as the R-tree [52]. We could also factor out the constant factors in the clustering function, which cannot be done transparently by a C-compiler.

5.2.1 Mathematical Optimization

Solutions as described by [40, 76, 64] use functions to calculate a clustering value for a set of segments, i.e. they all use similarity measures. The clustering function combines the clustering factors into a single value that quantifies the support for the hypothetical straight line. A better approach is in the first step to use these factors to filter out approximately 90 % of non-interesting cases, and then solve the remainder using these expensive measures.

The clustering algorithm discussed uses a clustering hierarchy, but specific applications will use a threshold value to select only a subset of this hierarchy, i.e. all clusterings $\{s_i\}$ where $M(\{s_i\})$ exceeds the threshold δ. Having this threshold we could reverse engineer the clustering function and find information to reduce our search space.

Two optimization methods could be applied: search optimization and filtering. The first, improves searching for candidate clusterings. For example a search for candidates based on spatial locality. In the naive algorithm the candidates considered are all segment set combinations. A filtering optimization would significantly reduce the set of candidate clusterings using a cheap operation as follows. Knowing two segments can never be clustered when their orientations differ too much, we could cheaply filter these out using a selection on their orientation difference. To see how we apply these optimizations, take a closer look at the clustering function 5.1.

Consider Equation 5.1 and assume $n-1$ segments perfectly fit the hypothetical line s_l. This together with threshold δ leads to Equation 5.4.

$$\sqrt[n]{\lambda(s_l, s_n)} * \frac{L(s_l, \{s_i\})}{l_{s_c}} \geq \delta \tag{5.4}$$

When only considering the angular displacement, i.e. assume coverage is perfect and orthogonal displacement is 0, leaves Equation 5.5. Only considering the orthogonal displacement gives Equation 5.6.

$$\sqrt[n]{\frac{1}{\sqrt{\pi\sigma^2_{a_{l(s_n)}}}} * e^{\frac{-(\theta_l-\theta_n)^2}{\sigma_{a_{l(s_n)}}{}^2}}} \geq \delta \Rightarrow \theta_l - \theta_n \leq \sqrt{-log(\delta^n * \sqrt{\pi\sigma^2_{a_{l(s_n)}}})} * \sigma_{a_{l(s_n)}} \tag{5.5}$$

$$\sqrt[n]{\frac{1}{\sqrt{\pi\sigma^2_o}} * e^{-\frac{d(s_l,s_n)^2}{\sigma_o{}^2}}} \geq \delta \Rightarrow d(s_l, s_n) \leq \sqrt{-log(\delta^n * \sqrt{\pi\sigma^2_o})} * \sigma_o \tag{5.6}$$

Assuming no rotational and orthogonal displacement leads to Equation 5.7. The maximum gap between two segment sets, S_1 and S_2, can be obtained when both the sets are perfectly aligned, then $L(s_l, \{s_i\})$ is the sum of all segment lengths and l_{s_c} is the sum of all segment lengths plus the gap size.

This gives an upper bound on the distance between to perfectly aligned segments, i.e. a maximum gap size.

$$\frac{L(s_l, \{s_i\})}{l_{s_c}} \geq \delta \Rightarrow gap \leq \frac{1-\delta}{\delta} * (L(S_1) + L(S_2)) \tag{5.7}$$

To allow for candidate search based on segment distance we need to combine this with the maximum rotational displacement and maximum orthogonal distance because these factors could influence the distance between segments. There are three extremes to consider when searching the maximum segment distance, maximum gap size, maximum orthogonal distance and maximum rotational displacement. The maximum segment distance is the maximum of these three extremes. See Figure 5.2.1 for these extremes. The gap size and orthogonal distance can be calculated directly.

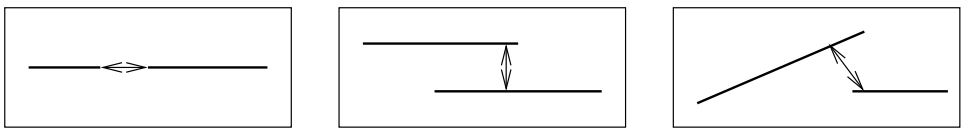


Figure 5.4: (a) max. gap size (b) max. orthogonal distance (c) max. rotational displacement

The angle θ between the hypothetical line s_l and the segment s_n depends on the segment distance, $d(s_l, s_n)$. This leads to Equation 5.8.

$$d(s_l, s_n) \leq \sqrt{max\_gap^2 + (sin(\max\_angle) * 1/2l(s_n))^2} \tag{5.8}$$

5.2.2 Split based algorithm

The problem of the clustering algorithms discussed so far is their complexity. For each level of the clustering hierarchy n^2 combinations are checked.

One way to reduce the possible combinations is to look at the data characteristics at hand. For example Figure 5.2.2 shows the segments length and orientation of an A4 sized utility map. This figure clearly shows that the large segments can be easily split into two groups around the peaks, 60 and 175 degrees. This could give two (disjoint) input sets for the optimized clustering algorithm.

When we apply the same method for the segment width and polar r coordinate we can identify groups of segments, which are strong candidates for clustering. The splitting in groups should allow for overlapping groups, since not all clustering factors have clear split positions.

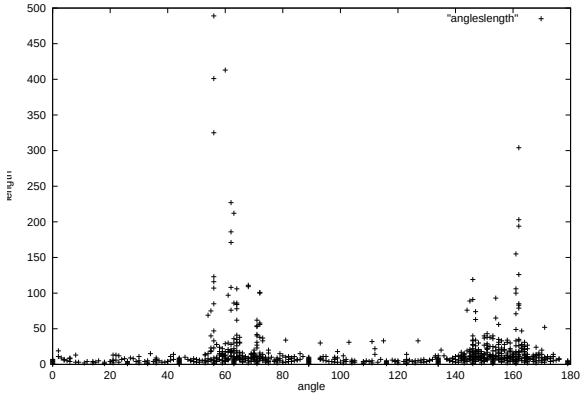


Figure 5.5: Angle Length

5.3 A hybrid solution

In this section we illustrate integration of an existing algorithm for line clustering with our extensible database management system Monet. The approach taken is to store the segments and clustering hierarchy in a database table. The clustering algorithm extracts portions of this table for processing and stores the results back into the database. This hybrid algorithm is directly based on [64], and illustrates use of a DBMS as a single object server. It does not involve delegation of work.

5.3.1 Database representation

Integration of the line clustering algorithm and database technology requires definition of a data model. The ideal situation, from an image processing

point of view, is when the C++ structures in the application program are understood and managed by the DBMS. This ideal is pursued by object-oriented systems, such as persistent C++ and ODBMS.

Although this leads to a seamless interconnection, i.e. no impedance mismatch, the state of the art DBMS optimization techniques have limited effect. The underlying reason is the iterative processing model still adhered to in the application code, rather than a more declarative set-based approach.

The solution is to use an extensible relation DBMS, which offers a declarative query language which could be extended with new abstract data types, commands and search accelerators. Examples are Postgres and Informix, where the user can introduce abstract data types, whose representation is a byte sequence. Conversion of application data structures into their database equivalents then merely amounts to conversions into/from a byte sequence.

Another example is our extensible database system Monet[8], which supports user-defined abstract data types, called atoms, search accelerators, and new commands. These are introduced using the Monet extension and C/C++ programming language.

5.3.2 Data Model

Recall that in Monet the data is fully vertically decomposed into Binary Association Tables (BATs), see Section 2.3. The data model for this application domain consists of object identifiers (oids), segments, and sets of oids. Oids are part of the standard data types of Monet, and thus all operations needed are at hand. For the segments a new atom (abstract data type) was needed, see Figure 5.6 for the module specification.

The input for the line clustering problem is a large set of line segments obtained by scanning a grey-value image, using an edge detector followed by a straight line extraction algorithm. This initial information is placed in a BAT, which contains for each segment an object identifier and a segment representation, see Figure 5.7 for a slice taken from this BAT.

The nodes of the hierarchy are also stored in a BAT. For each node again an object identifier is used together with a set of oids. These sets of oids represent the sets of segments. See Figure 5.3.2 for the required schema.

5.3.3 Clustering Algorithm

The subsequent step is to recode the algorithm [64] in MIL. Although this is a straight forward mapping, it pays off to judiciously use indices to trim the space explored. The cost incurred by combinatorial explosion encountered in the naive line clustering algorithm can be reduced using a spatial filtering technique. The rationale is that two segments are merge candidates if they are spatially local.

```
.MODULE segment;
.USE point;

.ATOM segment[32,8];
.TOSTR   = segment_tostr;
.FROMSTR = segment_fromstr;
      .NEQUAL  = segment_comp;
       .HASH    = segment_hash;
       .NULL    = segment_null;
.END;

.COMMAND direction( segment ) : dbl = segment_direction;
"get segment direction"

.COMMAND center( segment ) : point = segment_center;
"get segment center point"

.COMMAND length( segment ) : dbl = segment_length;
"get segment length"
.END segment;
```

Figure 5.6: The segment atomic type

The database structure to support location of spatial objects is the R-tree, supported by one of the Monet extension modules [13]. An R-tree clusters spatial local boxes together. We used this index on the bounding boxes of the segments. These boxes were blown up, because the maximum segment distance depends on the segment length. We used a R-tree join to find overlapping bounding boxes and therefore find close by segments. We can use this search accelerator without any further programming.

The original algorithm finds all clusterings for a given threshold δ. Based on the threshold we find appropriate maxima for the clustering factors, see Section 5.2.1. We calculate the maximum allowed distance between two segment sets. A maximum orientation, width and gap size difference which could still lead to a possible clustering. Based on these maxima we filter out possible clusterings. These filters reduce a lot of expensive calculation of the support function.

The hybrid algorithm follows the original hill-climbing process, which will go through the following steps.

The possible clusterings are found using a R-tree overlap join, which clusters all overlapping boxes and, thus, all spatially local segments. The set found is further reduced using the angle and width heuristic filters. In

```
#--------------------------#
# BAT:          tmp_28     #
# (oid)         (segment)  #
#--------------------------#
[ 4286, ((3844.000000,15573.000000),(3834.000000,15578.000000))]
[ 4287, ((3834.000000,15578.000000),(3829.000000,15572.000000))]
[ 4288, ((3829.000000,15572.000000),(3841.000000,15564.000000))]
[ 4289, ((3844.000000,15563.000000),(3852.000000,15560.000000))]
[ 4290, ((482.000000,15590.000000),(491.000000,15596.000000))]
[ 4291, ((491.000000,15596.000000),(478.000000,15602.000000))]
[ 4292, ((478.000000,15602.000000),(475.000000,15600.000000))]
[ 4293, ((475.000000,15600.000000),(476.000000,15590.000000))]
[ 4294, ((480.000000,15591.000000),(480.000000,15591.000000))]
[ 4295, ((480.000000,15589.000000),(480.000000,15589.000000))]
[ 4296, ((5823.000000,15651.000000),(5830.000000,15660.000000))]
[ 4297, ((5830.000000,15660.000000),(5822.000000,15660.000000))]
[ 4298, ((5824.000000,15647.000000),(5832.000000,15638.000000))]
```

Figure 5.7: Slice from the bat with segments

Bat	**head type**	**tail type**
segments	oid	segment
boxes	oid	box
segment sets	oid	oidset
nodes	oid	oid

Figure 5.8: Schema

other words, only the clusterings segments which are spatially local, and in the same angle and width ranges, are the candidates considered.

In stage 3 all clustering values are calculated using the original cluster function. This cluster function has been shown to be accurate, but it is also rather expensive. It needs to find the hypothetical line with the maximum support value. Finding the best hypothetical line through the set of segments is done using a least square fitting algorithm. Because the support function depends on the hypothetical line, which changes when new segments are added, no reuse of the calculations can be done.

The support values are known, the clusterings with a maximum support value for each of the containing clusters should be selected. Thereafter the clustered segments and corresponding boxes are deleted from their BATs, and the new sets are inserted in those BATs. For each new cluster a new bounding box is calculated and inserted into the R-tree BAT. The bounding box of the set of segments is that box containing all segments. This stepping process will continue until nothing more can be clustered.

Algorithm hybrid

1. **Spatial join**, join all possible clusterings based on spatial locality
2. **Heuristic selection**, make a selection based on the angle heuristics. Filter out pairs with very dissimilar orientation.
3. **Support values**, calculate the precise support values for the hypothetical line.
4. **Select maximums**, select clusterings which have a maximum support value
5. **Update**, update the BATs containing the nodes and boxes

Since segment clusters are represented by there segment identifiers, interface functions should find the corresponding segments and convert those into the data structures needed for the clustering functions. This means there is a lot of conversion overhead. For the different parts of the calculation of the heuristics and support values a number of segment and node characteristics are needed, like center point, angle, and mean angle. Also these characteristics have to be recalculated each time the different functions are called.

Using different selection criteria has proven to significantly reduce the number of times the support value is calculated. As a consequence the support function is no longer the dominant performance overhead. The heuristics are now responsible for most of the execution time.

The use of the R-tree reduces the search space depending on the threshold. Since the tree is build only once the cost is low compared to the costs of the other functions.

5.4 Database Solution

In this section we demonstrate how a modern DBMS can be used to tackle the line clustering problem. In Section 5.4.2 we show that algorithms based on these primitives outperform the solutions given in the previous section.

5.4.1 Line cluster model

The input to the line clustering problem is a large set of line segments obtained by scanning a grey-value image. The segments are represented as atomic values in the database using the extensions introduced in Section 5.3.

The line clustering problem can be rephrased as finding (disjoint) subsets that provide maximum support for a hypothetical line derived from the

Operation	**functionality**
[op](BAT[ht,t1] α, BAT[ht,t2] β)	$\{at : ab \in \alpha \wedge ac \in \beta \wedge t = op(b,c))\}$
{op}(BAT[ht,tt] α)	$\{ab : \beta = \{c : ac \in \alpha\} \wedge b = op(\beta)\}$

Table 5.1: Monet's BAT Update Operations

segments in the subset. Alternatively, each segment is assigned to a single subset and moving a segment from one cluster to another reduces the clustering value for both.

The key operation of the line clustering algorithms is to analyze a segment set. That is, the basic object of manipulation is a set of segments. In the previous algorithm in fact, the object was constructed as soon as transfer occurred to the clustering algorithm. This is not necessary. Within the database environment a line can be represented by a multi-valued function from a group identifier to a set of segments. How these segments are glued together to form a line is a separate issue.

A group can be reduced to a single value using the Reduce command. Using the Map command a function can be applied to each group member and a group parameter, see Table 5.4.1.

The algorithms are all linear in the size of their operands. They can also be parallelized easily.

The easiest way to represent the initial segment set within Monet is to introduce a mapping from $g_i \rightarrow \{s_j\}$, where g_i denotes a group identifier and s_j a segment identifier. This can be represented with a single Binary Association Table called *groups* and forms the first level clustering of segments. For each group we calculate a bounding box $g_i \rightarrow \{b_i\}$. The bounding boxes are stored in the BAT named *boxes*. The remainder of the clustering runs as follows:

Algorithm DBMS

Filtering on spatial locality Determine the candidate pairing of groups using an overlap operator over their bounding boxes, i.e. C := overlap(boxes, boxes.reverse)

Create Groups Create a candidate group for each joined pair and store it in a BAT, i.e. CG := C.mark().join(nsegs)

Filtering on max angle diff Determine the maximum angle difference between the average and the segments angle for all groups, and select those segments where the angle difference is less than than max_angle_diff, i.e.
Cangle := {sum}(NG)
Cnr := sum(C.join(nnr))

Cmangle := [/](Cangle, Cnr)
Cmadiff := max([angle_diff](Cangle, Cmangle))
CG := CG.semijoin(Cmadiff.select(0.0, max_angle_diff))

Calculate Hypothetical line Calculate interpolated straight line through the center points, First get center of all line segments in the candidate group, from this calculate the center point for the hypothetical line, a second point is calculated using interpolation, this point is used to find the lines orientation i.e. *CCenter := [center](CG)*
NCenter := [/]({sum}(CCenter)),Cnr)
dx := [-]([x](CCenter),[x](NCenter))
CX := [*](dx,dx)
CY := [*](dx,[y](CCenter))
CXsum := {+}(CX)
CYsum := {+}(CY)
Angle := [atan2]([-](CXsum,[x](NCenter)),[-](CYsum,[y](NCenter)))
Line := [new_line](NCenter,Angle)

Calculate support values Calculate the product of the width, rotational, and orthogonal distance heuristics

Select new clusters Select groups with maximum support values for both containing clusters

Update retained information Delete clustered groups and insert the new groups

The result of the distance join is used to construct groups of segments. This again uses the search optimization based on spatial locality. These candidate groups are further reduced using filters on orientation and segment width. The operations needed for the calculation of the hypothetical lines are done in a setwise fashion, which gives us the possibility to reuse the intermediate results, such as the sum of the segment centers and the mean orientation. The calculations needed for the filters can be reused during the calculation of the support values. With these support values the best clusterings could be selected. At the last step we have to update the retained information, such as bounded boxes and mean orientation.

We can reuse intermediate results because we changed from a single segment at a time approach to a more database approach of set at a time bulk operations. There are no extra implementation efforts involved for this set-at-a-time approach, because the primitives needed are already supported by the database kernel.

5.4.2 Experiments and Results

Extensive experiments were performed to compare both effectiveness and efficiency of the algorithms. The input for those experiments were segment

sets obtained from real-life utility maps. Following [64] we used the standard deviation σ_o and σ_a 1.5 and 0.18 respectively.

# segments	time(s) sequential	time(s) DBMS
300	23	2.3
600	76	2.9
1200	298	4.7
1600	454	5.4
2000	660	6.9
2736	1330	8.7
4000	2749	15
5472	5280	93

Table 5.2: Results for the sequential and DBMS algorithms

The first experiments show the efficiency of the database approach against an algorithm used by [64] in image analysis. It enables comparison of the traditional C++ based implementation with the DBMS algorithm. The set of segments S is extracted from a single utility map. A family, of sets F is constructed, such that the elements in F form a subset lattice, i.e. $\forall f_i \in F$ $\exists f_j \in F : f_i \subset f_j$ The experiment was done with a fixed threshold δ of 0.5. The result are shown in Table 5.2.

As can be seen form the first experiment the DBMS set at a time algorithm is an order of magnitude faster than the sequential algorithm. This result can be attributed to effectiveness of the spatial index.

The second set of experiments with the hybrid and DBMS algorithms was focussed on the efficiency of these algorithms. Different input sets were taken. The sets were extracted from different utility maps each with its own segment density. Figures 5.4.2, 5.4.2 and 5.4.2 show the histograms of the segment orientation. The experiment was done with various thresholds to evaluate the performance degradation under larger clustering hierarchies. Table 5.3 shows the results.

The set at a time algorithm out performs the Hybrid algorithm on large data sets, because then it profits from the information retained. With small sets recalculation is less expensive.

All the experiments were done on a Sun SPARC-X running the Solaris operating system and using an early Monet V3 version.

5.5 Conclusion

It has been shown that an extensible DBMS can be used to tackle the line-clustering problem. The overhead of the conversion between database

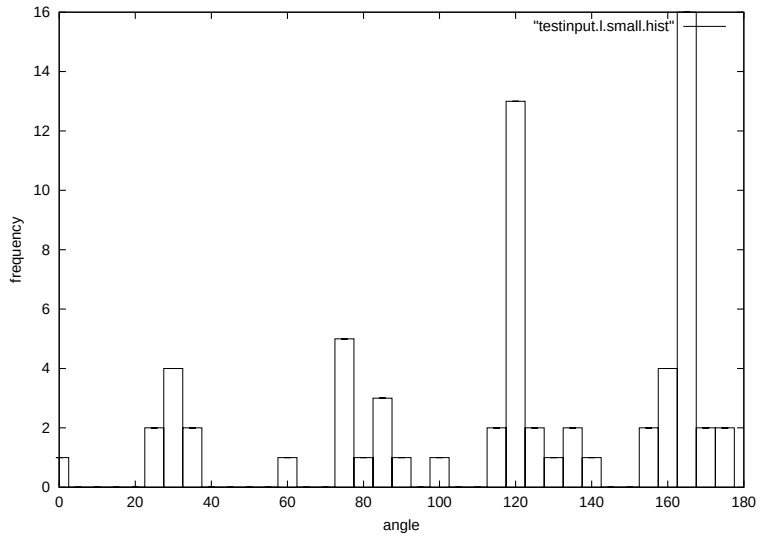


Figure 5.9: Histogram of the line segment angles, small image

structures / application structures is not a dominant factor. Moreover, there exists a small algebraic extension to the DB core functionality, which enables us to tackle the line clustering problem with database kernel support. The performance is promising compared to the original solution written in C++.

# segments	threshold	hybrid	DBMS
67	0.25	1.1	1.4
67	0.4	0.9	1.2
67	0.75	0.5	0.8
2585	0.25	82	67
2585	0.4	56	36
2585	0.75	13	7
4362	0.25	77	45
4362	0.4	50	35
4362	0.75	9	4

Table 5.3: Result for the Hybrid and DBMS algorithms

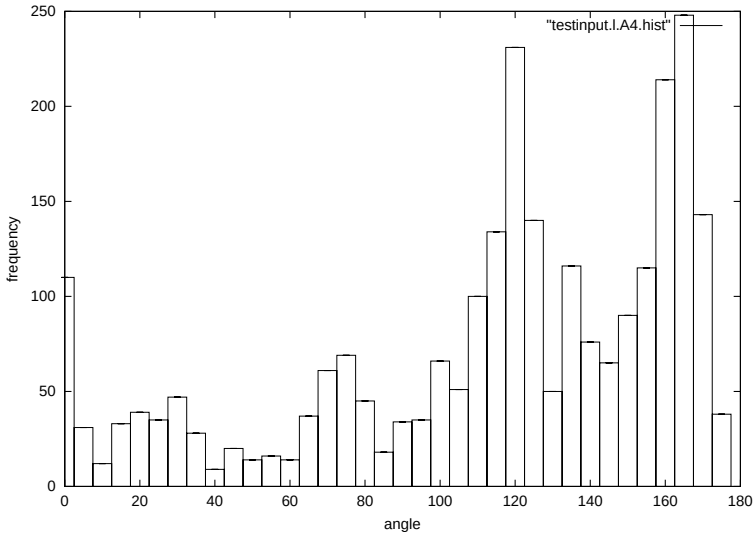


Figure 5.10: Histogram of the line segment angles, A4 sized image

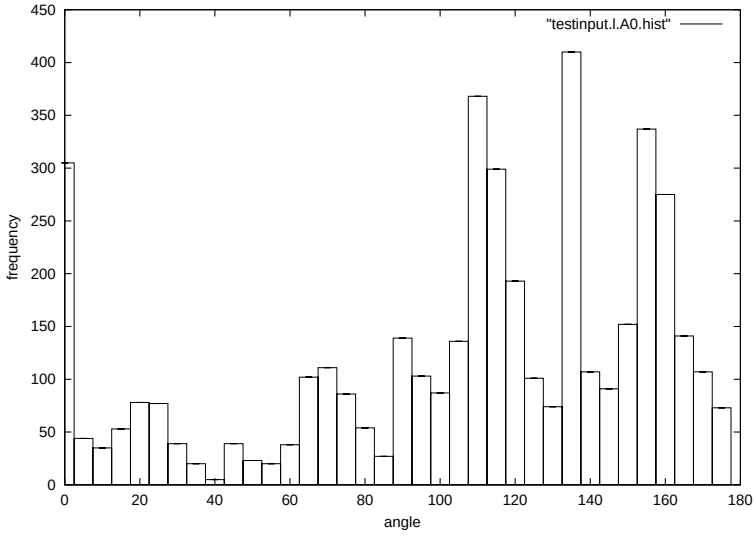


Figure 5.11: Histogram of the line segment angles, A0 sized image

Chapter 6

Fitness Join

6.1 Introduction

Join algorithms have a long tradition of interest in the database community. By the late-70s the key algorithms were published [7]. Nested-loop, sort-merge, and hash-based join algorithms have successively been explored extensively to reduce their running time, including their parallel versions. For an overview of these results see [73, 67]. In the object-oriented context joins based on path traversals have been supported using join indices [111] and pointer-based algorithms [33]. The results have been generalized into a methodology for index structures[38, 23].

The common denominator of the join studies is that they largely deal with equi-join conditions. Algorithms with more complex conditions, i.e. theta-joins, have been barely scratched upon [34], let alone their embedding in real-life applications.

Band-joins have been identified as a crucial asset for engineering applications, which require constraints on intervals around the join attribute. A traditional sort-merge algorithm has been put forward as the best attack, which brings the problem back into the realm of the now classical algorithms [34].

More recently, Helmer and Moerkotte [97] have investigated the extension of main-memory join algorithms to deal with subset-join predicates. From an application perspective, near match solutions are considered of more importance than total match. Further progress has been reported for optimizing cross products in [97] and universally quantified predicates in [24].

These join activities are exemplary to better support novel applications in a bottom-up fashion. Namely, the core relational algebra is extended gradually with new features. However, starting from the way novel (non-database) applications are being built, we have encountered the need for *fitness joins*, which take the form of identifying join pairs optimally placed

or ranked under a given metric condition. The naive solution for a fitness join starts with producing the cross product followed by calculation of a fitness value for each pair using a (sub-)query. Thereafter the best (worst) from the perspective of one of its operands is (are) retained (for ranking).

The contributions of this chapter rest on introducing a reference example to study the class of fitness joins, followed by chartering a road to develop new algorithms and optimization schemes. The search space for effective solutions is trimmed by presenting a concrete operator for fitness joins, called the *bounded theta join* operator implemented in Monet [9]. The scope for further optimization is illustrated using the mathematical properties as guidance for search heuristics. The bottom line being to extend a query optimizer with a limited form of mathematical reasoning to derive effective processing heuristics.

The remainder of this chapter is organized as follows. Section 6.2 provides a motivational example to study the fitness joins. Section 6.3 illustrates how fitness joins can be handled in both a traditional and extensible database setting. It also introduces the bounded-theta solution in Monet. Section 6.4 charters the contours of mathematical query optimization. Section 6.5 summarizes a concrete implementation and its evaluation. We conclude with challenges for subsequent research.

6.2 Motivation

In this section we introduce the class of joins considered with an artificial example, *the ballroom*, followed by an indication of the application domains in which they appear.

6.2.1 The Ballroom Example

Consider the database shown in Figure 6.1, which lists the participants of a yearly ballroom contest. All persons are qualified dancers as illustrated by their repertoire. In the training session for the contest it makes sense to identify (or rank) potentially good teaching partners, i.e. those with a good overlap of dance repertoire and partners of similar age. During a dance session it is mandatory to quickly find a partner with the proper dance repertoire on the dance floor when the music changes, because dancing with an inexperienced person does not contribute to the training experience, especially with the Tango. Positions of the dancers are also shown (`[x,y]`).

These questions can be rephrased to database queries as follows:

Q_1: *Find a partner of opposite gender and closest in age.*
Q_2: *Find a partner with the best overlap in dance repertoire.*
Q_3: *Find a male partner with a larger dance repertoire.*
Q_4: *Find the closest Tango partner on the floor.*
Q_5: *Rank the dance partners by experience.*

Person					
name	gender	age	[x,y]	repertoire	experience
Tina	female	21	[4,4]	{rumba, samba, jive}	6
Brian	male	24	[0,5]	{tango, foxtrot, rumba, samba, waltz, jive}	10
Mary	female	25	[4,2]	{tango, foxtrot, walt}	8
Susan	female	28	[1,4]	{foxtrot, waltz}	4
Edward	male	30	[5,4]	{rumba, foxtrot, jive}	8
Owen	male	32	[5,2]	{foxtrot}	1
Ruth	female	33	[3,4]	{rumba, jive}	3
Peter	male	36	[5,1]	{foxtrot, waltz, jive}	8

Figure 6.1: The Ballroom database

Let's have a more detailed look at the requirements. Q_1 can be answered using the age difference over persons, i.e. $f_1(r,s) = abs(r.age - s.age)$, and to retain those that minimize this function for all possible couples. Query Q_2 calls for comparison of set-based attributes, where we have to map dance capabilities into a relevant information quantifier. Good partners are those that have an identical dance repertoire. The best dance partner would be one that maximizes the fitness function that compares the repertoire sets of couples:

$$f_2(r,s) = \frac{|r.repertoire \cap s.repertoire|}{|r.repertoire \cup s.repertoire|}$$

Choosing a good teacher calls for inspection of the repertoire of the potential partners. Their dance repertoire should exceed your own and the best teacher is selected by repertoire count. This can be answered with the following simple fitness function:

$$f_3(r,s) = \begin{cases} |repertoire(s.name)| & \textbf{if } \text{r.repertoire} \supset \text{s.repertoire} \\ 0 & \textbf{otherwise} \end{cases}$$

The teacher selection procedure can be further refined by keeping a tally on the experience of each dance performed by a person. However, then we run into a semantic problem as to precisely define by what is meant with a good teacher. The predominant approach is to look at the experience histogram distributions for two repertoire sets and determine a metric similarity. The best teacher is then someone with larger repertoire and a maximal distance in the experience space. This, however, quickly breaks down. An Euclidean metric would favor a super expert on a single dance over someone who has adequate dance experience in multiple dances. The way out of this dilemma is to consider alternative (standard) metrics or

to call upon the user to give a fuzzy mathematical definition of the metric intended.

Query Q_4 combines a simple predicate with a spatial term using the Euclidean distance in the ballroom and we retain those minimizing the fitness function.

$$dist(r,s) = \sqrt{(r.x - s.x)^2 + (r.y - s.y)^2}$$

$$f_4(r,s) = \begin{cases} dist(r,s) & \textbf{if } \text{"tango"} \in r.repertoire \cap s.repertoire \\ & \wedge\ r.gender \neq s.gender \\ \textbf{nil} & \textbf{otherwise} \end{cases}$$

Ranking the dance partners by experience (Q_5) calls upon sorting the possible couples using e.g. the function $f_5(r,s) = s.experience$ or $f_r(r,s) = s.experience/|r.repetoire \cup s.repetoire|$. Sorting boils down to an iteration over the candidates, finding the next best partner not ranked yet. Although sorting can be implemented with repeated joins, it is certainly not the best solution.

In practice, arbitration may also be required to assure a maximum number of couples on the dance floor, i.e. minimizing the total dissatisfaction. This aspect is not dealt with in this thesis.

6.2.2 Fitness Joins

The ballroom examples leads to the class of *fitness* joins. The basis is formed by two sets of objects R and S organized into a bi-partite graph. This graph is obtained by taking the cross-product over R and S retaining pairs that satisfy a priori selection criteria. Each edge carries a value obtained from evaluation of a fitness function. From this enriched graph we retain all edges that minimize (maximize) the edge weight from the perspective of the first operand.

For example, the figures below represents the bi-partite graphs and fitness join result for the dancers under fitness functions for f_1, f_2 and f_4, respectively. It illustrates that persons may be assigned several candidates and that (in principle) the function is not symmetric.

The functions are built from the standard repertoire of mathematical functions in a DBMS. Furthermore, we have encountered the need to also describe discontinuous and constrained functions, e.g.

$$f(r,s) = \begin{cases} Expr_1 & \textbf{if } Cond_1 \\ Expr_{n-1} & \textbf{if } Cond_{n-1} \\ Expr_n & \textbf{otherwise} \end{cases}$$

The function lines being interpreted in sequential order until a condition holds. For example, in case the ballroom lacks experienced dancers of opposite gender, we may relax the partner choice. This fuzzy function could be described as:

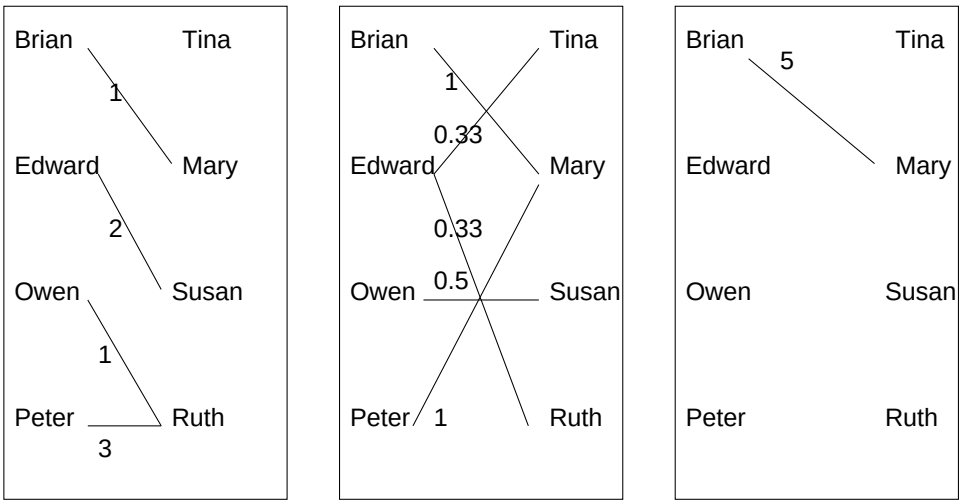


Figure 6.2: Query Fitness Graphs

$$f(x,y) = \begin{cases} 1 & \textbf{if } x.gender \neq y.gender \\ & \wedge\ x.repertoire \cap y.repertoire \neq \emptyset \\ 0.7 & \textbf{if } x.repertoire \cap y.repertoire \neq \emptyset \\ 0.1 & \textbf{otherwise} \end{cases}$$

Care should be taken on the definition of f. Besides being type correct we ignore all candidate pairs where either $f(r,s)$ or $f(r,t)$ is undecided or evaluates to **nil**. In general, this function can not be mimicked with an SQL CASE statement and the user is forced to create a query batch to simulate the behavior intended.

In the context of an extended relational model the fitness join can now be defined as follows:
Definition Let R and S be two (object) relations. Then the *minimal* fitness join $\lhd_f$ over R and S under function f is defined by:

$$R \lhd_f S = \{(r,s) | r \in R, r \neq s \in S\ \forall t \neq r \in S\ f(r,s) \leq f(r,t)\}$$

This definition combines tuples that minimize the fitness function f, it identifies (local) minima in the function space from R. If both arguments to a fitness join are one and the same table then the definition assures that we do not retain the identity pairs as being most fit. Likewise we can pair objects to maximize the fitness function.

Definition Let R and S be two (object) relations. Then the *maximal* fitness join $\lhd^f$ over R and S under function f is defined as follows:

$$R \lhd^f S = \{(r,s) | r \in R, r \neq s \in S \; \forall t \neq r \in S \; f(r,s) \geq f(r,t)\}$$

The consequence of our liberal definition of fitness functions is that, in general, the fitness joins are not symmetric ($R \lhd^f S \neq S \lhd^f R$), nor transitive ($(R \lhd^f S) \lhd^f T \neq R \lhd^f (S \lhd^f T)$). For a given point there may exist several minima (maxima).

6.2.3 Application domains

The Ballroom problem has notably many incarnations in literature. We have also encountered them in several applications developed for Monet [13, 79, 12]. They often appeared as heuristic functions written in a traditional language (C,C++). A pattern emerged that called for better support to analyze object pairs using a concise mathematical formulae, i.e. the $fitness$ function, as a necessary step in an iterative process to combine elements into larger semantic units (objects). Good database support for the fitness join alleviates these programs from using dedicated data structures and heuristic that makes re-use of the code base near to impossible. To illustrate:

- In *image databases* there is a need to compare high-dimensional data elements, such as color histograms under a distance metric[62]. A high-dimensional data structure is called upon, which is known to bring the dimensionality curse, while at the same time its semantic interpretation falls in the trap indicated for Q_3. A more precise fitness function specification may provide clues to weed out bad pairs by exploitation of the mathematical properties.

- In *image processing* there is a need to recognize complex-objects from primitive objects, e.g. triangularizations, grid-based decompositions, and more general image blobs. For example, lines should be recovered from line segments recognized during poor scanning [79, 81, 77]. The fitness function in this context binds segments that are angular similar, not too far apart, and of similar thickness. The clustering algorithm then attempts to maximize fitness to determine the candidates to pair into larger units.

- In *geographical information systems*, there is need to better support spatial joins and nearest neighbor search. To deal with it in an effective way emphasis is placed on spatial index structures, such as illustrated by [96, 42]. The fitness function in this context takes the form of an Euclidean distance metric using (a posteriori) filtering of candidate using attribute constraints.

- In *engineering* there is a need to deal with the imprecision of signals, calling for relaxation of the traditional equi-join condition. The concept of band-joins, e.g. $r.age - d_1 \leq s.age \leq r.age + d_2$, is a step into this direction [34], but also a special case of the fitness join.
- In *data mining* applications partial functions are used to classify information into coarse grain groups before the mining algorithms are fired to infer business models [54, 88] regrouping is needed. The groups depend on a fitness function.
- In *time series* applications, the fitness function translates into selecting a 'best-fit' between time series fragments [30, 29].

Although this list is by no means exhaustive, it highlights the need to consider specific support of the fitness join at the DBMS system level. Leaving the problem with the application programmer to deal with it on a case by case basis is from the database perspective not acceptable. We should find and assess algorithms and techniques to better support this large community.

6.3 Fitness Join Algorithms

In this section we analyze the necessity to extend a database kernel with a tailored implementation of the fitness joins. Such an extension should be weighted against required orthogonality of its instruction set and the opportunities to provide optimized versions otherwise not available or easily detectable by the query optimizer.

6.3.1 SQL Framework

From a computational perspective, the fitness joins can be readily supported by object-relational and O-O systems. However, due to the fitness join definition we have to fully exploit the DBMS's capabilities to group objects, to express the complex mathematical function (through a stored procedure), and to select specific elements from each group considered.

In the context of flat relational systems fitness joins lead to complex batches of SQL requests. Although multiple query optimization schemes may reduce the overhead incurred to some extent, the state-of-the-art in this field does not provide generic solutions in the near future.

To illustrate, in a relational system we can simulate the fitness join Q_1 using the SQL framework below. Its proper evaluation requires a nested query and exploits SQL's iterative semantics. Such queries are known to be notoriously difficult to optimize[1]. At best the optimizer can extract common sub-expressions or (erroneously) flatten it to a double cross-product expression.

```
select   r.name, s.name
from     Person r, Person s
where    r.gender <> s.gender
and      abs(r.age-s.age) ≤    (select abs(r.age-t.age) from Person t
                               where s.name <> t.name
                                 and r.gender <> t.gender)
```

Queries Q_2 and Q_3 are further complicated by a metric over set-valued attributes. Resolving this query in a flat relational framework is cumbersome. For an object-based framework with set operators it can be solved using a query to derive the table `temp(rname,sname,fitness)`. Subsequently, a simply aggregate query can extract the best partner. Traditional query optimizers have a hard time to optimize the cross-products and aggregates [94].

Query Q_4 uses the Euclidean distance metric, which is ideally supported by a Data Cartridge or Datablade for geographical information systems. The resulting basic SQL framework is shown below. Again not much can be done to improve response time. An R-tree index helps to solve point, region, and spatial joins, but it can not directly be used by a query optimizer to solve this fitness equation. The optimizer would have to detect that there is a better alternative for calling the two distance functions in the first place.

```
select   r.name, s.name
from     Person r, Person s
where    r.gender <> s.gender
and      distance(r.age,s.age) ≤   (select distance(r.age,t.age)
                                   from Person t
                                   where t.name <> s.name
                                     and r.gender= t.gender)
```

Note that this query depends on explicit naming of the distance function. Replacing it with the underlying definition would seriously jeopardize performance, because a query optimizer would not recognize easily the benefits of an R-tree. Likewise, the user could solve query Q_4 using a built-in function to access a nearest neighbor for any given point in the space covered. This solution works if, a priori, we split the Person into two tables, one for each gender, and to built a R-tree accelerator for fast access on locality. Thereafter, we can solve Q_4 with the following SQL query:

```
select   r.name, nearest neighbor(Females, r.x,r.y)
from     Males r
```

A query optimizer will typically generate a scan over Males and call the function for each instance. It will (normally) not consider building a search accelerator on Males first, followed by an index specific nearest neighbor algorithm, e.g. traversing the R-tree of both operands in parallel.

Person_name	
oid	name
0	Tina
1	Brian
2	Mary
3	Susan
4	Edward
5	Owen
6	Ruth
7	Peter

Person_age	
oid	age
0	21
1	24
2	25
3	28
4	30
5	32
6	33
7	36

Person_gender	
oid	name
0	female
1	male
2	female
3	female
4	male
5	male
6	female
7	male

Person_[x,y]	
oid	point
0	[4,4]
1	[0,5]
2	[4,2]
3	[1,4]
4	[5,4]
5	[5,2]
6	[3,4]
7	[5,1]

Person_dance	
oid	dance
0	rumba
0	samba
0	jive
1	foxtrot
1	rumba
1	samba
1	waltz
1	jive
2	tango
2	foxtrot
2	walt
3	foxtrot
3	waltz
4	rumba
4	foxtrot
4	jive
5	foxtrot
6	rumba
6	jive
7	foxtrot
7	waltz
7	jive

Person_experience	
oid	experience
0	6
1	10
2	8
3	4
4	8
5	1
6	3
7	8

Figure 6.3: The Monet Ballroom database

6.3.2 Monet solutions

To improve upon the situation sketched, we study the opportunities of extending a relational algebra engine. The system being considered here is Monet, a binary relational algebra engine, including powerful grouping primitives and facilities to extend its behavior through dynamic loadable modules [9, 12]. The Monet schema for the Ballroom database is shown in Figure 6.3.2.

The naive solution for the fitness joins $R \lhd_f S$ is to generate an iterative program in the Monet Interface Language (MIL) [9][1], that collects the minima for each $r \in R$. It would be the default route taken by most relational query optimizers.

In the remaining sections we focus on a more general scheme, using the fitness function of Q_1 as the focal point. Support for set-based operations

[1] The papers can be accessed through http://www.cwi.nl/∼monet

at the kernel level has been dealt with in the context of datamining support [11]. A synopsis is beyond the scope of this chapter. Furthermore, the spatial operators and their implementation in the context of Monet are reported in [13].

Cross Table Solution

The next step is to identify the necessary extensions to the relational algebra, such that performance gains can be obtained from bulk operations. In this quest, we follow Monet's approach to fully materialize results of binary operators. This has proven to be effective in most situations, because it trades storage space against memory cycles lost by MIL interpretation and context switching [12, 10]. The key problems to be addressed for the fitness joins are then to build a cross-table (sparse matrix) of the candidate pairs, calculate the fitness value per element, and retain the minima per row (R) order.

The first step is to extend the algebra with the notion of cross table together with primitives to query it. In Monet the cross table could be represented by two binary tables $Rmap : idx \rightarrow oid$ and $Smap : idx \rightarrow oid$ where their idx constitutes an index into the 2-dimensional space spanned by $|R| \times |S|$. The OIDs can be used to access the attributes of R and S tuples, respectively. These tables are produced with a tagging operator, $R.tag(X, Y, Z)$, which assigns the index value to each element in $R \times S$ using a conventional array-layout algorithm as shown in Figure6.4. The $tag-$operation can be implemented cheaply using a single scan over its operand.

Subsequently, cross table specific operators can be introduced, such as fetching elements using a idx, slicing a portion of the cross table for further processing, transformation of the cross table, and other matrix-like operations. For solving the fitness problem we need cross-table aggregates such as $rowMin$, $rowMax$,... to work on rows and columns. These are straightforward extensions of their regular implementations.

With the algebraic extensions in place, we can solve the fitness join query Q_1 in the Monet Intermediate Language (MIL) as follows.[2]

[2] See for details on MIL [9] and http://www.cwi.nl/∼monet

```
proc tag (R,X,Y,Z) := {
      var answer := new(int,int);
      var i :=0;
      var k :=0;
      R@batloop(){
        var j:=0;
        i:= k;
        while(j<X){
          answer.insert(i,$h);
          j:= j+1; i:= i+Y;
        }
        k:= k+Z;
      }
      return answer;
}
```

Figure 6.4: Monet Tagging Operation

	$R \lhd_f S = \{(r,s) \mid r \in R, s \in S\ \forall t \in S\ f(r,s) \leq f(r,t)\}$		
0	R	:= gender.select("male")	# R:oid → str
1	S	:= gender.select("female")	# S: oid → str
2	Rmap	:= R.tag(count(S),1,count(S))	# Rmap:idx → oid
3	Smap	:= S.tag(count(R),count(R),1)	# Smap:idx → oid
4	Rage	:= Rmap.join(age)	# Rage:idx → int
5	Sage	:= Smap.join(age)	# Sage:idx → int
6	delta	:= [abs](Rage[-]Sage)	# delta: idx → int
7	best	:= delta.minRow(count(S))	# best: idx → int
8	pairs	:= best.duplicate()	# pairs:idx → idx
9	v1	:= Rmap.reverse.join(pairs)	# v1:oid → idx
10	answer	:= v1.join(Smap)	# answer:oid → oid
11	a1	:= R.reverse.join(answer)	# a1:name → oid
12	a2	:= a1.join(S)	# a2:name → name

This script uses the Monet core algebra, multi-cast, and cross-table group operations. Lines 0,1 dissects the gender (binary) table into one for males (R) and females (S). Lines 2,3 constructs the candidate pairs mapping both tables into the cross-table space using the *tag* operation. Lines 4,5 associates the age values with each cell.

The multi-cast expression $[abs](Rage[-]Sage)$ in line 6 evaluates the age subtraction against all elements $Rage$ and $Sage$ with corresponding keys producing the table $\{Rage.idx, Rage.age - Sage.age\}$. Subsequently, the $abs()$ function is applied to all tuple tails, producing the table $\{Rage.idx, abs(Rage.age - Sage.age)\}$.

The minimum value per row is retained in line 7 using the cross-table enhancement. Line 8 replicates the key to prepare for the selection of couples

oids in line 9 and 10. Finally, line 11 replaces the oid by the person's name.

The storage required for this script to work is $|R| + |S| + 3 * |R| * |S|$ using eager garbage collection of intermediates. The processing cost is in the order of $4 * |R| + 4 * |S| + 5 * |R| * |S|$ steps. Evidently still too expensive to consider as the default for evaluation for large ballroom contests, because this algorithm still uses complete cross-products. Fortunately, for a large class of fitness functions a better solution exists.

Bounded Theta-Join Algorithm

Both the iterator and cross-table solutions ignored the mathematical properties of the fitness function. Yet, exploitation of these semantics may prove valuable in reducing the processing cost even further. For example, reconsider query Q_1, the minimum age difference, where the following equations hold:

$$abs(r.age - s.age) \leq abs(r.age - t.age) \Leftrightarrow \begin{cases} r.age \leq s.age \leq t.age \\ r.age \geq s.age \geq t.age \\ s.age \leq r.age \leq t.age \end{cases}$$

They illustrate that ordering on the age domain of s and t could be used to reduce the number of candidates. Furthermore, a pair (r, s) such that $r.age \leq s.age$, can be determined with a theta-join. The algorithm could be rephrased accordingly and it halves the space of candidates considered. However, this is still too many, because the fitness join requires for each r just one s; its closest neighbor under the fitness expression.

To tackle this problem, we have extended the relational algebra with a *bounded theta join* operation, e.g. $btj(R, S, \theta, n)$ where θ is one of the relational operators $(<, >, \leq, \geq)$ and n is the bound on the number of results retained per left operand value. An efficient implementation of the theta-join is already available in the Monet engine. It uses an index on one of the operands to speed up the search. If necessary, this index is created on the fly. Looping through the second operand, it produces the qualifying pairs, which are copied into the result table.

Its refinement for the bounded case $n = 1$ is relatively straightforward. It merely has to check the result table for duplicate insertion on the r component and to retain the minimum s encountered. The general case $n > 1$ requires slightly more work to retain the $n-$set of minimal values. The easiest way is to also create a sorted access path on the join attribute of S (if not already there). Then we can optimize the btj by merely doing a index lookup followed by a constrained (n)-step walk along the index to find the candidates of interest. Such tactical decisions are commonplace in the Monet kernel and prove to be highly efficient [9].

Using the bounded theta-join the Monet script of can be augmented as follows.

	$R \lhd_f S = \{(r,s) \mid r \in R, s \in S\ \forall t \in S\ f(r,s) \leq f(r,t)\}$		
0	R	:= gender.select("male")	# R:oid → str
1	S	:= gender.select("female")	# S: oid → str
2	Rage	:= age.semijoin(R)	# Rage:oid → int
3	Sage	:= age.semijoin(S)	# Sage:oid → int
4	right	:= thetajoin(Rage,Sage,≤,1)	# right : oid →oid
5	left	:= thetajoin(Rage,Sage,≥,1)	# left : oid → oid
6	cand	:= union(right,left)	# cand: oid → oid
7	Rmap	:= cand.mark()	# Rmap: idx → oid
8	Smap	:= cand.reverse.mark()	# Rmap: idx → oid
9	Rage	:= Rmap.join(Ra)	# Rmap: idx → oid
10	Sage	:= cand.reverse.mark()	# Rmap: idx → oid
11	delta	:= [abs](Rage[-]Sage)	# delta: idx → int
12	best	:= delta.minRow(count(S))	# best: idx → int
13	pairs	:= best.duplicate()	# pairs:idx → idx
14	v1	:= Rmap.reverse.join(pairs)	# v1:oid → idx
15	answer	:= v1.join(Smap)	# answer:oid → oid
16	a1	:= R.reverse.join(answer)	# a1:name → oid
17	a2	:= a1.join(S)	# a2:name → name

Lines 0,1 again identifies the males and females and line 2,3 obtain their age attribute. The left and right candidates are obtained with the bounded theta join in lines 4 and 5. They are combined to form the candidates in line 6. Line 7 and 8 construct the cross-table representation by introducing the cell identifier using a builtin routine *number()*. The remainder of the algorithm is identical to the previous version.

The prime effect of this preparatory step is that the storage cost is significantly reduced. Instead of $|R| * |S|$ candidates there are only $2 * |R|$ candidates in the main part of the algorithm. The processing cost is reduced accordingly to $2 * |R| * 5 + 2 * |R|$.

6.4 Query Optimization Schemes

In this section we indicate the opportunities to exploit the fitness functions at query optimization time and accelerator data structures to support complex cases.

6.4.1 Mathematical Query Optimization

The fitness join expressions indicates a route for further exploration. For, when we compose functional expression over the operator set {+,-,*,/,log,exp, sqrt}, a single attribute, and constants, we maintain monoticity of the result. This means that the *bounded-theta-join* solution presented in Section 7 can readily be applied.

The case considered for fitness function f_1 illustrates how a query optimizer can deal with discontinuous single attribute functions. It should break the underlying domains into pieces, such that within each piece the bounded-theta-join becomes applicable. Actually the optimizer should produce the first derivative in each point and determine the domain ranges where it can use the btj method. This analysis is relatively straightforward for the operator set considered.

Beyond these cases, a symbolic analysis of the fitness function could explore the following routes:

- *Sandwich method.* A fitness expression can be approximated using bounding functions that may be easier to compute, i.e.

$$\check{f}(r,s) \leq f(r,s) \leq \hat{f}(r,s)$$

- *Transformation method.* If the fitness expression can be subjected to an affine transformation (rotation, translation) with clear boundaries, we could solve the function in a fraction of the domain and use a lookup table to speed up evaluation. For example,

$$f(r,s) = f(r+\delta, s+\delta)$$

- *Candidate method.* If the operands involved 'identical objects' with respect to the attributes of concern in the fitness expression, it suffice to use one in solving the fitness problem.

The sandwich method is of interest if the underlying domain deals with sequence data. For example, in a stock exchange time series the band in which the stock price function fluctuates may be determined with a single scan over the underlying table. It provides a crude, but effective approximation of the function to filter candidates by looking at $\hat{f}(r,s)$ (or $\check{f}(r,s)$) first. It is even possible to break the sequence into bull and bear market segments before fitness expressions are calculated.

The transformation method could be used in those cases where the underlying object incurs repetition. For example, in a fractal encoding of an image the fitness expression needs to be solved once for each fractal component. Subsequent join results can be found by applying the fractal transformation to the operands.

The candidate method calls for a projection over the attributes of interest and to keep the identity of one record r (or s) to partake in the fitness test. We then know that the result obtained holds for all other members in the same group.

The price paid for such optimizations is to include a mathematics reasoning system as part of the query optimizer. For elementary analysis this is not more difficult than finding common subexpressions. Finding appropriate sandwich functions could be encoded as static optimization rules for the classes of operators considered.

6.4.2 Data structures for fitness joins

Once we enter the realm of multi-attribute expressions, e.g. the distance fitness (f_4), or set-based expressions (f_2) the bounded-theta-join solution should be generalized to multiple dimensions. For example, the bounded-theta-join with distance 1 over a spatial domain equates with the nearest neighbor operation often deployed.

The set-based expressions can benefit from the partial order of the subset relation to reduce the number of candidates. Searching for the sets with maximum overlap, as defined by f_2, can be answered using the *po*-tree induced from this partial order. The bounded theta join can then use the order to speed-up matching as follows. To find pair (r, s) maximizing $f_2(r, s)$ we start from the set closest to the repertoire of r. This starting point can be found in a single walk through the *po-tree*. A bounded number of sets with maximum overlap could then be determined by traversing the *po*-tree from this point in sub and super-set direction. Each newly found set should, if the bound is not reached, also be used as a new start point for traversal.

6.5 Evaluation

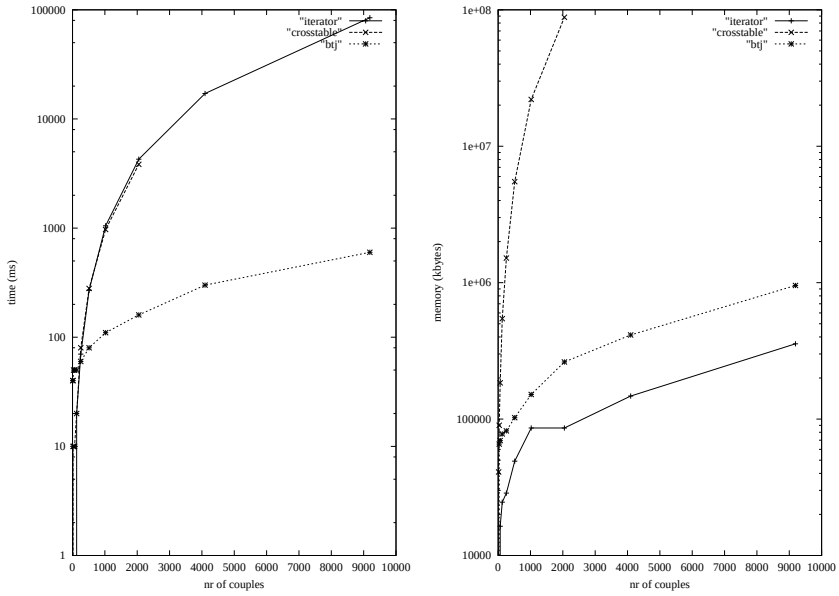
To assess the impact of the techniques described, we set out for a first experimental validation of the bounded-theta-join. For this purpose, we have implemented the ballroom contest in Monet. This system provides sufficient hooks to extend the algebraic engine with new operators and search accelerators. The implementation involved coding the naive, cross-table, and bounded-theta-join approach using a C-module. Moreover, we added a module for the *po*-tree over sets.

6.5.1 Dance partners by age

To evaluate the performance results, we faked a large ballroom contest with a party of trolls, dwarfs and elfs to obtain a sizeable collection of different ages. The database was initialized with a equal number of males and females. We experimented with subsets in the range of 8 to 9192 dance couples. The dance couples are formed using query Q_1.

This choice assumes that traditional optimization steps, such as reducing the number of individual candidates as quickly as possible using attribute based selection, has already been performed. The second optimization step assumed, is to solve the fitness join for just one distinctive element in each group. Therefore, we project the operands over the attributes mentioned in the fitness join expression, keeping one person for the evaluation. After the fitness join has been evaluated, the equivalent persons can be joined back into the result to find for each male the group of females of the same minimal age.

The performance of the three algorithms for query Q_1 is shown in Figure 6.5(a). The incremental memory requirements (Kb) of the algorithms are shown in Figure 6.5(b).



(a) (b)

Figure 6.5: Execution Times and Memory Requirements.

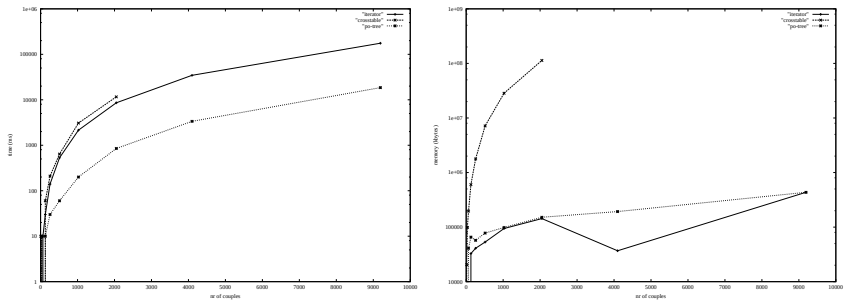
The experiments confirmed the expected behavior. The iterative solution is relatively fast for small dance parties (up to 64). Thereafter, the quadratic complexity of the algorithm results in poor performance. The Cross table solution performs even worse, because it consumes large amounts of memory, but also its performance follows the space consumption. Above 2048 dance couples the memory requirements even exceeded the available resources. The bounded-theta-join stands out as a winner, despite the overhead incurred in construction of an accelerator on the fly. The investment is quickly earned back in improved response time. The bounded-theta-join only needs to construct an accelerator when none of the two given tables is sorted or already contains an index structure. If the accelerators are available up front, the performance is even much better.

6.5.2 Dance partners by repertoire match

In a second dance contest problem couples were formed using a query Q_2: find the partner of opposite gender and with best overlap in dance repertoire. Again the operands are trimmed using the rules applied above. In the next step the search for the best overlapping dance repertoire is made. using

function f_2 to compare partners abilities.

Query Q_2 was solved using three algorithms again; iterator, cross table and *po*-tree. All three use bit sets to represent the repertoire. Their performance is shown in figure 6.6(a). The incremental memory requirements (Kb) of the algorithms are shown in figure 6.6(b).



(a) (b)

Figure 6.6: Execution Times and Memory Requirements.

Again these experiments confirm the expected behavior. The iterator solution is optimal only for very small dance parties. Even smaller than for query Q_1. This stems from the more expensive f_2 for finding the set overlap. The cross table suffers even more from memory consumption, because more intermediate results are needed to find the set overlap. The *po*-tree is a clear winner, especially for larger dance parties. The investments of building it on the fly are quickly earned back. The tree size is only related to the number of dances involved; it is unrelated to the number of dancers. Therefor the cost of building the po-tree becomes even a relatively smaller investment for larger dance parties. Even better performance can be obtained when the *po*-tree is built up front.

6.6 Conclusion

In this chapter we have introduced the class of *fitness join*s, which appear regularly as building blocks in advanced database applications. They extend the traditional equi-, theta- and set-joins by a mathematical complex formula in the join condition combined with a selection from a (ranked) group.

We have shown that this class can be handled efficiently for relatively simple fitness functions using moderate extensions to the relational algebra. A method to manipulate cross-tables or sparse matrices provides the hook to represent results from sparsely populated cross-products. Furthermore, the *bounded theta-join* appears a valuable addition to the standard repertoire of algebraic operators and it can be implemented using traditional optimization

techniques. It extends early work on theta-joins [34] by uncovering the real handle to tackle the problem efficiently. Namely, judicious use of the monoticity properties of compound mathematical functions combined with a variation of the theta-join.

Further optimization by exploring the mathematical properties of the fitness expressions have been indicated. It is an open research area, while most interactions with a database system come from applications where mathematical analysis is a integral activity. The optimizer framework of Monet is extended along this line and we plan to isolate and include the primitives for further experimentation in its algebra. Better support for fitness-based ranking remains on our wish list as well.

Chapter 7

Metric Indexing

[1]

7.1 Introduction

An emerging class of database applications heavily relies on spatial information, e.g. geographical and multi-media information systems. The majority of queries involves exploration of the spatial information, such as finding elements in a given region. Furthermore, join queries over a spatial domain often boil down to calculation of a distance function to locate point pairs within close vicinity of one-another. They are conventionally implemented using a spatial index (like R-tree) to filter candidate pairs.

Another important field for spatial joins is Multi-Media. In these fields the queries passed are not always exact but often fuzzy. Fuzzy queries need different index structures for performance improvement.

Since these application areas are still very new the queries will change rapidly. So reuse of previous intermediate results which showed to be very profitable in traditional systems will only result in loss of storage. Because of the low reuse possibility index structures should be generated on the fly.

In this chapter we introduce a cheap and fast index structure to speed up such *distance joins*. In particular, we demonstrate that an index structure based on the distance metric is both cheap to construct and competitive in performance.

Because the index structure can be built very fast and its storage overhead is minimal, it is it a good candidate for on the fly construction. and can hidden behind Monet operations, as part of a query processing plan. Since the index structure captures the metric information to answer distance joins directly, it reduces the number of times the expensive distance function is evaluated considerably.

[1] Metric Indexing is developed together with W. Quak

The key property exploited is that distance joins involve a well-defined, but expensive, (Euclidean) distance function. Moreover, these functions satisfy the mathematical triangular inequality property, i.e. the distance between two points is always smaller or equal than the distance between the points considered and a third point. Based on the triangular inequality it is possible to build an index structure to speed up several query classes, such as:

- **point-match**, for each A find the points B positioned at the same location (e.g. A.pos=B.pos).
- k**-nearest neighbor**, for each A find the k nearest elements B under the distance function.
- δ**-search**, find all B points within delta δ range away from A.
- δ**-join**, find all A,B point pairs within delta δ range.

In this chapter we ignore the point-match and k-nearest neighbor queries, because they are special cases of the δ-search. The point-match can be rephrased as a delta-join with $\delta = 0$, while the k-nearest neighbor can be implemented using a binary traversal over delta values until k values have been obtained. The δ-search is part of the inner-loop of the δ-join algorithm.

The remainder of this chapter is organized as follows. In Section 7.2 we review index structures that deal with spatial joins in high dimensions. Section 7.3 explains the use of the triangular inequality in a multi-dimensional space. In Section 7.4 the metric index data structure and algorithms are explained. Section 7.5 provides a mathematical estimation of the effectiveness of the new index structure. Section 7.6 reports on experimentations to validate the approach taken. Finally, in section 7.7 we draw our conclusions and pointers for future research.

7.2 Index Structures for Spatial Joins

Most previous work on searching in multi-dimensional spaces is concentrated on low dimensional data-structures, such as R-tree [52, 4] and and K-D-trees [5]. These structures can be extended to higher dimensions, but this results in two problems. The performance degrades, because as the dimension increases the querying cost often increases exponentially; it is called the *dimensionality curse*. Consequently, the index structures deployed become less effective as a pre-filter for selections and join operations.

This curse also stems from the metric effects in higher dimensions, which leads to a clustering of objects at the 'edge' of the n-dimensional space, while all points theoretically become placed at 'equal' distance of any other point in this space. This holds under the assumption of homogene distributed objects.

The prime route explored in literature to tackle the former deals with the scalability limitations of most data structures. Examples considered here are the X-tree, SS-tree and TV-tree:

The X-tree [6] tackles the dimensionality problem by observing that the performance degradation in the R-tree based index structures is mainly due to the high overlap between the nodes in the R-tree itself. This overlap causes an increased number of nodes to be visited when querying the R-tree. The X-tree solves this problem by allowing nodes of the X-tree to be bigger than one disk block (the so-called super nodes) if a split node would generate a high overlap. This technique makes an X-tree behave like an R-tree in low dimensions, while in higher dimensions the join behavior converges to that of a nested loop.

Another data structure for indexing high-dimensional vectors is the SS-tree [118]. The SS-tree is an R*-tree based structure using bounding (hyper)balls instead of rectangles. In 2-dimensions bounding circles are more appropriate for performing similarity queries. Furthermore, they store some additional statistical data in the nodes to support various operations used in image retrieval.

The TV-tree [69] reduces the size of internal nodes by projecting the data in internal nodes to a lower dimension. By using different projections in different parts of the tree, all parts of the original vectors are used. If some dimensions of the input data are more important than others a big speedup can be gained. This is done by first projecting the data to these important dimensions. It is unclear how well the TV-tree performs when all dimensions are equally important.

Despite the progress reported in reducing the storage/processing cost in moving to higher dimensional indices, these data structures are focussed on the spatial organization. We focus on point and region-based retrieval operations. Our key operation, δ-joins, requires a relaxation of the spatial joins supported by several systems. It behaves more like a theta-join within a spatial context. The role of the index structures in this case are primarily aimed at reducing the number of candidates for consideration.

7.3 Triangular Inequality

As mentioned before, distance functions play an important role in real life applications, e.g. GIS, CAD/CAM, Image Retrieval and multi-media applications. For example in GIS and CAD/CAM applications require spatial queries, like "find me the closest restaurant to a given location" and "find objects that are so closely placed that they generate electro-static interference". Many content based text, image and multi-media applications use similarity based queries, like "find similar colored objects". To illustrate, the functions encountered in the areas considered are:

1. The Great Circle Distance is used in GIS to calculate the 'as the crow flies' distance between two places in the world.

2. A distance function amongst customer profiles (time series) in the datamining area.

3. The Weighted Euclidean Distance over a vector space:

$$d(V,W) = (V-W)^T A(V-W) = \sum_i \sum_j A_{i,j}(V_i - W_i)(V_j - W_j)$$

4. Histogram Intersection

$$d(V,W) = 1 - \frac{\sum_i min(V_i, W_i)}{\sum_i (V_i)}$$

Most distance functions are expensive to calculate and, because they are called repeatedly, they contribute considerably to the total querying cost. Our index structure aims to reduce the number of calls to these functions in a naive implementation of the δ-join. This is achieved by using the metric properties. The key to the solution proposed relies on the *triangular inequality* relationship.

The mathematical properties for a metric, $|xy|$, where x, y and z are multi-dimensional vectors, are:

- **Positivity** $|xy| \geq 0 \wedge |xx| = 0$
- **Symmetry** $|xy| = |yx|$
- **Triangular inequality** $|xy| \leq |xz| + |zy|$

It enables to set-up an index that is both effective (on low selectivities) and fast to construct.

7.3.1 Using the Triangular Inequality

The naive implementation of the envisioned δ-join is a nested loop. For each pair considered the distance should be calculated. Since this results in many expensive calculations it becomes mandatory to reduce candidate pairs to reuse results being calculated. This is achieved by taking a reference point (or set of reference points) and to calculate the distances between the reference point and each vector in a join operand first.

Now consider a query looking for all points within distance δ from a query point q, i.e. all points p with $|pq| < \delta$. The metric properties enables reuse of distances calculated between p and reference point r.

Assume that for points in our space we know its distance to a reference point r. Then the query $|pq| < \delta$ could use this information as follows. The

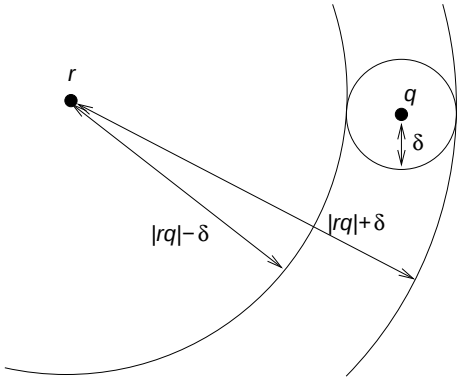


Figure 7.1: Circles

triangular inequality provides us with $|rq| \leq |rp| + |pq|$. So we have an upper bound $|rq| \leq |rp| + \delta$. Since also $|pq| \leq |pr| + |rq|$ holds, we also know that $|pr| - \delta \leq |rq|$ holds. Using the metric symmetry we can use $|rp|$ directly, else we could also store $|pr|$. Figure 7.1 shows the use of the triangular inequality in the 2 dimensional case.

If the query point is already close to the reference point, we can remove all possible points with large distance from r from consideration. They typically fall behind the horizon of $2 * \delta$. Alternatively, if the query point is far away from the reference point we can remove all possible points which are close to the reference point r.

7.4 Metric index structure

In this section we will explain how the metric index structure is built and how it is used in the select and join algorithms.

7.4.1 The reference points

There are various ways to select a reference point: random, center of gravity, or middle point. A randomly selected reference point would be the prime choice considered when it is a priori known that the space is large. The center of gravity would be of interest if the space contains several clusters. Then each cluster leads to a reference point. Unfortunately, cluster detection and, subsequently, the reference point is expensive to calculate. Instead we use the heuristic to randomly select reference points.

Given a reference point, we calculate its distance to each point in the table. The points are subsequently sorted by distance using a tree-like structure. This will speed-up searching later for elements at a given distance from the reference point.

7.4.2 The optimized distance select

Selection using the metric index follows the traditional route of pre-filtering; the index is used to reduce the candidates to consider to solve the δ-select. The following pseudo code routine explains how this can be done.

δ-select (ps, q, δ){
    select p **from**
        select p **from** ps
        where $|pr| - \delta < |rq| < |pr| + \delta$
    where $|pq| < \delta$
}

The inner **select** selects all points p form the point set ps where the distance of the reference point r to the query point q is between $|pr| - \delta$ and $|pr| + \delta$. This identifies candidates in a cylinder around the reference point. It can be solved with a single lookup because we know the distance to r. The outer **select** filters this set by checking for the actual distance between p and q.

Similarly, we can use the metric index to speed-up the δ-join using the generalized triangular inequality.

$$|pq| \leq |pr_0| + |r_0r_1| + \ldots + |r_nq|$$

δ-join (ps, qs, δ){
    select p, q **from**
        select p, q **from** ps, qs
        where $||pr_p| - |r_pr_q| - \delta| < |r_qq|$
            $< |pr_p| + |r_pr_q| + \delta$
    where $|pq| < \delta$
}

The algorithm can be simplified when both tables use the same reference point.

The use of multiple reference points merely leads to a few **and** parts in the query. The following code shows the simplified version for a three reference points query.

δ-join (ps, qs, δ){
    select p, q **from**
        select p, q **from** ps, qs
        where $||pr_0| - \delta| < |r_0q| < |pr_0| + \delta$
        and $||pr_1| - \delta| < |r_1q| < |pr_1| + \delta$

and $||pr_2| - \delta| < |r_2q| < |pr_2| + \delta$
where $|pq| < \delta$
}

7.5 Effectiveness of the metric index

In this section we give an estimate on the expected hit ratio of the candidates selected, i.e. "Is the metric index a good filter?". This estimate is only given for the case where one reference point is chosen. Moreover, all estimates are based on the (simplifying) assumption that the vectors are uniformly distributed in space; this means that the size of a query result is linear with the volume covered by the query. A formula for the volume of a hyperball with dimension d and volume r, is denoted $V_{r,d}$. First, we give a formula for balls with $r = 1$. This formula is defined recursively where the volume in one dimension is expressed in volumes of the lower dimensions. The volumes in dimensions 1 and 2 are given:

$$V_{1,1} = 2, \qquad V_{1,2} = \pi, \qquad V_{1,d} = \frac{2\pi}{d} V_{1,d-2}$$

In fact $V_{1,1}$ is the length of the interval $[-1, 1]$ and $V_{1,2}$ is the area of the circle with radius 1. The formula for balls with given r becomes:

$$V_{r,d} = r^d \; V_{1,d}$$

The next step is to estimate the size of the query result and the size of the filter set of the range query with range δ around query point q with reference point r. The two dimensional case of this query is depicted in Figure 7.1. Due to the uniform distribution the size of the query result is equivalent to the volume of a ball with radius δ around point q. The candidate points (points which pass the filter step) are all the points in the (hyper)disc of all points with distance between $|rq| - \delta$ and $|rq| + \delta$. Now the effectiveness of the filter is the number of hits divided by the number of candidates.

$$\frac{\#\text{hits}}{\#\text{candidates}} = \frac{V_{\delta,d}}{V_{|rq|+\delta,d} - V_{|rq|-\delta,d}} =$$

$$\frac{\delta^d}{(|rq| + \delta)^d - (|rq| - \delta)^d}$$

In Figure 7.2 the effectiveness of a few selectivity values is shown. In this Figure we keep the answer set constant by increasing δ for higher dimensions. As can be seen, the filter effectiveness degrades for high dimensions. But there are also some aspects to take into account to make life bearable in practice.

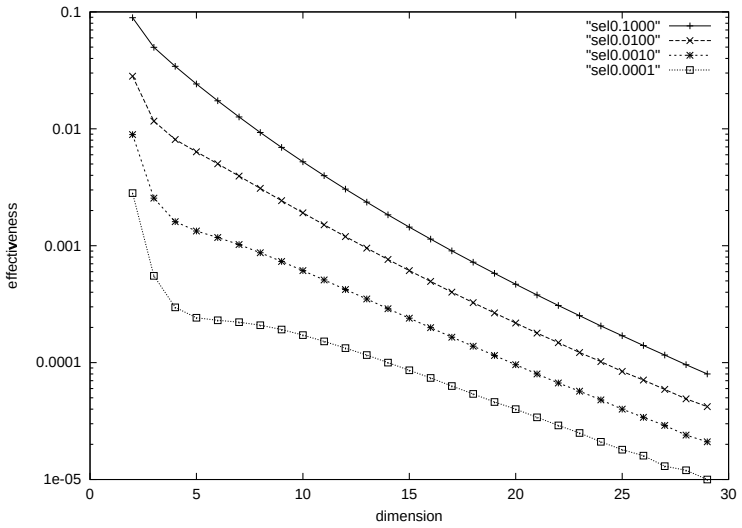


Figure 7.2: Effectiveness

- In this analysis only one reference point is taken into account. Improved gain may come when more reference points are used, because they break the cylinders into pieces. This analysis will be done experimentally.

- The effectiveness of the filter is still good for small query results. A situation likely to occur in large multi-dimensional applications.

- The uniform distribution assumption is not likely to hold in practice. Clustered spaces will lead to more opportunities to filter out irrelevant points.

To assess the impact of the choice of the reference point on the effectiveness, we calculated the expected performance while varying the distance $|rq|$ between 0.0 and 0.1. In Figure 7.3 we plot the effectiveness for various dimensions while fixing δ on 0.001.

Although the analysis in this section re-enforced the existence of the dimensionality curse when dealing with distance joins in high-dimensional spaces, it also indicates good behavior for low selectivity values and a small δ. We conjuncture that it will further improve with sparsely (skewed) populated in real-life applications.

One way to improve the filter effectiveness of the metric index is the use multiple reference points. For two reference point, the filtering step becomes a windowing query on points in $\mathbb{R}^2$. See Figures 7.4 and 7.5. Adding more reference points yields windowing queries in n-dimensional space. In fact this leads to a filter step which converts an n-dimensional range query into

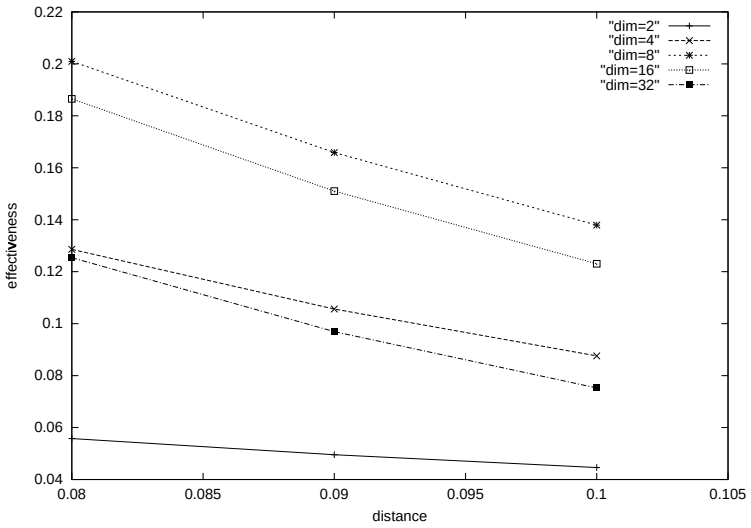


Figure 7.3: Impact of location reference point

a windowing query of any dimension (depending on the number of reference points).

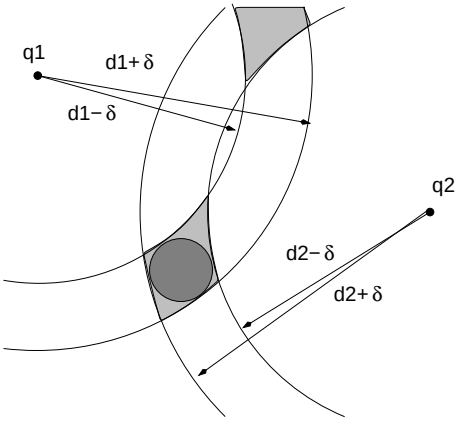


Figure 7.4: Distance Query

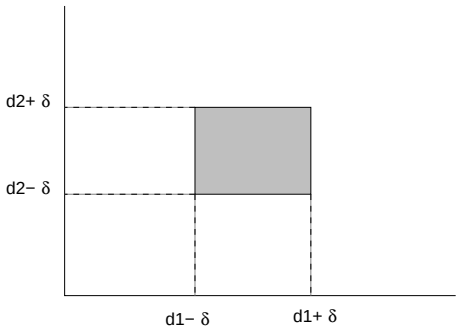


Figure 7.5: Range Query

7.6 Experimentation

To assess the performance of the metric index in a real setting, we have extended the Monet [13] system with a software module for δ-joins, δ-select, and a metric index. Subsequently, we conducted experiments on data sets generated using a standard pseudo-random number generator. All vector

fields domains are [0..1).

The first experiment conducted was geared to get a handle on the cost of the distance function. Therefore, we measured the execution time of a naive implementation –with a simple loop– of the distance select. The results are shown in Figure 7.6. It shows the execution time of a distance select for databases sizes ranging from 10 to 100K with vectors of dimensions 2,4,8,16,32 and 64. All selects were done with an equal distance of 0.1. So only very close points were retrieved.

The cost of this naive loop could be invested in construction of a metric index. Once it is available, it can be used as a pre-filter. Figure 7.7 shows the results against the same query using the index structure. The benefit from the index is evident. For the low selectivity considered it leads to an overall performance improvement. Since the investment in the index structure are already recovered with 2 δ-selects.

For the δ-join a similar experiment was conducted. Figures 7.8 and 7.9 show the results of the naive nestedloop- and metric index based implementations. The cost of metric index construction is neglectable compared to the gain. Again the benefit of the index is evident.

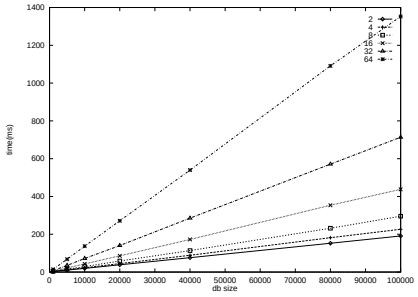


Figure 7.6: Naive δ-search

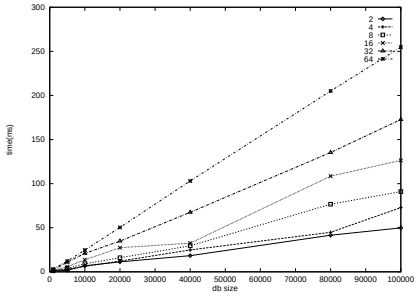


Figure 7.7: δ-search using Metric Index

To assess the degradation caused by increasing the result size we conducted an experiment with fixed database size of 100K and with dimensions 2 to 64, but with increasing query selectivity ranging from 0.01% to 1.0%. Figure 7.10 shows the execution times of the distance selects using the index structure. It clearly shows the reduced usability of the index structure for larger answer sets. Only for low dimensions the index structure seems effective. This stems from the uniform generated data.

To show that the index structure is a cheap alternative for spatial queries in low dimensions we also compared our method with the R-tree data structure. Because our current version of the R-tree only works on two-dimension vectors, this test is only run on two dimensional data. The results of this

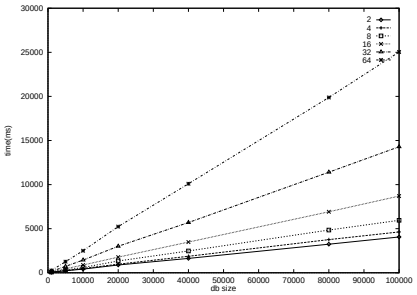


Figure 7.8: Naive δ-join

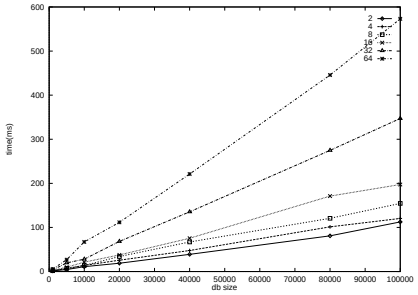


Figure 7.9: δ-join using Metric Index

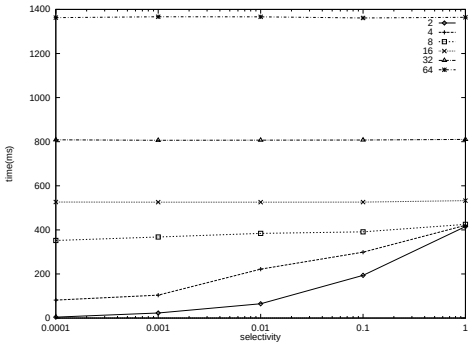


Figure 7.10: increasing selectivity

experiment are show in Figures 7.11 and 7.12. They show the construction and execution times of the δ-join for naive (nor), optimized (opt), optimized with 2 reference points (opt2) and Rtree (rtree). From the figure we can conclude that the metric index with two reference points is overall better and that for relative low selectivity also the single reference point performs well.

The last experiment was conducted to investigate the improved selectivity of the index structure when using multiple reference points. We experimented with 1 up to 16 reference points. Three synthetic data sets were used. We generated one using a uniform distribution, two using uniformly distributed clusters, where the clusters internal distribution was zip-f or gauss around the cluster center.

We measured the ratio between the number of points selected based on the index only i.e. without the real distance post filter, and the actual result

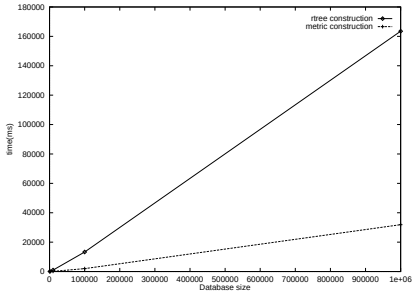


Figure 7.11: Construction Cost

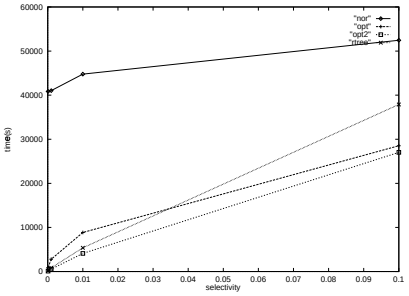


Figure 7.12: Two Dimensions

size. Figure 7.6 shows the results for the various data sets. It clearly shows that multiple reference points in the clusters with zip-f internal distributions will clearly not improve. This was to be expected since most of the points will be close around the cluster centers. In case of the gauss internal distribution multiple reference points improve the selectivity enormously. We see a similar effect for the uniform data.

7.7 Conclusions

In this chapter we presented a cheap index structure to improve the query performance of joins involving a distance metric. This index structure works on any distance metric, as long as it obeys the triangular inequality. There is no need for a full metric. We showed that the index structure is profitable in higher dimensions for small selectivities.

Several areas require further investigation. First, our assumption of uniform distribution of points in the space leads to a worst-case behavior, especially in high dimensions. All points appear at the border of the space and are equally spaced. We conjuncture that data obtained from real-life applications are extremely sparse and that clustering of points (the focus of the query) lead to good performance for acceptable ranges of selectivity.

Second, the implementation of the n-dimensional R-tree in Monet should be finished to balance the results obtained so far. We conjuncture that the effects of the dimensional curse for such data structures are worse than those experienced in our metric index. Experiments in both directions are under way.

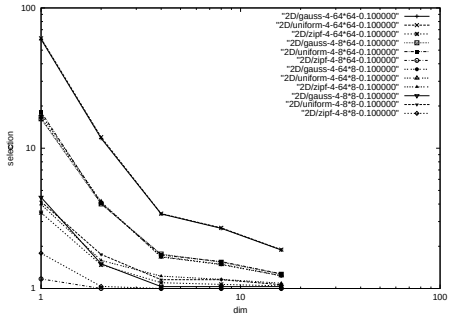


Figure 7.13: 4 dimensional data

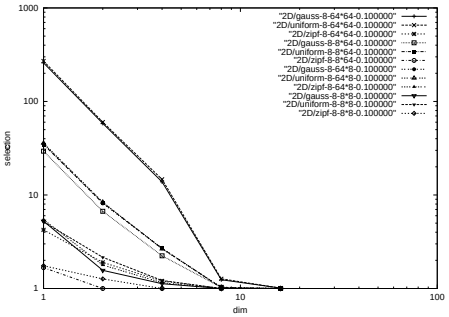


Figure 7.14: 8 dimensional data

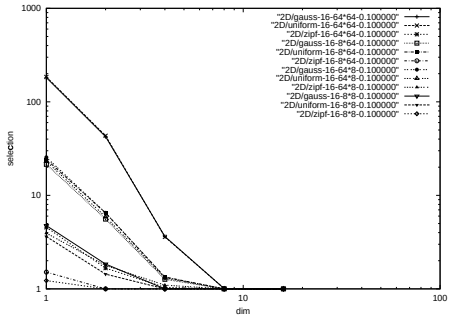


Figure 7.15: 16 dimensional data

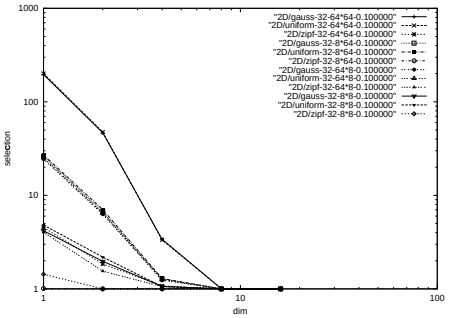


Figure 7.16: 32 dimensional data

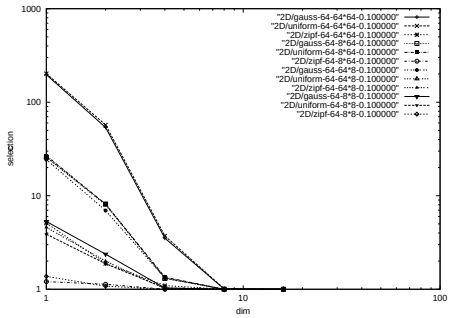


Figure 7.17: 64 dimensional data

Chapter 8

Summary

The objective of this thesis was to design an architecture for image database systems. In this quest we explored many techniques effective in both image retrieval and image analyses systems. The exploration lead to several refined objectives.

The first refined objective is how to incorporate images and image operations in an extensible DBMS ? The extensible DBMS was Monet. In chapter 3 we showed a mapping of images to BATs, i.e. binary tables. We indicated how a default implementation of the image algebra operations could be readily achieved. This also proved the completeness of our approach. Using this representation we showed many possible optimizations to be used by the query optimizer to find better query plans.

It convinced us that pixel-set based image processing in a DBMS context is a viable alternative against the image (C++) data structure approach, the dominate approach taken in image analyses domain. It permits one to focus on effectiveness and lets the query optimizer take care of the efficient evaluation.

The second refinement of the global objective dealt with queries, i.e. how should an image database system support image retrieval queries ?

In chapter 4 we introduced an algebraic framework to express queries on images, pixels, regions, segments and objects. We showed 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.

In Section 4.2.1 we showed our prototype image retrieval system and explained the Multi-level signature image description. The multi-level signature approach shows that an image algebra should accommendate multiple image descriptions and requires multiple index-structures.

In Section 4.3 we have introduced the necessary data structures and op-

erators to build an image database system aimed at supporting embedded image querying. We have experimentally demonstrated that a bottom-up index construction outperforms a top-down approach terms of storage requirements and performance.

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.

The third refinement dealt with image analysis, could an image database system be used for image analyses tasks ?

In chapter 5 we showed that an image database could be used profitably to support image analyses researchers. In a case study it has been shown that an extensible DBMS can be efficiently used to tackle the line-clustering problem. The overhead of the conversion between database structures and application structures is not a dominant factor. Moreover, there exists a small algebraic extension to the DB core functionality, which enables us to tackle the line clustering problem. The performance is promising compared to the original solution written in C++.

Although we proved the effectiveness of using a database system for image analyses, the image community is far from taking up this route. The main reason for this is a mentality issue. It is hard to change a researchers approach which has been used for over a decade. In due time we expect that the object-at-a-time approach, as dictated by the imperative languages like C++, is replaced with a set-based approach.

The next refinement to the global objective dealt with similarity queries, how to support similarity queries ?

In chapter 6 we have presented and analyzed the class of *fitness join*s, which appear regularly as building blocks in advanced database applications. They differ from the traditional equi-, theta- and set-joins by a mathematical complex formula in the join condition combined with a selection from a group.

We have shown that this class can be handled efficiently for relatively simple fitness functions using moderate extensions to the algebra. In particular, the *bounded theta-join* appears a valuable addition to the standard repertoire and can be implemented using traditional optimization techniques. It extends early work on theta-joins [34] by uncovering the real handle to tackle the problem efficiently. Namely, judicious use of the monoticity properties of compound mathematical functions combined with a variation of the theta-join.

Further optimization, along the line of exploring the mathematical properties of the fitness expressions have been indicated. Its scope has been barely scratched upon and we foresee much better support of advanced applications when their mathematical properties are properly accessed by an optimizer. The optimizer framework of Monet is extended to cope with the

information presented and we plan to isolate and include the primitives for further experimentation in its algebra.

Derived from the objective to deal with similarity queries was the objective to improve the performance of similarity queries. In chapter 8 we presented a cheap index structure to improve the query performance of joins involving a distance metric. This index structure works on any distance metric, as long as it obeys the triangular inequality. There is no need for a full metric. We showed that the index structure is profitable in higher dimensions for small selectivities.

8.1 General Research Directions

This thesis opens new research directions. The first direction, there is a need for more investigation in *Region-based Querying*. We defined regions as the basic building blocks for image descriptions. The regions describe small parts of the image by associating region features. The query implications of such regions needs more investigation.

The multiple-features associated with regions will form a single high dimensional space. All problems associated with the *high dimentionality curse* are valid. Therefore, all regions will be evenly distributed over this space.

The predicate query model as available in current database management systems proved inadequate for image retrieval queries. More advanced query models such as *Proximity-based Querying* and *Relevance feedback* are considered good alternatives and need more research.

Opposed to this we could question such a fuzzy query evaluation model. Is the current accepted query model, i.e. image retrieval based on similarity, the proper solution. Is such a Fuzzy model really needed? Maybe we could guide the user to better understand the image descriptions used to describe the image content. A better understanding of the image descriptions could lead to precise image queries which could be handled using the predicate query model.

The main problem of image retrieval systems is that the results obtained through an index using similarity measures are often only understood by the the image analyses experts. For naive users, it is often difficult to understand why the answer to a query image showing e.g. a sunset should return nice images about the African savanne. This is typically an artifact of non-precise query formulation and weak indexing structure.

A way out of this problem is to give the user more insight in the features used to solve the query against the image database. This insight is given by specifying the query and showing the results in terms of the indexed image features directly.

The usage of image databases by image analyzing researchers will give

the database researchers more input on there query behavior and requirements. This input could be used to improve the performance of image analyzing applications. Once the image analyzing community has accepted set based processing they can again focus on the image analyzing problems and let the database take care of the performance issues.

Acknowledgements

First I like to thank both Arnold Smeulders and Martin Kersten for giving me the change to do this research. Martin, your enthousiasm helped me through the difficult times of writing text. As you know I am easily distracted, there is always something more interesting, but you kept me focused. Now this is finished we can hopefully both enjoy our hobby again (hacking).

As Monet user I like to thank Peter for his bug fixes. Wilko, thanks for the first tutorial on Monet usage and for the bug fixes in the gis modules. Peter, Martin and Wilko thanks, I am proud to be part of the Monet team.

During my research I had lots of room mates, they made daily live a lot more fun. Thanks Dennis, I now know what you were going through. Silvia and Audrey, lucky me having all the girls in my room. The list goes on with Benno, Marcos, Jeroen, Menzo, Stefan and Arjen.

Verder wil ik mijn vrienden en familie bedanken voor hun steun en belangstelling tijdens de afgelopen vier jaar.

Pa en Ma bedankt. Bedankt voor de steun die jullie mij gaven ondanks dat het onderzoek voor jullie verre van begrijpelijk is. Jullie geloof in mij heeft de doorslag geven om deze richting te kiezen. En nu kan ik eindelijk jullie vraag, "is je boekje al af", positief beantwoorden.

Lilian, heel graag wil ik je bedanken voor jouw geduld en steun de afgelopen jaren. Het leven met een 'computer-verslaafde' zoals jij dat noemt, is niet altijd even gemakelijk. Maar het liefst bedank ik je voor het afgelopen jaar. Het was zwaar (vooral voor jou), maar wat zij we trots op onze ...!

Bibliography

[1] Rosenthal A. and U.S. Chakravarthy. Anatomy of a modular multiple query optimizer. In *Int. Conf. on Very Lage Databases (VLDB)*, pages 230–239, 1988.

[2] P. M. G. Apers, C. A. van den Berg, J. Flokstra, P. W. P. J. Grefen, M. L. Kersten, and A. N. Wilschut. PRISMA/DB: A parallel main memory relational DBMS. *IEEE Trans. on Knowledge and Data Eng.*, 4(6):541, December 1992.

[3] M. Arya, C W. Cody, Faloutsos, J. Richardson, and A. Toga. The QBISM Medical Image Database. Multimedia Database Systems, pages 79–97, 1996.

[4] N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the SIGMOD Conference*, pages 322–331, 1990.

[5] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Comm. ACM*, 18:509–517, 1975.

[6] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel. The X-tree: An index structure for high-dimensional data. In *Proceedings of the 22nd VLDB Conference*, 1996.

[7] M.W. Blasgen and K.P Eswaran. On the evaluation of queries in relational system. Research report, IBM, 1976.

[8] P. A. Boncz and M. L. Kersten. Monet: An impressionist sketch of an advanced database system. In *Proc. IEEE BIWIT workshop, San Sebastian (Spain)*, july 1995.

[9] P. A. Boncz and M. L. Kersten. Mil primitives for querying a fragmented world. *VLDB Journal 8(2)*, pages 101–119, 1999.

[10] P. A. Boncz, S. Manegold, and M. L. Kersten. Database architecture optimized for the new bottleneck: Memory access. In *VLDB Conference*, pages 54–65, 1999.

[11] P.A. Boncz, T. Rühl, and F. Kwakkel. The drill down benchmark. In *Proc. 24th Int. Conf. Very Large Data Bases, VLDB*, pages 628–632, 24–27 August 1998.

[12] P.A. Boncz, A. Wilschut, and M.L. Kersten. Flattening an object algrebra to provide performance. In *Conf. ICDE'98*, 1998.

[13] Peter A. Boncz, Wilko Quak, and Martin L. Kersten. Monet and its Geographic Extensions: a novel Approach to High Performance GIS Processing. In *EDBT proceedings*, 1996.

[14] J.B. Burns. Extracting straight lines. *IEEE PAMI 8(4)*, pages 425–455, 1986.

[15] J. Canny. A computational approach to edge detection. *IEEE PAMI 8(6)*, pages 679–698, 1986.

[16] C. Carson, M. Thomas, S. Belongie, J. M. Hellerstein, and J. Malik. Blobworld: A system for region-based image indexing and retrieval. In *Third International Conference Visual Information and Information Systems*, pages 509–516, Amsterdam, the Netherlands, 1999.

[17] R. G. G. Cattell. ODMG-93: A standard for object-oriented DBMSs. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 23(2):480–480, June 1994.

[18] N. S. Chang and K. S. Fu. Query-by pictorial example. In *IEEE transactions on Software Engineering 6(6)*, Nov 1980.

[19] S. K. Chang. Pictorial data-base systems. In *IEEE Computer*, Nov 1981.

[20] S. K. Chang, C. W. Yan, D. C. Dimitroff, and T. Arndt. An intelligent image database system. In *IEEE transactions on Software Engineering 14(5)*, May 1988.

[21] J. Chanussot and P. Lambert. Total ordering based on space filling curves for multi-valued morphology. In *Proceedings of the International Symposium on Mathematical Morphology (ISMM'98)*, pages 51–58. Kluwer Academic Publishers, Amsterdam, 1998.

[22] R. Chellappa and R. Bagdazian. Fourier Coding of Image Boundaries. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:102–105, 1984.

[23] Vassilis Christophides, Sophie Cluet, and Guido Moerkotte. Evaluating queries with generalized path expressions. In H. V. Jagadish and

Inderpal Singh Mumick, editors, *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996*, pages 413–422. ACM Press, 1996.

[24] Moerkotte G. Claußen J., Kemper A. and K. Peithner. Optimizing queries with universal quantification in object-oriented and object-relational databases. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 286–295. Morgan Kaufmann, 1997.

[25] E. Clementini, P. Felice Di, and P. Oosterom van. A Small Set of Formal Topological Relationships Suitable for End-user Interaction. In *SSD: Advances in Spatial Databases*. LNCS, Springer-Verlag, 1993.

[26] Computer Associates. *Jasmine ii: The intelligent information infrastructure*.

[27] G. Copeland and S. Khoshafian. A Decomposed Storage Model. In *Proc. ACM SIGMOD Conf.*, page 268, Austin, TX, May 1985.

[28] J. M. Corridoni, A. Del Bimbo, and P. Pala. Image retrieval by color semantics. *ACM Multimedia Systems 7(3)*, pages 175–183, 1999.

[29] Rafiei D. On similarity-based queries for time series data. In *Conf. ICDE'99*, pages 410–417, 1997.

[30] Rafiei D. and A.O. Mendelzon. Similarity-based queries for time series data. In *SIGMOD Conf*, pages 13–25, 1997.

[31] I. Daubechies. *Ten Lectures on Wavelets*. Society for Industrial and Applied Mathematics, 1992.

[32] A.P. de Vries. *Content and multimedia database management systems*. PhD thesis, University of Twente, Enschede, The Netherlands, December 1999.

[33] D. DeWitt, D. Lieuwen, and M. Metha. Pointer-based join techniques for object-oriented databases. In *PDIS*, 1993.

[34] D. DeWitt, J. Naughton, and D. Schneider. An evaluation of non-equijoin algorithms. In *Int. Conf. on Very Large Databases (VLDB) Barcelona, Spain*, pages 443–452, 1991.

[35] M. C. d'Ornellas and N.J.Nes. Image retrieval using linear greyscale granulometries. In *In ASCI conference, Lommel, Belgium*, pages 109–114, 1998.

[36] M. C. d'Ornellas and N.J.Nes. Color image texture indexing. In *In Visual'99 conference, Amsterdam, The Netherlands*, 1999.

[37] M. C. d'Ornellas, R. v.d. Boomgaard, and J. Geusebroek. Morphological algorithms for color images based on a generic-programming approach. In *Proceedings of the Brazilian Conference on Computer Graphics and Image Processing (SIBGRAPI'98)*, pages 323–330, Rio de Janeiro, 1998. IEEE Press.

[38] Boris Shidlovsky Elisa Bertino, Barbara Catania. Towards optimal indexing for segment databases. In *EDBT(Spain)*, pages 39–53, 1998.

[39] A. Etemadi. Robust segmentation of edge data. In *Proc. of the 4th international conference on image processing and its applications*, 1992.

[40] A. Etemadi, J-P. Schmidt, J. Illingworth, and J. Kittler. Low-level grouping of straight line segments. In *Proc. of the British machine vision conference*, 1991.

[41] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz. Efficient and Effective Querying by Image Content. *Intelligent Information Systems 3*, pages 231–262, 1994.

[42] C. Faloutsos and et.al. Efficient and effective querying by image content. *Journal of Intelligent Information Systems*, 3:231–262, 1994.

[43] C. Frankel, M. J. Swain, and V. Athitsos. Webseer: An image search engine for the world wide web. Technical report, University of Chicago, 1996.

[44] H. Freeman. On the encoding of arbitrary geometric configurations. *Transactions on electronic computers*, 10:260–268, jun 1961.

[45] T. Gevers. Color in image search engines. *Multimedia Search*, 2000.

[46] T. Gevers and A. W. M. Smeulders. Evaluating Color and Shape Invariant Image Indexing for Consumer Photography. In *Proc. of the First International Conference on Visual Information Systems*, pages 293–302, 1996.

[47] T. Gevers, A.W.M. Smeulders, and H. Stokman. Photometric invariant region detection. In P. H. Lewis and M. S. Nixon, editors, *Proceedings of The Ninth British Machine Vision Conference*, pages 659–670, 1998.

[48] S W Golomb. Run-Length Encodings. *IEEE Transactions on Information Theory 12(3)*, pages 399–401, july 1966.

[49] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing.* Addison-Wesley, Reading, MA, USA, 3rd edition, 1992.

[50] G. Graefe. Encapsulation of parallelism in the volcano query processing system. In *19 ACM SIGMOD Conf. on the Management of Data, Atlantic City*, May 1990.

[51] R. H. Güting. Gral: An extensible relational database system for geometric applications". In *Proceedings of the 15th Conference on Very Large Databases, Morgan Kaufman pubs. (Los Altos CA), Amsterdam*, August 1989.

[52] A. Guttman. R-trees: a dynamic index structure for spatial searching. *Proc. ACM SIGMOD*, pages 47–57, 1984.

[53] Lu H., Ooi B.-C, and Tan K.-L. Efficient image retrieval by color content. Dept of Information Systems and Computer Science, National University of Singapore, 1994.

[54] M. Holsheimer, M.L. Kersten, and H. Mannila. A perspective on databases and data mining. In *Knowledge Discovery in Database (KDD95), Montreal, (Can.)*, 1995.

[55] The tpc benchmark series.

[56] IBM. *DB2 Universal Server.*

[57] Informix. *The Informix Universal Server.*

[58] Y. Ishikawa, R. Subramanya, and C. Faloutsos. Mindreader: Querying databases through multiple examples. In *Proceedings of the 24th VLDB Conference*, New York, USA, 1998.

[59] C. E. Jacobs, A. Finkelstein, and D. H. Salesin. Fast multiresolution image querying. In *In proceedings of SIGGRAPH 95, Los Angels*, 1995.

[60] Bernd Jähne. *Digital Image Processing: Concepts, Algorithms, and Scientific Applications.* Springer-Verlag, Berlin, Germany, 2 edition, 1993.

[61] Anil K. Jain. *Fundamentals of Digital Image Processing.* Prentice-Hall, Englewood Cliffs, NJ, 1989.

[62] R. Jain. Visual information management—introduction. *Communications of the ACM*, 40(12):31–32, December 1997.

[63] R. Jain, S. Murthy, P. Chen, and S. Chatterjee. Similarity Measures for Image Databases. In *SPIE, Storage and Retrieval for Image and Video Databases III*, pages 58–65, 1995.

[64] A. Jonk and A.W.M. Smeulders. An axiomatic approach to clustering line-segments. In *Proc. of the Third International Conference on Document Analysis and Recognition*, pages 386–389, 1995.

[65] J. Kender. Saturation, hue and normalized color: calculation digitization, and use. Computer science technical report, Carnegie-Melloni University, 1976.

[66] M.L. Kersten and N.J. Nes. Fitness join is the ballroom. In *CWI report*, July 1998.

[67] M. Kitsuregawa, M. Nakayama, and M. Takagi. The effect of bucket size tuning in the dynamic hybrid grace hash join method. In *Int. Conf. on Very Large Databases (VLDB)*, pages 257–266, 1989.

[68] C. Kohl and J. Mundy. The development of the image understanding environment. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 1994.

[69] King-Ip Lin, H. V. Jagadish, and Christos Faloutsos. The TV-tree: An index structure for high-dimensional data. *VLDB Journal*, 3(4):517–542, Januari 1994.

[70] Hong-Chih Liu and M. D Srinath. Corner Detection from Chain-Code. *Pattern Recognition(1-2), 1990*, 23:51–68, 1990.

[71] B. B. Mandelbrot. *The Fractal Geometry of Nature*. W.H. Freeman and Co., New York, rev 1983.

[72] J. McPherson and H. Pirahesh. An overview of extensibility in starburst. *IEEE Database Engineering, vol. 10, no. 2*, pages 32–39, june 1987.

[73] P. Mishra and M.H. Eich. Join processing in relational databases. *ACM Computing Surveys*, 24(1):63–113, 1989.

[74] A. H. Munsell. A color notation. Technical report, Munsell Color Group, 1905.

[75] M. Nabil, A.H.H Ngu, and J. Shepherd. Picture similarity retrieval using the 2d projection interval representation. IEEE Trans. Knowledge and Data Engineering*, Vol 8, nr 4*, pages 533–539, August 1996.

[76] P.F.M. Nacken. A metric on line segments. *IEEE PAMI 15(11)*, pages 1312–1318, 1993.

[77] N.J. Nes, C. van den Berg, and M.L. Kersten. Database support for image retrieval using spatial-color features. In *First International*

Workshop on Image Databases and Multi-Media Search, pages 210–217, Aug 1996.

[78] N.J. Nes, C. van den Berg, and M.L. Kersten. Database support for image retrieval using spatial-color features. In *Image Databases and Multi-Media Search*, pages 293–300. World Scientific, August 1997.

[79] N.J. Nes and M.L. Kersten. Region-based indexing in an image database. In *The International Conference on Imaging Science, Systems, and Technology, Las Vegas*, pages 207–215, June 1997.

[80] N.J. Nes and M.L. Kersten. The acoi algebra: A query algebra for image retrieval systems. In *BNCOD'16, cardiff*, pages 77–88, July 1998.

[81] N.J. Nes, M.L. Kersten, and A. Jonk. Database support for line clustering. In *ASCI conference, Vosse-Meren, Belgium*, pages 277–282, June 1996.

[82] N.J. Nes, W. Quak, and M.L. Kersten. Metric indexing to improve distance joins. In *ASCI conference, Lommel, Belgium*, pages 133–139, June 1998.

[83] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. The grid file: An adaptable, symmetric multi-key file structure. In *Conference of the European Cooperation in Informatics*, pages 236–251, 1981.

[84] Oracle. *Oracle 8 user manual*.

[85] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison Wesley, Reading, 1994.

[86] A. Pentland, B. Moghaddam, and T. Starner. View-based and modular eigenspaces for face recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, July 1994.

[87] A. Pentland, R. W. Picard, and S. Sclaroff. Photobook: Content-based manipulation of image databases. In *SPIE Storage and Retrieval for Image and Video Databases II, No. 2185*, pages 34–47, 1994.

[88] Piatetsky-Shapiro and Frawley, editors. *Data Surveyor: Searching the Nuggets in Parallel*, chapter 4. MIT Press, Menlo Park, California, 1995. M. Holsheimer and M.L. Kersten and A. Siebes.

[89] E. M. Riseman and M. A. Arbib. Computational Techniques in the Visual Segmentation of Static Scenes. *Computer Graphics and Image Processing*, 6(3):221–276, June 1977.

[90] G. X. Ritter. *Image Algebra*. North-Holland, 1993.

[91] Y. Rui, T. S. Huang, and S. F. Chang. Image retrieval past, present and future. In *Symposium MM I*, 1997.

[92] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison Wesley, 1990.

[93] H. S. Sawhney and J. L. Hafner. Efficient color histogram indexing for quadratic form distance functions. *IBM report*, 1993.

[94] Wolfgang Scheufele and Guido Moerkotte. On the complexity of generating optimal plans with cross products. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 12-14, 1997, Tucson, Arizona*, pages 238–248. ACM Press, 1997.

[95] J. Segman and Y. Y. Zeevi. Spherical wavelets and their applications to image representation. *Journal of Visual Communication and Image Representation*, 4(3):263–70, 1993.

[96] T. Seidl and H-P. Kriegel. Efficient user-adaptable similarity search in large multimedia databases. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 500–515. Morgan Kaufmann, 1997.

[97] S.Helmer and G.Moerkotte. Evaluation of main memory join algorithms for joins with subset join predicates. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 386–395. Morgan Kaufmann, 1997.

[98] Chang S.K., Shi Q.Y., and Yan C.W. Iconic indexing by 2d strings. IEEE Trans. Pattern Analysis and Machine Intelligence *, Vol 9, nr 3*, pages 413–428, May 1987.

[99] J. R. Smith and S. Chang. Quad-tree segmentation for texture-based image query. In *ACM Multimedia*, pages 279–286, 1994.

[100] J. R. Smith and S. Chang. Tools and Techniques for Color Image Retrieval. In *SPIE Storage and Retrieval for Image and Video Databases IV, No 2670*, 1996.

[101] J. R. Smith and S. Chang. Tools and Techniques for Color Image Retrieval. In *Storage and Retrieval for Image and Video Databases (SPIE)*, pages 426–437, 1996.

[102] J. R. Smith and S. Chang. Visualseek: A fully automated content-based image query system. In *ACM Multimedia*, pages 87–98, 1996.

[103] M. Stonebraker and G. Kemnitz. The POSTGRES next-generation database management system. *Comm. of the ACM, Special Section on Next-Generation Database Systems*, 34(10):78, October 1991.

[104] G. Strang and T. Nguyen. *Wavelets and Filter Banks*. Wellesley-Cambridge Press, 1996.

[105] Sun Micro Systems. *The Java 2D^{TM} API*.

[106] Swain and Ballard. Color indexing. *International Journal of Computer Vision*, 7, 1991.

[107] H. Talbot, C. Evans, and R. Jones. Complete ordering and multivariate mathematical morphology: Algorithms and applications. In *Proceedings of the International Symposium on Mathematical Morphology (ISMM'98)*, pages 27–34. Kluwer Academic Publishers, Amsterdam, 1998.

[108] S. L. Tanimoto and T. Pavlidis. A Hierarchical Data Structure for Picture Processing. *Computer Graphics and Image Processing*, 4(2):104–119, June 1975.

[109] L. Uhr. Layered recognition cone networks that preprocess, classify, and describe. *IEEE Transactions on Computers*, 21:758–768, 1972.

[110] http://www.wins.uva.nl/research/isis/isisns.html.

[111] P. Valduriez. Join indices. *ACM Trans. on Database Systems*, 12(2):13–18, Dec 1994.

[112] Carel van den Berg, Rein van den Boomgaard, Marcel Worring, Dennis Koelma, and Arnold Smeulders. Horus: Integration of image processing and database paradigms. In *Image Databases and Multi-Media Search*, pages 309–317. World Scientific, August 1997.

[113] C. J. van Rijsbergen. The best-match problem in document retrieval. *Communications of the ACM 17(11)*, pages 648–649, 1974.

[114] M. Vetterli and J. Kovacevic. *Wavelets and Subband Coding*. Prentice Hall, 1995.

[115] J. Z. Wang, G. Wiederhold, O Firschein, and S. X. Wei. Wavelet-based image indexing techniques with partial sketch retrieval capability. In *In proceedings of the Fourth Forum on Research and Technology Advances in Digital Libraries*. IEEE, 1997.

[116] J. Z. Wang, G. Wiederhold, O Firschein, and S. X. Wei. Content-based image indexing and searching using daubechies' wavelets. *International Journal of Digital Libraries 1 (4)*, pages 311–328, 1998.

[117] Aref. W.G., Barbara D., and D. Lopresti. Ink as a First-Class Datatype in Multimedia Databases. Multimedia Database Systems, pages 113–160, 1996.

[118] David A. White and Ramesh Jain. Similarity indexing with the SS-tree. *Proc. 12th IEEE International Conference on Data Engineering*, pages 516–523, 2 1996.

[119] M. A. Windhouwer, A. R. Schmidt, and M. L. Kersten. Acoi: A System for Indexing Multimedia Objects. In *International Workshop on Information Integration and Web-based Applications and Services*, Yogyakarta, Indonesia, November 1999.

[120] Yamamuro, M. K. Kushima, H. Kimoto, H. Akama, S. Konya, J. Nakagawa, K. Mii, N. Taniguchi, and K. Curtis. Exsight-multimedia information retrieval system. In *20th Annual Pacific Telecommunications Conference*, pages 734–739, Honolulu, HI, Jan 1998.