UNIVERSITY OF JOENSUU
COMPUTER SCIENCE
DISSERTATIONS 4

EUGENE AGEENKO

CONTEXT-BASED COMPRESSION OF BINARY IMAGES

ACADEMIC DISSERTATION

To be presented, with the permission of the Faculty of Science of the University of Joensuu, for public criticism in Auditorium M1 of the University, Yliopistokatu 7, Joensuu, on May 5th, 2000 at 12 noon.

UNIVERSITY OF JOENSUU
2000

Julkaisija	Joensuun Yliopisto
Publisher	University of Joensuu
Vaihto	Joensuun yliopiston kirjasto / vaihdot PL 107, 80101 Joensuu puh. 013-251 2677, fax. 013-251 2691 email: Riitta.Porkka@joensuu.fi
Exchanges	Joensuu University Library / Exchanges P.O. Box 107, FIN-80101 Joensuu, Finland tel. +358-13-251 2677, fax. +358-13-251 2691 email: Riitta.Porkka@joensuu.fi
Myynti	Joensuun yliopiston kirjasto / vaihdot PL 107, 80101 Joensuu puh. 013-251 2652, 013-251 2662, fax. 013-251 2691 email: Armi.Lavikainen@joensuu.fi
Sales	Joensuu University Library / Exchanges P.O. Box 107, FIN-80101 Joensuu, Finland tel. +358-13-251 2652, fax. +358-13-251 2691 email: Armi.Lavikainen@joensuu.fi

ISSN 1238-6944
ISBN 951-708-880-9
UDK 681.3.05, 681.327.12
Computing Reviews (1998) Classification: E.4, I.4.2, I.4.3, I.4.6, I.4.10
Yliopistopaino
Joensuu 2000

CONTEXT-BASED COMPRESSION OF BINARY IMAGES

Eugene Ageenko

Department of Computer Science
University of Joensuu
P.O.Box 111, FIN-80101 Joensuu, FINLAND
ageenko@cs.joensuu.fi

University of Joensuu, Computer Science, Dissertations 4
Joensuu, 2000, 110 pages
ISSN 1238-6944, ISBN 951-708-880-9

Keywords: binary image compression, statistical context-based modeling, feature-based filtering, noise removal, document images, image recognition, hybrid raster/vector modeling, digital spatial libraries, spatial access.

Binary images are used in a wide variety of digital imaging applications including Document Imaging, Image Communications, Engineering Document Management, Digital Spatial Libraries and Geographic Information Systems. The storage size of the images has been a major restriction in digital imaging systems for decades. Existing image compression methods do not provide an efficient universal solution, nor satisfy all application-specific requirements, such as on-line retrieval, high quality restoration, spatial access and intellectual post-processing. Better solutions must therefore be developed.

In this thesis, the compression of bi-level document images is studied using statistical context-based modeling and arithmetic coding. First, we consider variable-size context models and propose several improvements for image compression applications. Next, we apply statistical context modeling to design better filtering methods for the enhancement of the digitized textual documents for compression and recognition. Experiments show that the proposed technique alleviates the loss in compression performance caused by digitization noise, while preserving the image quality, and readability by the Optical Character Recognition system. For line-drawing images, we develop semantic modeling. We extract vector features from the images using a Hough transform and raster-to-vector conversion. The extracted semantic information is used for efficient noise removal using feature-based filtering, and for improving statistical modeling in hybrid raster/vector compression systems. Finally, we propose an image storage system that enables real-time operations with large, remotely located images. The system supports instant image preview, fast decompression, and direct access to the image fragments.

For my wife Elena and daughter Erika

PREFACE

I would like to kindly thank my good friends and colleagues for the help in the creation of this thesis: Pasi Fränti, the supervisor of my study and co-author of our papers; Olli Nevalainen and Søren Forschammer, the reviewers; Jussi Parkkinen, Jorma Tarhio, and Martti Penttonen, all the heads of the Computer Science department during my studies; Alexander Kolesnikov, Heikki Kälviäinen, and Saku Kukkonen for cooperative work; Juha Hakkarainen, Martti Forsell, and computer administration personnel for the help with UNIX and Mac computers; staff of the Computer Science department of the Moscow State University in Ulyanovsk, where I have completed my MS; my parents Igor and Valentina, and brother Robert, who gave me respect for education and prodded me to complete the study; Elena and daughter Erika for understanding and love; Center for International Mobility, department of Computer Science and East Finland Graduate School in Computer Science and Engineering for financial support; National Land Survey of Finland for the set of cartographic images we have used in evaluation of our methods; and all my friends including Gerold Weinmann, Ivan Ignatieff, Andrew Alekseeff, Paul Kopyloff and other for their infinite support.

CONTENTS

1. Introduction

Document Imaging (DI) refers to the management of paper documents by capturing, indexing, archiving, retrieving, and distributing them electronically, as shown in Figure 1-1 [Lun90, Will92, Saf93, JLG96]. It has been estimated that over 10^{12} paper documents exist in the world, and the quantity is estimated to double every three years [OAS98]. The current industry is oriented towards producing and reproducing paper documents, and it does it faster than the papers can be digitized. Document Imaging aims at stopping (or at least slowing down) the growth of the paper piles and substitutes for paper in storing and accessing information. It provides easier access to the electronic replicas of documents, and minimizes the storage cost, compared with other document storage solutions, such as paper or microfilm.

A *document image* is the raster digital image that is the exact digitized replica of an original document [Lyn90]. Images are superior to paper documents because they can be economically stored, efficiently searched and browsed, copied without loss of quality, and quickly transmitted. Moreover, image is a media, satisfying both legal requirements and library preservation standards [JLG90, Les92, Har93].

We consider the following four major categories of imaging applications, according to the type of documents dealt with:

- *Document Digitization and Archiving System* (DDAS) – text documents, such as forms, records, and publications;
- *Image Communications* – facsimile and visual data;
- *Engineering Document Management* (EDM) – line-drawings, such as engineering drawings, cartographic maps, architectural and urban plans, schemes, and circuits (radio-electrical and topological);
- *Digital Spatial Libraries* (DSL) and *Geographic Information Systems* (GIS) – spatial images, maps and plans.

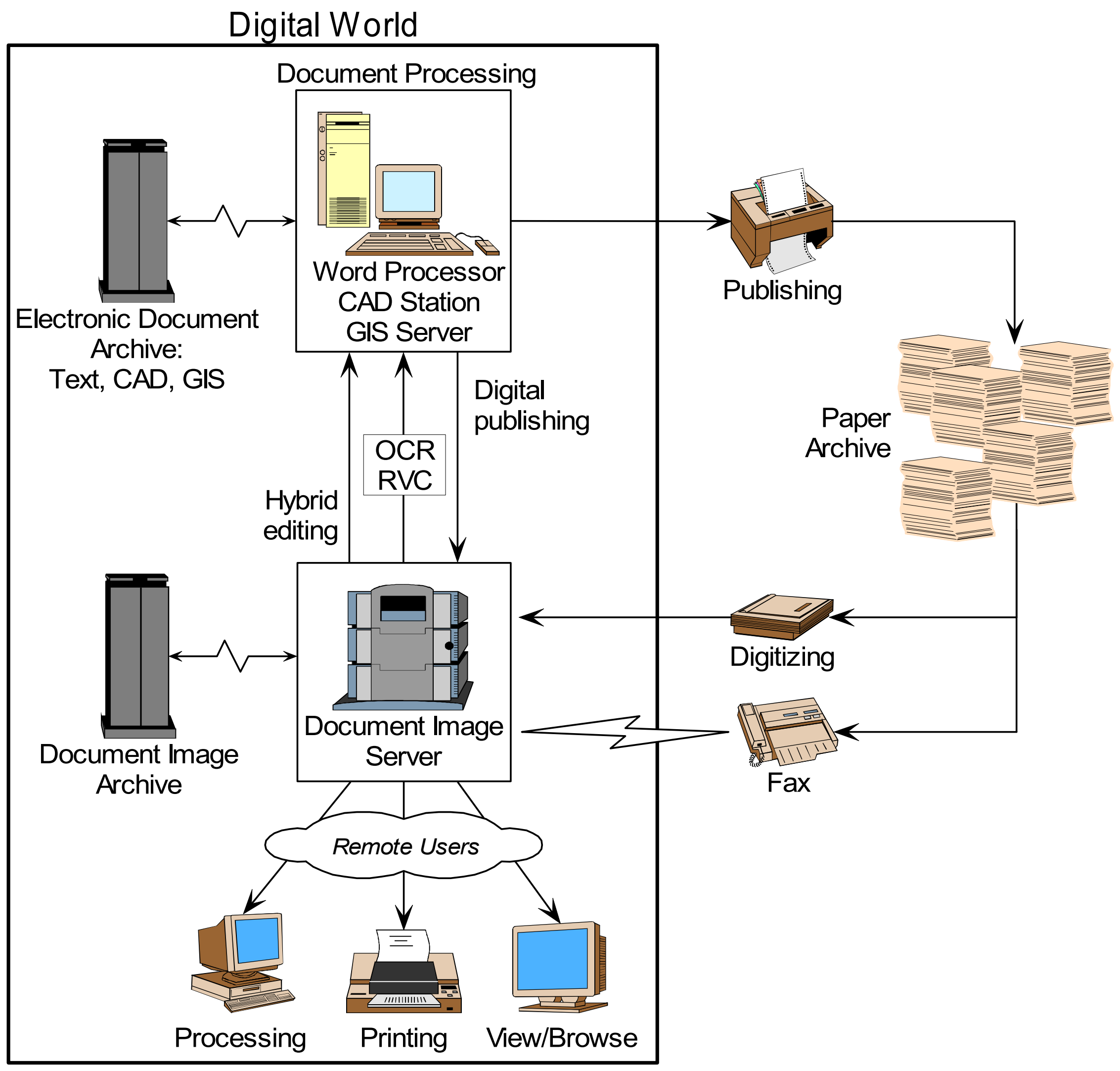


Figure 1-1. Life cycle of document images.

1.1. **Overview of the Application Areas**

Document Digitization and Archiving System

In DDAS, incoming documents are digitized (unless they are initially in digital form), categorized, and archived in electronic form. The whole process may be set for fully automatic operation without human intervention. The *digitization* phase can be efficiently performed using scanners and facsimile, which are also relatively inexpensive technology. The *archiving* phase includes *image enhancement*, *compression*, *recognition* and *indexing* operations.

Using *Optical Character Recognition* (OCR), it is possible to recognize the content of a digitized document and convert it to native text format so that it can be manipulated as though it had been typed in manually [OO92, MSY92]. However, at the current state of technology, the existing OCR solutions are characterized by high error rates and complexity [JLG93]. On the other hand, using the current compression technology [JBIG1, Haskel98], images can be compressed to approximately the same size as those used by file formats of common word processing software, or by PostScript files, see Table 1-1.

Table 1-1: Storage sizes of a one-page document in different document formats.

File Format	File Size (Kbytes)	
	200 dpi	300 dpi
Raster document image	470	> 1000
JBIG compressed image	~ 40	50-80
Word-processing file	30-40	
Post Script file	~ 90	
ASCII text	~ 4	

After the document has been archived, its further processing depends on the application. The document may be converted to word-processing compatible format using OCR or be indexed [Saf95]. *Document indexing* [WMB94] stands for the categorizing of the documents by some criteria or field (e.g. account number, date, and name) and usually requires OCR of the predefined text regions. *Full-text indexing* serves as a text search on the actual document content and requires OCR to convert the document into searchable ASCII text, which is usually stored together with the document image [Will92].

Image Communications

In an image communication system, such as facsimile, image serves as a communication medium. The document is first digitized using an optical scanning device, and is then compressed and transmitted to the recipient, where it is re-printed or archived in an electronic form [Hun80, ITU T.4, T.6]. The main difference between this and DDAS is that the sender and the recipient are separated by a communication channel, usually a telephone line, which is the bottleneck of the system. The sender may not have sufficient memory to hold the entire image for the time between digitization and transfer. Image scanning, compression and transmission are therefore performed simultaneously, and no intermediate image pre-processing pass over the entire image can be applied.

Engineering Document Management

It has been estimated by International Data Corporation (IDC) that about 8,000,000,000 line drawings exist in the world [Wils96, 99]. Only about 13% of them have been designed and stored in digital form using vector representations such as Computer Graphics Metafile (CGM) or AutoCAD drawings (DWG), see Figure 1-2. Nevertheless, there are still (and will continue to be) a large number of drawings that are stored as paper documents.

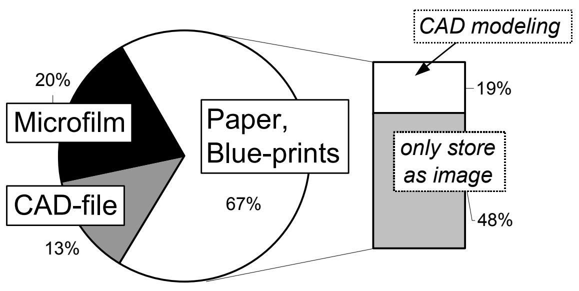


Figure 1-2. The world of engineering drawings [Wils96].

A possible solution for engineering image compression is to perform a *raster-to-vector conversion* (RVC), where the bitmap image is segmented into CAD primitives such as line segments, circles, and circular arcs and stored with any CAD/CAM format, see Figure 1-3 [Kas90, WD99]. Vectorized images are suitable for editing and they can be scaled without loss in quality. The storage size of an engineering drawing in CAD format takes about 2 % compared to a raster format with 300 dpi; this corresponds to a compression ratio of 50:1.

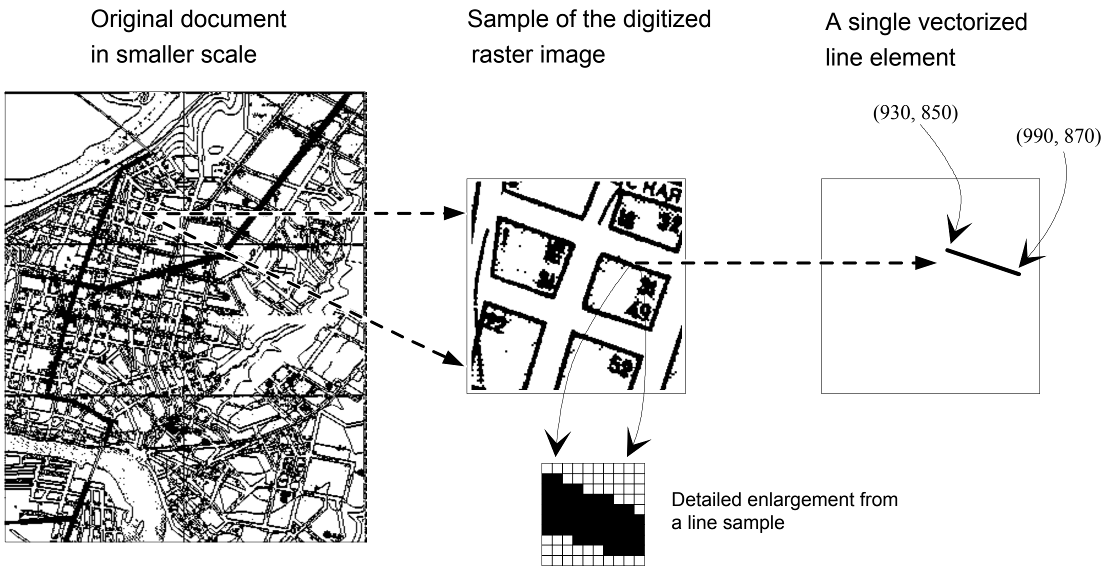


Figure 1-3. An example of raster-to-vector conversion.

Raster-to-vector conversion, however, may be problematic because of the high complexity and insufficient accuracy of conversion systems. The conversion process does not necessarily produce a faithful copy of the original document and loss of data is apparent. Moreover, the process is not often automatic, and requires expensive human interaction. Industrial projects have shown that the costs for such data acquisition exceed the hardware and software costs of operational information systems by a ratio of 100:1, according to [RM95].

It has been shown that there is a strong requirement for raster images to be vectorized only for parametric modeling and control system applications. These applications represent less than 15 % of all applications where engineering documents are used [Wils96]. In most other applications, the raster format is often sufficient, especially if *hybrid editing* is supported [Wils99, SEA99]. Hybrid editing means using both raster and vector data simultaneously, see Figure 1-4. Information can be exchanged back and forth between the two distinctive formats. Typically, the old (digitized) data is kept in the raster background, and new edits are maintained in vector format. These can be drawn either by hand or extracted from the raster image using semi-automatic vectorizing. No resources would be wasted on converting every document into CAD format, and the conversion would be made only when so desired. Nor would there be any loss of data without the control of the user.

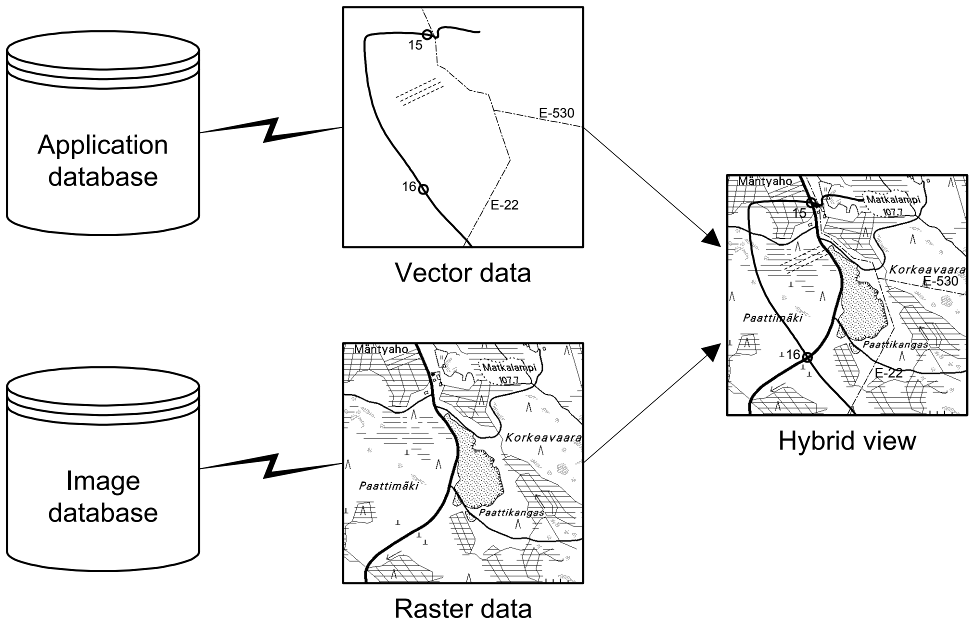
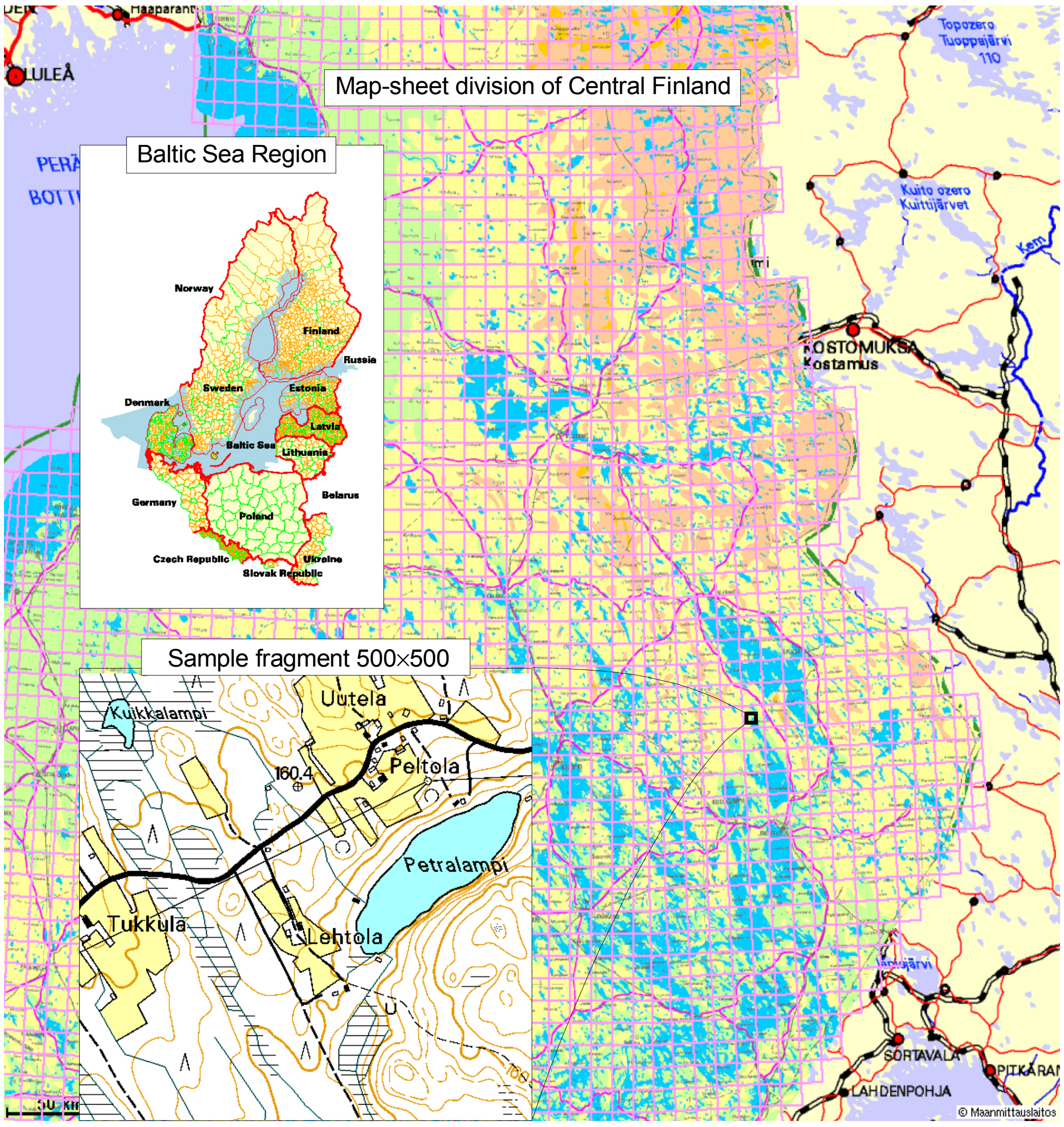


Figure 1-4. An example of hybrid raster/vector representation.

Digital Spatial Libraries and Geographic Information System

In *Digital Spatial Libraries* (DSL), raster map images are usually generated from a map database for digital publishing on CD-ROM or on the Web [ESRI94, 98; Fox+95]. The images consist of several binary layers, which together form the computer-generated color image, see Figure 1-5. The number of layers is limited, but the size of a single image is typically very large.

As an example of DSL, we consider digital maps produced in several international projects headed by the *National Land Survey of Finland* (NLS): *MapBSR*, a project covering the entire Baltic Sea region; *Barents GIT* (Geographic Information Technology), a joint project between Finland, Sweden, Norway and Russia; and *GIS-Server*, a project aimed at spanning the border between Finland and Russia. The objective of the projects is to produce uniform geographic information that can be used in planning and decision-making about communication, infrastructure, technical, economic and cultural cooperation, tourism and security interests [NLS].



BINARY LAYERS

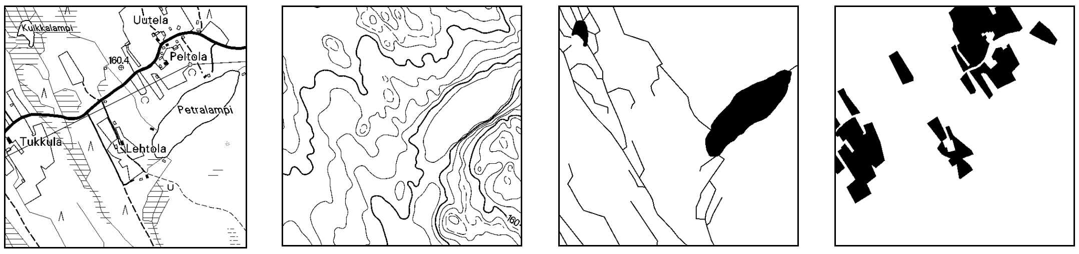


Figure 1-5. An example of Digital Spatial Library.
(with the permission of National Land Survey of Finland).

1.2. **Previous Work and the Scope of the Thesis**

Image Compression

Every document imaging application has a raster image as the basic component. A few color tones are usually sufficient to represent the original document, and only two tones are widely used [JBIGWP]. A multi-color document can be decomposed into bi-level planes, and be processed as the collection of bi-level images [RM92, JKW98].

The storage size of digitized images has been a major restriction in document imaging systems for decades. The storage problem is obvious: a standard A4-size document scanned at a relatively low resolution of 200 dpi (1728×2376 pixels) takes about 0.5 Mb, whereas a high quality engineering drawing of size A1 at 400 dpi requires 16 Mb. A typical digital map image of 5000×5000 pixels, representing a single map sheet of 10×10 km^2 in the NLS library, requires about 12 Mb; and there is no upper limit [NLS]. The GIS images may take hundreds of megabytes [PW96, Sam89]. The storage size impacts on nearly every aspect of a digital imaging system. The necessity of compression for saving storage space is, therefore, obvious. Cost savings emerge from several areas: fewer storage resources are needed and less network bandwidth required. Faster transfer implies a productivity gain because it makes Internet and LAN access more useful; less time is spent in waiting, and fewer resources are required to retrieve the files.

The compression of bi-level images has been extensively surveyed in the literature [Hun80, Ur92, AT94, Sal97, Haskel98]. The general idea of compression is to reduce the redundancy in the compressed data. The compression is usually considered as the process of assigning codes to the symbols of the compressed message according to its model, which is an assemblage of some rules or data that describe the message [RL81]. The initial advancements in compression of one-dimensional signals were quickly extended to the image domain by concatenating the image pixels in a single stream in an appropriate order, e.g. *raster-scan* or *row-major* order, in which separate lines of the image were processed in a left-to-right top-down manner. There have also been other sophisticated pixel-ordering techniques resulting in a more efficient utilization of one-dimensional sequential compression [NW80].

There are many approaches to reducing the redundancy in the images. Capon was one of the first to introduce the *run-length encoding* (RLE), which replaces the runs of same-colored pixels by two numbers: color and length of the run [Cap59]. *Statistical* approaches are based on modeling the image according to the probability distribution of the pixel values. Techniques such as Shannon-Fano [Sh48] and Huffman coding [Huf52, Vit87] assign shorter codes to more frequent pixel color values and vice versa, as in a Morse alphabet. The codes have a unique prefix property, enabling correct decoding.

The codes mentioned above, as do most of the other codes [Col66, Ric79, Will91] belong to the family of *integral* codes that have an integral number of bits per code. Huffman codes are known as *minimal-redundancy* codes, since they deliver the minimal possible bit-rate value among other integral codes. They are based on the *entropy* concept introduced by Shannon in 1948 [Sh48] that estimates the optimal code-length value.

The Huffman encoding process is usually very fast and not complex. However, because of the integral property, the probability distribution of the code does not necessarily match the distribution of the source. It also requires a minimum code length of one bit, which may produce a significant deterioration in the compression ratio if the probability distribution is much skewed. It therefore prevents the direct application of integral codes for the compression of binary images. To solve the problem, it has been proposed to apply the Huffman code, not directly to the pixels, but to runs of pixels, resulting in the *Modified Huffman* (MH) algorithm [ITU T.4, Hun80].

The following techniques have been developed to take advantage of a two-dimensional correlation between the image lines. *Vector run-length coding* proposed by Wang and Wu [WW92] applies run-length coding two-dimensionally to $m\times n$-sized blocks of pixels, instead of to single pixels.

The *block coding* introduced by Kunt and Johnsen [KJ80] divides the image into rectangular blocks of pixels. A totally white block is coded by a single 0-bit, whereas all other blocks are coded by a 1-bit as a prefix, followed by the content of the block. The process can be iterated, resulting in a *hierarchical block coding* algorithm. This technique has been improved by encoding the bit patterns of the 2×2-blocks with Huffman codes. Another improvement has been achieved using the prediction technique [NW80]. The idea is to form a so-called *error image* from the original one, by comparing the value of each original pixel to the value given by a *prediction function*. If these two are equal, the pixel of the error image is set to white, otherwise to black. The encoding is then applied to the error image instead of to the original one, and gains were achieved from the increased number of white pixels [FN95].

Another approach is to exploit the correlation between successive lines of the image. This idea is implemented in the method called *relative element address* *designate* (*READ*). Instead of coding the lengths of the runs, this method codes the location of the boundaries of the runs (point of transitions from black to white and vice versa) relative to the corresponding positions in the previous line. If there are no such positions within three pixels in the reference line, one-dimensional run-length coding is used. The READ method is defined in the international standard for facsimile communications [ITU T.4]. In this, every *k*-th line ($k = 1, 2,$ or 4) is coded using one-dimensional MH-coding, and two-dimensional READ-code (more accurately referred to as *Modified READ*) is applied to the remaining lines.

A technique, which is completely different to integral coding, is called *arithmetic coding*, as proposed by Rissanen and Langdon [RL79]. The idea of arithmetic coding is to represent the entire input message as a small interval in the range [0,1]. The resulting codeword is the binary code representation of the interval. Arithmetic coding is an optimal coding method as regards the model, and it fits for compression of binary images. It is also well suited for dynamic modeling, because there is no need to store and update complex data structures such as Huffman trees.

Arithmetic coding does not have the limitation of integral codes. The method is, therefore, fully applicable directly to the pixels, instead of pixel blocks or runs. In [LR81], Rissanen and Langdon proposed that Shannon's context-based statistical modeling be used in conjunction with arithmetic coding. The idea of context-based modeling is to obtain the statistical model of the image by conditioning the probability distribution of the pixels on the context. The context is determined by the combination of the pixels in the local neighborhood, which is defined by the template. Context-based statistical modeling and arithmetic coding are implemented in the latest international standard for compression of binary images, *JBIG (Joint Bilevel Image Experts Group)* [JBIG1]. To distinguish it from the emerging JBIG2 standard, we will refer to it as to JBIG1.

Although JBIG1 is originally designed for bi-level facsimile images, the solutions for multi-tone and grayscale images have also been presented. The multi-tone images can be separated into several bit-planes, and each bit-plane compressed separately. The separation can be performed using the binary representation of gray-level values or the gray-code words [WRA96, JKW98]. Another approach to deal with multi-tone images has been developed by At&T and is called *"DjVu"*. It classifies the pixels of the image either as foreground (text, drawings) or background (pictures, photos, paper texture). The classification is compressed as a binary image, whereas a progressive, wavelet-based lossy compression technique is applied to the foreground and background images [Haf+99].

To improve the compression of binary images, Moffat has experimented with various sizes and shapes of the context templates and has noticed that the bit-rate does not decrease further after the templates exceeds 14 pixels [Mof91]. However, he has demonstrated the potential of larger templates of up to 24 pixels and has proposed the two-level modeling technique using both 10- and 24-pixel templates. The variable-size modeling based on a context tree has been introduced by Rissanen in [Ris]. Martins and Forchhammer have recently studied variable-size context modeling and proposed both context tree and "free tree" techniques, in which the number and the positions of the context pixels are varied depending on the pixel and its neighborhood [MF98].

Remarkable improvements in image compression have been achieved by specializing in some known image types (e.g. text images) and exploiting global dependencies. The emerging standard JBIG2 [JBIG2] will segment a page into different classes of image data, in particular, textual, halftone and generic (other) [TK99], and utilize the repetitive nature of the textual and halftone images. For textual data, JBIG2 uses pattern matching techniques, which are based on the following works. Ascher and Nagy [AN74] have proposed the pattern matching technique to extract symbols and marks from the image into the dictionary, which is a collection of bitmaps. Witten et al. expanded on this approach to address the extraction of marks, indexing their location within the image, compressing the indices, and coding the residuals left after the replacement of marks by library prototypes [IW94, WMB94]. Howard has proposed soft pattern matching, which compresses the original image instead of the residual, and uses the image composed of prototypes in improved context modeling [How97].

JBIG2 will also address the compression of halftone image data using either of the two following methods. The first is similar to JBIG1, but it uses larger context templates (up to 16 pixels) with multiple adaptive pixels [MF98, 99]. The larger templates are intended to exploit specific types of redundancies that exist in halftone images. The second method involves descreening the halftone image (converting it back to grayscale) and transmitting the grayscale values [VETK99, FJ94]. Some data, such as line art data, may not be identified as either textual or halftone, and will be coded by a *cleanup coder*, which is essentially a bitmap coder similar to JBIG1. JBIG2 will also provide *lossy* image compression [MF99], and *quality* and *content progressive* coding [How+98].

Image enhancement

The quality of document images may have faded during the document life cycle and digitization process, while noise introduces unnecessary details in the images. It degrades the image quality and weakens image compression. Several filtering methods have been considered in the literature for image pre-processing [TP80, Ber87, AKS90, ZD96]. These filters include *logical smoothing*, variations of *median filtering*, *isolated* pixel removal, and "crisp" and soft *morphological filters* [Ser82, Hei94]. All these analyze the local pixel neighborhood defined by a filtering template. To accept or reject the pixel, they use a set of rules, such as predefined masks or quantitative description of the local neighboring area. Recent research in mathematical morphology has shown that morphological filtering can be used as an efficient tool for pattern restoration in an environment of heavy additive noise, but it is not necessarily suitable for filtering the content-dependent noise introduced by the image digitization process [SG91, Hei94, KA94, DA97].

Hybrid raster/vector modeling

Numerous techniques and systems have been proposed for line-drawing images, to extract semantic information and perform a conversion of the document to vector space [Hou69, HK83, Ab89, Lea93, NL90, Kas90, RM95, KBO96, WD99]. However, no one solution creates a faithful copy of the original document, and expensive human interaction is often required [RM95].

The hybrid raster/vector compression system has been proposed to eliminate the necessity of converting the engineering drawings into vector format in order to process them in CAD applications [Wils96, 99]. Typically, vector and raster representations of the image are stored together in the same file. Existing information is kept in original raster form when new edits are made in vector form native for CAD. The vector features of the raster object can be extracted on demand, so that this raster object can be processed (moved, scaled, rotated or removed) directly in the raster [SEA].

Spatial access

The document image archive may not be physically present at the viewing location, but may be accessed through a communication channel, which could be nothing more than a slow telephone connection [Lun90]. Compression reduces the amount of data to be transferred and makes the image retrieval faster. However, the time required for image transmission and decompression may make the access to the images significantly slower. Spatial access is, therefore, another highly desired property of an imaging application, such as GIS, that deals with spatial data [Sam89, Fox+95]. Spatial access means direct access to an image fragment in a compressed file and enables an efficient and precise retrieval of the desired image fragments. Imaging applications using large format images (e.g. EDM) may also benefit from spatial access [PW96, AF98]. It allows the user to eliminate unnecessary delays caused by retrieval and decompression of the entire image. Spatial access has received relatively little attention in the literature; for solutions in text compression, see [WMB94].

1.3. **Structure and Contribution of the Thesis**

Although the lossless compression of binary images has been extensively studied during the last decades, and several compression standards already exist, they do not provide an efficient universal solution, nor satisfy all application-specific requirements, such as on-line retrieval, high quality restoration, spatial access and intellectual post-processing. Better solutions must therefore be developed. In this thesis, we study context-based methods for enhancement, storage and processing of binary images of documents and line drawings. We aim at improving compression performance and interactive processing. The organization of this thesis is as follows.

Chapter 2 contains the basic concepts and definitions for statistical image compression. We start by recalling the concept of statistical context modeling introduced by Shannon, and define static, semi-adaptive and dynamic modeling approaches. Next, arithmetic coding and its implementation aspects are briefly discussed. We review also the process of the probability estimation derived from arithmetic coder renormalization, which is implemented in the QM-coder, a binary arithmetic coder used in JBIG1. Thereafter, we introduce a new forward-adaptive modeling technique for the QM-coder [AF99a]. The technique is useful when large images are subdivided into smaller parts. Finally, the JBIG1 standard is briefly described, including the algorithm for resolution reduction.

Chapter 3 deals with context-based modeling and aspects of variable-size context models. We start by discussing the fixed size context templates and show the limitations of this approach. Next, we proceed to variable-size context modeling, in which the number of context pixels depends on a combination of neighboring pixel values. We define the concept of context tree and study the aspects of its construction. We define a splitting criterion, and consider both static and semi-adaptive construction alternatives. Thereafter, we review and compare two strategies for building the context tree: top-down and bottom-up approaches. For top-down tree construction, we show the locality problem of the tree splitting and present a new delayed pruning technique [FA99]. For bottom-up tree construction, we present a new space efficient two-stage pruning algorithm [AF00b]. We also show how variable-size context modeling and forward-adaptive statistical modeling can be combined [AF00b]. Finally, we give an empirical comparison of the strategies presented and draw conclusions.

In Chapter 4, we study global modeling of the images. We start by discussing various techniques for extracting semantic information from the image. First, we review the pattern matching techniques used in the emerging standard JBIG2 for extraction of the common symbols from document images, and utilizing this information in improved image compression. Next, we switch to line drawings, and propose two techniques for the extraction of vector features: the first one is based on a Hough transform [FAKK98a], and the second one is based on raster-to-vector conversion [FAK99]. Finally, we consider hybrid raster/vector storage systems and propose a hybrid modeling technique, in which vector information is used for improving compression of raster images [FAKK98b].

Chapter 5 is devoted to image enhancement and noise removal. We present here two concepts for image filtering: context-based statistical filtering for document images, and feature-based filtering for line-drawing images. We start by defining context-based filtering, and introduce the *Simple Context filter* that unconditionally changes uncommon pixels in low entropy contexts. Thereafter, we introduce a new *Gain-Loss filter* that takes into account the effect of filtering on compression (the gain,) as well as the error introduced by filtering (the loss) [AF00a]. Next, we introduce the new concept of feature-based filtering based on semantic image modeling [FAK99]. We present an algorithm for removing content-dependent noise along the line contours of the image, which is difficult to remove using traditional filtering methods without smoothing the image.

Chapter 6 deals with interactive image browsing, retrieval and spatial access. First, we establish the objectives in order to support real-time access to the image archive: instant preview, fast decompression, and spatial access, i.e. direct access to the image fragments [AF98]. Next, we present a new storage system architecture that combines the compression methodologies of Chapter 2 with the properties of instant preview (by block-coding) and interactive access to the compressed image (by image tiling). We thoroughly discuss the implementation aspects of the spatial access, and its advantages and disadvantages for the compression [AF98].

In Chapter 7, we present empirical evaluation of the proposed modeling techniques, filtering methods and storage system. We start by studying the performance of variable-size context modeling in image communication [FA99]. Next, we analyze the effect of context-based filters introduced in Chapter 3 on document images [AF00a]. We study the possibility of improving the compression performance of filtered images, while preserving document recognition. Thereafter, we evaluate the feature-based filtering method as a pre-processing stage in an image compression system, which uses either of two standard compression components, JBIG1 or ITU Group 4, and compare it with the traditional filtering methods [FAK99].

Finally, we evaluate the efficiency of the storage system outlined in Chapter 6. In the study, we first compare two techniques for generating thumbnail images, block-coding and JBIG resolution reduction; and analyze the effect of block codes on the decompression time, and the effect of image tiling on compression performance. We then evaluate five compression methods based on the methodologies presented in Chapters 2 and 3, and compare them with ITU Group 3/4 and JBIG1 [AF00b].

In Chapter 8, we make conclusive remarks on the study.

The image test sets used in the evaluation are included in the Appendix. These sets include: CCITT facsimile documents, digitized text documents and newspaper images, real-life line drawings, including engineering drafts, electrical circuits, GIS maps and architectural plans, and, finally, topographic maps from the digital spatial library of the National Land Survey of Finland.

The main contributions of this work can be summarized as:

- a consistent presentation of the fundamental concepts and methods concerning context-based statistical modeling;
- a new forward-adaptive modeling technique for the QM-coding algorithm aimed at the construction of a better initial model for the coder in order to alleviate the learning cost problem caused by tiling the image into small parts (Section 2.7);
- new results and development work concerning variable-size context modeling (Chapter 3), including the solution for the locality problem of the tree splitting (Section 3.3), and a new space efficient two-stage tree construction algorithm (Section 3.4);
- application of variable size modeling in image communication (Section 7.1), and in conjunction with forward-adaptive modeling in digital spatial libraries (Sections 3.5 and 7.4.2);
- the concept of global modeling for line drawings (Chapter 4), and the study of methods for extracting semantic features by Hough transform (Section 4.2) and raster-to-vector conversion (Section 4.3);

- a new method for compression of the line-drawing images stored in raster-graphic format in hybrid raster/vector storage systems (Section 4.4.4);
- a new feature-based filtering technique for removing quantization noise from line-drawing images (Section 5.2);
- a new context-based filtering algorithm for enhancing document images for compression, while preserving document readability (Section 5.1);
- a new approach for evaluating filtering methods by the OCR technique (Section 7.2); and
- a new storage system architecture supporting instant preview, fast decompression, and direct access to image fragments (Chapter 6).

2. Statistical Image Compression

The aim of compression is to remove redundancy of the data. Statistical image compression consists of two distinct phases: *statistical modeling* and *coding* [RL81]. In the modeling phase, we construct the probability distribution for the occurrence of the symbols to be compressed. The coding process assigns the variable length code words to the symbols according to the probability model, so that shorter codes are assigned to symbols that are more probable and vice versa. The main problem of the compression is to find a good model, describing the data with high precision. The coding can be efficiently performed using *arithmetic coding*, which is an optimal coding for a given probability model [RL79].

2.1. **Statistical Modeling**

A binary image can be considered as a message, generated by an information source. The idea of statistical modeling is to describe the message symbols (pixels) according to the probability distribution of the source alphabet (binary alphabet, in our case). Shannon has shown in [Sh48] that the information content of a single symbol (pixel) in the message (image) can be measured by its *entropy*:

$$H_{pixel} = -\log_2 p \, , \tag{2.1}$$

where p is the probability of the pixel. Entropy of the entire image can be calculated as the average entropy of all pixels:

$$H_{image} = -\frac{1}{n}\sum_{i=1}^{n}\log_2 p_i \, , \tag{2.2}$$

where p_i is the probability of i-th pixel and n is the total number of pixels in the image. If the probability distribution of the source alphabet (black and white pixels) is a priori known, the entropy of the probability model can thus be expressed as:

$$H = -p_W \log_2 p_W - p_B \log_2 p_B \, , \tag{2.3}$$

where p_W and p_B are the probabilities of the white and black pixels, respectively.

Entropy gives the optimal number of bits required for encoding a single pixel with a given model. A model with skewed probability distribution will have low entropy, see Figure 2-1. Respectively, the codes with the lengths equal to the entropy values will provide an optimal compression in respect of the model.

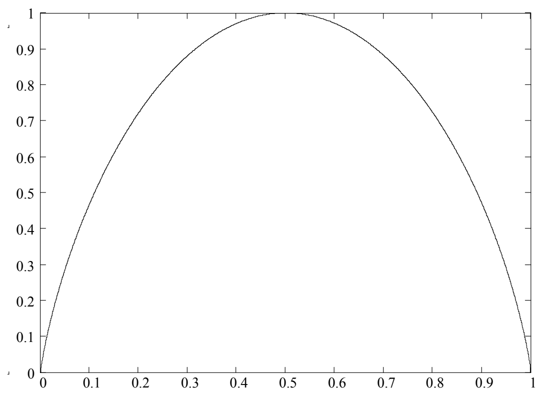


Figure 2-1. Entropy of the binary probability model, as function of white pixel probability.

2.2. **Static, Semi-adaptive and Adaptive Approaches**

The modeling schemes can be classified as *static*, *semi-adaptive* or *adaptive* (*dynamic*). In the static modeling, the probability distribution of a source alphabet is a priori known (or suggested) and the same, non-changing model is applied to every pixel during the compression. The advantage of static modeling is its simplicity, and that no side information has to be passed to the decoder.

Semi-adaptive modeling uses a preliminary pass on the input data to gather statistics and construct the model. The model is passed to the encoder, which performs data compression as in the static variant. The model must also be passed to the decoder to make decompression possible.

Dynamic (adaptive) modeling takes one step further and eliminates the need for an extra pass over the image to construct the model, and no model overhead is required. Both encoder and decoder dynamically estimate the model during the compression/decompression adapting to the preceding data. Usually an equal initial probability distribution for black and white pixels is assumed. The time-dependent cumulative pixel counts n_B^t and n_W^t are therefore initialized to 1, and are subsequently incremented by 1 each time black or white pixel value appears, respectively.

The more sophisticated *Bayesian sequential estimator* calculates probability of the pixel on the basis of the observed pixel frequencies as follows [MF98]:

$$p(x^t)=\begin{cases} p_W^t=\dfrac{n_W^t+\delta}{n_W^t+n_B^t+2\delta}, & \text{if } x^t \text{ is } \textit{white} \\ p_B^t=1-p_W^t, & \text{if } x^t \text{ is } \textit{black} \end{cases} \tag{2.4}$$

where n_W^t, n_B^t are the time-dependent counters, p_W^t , p_B^t are the probabilities for white and black colors respectively, and δ is a constant. Counters n_W^t and n_B^t start from zero and are updated after the pixel has been coded (decoded). As in [JBIG1], we use $\delta = 0.45$. The cumulative equation for entropy (2.2) is used to estimate the average bit rate and calculate the ideal code length.

Dynamic modeling is inefficient at early stage of compression, since it takes time to adapt to the correct model, but highly applicable for compression large volumes of data, such as document images. The loss in compression rate caused by the model adaptation is known as the *learning cost*.

2.3. **Context Modeling**

The pixels in an image form geometrical structures with appropriate spatial dependencies. The dependencies can be localized to a limited neighborhood, and described by a *context-based statistical model* [LR81]. In this model, the pixel probability is conditioned on the *context C*, which is defined as distinct black-white configuration of neighboring pixels within the local template. For binary images, the pixel probability is calculated by counting the number of black (n_B^C) and white (n_W^C) pixels appeared in that context in the entire image:

$$p(x)=\begin{cases} p_W^C=\dfrac{n_W^C}{n_W^C+n_B^C}, & \text{if } x=\textit{white} \\ p_B^C=1-p_W^C, & \text{if } x=\textit{black} \end{cases}, \tag{2.5}$$

Here, p_B^C and p_W^C are the corresponding probabilities of the black and white pixels. The entropy $H(C)$ of a context C is defined as the average entropy of all pixels within the context:

$$H(C)=-p_W^C\cdot\log_2 p_W^C-p_B^C\cdot\log_2 p_B^C \tag{2.6}$$

A context with skew probability distribution has smaller entropy and therefore smaller information content. The entropy of an *N*-level context model is the weighted sum of the entropies of individual contexts:

$$H_N = -\sum_{j=1}^{N} p(C_j) \cdot \left(p_W^{C_j} \cdot \log_2 p_W^{C_j} + p_B^{C_j} \cdot \log_2 p_B^{C_j}\right) . \tag{2.7}$$

In principle, a skewed distribution can be obtained through conditioning of larger regions by using larger context templates. However, this implies a larger number of parameters of the statistical model and, in this way, increases the model cost, which could offset the entropy savings. Another consequence is the “context dilution” problem occurring when the count statistics are distributed over too many contexts, thus affecting the accuracy of the probability estimates.

2.4. **Arithmetic Coding**

Arithmetic coding is a statistical compression method that assigns one long code to the entire input stream, instead of assigning codes to the individual symbols [RL79, WNC87]. It is an optimal coding method for a given probability model, because it can achieve a bit-rate approximately equal to the entropy value.

The basic idea of arithmetic coding is to represent the entire input data as a small sub-interval in range [0,1). The coding process starts by dividing the interval [0,1) into two sub-intervals according to the probability distribution of the black and white pixels. Depending on the pixel color, the upper or lower sub-interval is chosen, and the process is repeated for the next symbols, resulting in smaller and smaller intervals. The final interval describes the source uniquely. The length of this interval, *L*, is the cumulative product of the probabilities of the coded symbols:

$$L_{\text{final}} = p_1 \cdot p_2 \cdot p_3 \cdot \ldots \cdot p_n = \prod_{i=1}^{n} p_i , \tag{2.8}$$

and it can be coded by the following number of bits:

$$Codesize(L_{\text{final}}) = -\log_2 \prod_{i=1}^{n} p_i = -\sum_{i=1}^{n} \log_2 p_i \tag{2.9}$$

The implementation aspects of the binary arithmetic coding follow [Sal97]. The encoding process starts by defining two variables, *Low* and *High*, in order to describe the coding interval. The *Low* is initialized to 0, and *High* to an infinite fraction .999…, since it has to be interpreted as a fraction less than 1. Usually, *Low* and *High* are represented as integer (binary) variables holding the most significant part of the real numbers. After the pixel has been coded, the interval is reduced by a factor that equals the pixel probability, and *Low* and *High* are updated accordingly. Very soon the interval becomes too small to be expressed by the two variables, and the interval scaling procedure is therefore applied. When the interval falls below or above the *half point*, the codeword is known to start with the bit 0 or 1, respectively. In both cases, the starting bit can be shifted out of the interval variables and output to the compressed stream, and the interval is rescaled, see Figure 2-2.

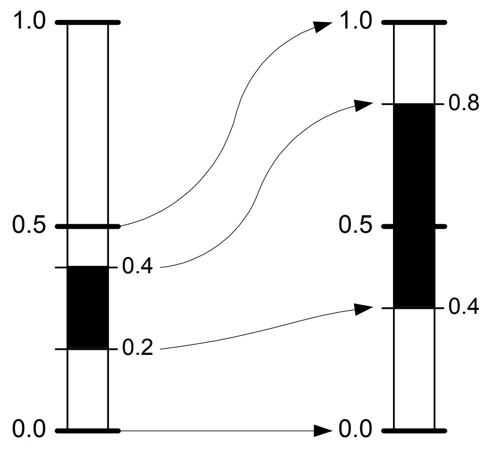


Figure 2-2. Example of *half point scaling* [RL79].

Underflow can occur when the size of the interval becomes too small, but the interval still covers the half point. To solve this problem, *quarter point scaling* is applied, see Figure 2-3. In this case, neither bit is output. Later, when the half point scaling occurs, an appropriate bit will be added to the code stream, see [RL79] for details.

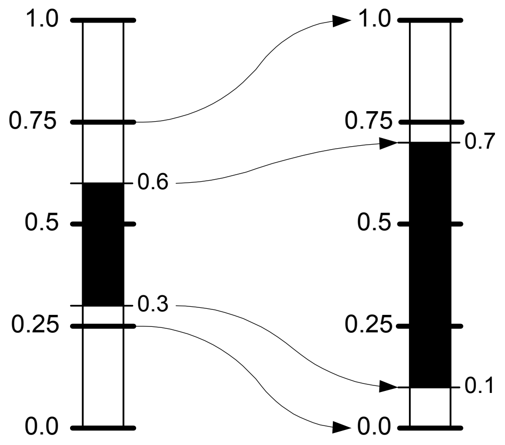


Figure 2-3. Example of *quartet point scaling* [RL79].

2.5. **QM-coder**

The QM-coder is the arithmetic coder used in JBIG1 [PMLA88, PM93]. It is an approximate implementation of arithmetic coding tailored for binary data. Its sub-optimality is compensated by the sophisticated automaton-based probability estimation (see Section 2.6), providing fast adaptation to the source data.

The QM-coder uses the following variables to describe the interval: *interval base* and *interval size*. If the encoded pixel value (color) is the one with higher probability, it is denoted as *most probable symbol* (MPS), when the opposite value is denoted as the *least probable symbol* (LPS). The interval is always divided so that the LPS sub-interval is above and MPS sub-interval is below as shown in Figure 2-4. Here *C* is the *interval base*, *A* is the *interval size*, and *Qe* is the LPS probability estimate [PMLA88].

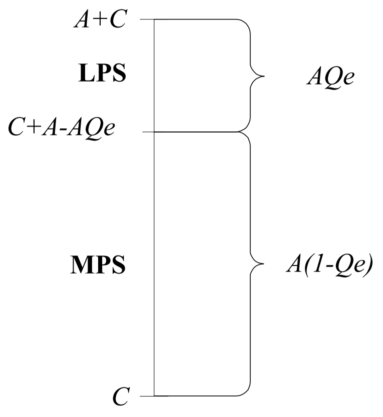


Figure 2-4. Interval subdivision of the QM-coder.

Altering the interval size involves multiplication. The QM-coder accepts the element of approximation by replacing the interval multiplication by suitable scaling. It assumes that the interval size is roughly constant and equals to 1. In this case, the coding of a pixel changes the interval as follows.

After MPS:

$$\begin{aligned} &C \text{ is unchanged} \\ &A \leftarrow A \cdot (1 - Qe) = A - A \cdot Qe \approx A - Qe \end{aligned} \tag{2.10}$$

After LPS:

$$\begin{aligned} &C \leftarrow C + A \cdot (1 - Qe) = C + A - A \cdot Qe \approx C + A - Qe \\ &A \leftarrow A \cdot Qe \approx Qe \end{aligned} \tag{2.11}$$

The interval size is maintained between 0.75 and 1.5, centered on 1. When the interval size falls below lower bound, the interval is renormalized by a series of consecutive duplications performed by bit-shifting operations. The renormalization occurs always after the LPS, and if necessary, after the MPS is encountered. At each renormalization, the encoder generates output bits (0 or 1) regarding to MPS or LPS and number of duplications in the renormalization process.

2.6. **Automaton-based Probability Estimation**

Probability estimation can be derived from arithmetic coder renormalization, as in the QM-coder [PM88]. Instead of maintaining pixel counts, the estimation process is implemented as a state automaton consisting of 226 states. Each context has its own 8-bit pointer to the automaton, where one bit indicates the color of MPS. The automaton has mirror symmetry about the change in the sense of MPS color, and we therefore consider only 113 states, see Figure 2-5. The automaton is a Markov-chain containing one state for each probability estimate. The states are organized in rows that are ordered by the level of adaptation. The states in the upper rows are more sparsely distributed throughout the probability range and therefore they allow faster adaptation.

The adaptation process starts from the zero-state. In each state, the automaton can perform a transition to two other states, see Figure 2-5. After each MPS renormalization, a transition is made to the next state situated to the right in the same row, having a smaller LPS probability. After each LPS renormalization, a transition is made to the state with a larger LPS probability, which is the appropriate state in the row at the next level in the case of the *transient* state, or to the preceding state in the same row in the case of *non-transient* states. *Transient* states are, therefore, visited only during the learning stage, and the pointers stabilize eventually to the *non-transient* states. If the statistics change later, the non-transient states can be re-entered from other non-transient states, making local adaptation possible.

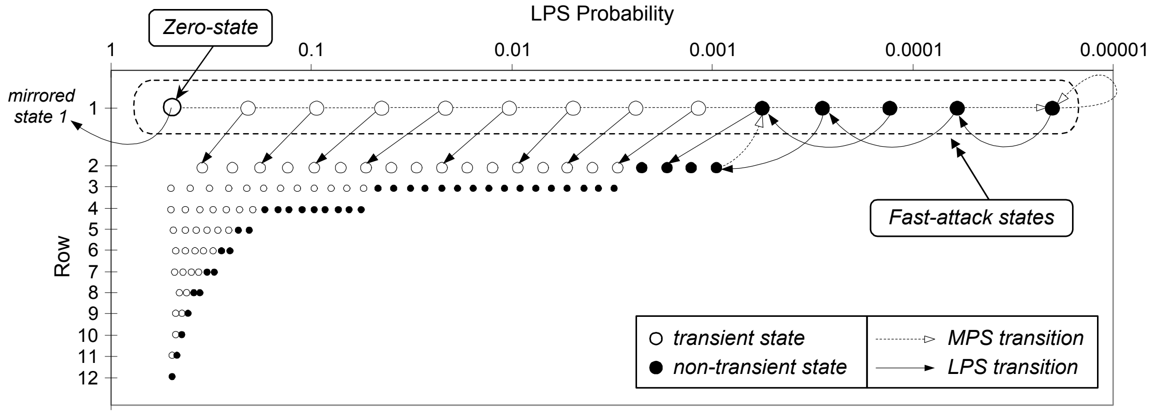


Figure 2-5. Spatial organization of the QM-coder state automaton and transition sketch for the fast-attack states. Because of the mirror symmetry regarding the change in sense of MPS, only half of the states are depicted.

2.7. **Forward-adaptive Modeling**

In [AF99a] we proposed the forward-adaptive variant of statistical context-based modeling for the QM-coding algorithm. The technique is a two-stage combination of *forward-adaptive* and *backward-adaptive* strategies. Statistics are first collected globally over the image (as in the semi-adaptive approach) to construct a better initial model. The model is stored in the compressed file. In the second stage, the image is coded using QM-coder and initializing the statistics according to the constructed model.

The initial model serves to enable faster adaptation and helps to alleviate the coding inefficiency caused by learning cost, which is typical when coding small portions of data. This can be very useful if the coding must be restarted periodically (see Chapter 6). The forward-adaptive method can be implemented with minor modifications to the existing software implementations of the QM-coder. This scheme requires two passes over the image even though the decompression can be performed with one pass only.

The implementation of the method is outlined in Figure 2-6. The input image is first analyzed and the probability distribution of black and white pixels is calculated for each context. The calculated probabilities are mapped to the 26 fast-attack states in the state automaton using a look-up table. The fast-attack states (first row of state in Figure 2-5) can represent all probabilities with sufficient accuracy, allow faster adaptation than from the zero-state, or re-adaptation from non-transient states. The choice of the fast-attack state can be coded by five bits each. The LPS probabilities of the fast-attack states are shown in Table 2-1. The result of the mapping is the model table formed by the five-bit indices.

Figure 2-7 shows the changes in the QM-coder caused by the forward-adaptive modeling. The probability mapping is implemented using the *GetFastAttackStateIndex* function. The state index is found by a sequential search implemented in the *FindFastAttackState* function. The QM-coder is initialized using *RestoreState* function. It takes the context number (*context*) and the state index (*index*) as input and accordingly restores the fields (*mps* and *cstate*) for the appropriate context in the QM-coder.

```
// MODELING STAGE
for (each pixel x of t in raster scan order)                    // gather statistics
    {
      c = GetContext (x);                                  // determine pixel's context c
      n_total[c] ++;                                       // update statistics of context c
      if (x == white) n_whites[c] ++ ;
    }
for (i = 0, i < NumberOfContexts, i ++)                    // construct and store the model
    {
      index[i] = GetFastAttackStateIndex (n_whites[i] / n_total[i]);
      StoreModelIndexIntoFile (index[i]);
    }
// CODING STAGE
for (i = 0, i < NumberOfContexts, i ++)                          // initialize the QM-coder
    RestoreState(i, index[i]);
for (each pixel x of t in raster scan order)                    // compress the cluster t
    {
      c = GetContext (x);                                  // determine pixel's context c
      EncodePixelByQM (x, c);                              // encode pixel x by QM-coder
    }
```

Figure 2-6. FA-M algorithm.

Table 2-1: LPS probabilities of the fast-attack states.

State:	0	1	2	3	4	5	6
p_{LPS}:	0.49690	0.20691	0.09417	0.04435	0.02120	0.01021	0.00493
State:	7	8	9	10	11	12	13
p_{LPS}:	0.00239	0.00116	0.00056	0.00028	0.00013	0.00006	0.00002

```
float FastAttackStateBounds [13] = {
        .30891, .14590, .06891, .03255, .01537, .00726, .00343,
        .00162, .00076, .00036, .00017, .00008, .00004 };

int FindFastAttackState (float Prob)
  {
    int i;
    for (int i = 0; i < 13; i ++)
          if ( Prob > FastAttackStateBounds [i] )  return (i);
    return (13);
  }

int GetFastAttackStateIndex (float WhiteProb)
  {
    float LpsProb;
    int    index;
    if (WhiteProb < 0.5)    LpsProb = WhiteProb;
    else                    LpsProb = 1 - WhiteProb;
    index = WhiteProb < 0.5 ? 0x00 : 0x10;
    index = index | FindFastAttackState (LpsProb);
    return (index);
  }

void RestoreState (int context, int index)
  {
    mps[context] = (index & 0x10) ? 0 : 1;
    cstate[context] = (index & 0x0f);
  }
```

Figure 2-7**.** Extensions for the QM-coder.

2.8. **JBIG1**

JBIG1 is an International Standard for compression of bi-level images in communications [JBIG1]. The standard defines two methods for bi-level compression, *progressive* and *sequential*. In sequential coding, the image is coded in raster scan order using a *context-based probability model* and *adaptive arithmetic coder* (QM-coder), see Figure 2-8. The probability distribution of the black and white pixels is conditioned on the context, which is defined by the combination of already coded neighboring pixels. A three-line ten-pixel template is used by default, see Figure 2-9. Both encoder and decoder estimate the model dynamically during the compression. The estimation starts from scratch and adapts the model to the input data. The probability estimation in the QM-coder is derived from the arithmetic coder renormalization and is based on the Bayesian estimation concept [PM88].

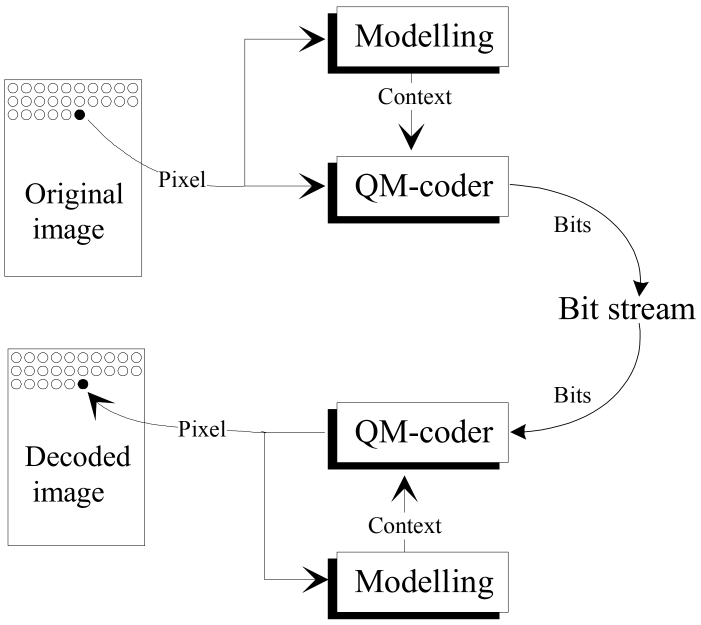


Figure 2-8. Block diagram of JBIG1.

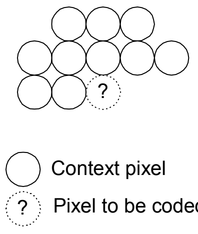


Figure 2-9. Default ten-pixel three-line context template of JBIG1.

JBIG1 has also progressive mode, in which the encoded image is stored in several resolutions. A reduced resolution version of the image, which is usually not larger than 640×480 pixels, is compressed first. It is followed by the layers with progressively increasing resolutions so that each successive layer has twice the number of horizontal and vertical pixels than the previous layer. Pixels from the previous resolution layer are added to the context template to improve the compression performance. A drawback of the progressive mode is the redundancy that it adds to the code stream. The redundancy amounts to 15-25 % according to our experiments, see Figure 2-10. JBIG1 can also compress gray-scale images can be compressed using the binary representation of gray-level values or the Gray-code words [WRA96, JKW98].

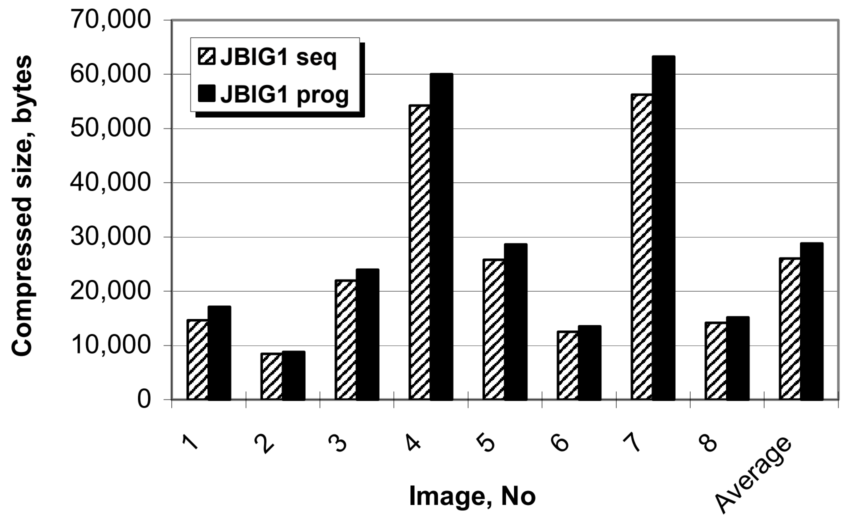


Figure 2-10. Compression rates of JBIG1 in sequential and progressive mode for the CCITT test set (see Appendix A).

JBIG1 separates an image into several horizontal stripes, resolution layers and planes, were each plane contains one bit per pixel. The resolution layers are stored all in a single *bi-level image entity* (BIE) file or they can be stored in several separate BIE files. Separate bitmap planes are stored in a multi-bitplane BIE. One single stripe in one plane and layer is encoded as a data unit called *stripe data entity* (SDE) inside the BIE. There are 12 different possible orders in which the SDEs can be stored inside the BIE. This order is only relevant for applications, in which we want to decode a JBIG1 file, which has not yet completely arrived from e.g. a slow network connection. For instance, some applications prefer that the outermost of the three loops (stripes, layers, planes) is over all layers so that all data of the lowest resolution layer are transmitted first.

Resolution reduction in JBIG

Here we briefly recall the principles of the JBIG resolution reduction algorithm [JBIG1], which we will consider for generating the thumbnail images in our storage system described in Chapter 6. A series of images with decreasing resolutions is generated from the original image prior to compression. The process continues until size of the final, lowest resolution image becomes smaller than a predefined size. At each iteration, the input image is processed in the raster-scan order, and the value of each target pixel is calculated as a linear function of the preceding neighboring pixels from the high-resolution (input) and low-resolution (target) images. The already-committed pixels at the low-resolution image participate in the sum with negative weights that offset the corresponding positive weights.

Specifically:

$$L = 4x_{22} + 2(x_{23} + x_{32}) + x_{33} + (x_{11} - y_{00}) + 2(x_{21} - y_{10}) + \\ +(x_{31} - y_{10}) + 2(x_{12} - y_{01}) + (x_{13} - y_{01}) \tag{2.12}$$

Or equally:

$$L = 4x_{22} + 2(x_{12} + x_{21} + x_{23} + x_{32}) + (x_{11} + x_{13} + x_{31} + x_{32}) - 3(y_{01} + y_{10}) - y_{00}. \tag{2.13}$$

If black and white pixels are equally likely and the pixels are statistically independent, the expected value of the target y_{33} pixel is 4.5. A pixel is therefore chosen to be black if the value is 5 or more, and white if it is 4 or less.

The method preserves the overall grayness of the image. However, problems occur with lines and edges because these deteriorate very rapidly. To address this problem, a number of exception patterns have been defined to reverse the polarity of the target pixel after the thresholding of the weighted sum (2.13). An example of such an exception pattern is show in Figure 2-11.

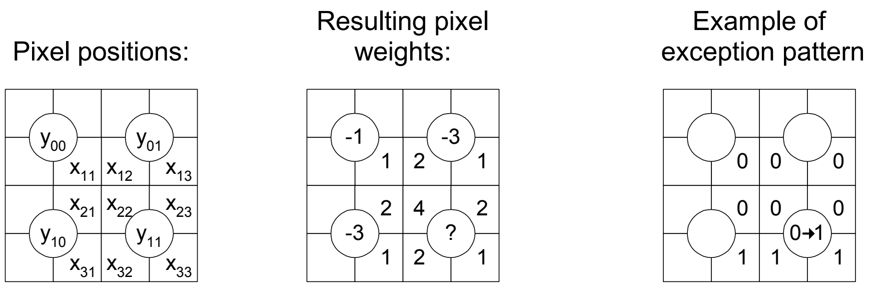


Figure 2-11. Resolution reduction in JBIG: participating pixels (left); pixel weights (center); an example of exception pattern (right).

3. Context Modeling

Binary images form a favorable source for context-based image compression because of the strong correlation between their neighboring pixels [LR81, TWMG93]. In context modeling, the pixel probabilities are conditioned on the context, which is defined by the combination of pixel color values within the local template.

3.1. Fixed Size Context Template

By default, JBIG1 uses the 10-pixel context template shown in Figure 3-1. This is referred to here as JBIG_{10}. With a 10-pixel template there are $2^{10} = 1024$ different contexts in total. Despite the high number of contexts, only a small fraction of them is really important. For example, in the case of the *CCITT-5* test image, 50 % of the code bits originate from only nine most common contexts. These most important contexts and their statistics are shown Figure 3-2. Furthermore, 99 % of the code bits originate from 183 contexts, and 429 out of the 1024 contexts are never used at all.

The context size is a trade-off between the prediction accuracy and learning cost (in dynamic modeling) or model overhead (in semi-adaptive modeling). A larger template size gives us a theoretically better pixel prediction. This results in a skewer probability distribution and lower bit-rates. However, with a large template the adaptation to the image statistics takes longer, which increases the coding deficiency in the early stage of compression, which is known as the *learning cost* [PM93]. The number of contexts grows as an exponential function of the template size, and the learning cost outweighs the benefit in compression for templates larger than 14 pixels, according to Moffat [Mof91]. In our experiments with different set of images, we have obtained higher compression rates for templates up to 18 pixels, but have noticed only marginal improvement for those templates greater than 14 pixels (see Figure 3-3). Moffat, on the other hand, has demonstrated the potential of even larger context templates up to 22 pixels. He has proposed two-level context modeling, using context templates of two sizes. The larger templates are used only when adaptation has been performed.

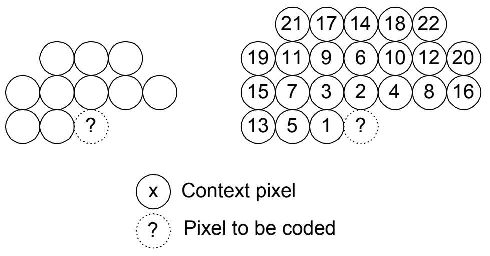


Figure 3-1. The default three-line context template of the sequential JBIG1 with default position of adaptive pixel (left), and 22-pixel *ordered neighborhood* used to determine an optimal context size (right).

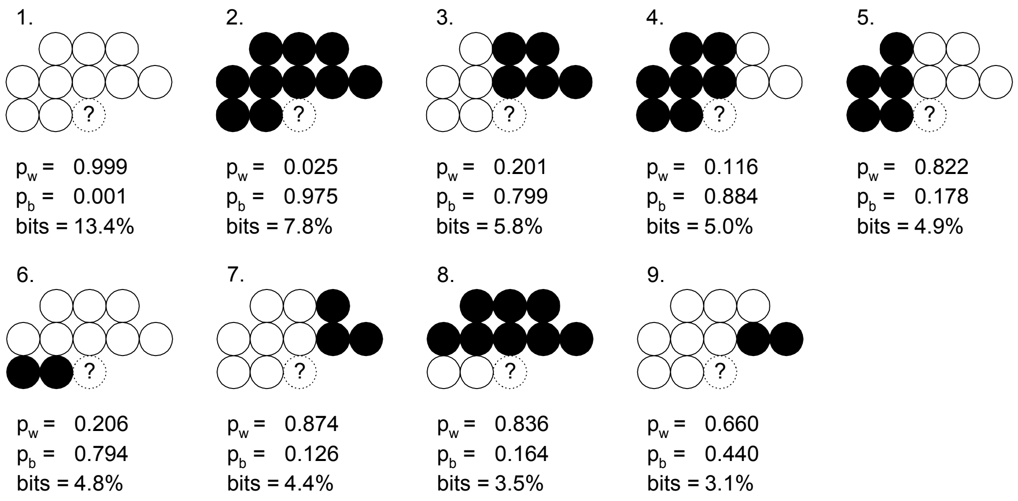


Figure 3-2. The most important contexts of JBIG1 in the case of *CCITT*-5 image at 200 dpi, according to [AF98].

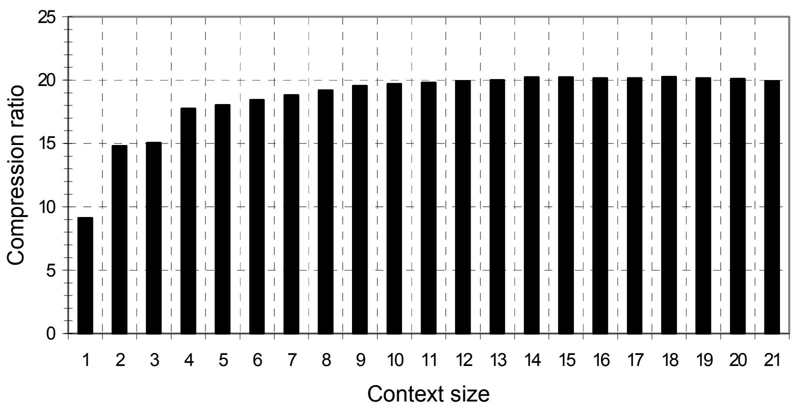


Figure 3-3. Compression performance as the function of context size (for CCITT images)

3.2. **Variable Size Context Model**

In variable-size context modeling, the number of context pixels depends on the combination of the neighboring pixel values. The context selection is made by traversing the *context tree* (CT) instead of checking a fixed size template [Ris83, Ris86]. Each node in the tree represents a single context, and the two children of a context correspond to the parent context augmented by one more pixel. The position of this pixel can be fixed in a predefined order, as shown in Figure 3-4, or optimized within a limited search area relative to the compressed pixel position [MF98]. We refer to the later case as a *free tree*. Only the leaves of the tree are used in the compression. An example of a context tree is shown in Figure 3-5.

3.2.1. *Splitting Criterion*

To construct a context tree, the image is processed and the statistics n_W^C and n_B^C are calculated for every context in the full tree, including the internal nodes. The tree is then pruned by comparing the children and parents nodes at each level. If compression gain is not achieved from using the children nodes instead of their parent node, the children are removed from the tree and their parent will become a leaf node. The compression gain is calculated as:

$$Gain(C, C_W, C_B) = l(C) - l(C_W) - l(C_B) - SplitCost, \tag{3.1}$$

where C is the parent context and C_W and C_B are the two children nodes. The code length l denotes the total number of output bits from the pixels coded using the context. The cost of storing the tree is integrated into the *SplitCost* parameter.

The code length can be calculated by summing up the entropy estimates of the pixels as they occur in the image:

$$l(C) = \sum_t \log p^t(C) \tag{3.2}$$

The probability of the pixel is calculated on the basis of the observed frequencies using a Bayesian sequential estimator:

$$p^t(C) = \begin{cases} p_W^t(C) = \dfrac{n_W^t(C) + \delta}{n_W^t(C) + n_B^t(C) + 2\delta}, & \text{if } t^{\text{th}} \text{ pixel is } \textit{white} \\ p_B^t(C) = 1 - p_W^t(C), & \text{if } t^{\text{th}} \text{ pixel is } \textit{black} \end{cases} \tag{3.3}$$

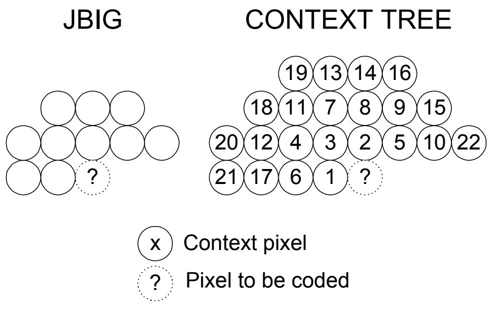


Figure 3-4.The 22-pixel *ordered neighborhood* used for the context tree (shown right). The first 10 pixels in the neighborhood constitute the default JBIG1 template (shown left). The template form and the pixel order in this example are optimized for topographic images [AF00b].

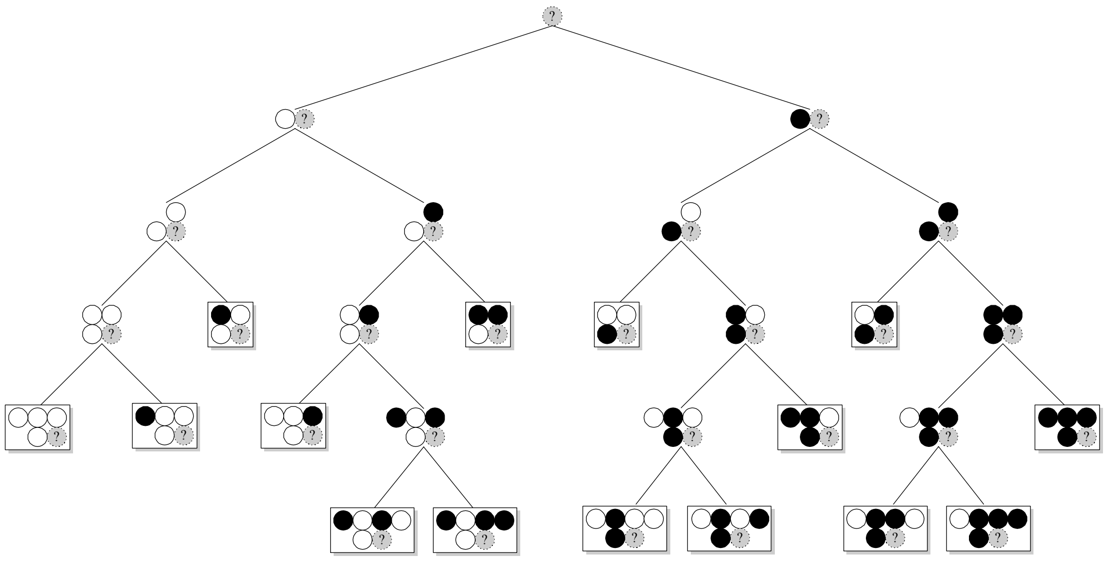


Figure 3-5. Illustration of a context tree. In practice, the context trees are much larger. But even this example can deliver significant compression performance.

where n_W^t , n_B^t are the time-dependent frequencies, and p_W^t , p_B^t are the probabilities for white and black colors respectively, and δ= 0.45, as in [JBIG1].

The code length can be efficiently calculated as the Bernoulli code length from only the final counts n_0 and $n_1$by using a fast approximate method as in [MF98]. However, if the tree is constructed off-line, we can accumulate the code length directly in the training phase by summing up the observed entropy estimates of (3.2).

3.2.2. *Static and Semi-adaptive Alternatives*

There are two alternative approaches for generating a context tree. In the *semi-adaptive* approach, the tree is optimized directly for the image that is be compressed. An additional pass (or passes) over the image will be required. The cost of storing the tree structure is one bit per node ('1' for indicating a divided node, and '0' for indicating a leaf node). This takes approximately 2 bits per context because the number of nodes in the three is twice the number of contexts (leaf nodes) minus one. For a free tree, the position of the next context pixel must also be stored. It can be represented as an index within the search area, and stored with $\lceil \log(window\_size) \rceil$ bits. The disadvantage of the semi-adaptive approach is that the on-line construction of the tree makes the compression an order of magnitude slower than JBIG1.

Another approach proposed in [FA99] uses a *static* tree, which is optimized on a training image [FA99]. This is possible because of the similarity of the trees with images of a similar type. The main problem of the static approach is to control the growth of the tree. There is no overhead from storing the tree and, therefore, we must add a progressively weighted constant to the *SplitCost* in order to prevent the tree from growing greedily.

3.3. **Top-down Tree Construction**

According to the direction of the pruning operation, the tree construction is classified either as *top-down* or *bottom-up*. In the top-down approach, the tree is constructed stepwise by expanding it one level at a time, starting from a predefined minimum level k_{MIN}. The process starts by constructing the models for all contexts at the level k_{MIN}. The contexts on the next level are tentatively constructed, compared to their parent contexts, and pruned. The process continues until a predefined maximum level k_{MAX} has been reached, or when no new nodes were created during the process of a single level. The top-down construction algorithm is outlined in Figure 3-6.

Another top-down approach is known as the *free tree* [MF98]. In this, the position of the next context pixel is not fixed during construction but it is determined adaptively. When a new level is constructed, all possible positions for the next context pixel are analyzed within a predefined search area. The position that results in maximal compression gain is chosen for each context separately. A drawback of this approach is that the position of the new context pixel must also be stored in the compressed file. The computational complexity of the free tree algorithm is an order of magnitude greater and it grows with a factor of the search area size.

```
ConstructContextTree (int $k_{MIN}$, int $k_{MAX}$)
    CONTEXTTREE CT;
    $k \leftarrow k_{MIN}$;
    CT ← GenerateTreeStructure (k);
    CollectStatistics (CT, k);
    repeat
        $k \leftarrow k + 1$;
        ConstructLevel(CT, k);
        CollectStatistics(CT, k);
        PruneLevel(CT, k);
    until ($k = k_{MAX}$ or no new nodes were created);
    return (CT);
```

Figure 3-6. Algorithm for top-down construction of the tree.

3.3.1. *Delayed pruning*

It may appear that a context delivers negative gain at some step of the iteration and it will not be expanded further, even though the expansion can deliver positive gain later. For example, let us consider the tree of Figure 3-7. The inclusion of the seventh pixel in the context only has a marginal effect on the model; the entropy values of all-white contexts at the first two levels shown in Figure 3-7 are practically equal. Therefore, the gain is overwhelmed by the learning cost because the third context pixel does not reduce the frequency of black pixels in the all-white context. The inclusion of the eighth pixel, on the other hand, provides a remarkable compression gain.

In [FA99] we have proposed delayed pruning technique to alleviate the *locality problem*. The tree expansion is not terminated directly after negative gain has been observed. Instead, the expansion is allowed to continue one level further. If neither of the children nodes produce an improvement in the case of a further split, the expansion is terminated and the children are actually removed from the tree. Delayed pruning is not applied in the case of deterministic contexts (when either of the counters n_1^t or n_0^t equals zero) because, in this case, a further improvement is impossible.

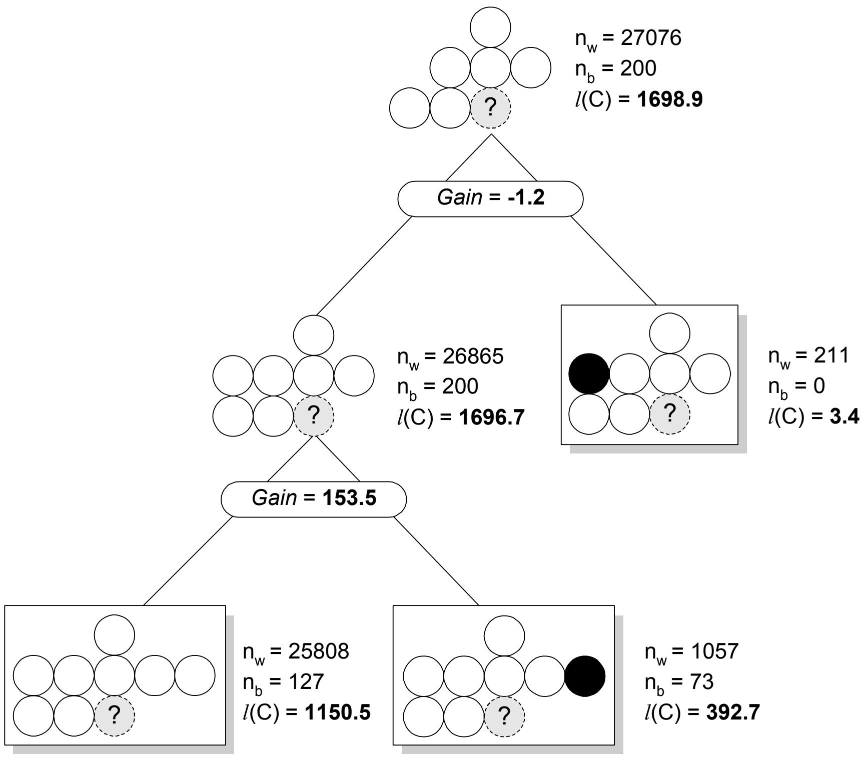


Figure 3-7. The locality problem of the splitting. The first split delivers negative gain (**-1.2** bits), though the second split provides remarkable gain of (**+153.5** bits).

3.4. **Bottom-up Tree Construction**

In the bottom-up approach, the tree is analyzed from the leaves to the root [AF00b]. A full tree of k_{MAX} levels is first constructed by calculating statistics for all contexts in the tree. The tree is then recursively pruned up to level k_{MIN}, using the same criterion as in the top-down approach. The gain is calculated using the equation (3.1) and code length $l(C)$ using (3.2). The code lengths from the children contexts $l(C_W)$ and $l(C_B)$ are derived from the previous level of the recursion. The sub-trees of the nodes that do not deliver positive compression gain are removed from the tree. A sketch of the implementation is shown in Figure 3-8 and the algorithm is illustrated in Figure 3-9.

```
PruneTree (CONTEXTTREE CT, int level)
  if (level = $k_{\text{MAX}}$)  // we have reached the end of tree
      return (CodeLength (CT ));
  else  // process the sub-trees recursively
      CLw ← PruneTree (CT→WhiteChild, level+1);
      CLb ← PruneTree (CT→BlackChild, level+1);
      if (level ≤ $k_{\text{MIN}}$)  // out of pruning range
          return (0);
      else  // check the node for pruning
          CL ← CodeLength (CT);
          Gain ← CL – CLw – CLb – SplitCost ;
          if (Gain > 0)  // split node
              return (CLw + CLb + SplitCost);
          else  // prune node
              RemoveTree (CT→WhiteChild);
              RemoveTree (CT→BlackChild);
              return (CL);
```

Figure 3-8. Recursive bottom-up tree pruning algorithm

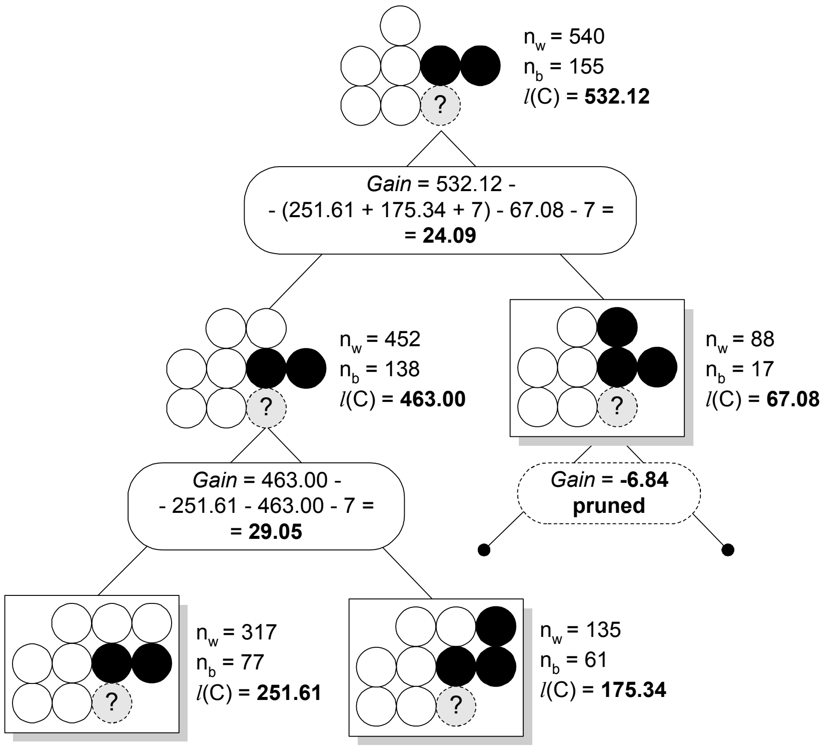


Figure 3-9. Illustration of bottom-up tree pruning

The bottom-up approach can be implemented using only one pass over the whole image. Unfortunately, high k_{MAX} values will result in huge memory consumption. For this reason, a two-stage bottom-up pruning procedure was proposed in [AF00b]. In the first stage, the tree is constructed from the root to level k_{START} and then recursively pruned until level k_{MIN}. In the second stage, the remaining leaf nodes at the level k_{START} are expanded up to level k_{MAX} and then pruned until level k_{START}. In this way, the memory consumption depends mainly on the choice of the k_{START} because only a small proportion of the nodes at that level remains after the first pruning stage. The starting level k_{START} is chosen as large as the memory resources permit.

3.5. **Combination of Variable-size Context Modeling and Forward-adaptive Statistical Modeling**

In forward-adaptive statistical modeling, the context size is a trade-off between the prediction accuracy and the overhead of the model. A larger context template results in a more accurate probability model, but the overhead grows exponentially with the size of the template. A proper choice of the context model is, therefore, even more important in forward-adaptive modeling than in dynamic modeling.

The variable-size context modeling can be efficiently combined with the forward-adaptive statistical modeling, as proposed in [AF00b]. In this technique (*CT-FAM*), the forward-adaptive model construction remains the same as in Section 2.7. The difference is that the context selection is made using a context tree. We apply this technique to the compression of small blocks of data (see Chapter 6 for details). We assume that if blocks are small enough, the fast attack states make an adequate approximation of the probability distribution of the pixels within the block. Therefore, we use the following equation to estimate the code length of a context in the splitting criterion of the context tree (3.1):

$$l(C) = n_W^C \cdot \log\left(\frac{n_W^C}{n_W^C + n_B^C}\right) + n_B^C \log\left(\frac{n_B^C}{n_W^C + n_B^C}\right). \tag{3.4}$$

In this way, the calculation of the code length is significantly faster than if the cumulative equation (3.2) had been used. It makes possible to construct the context tree on-line during compression. We also add the cost of storing the statistical model to the *SplitCost* parameter, providing the optimal tradeoff between compression improvement and overhead. The *SplitCost* is composed of the model cost (5 bits per context) and the cost of storing the tree structure (2 bits per context).

3.6. **Analysis**

We evaluate three different approaches (top-down, free tree and bottom-up) for building a context tree, see Table 3-1. In the comparison, we use the NLS test image Basic$_0$ (see Appendix E). For the free tree approach, the size of the search template is 40. The split cost is composed from the cost of storing the tree and the model (7 bits per context for context tree, and 12 bits for the free tree). The compression ratios are given for the CT-FAM method (see Section 3.5) when applied to a typical NLS binary map. The respective compression ratio of JBIG1 is 8.74, and the compression time is 1min 30s. We observe that the bottom-up tree construction is faster than the top-down approach but it requires more memory. For the bottom-up approach, the memory load grows exponentially with the size of the initial tree (k_{START}) but results in a larger tree and higher compression performance.

Table 3-1: A comparison of the tree-building strategies using Basic_0 NLS test image. The numbers in parenthesis are: (k_{MIN}, k_{MAX}) and (k_{MIN}, k_{START}, k_{MAX}) for 2-stage bottom-up pruning.

	Top-down		Free tree	Bottom-up		
	(6,22)	(10,22)	(2,22)	(2,18)	(2,22)	(2,18,22)
Contexts in the tree	1366	2373	2041	5596	8209	6527
Tree file size (bytes)	341	591	1786	1400	2053	1632
Passes over image	16	12	20	1	1	2
Creation time	30m 20s	26m 58s	1h 58m 33s	3m 8s	4m 56s	6m 31s
Memory load (bytes)	26K	51K	1M	8.5M	136M	8.5M
Compression ratio	10.04	10.40	11.30	11.14	11.65	11.44

The top-down construction of the tree can be performed with a small memory load (50 Kbytes) but it is very time consuming and, therefore, inapplicable for on-line compression. Another problem is that the expansion of some branches may stop too early because of the locality of the splitting criterion. The bottom-up method does not have this problem.

The free tree method does not give a significant improvement over the top-down approach with a fixed split pixel. The reasons for this are the high split cost, early termination of the tree expansion, and a limited search template (40 pixels). A delayed pruning technique and a significantly larger search template (about 500 pixels) could be applied to improve these results. However, it would cause a significant increase in memory consumption and running time, and is, therefore, not investigated here.

Bottom-up pruning requires only one or two passes over the image and gives better compression performance. The one-stage variant with $k_{\text{MAX}} = 22$ has the highest compression performance but the two-stage variant requires much less memory (8.5 Mbytes vs. 136 Mbytes). In the first stage, the tree is pruned from level 18 to 2. During this stage, 525,252 nodes are analyzed in total, and the number of leaf nodes is reduced from 256,548 to 5,596. Only 1,305 of these belong to the 18-th level. In the second stage, these nodes are expanded down to the 22-th level. In total, 20,880 nodes were analyzed and 2,236 new leaf nodes were created. Thus, most of the nodes are analyzed and pruned during the first stage.

4. Global Modeling

We define here the concept of global modeling as a process where the modeling is based on such global semantic information of an image, as could not be utilized by local context-based modeling. The extracted semantic features depend on the image type. They may be useful for better modeling, as well as in various applications containing image analyzing and understanding procedures, such as image segmentation, indexing, OCR and RVC [Kas90, KOG92].

An example of global modeling is JBIG2 [How+98, JBIG2]. It will include pattern matching techniques, used for extraction of common symbols from the image, and utilize this information in improved image compression. Our aim is to develop global modeling techniques that are appropriate for line-drawing images, which consist mostly of straight-line segments. We study two approaches of this type in Sections 4.2 and 4.3.

4.1. Pattern matching for text images and JBIG2

A text image contains many repeated symbols. Therefore, instead of coding all the pixels of every symbol occurrence, it is possible to code only the bitmap of one representative instance of the symbol. There are two encoding methods used in JBIG2: *pattern matching and substitution* (PM&S) and *soft pattern matching* (SPM).

The pattern matching and substitution method works as follows [AN74, WMB94]. The image is segmented into pixel blocks containing connected black pixels. These blocks are sequentially matched against representative symbol bitmaps from the adaptively constructed dictionary. If an acceptable match is found, the pointer to the corresponding bitmap in the dictionary and the position of the character on the page are encoded. If there is no acceptable match, the bitmap of the current pixel block is encoded using standard bitmap encoding techniques such as MMR or JBIG1, and added to the dictionary. The method allows high lossy compression levels, but results in infrequent but inevitable substitution errors. For cases where such errors are unacceptable, the *residue coding*, that is the refinement coding of lossy image back to the lossless original, or the SPM technique can be used.

The soft pattern matching differs from PM&S in that, in addition to a pointer to the dictionary and position information, it includes *refinement* data that can be used to recreate the original symbol, providing for lossless compression [How97]. The SPM method is illustrated in Figure 4-1. The only difference to PM&S (shown in *italics* in the figure) is that lossy direct substitution of the matched symbol is replaced by a lossless encoding that uses the matched character in the coding context. The refinement coding process is similar to the bitmap coding, with the difference that the two-layer context template is used. The template is shown in Figure 4-2. It consists of a combination of four neighboring pixels from the input block, and seven from the bitmap of the matching dictionary symbol.

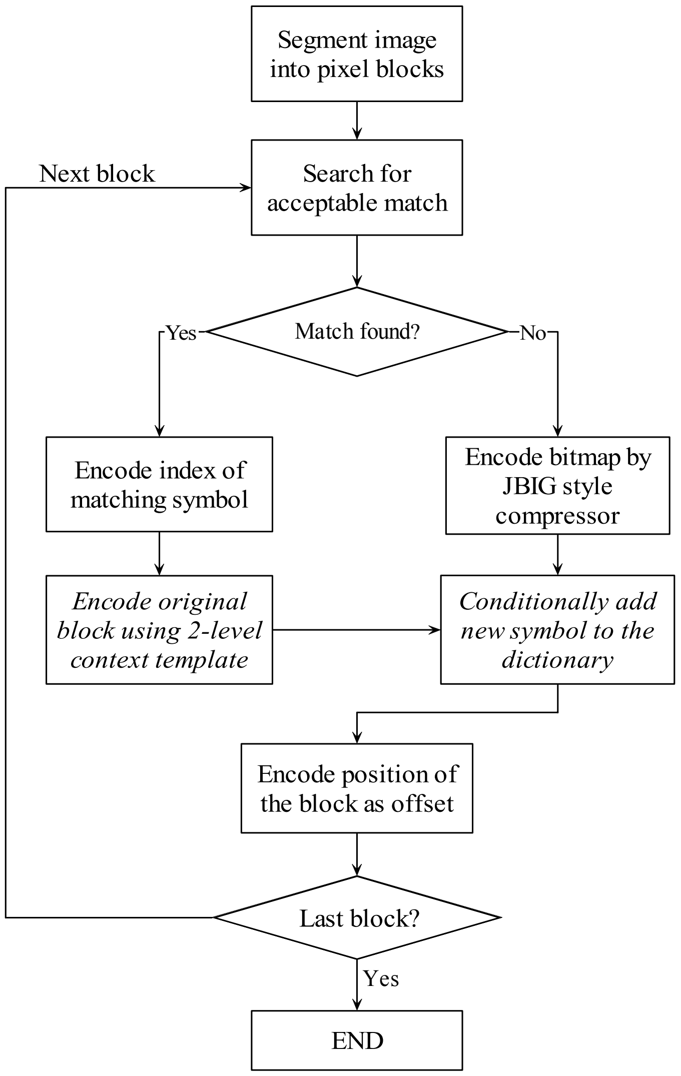

Segment image into pixel blocks
Next block
Search for acceptable match
Yes
Match found?
No
Encode index of matching symbol
Encode bitmap by JBIG style compressor
Encode original block using 2-level context template
Conditionally add new symbol to the dictionary
Encode position of the block as offset
Last block?
Yes
END

Figure 4-1. Block diagram of the soft pattern matching algorithm used in JBIG2 [How+98].

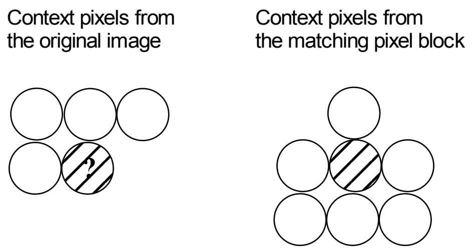

Context pixels from the original image
Context pixels from the matching pixel block
?

Figure 4-2. Two-layer context template for coding the pixel blocks [JBIG2].

The JBIG2 standard mainly defines the general file structure and the decoding procedure, but leaves some freedom in the design of the encoder. In effect, the decoder is guaranteed to be lossless in respect of the coded image data. However, the original image may be modified by the encoder during a preprocessing phase to increase coding efficiency. For example, some loss can be introduced by eliminating small pixel blocks that represent noise. This will improve compression efficiency while still maintaining a low probability of substitution errors [MF99]. If conditions permit, the symbol dictionary can be constructed and optimized off-line before the actual compression, so that the dictionary symbols are averaged among similar matching symbols, and infrequent symbols are pruned from the dictionary.

4.2. **Feature Extraction using Hough Transform**

The following two techniques are used for extraction of linear features from line drawing images. The first one uses Hough transform and is summarized in Figure 4-3. The motivation is to find rigid straight lines in the image. The extracted line segments are represented by their end-points and encoded into the feature file.

4.2.1. *Hough Transform*

Suppose that we have an image consisting of several samples of a straight line. Hough [Hou62] proposed a method (commonly referred to as *Hough transform*) for finding the line (or lines) among these samples. Considering a point (x_i, y_i), there is an infinite number of lines passing through it. However, they all can be described as

$$y_i = a \cdot x_i + b . \tag{4.1}$$

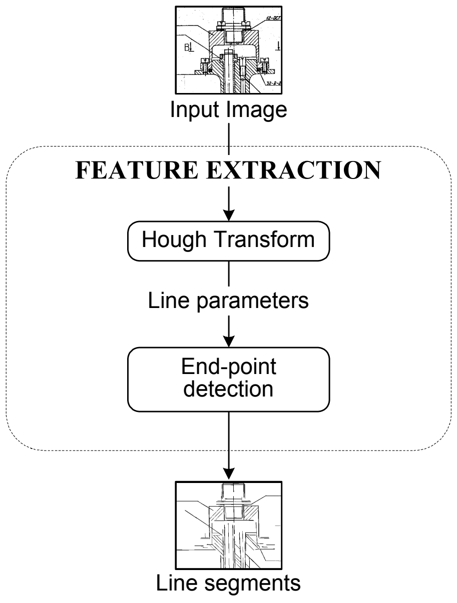


Figure 4-3. Block diagram of the feature extraction process.

It means that all the lines passing (x_i, y_i) defined by two parameters (a, b) can be expressed as

$$b = -x_i \cdot a + y_i. \tag{4.2}$$

The Hough transform is a process where each pixel sample (x, y) in the original pixel space is transformed to a curve in the parameter space, representing all-possible lines passing this pixel. If there is evidence of line presence in the image, which means that the pixel samples are located along the line, the Hough curves will intersect at the same point (a', b'), providing the parameter value for the line in question, as shown in Figure 4-4.

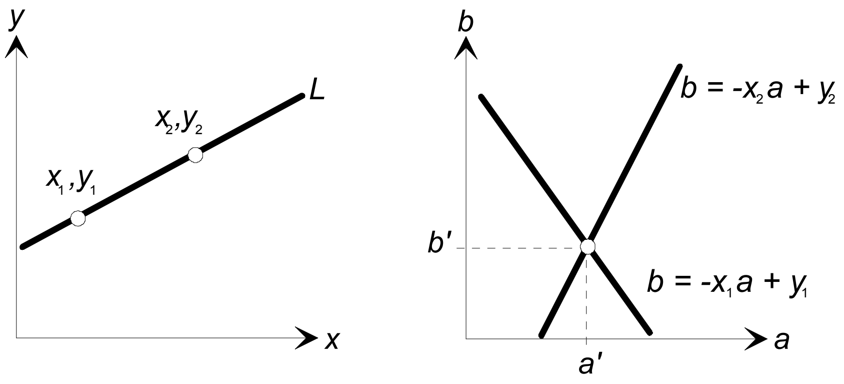


Figure 4-4. Hough transform: pixel xy-space (left), and parameter ab-space (right).

To implement the Hough transform, the parameter space is represented as a $k \times k$ accumulator array where k can be tuned according to the image size, see Figure 4-5. In each cell of the matrix, there is a counter of how many parametric curves are crossing that point. Each curve increases the counter of the cells located along its way. The lines are extracted from those positions of the array, for which the score exceeds a predefined threshold parameter.

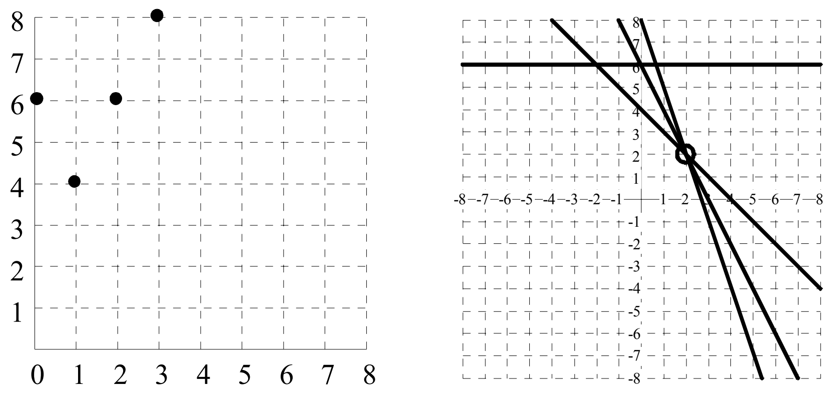


Figure 4-5. Small example of Hough transform.

A problem in this implementation is that both the slope (a) and intercept (b) approach infinity as the line approaches the vertical. One way around this difficulty is to use the normal representation of a line:

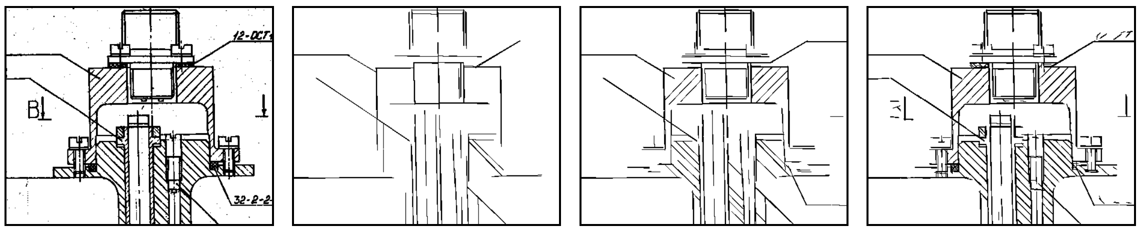
$$x \cdot \cos\theta + y \cdot \sin\theta = \rho, \tag{4.3}$$

were ρ represents the shortest distance between origin and the line, and θ represents the angle of the shortest path in respect to the x-axis. Their corresponding ranges are $\rho \in [0, \sqrt{2}D]$, and $\theta \in [-90°, 90°]$, where D is the distance between corners of the image [Lea93].

A drawback of the method is its high complexity. A straightforward implementation of HT requires $O(kn)$ time, where n is the image size and $k \times k$ is the size of the accumulator matrix. The method is therefore suitable for off-line applications, such as image archival. HT can also be made faster using the *randomized Hough transform* (RHT) as in [KHXO95]. Here, instead of processing individual pixels, the image is randomly sampled by selecting pairs of pixel. Each pair determines only one value in the parameter space. The sampling is repeated until an evident maximum is emphasized in the parameter space. RHT reduces the size of the parameter array and decreases the computation time since only a part of the pixels, possibly a small part is need to be transferred into the array.

4.2.2. *End-point Detection*

The Hough transform is capable to determine the location of a line as a linear function but it cannot resolve the end-points of the line. In fact, HT does not even guarantee that there exists any finite length line in the image but it only indicates that the pixels (x, y) along $y = a \cdot x + b$ may represent a line. The existence of a line segment must therefore be verified. The verification is performed by scanning the pixels along the line and checking whether they meet certain criteria. We use the scanning width, the minimum number of pixels, and the maximum gap between pixels in a line as the selection criteria. If predefined threshold values are met, a line segment is detected and its end-points are stored for later use. The features extracted with different parameter setup are shown in the Figure 4-6.



number of segments:	117 line segments	289 line segments	752 line segments
min segment length:	150	70	30
max segment width:	1	1	1
max length of a gap:	2	2	3
accumulator threshold:	20	20	17

Figure 4-6. Example of the feature images that are made using different parameter setup.

4.3. **Raster-to-Vector Conversion**

Raster-to-vector conversion (also known as *vectorizing*) process is outlined in Figure 4-7. The motivation is to extract semantic information from the image in the form of rigid line segments. Each line segment is represented by two end-points and the width of the line. The details of the vectorizing process are described in the following.

4.3.1. *Skeleton Construction*

The black-and-white raster image is first processed by a *distance transform* (DT) defined by 4-connectivity. We use the fast and memory efficient implementation of [KT95], which processes the image in smaller fragments. This eliminates the need for two passes over the image. The resulting distance labeled image is then thinned using the one-pass algorithm of [AB89]. Skeletal pixels are recognized by checking the 3×3 neighborhood of each pixel. The pixels satisfying one of the so-called "*multiplicity conditions*" are marked as skeletal pixels. The result of the algorithm is a width-labeled skeletal image.

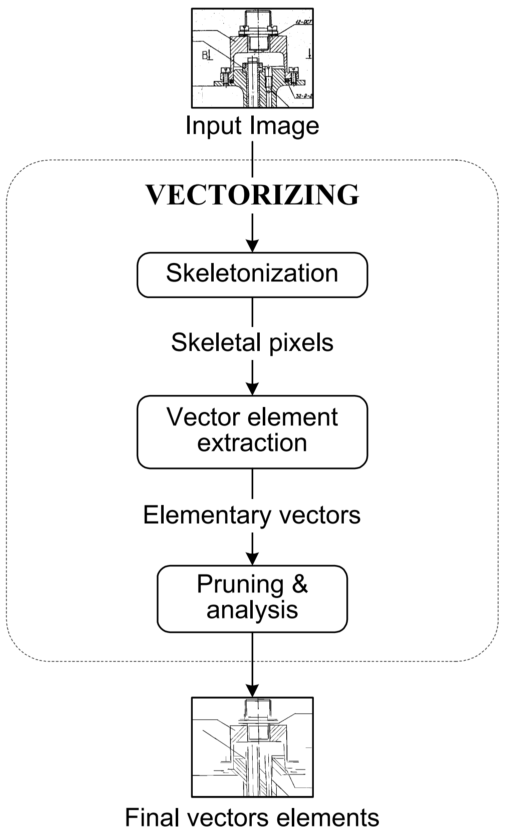


Figure 4-7. Block diagram of the raster-to-vector conversion process.

4.3.2. *Extraction of Vector Elements*

The vector elements are extracted from the skeletal image using a fast and simple line-tracing algorithm. The branches of the skeleton are traced pixel-by-pixel from one delimiter (line end or crossroad) to another, and stored as chain codes. The direction for tracing is derived from a pre-calculated two-dimensional look-up table (LUT), as in [KBO96]. The first index for accessing the LUT is the previous direction, and the second index is constructed from the 3×3 neighboring pixel values of the current pixel. The resulting chain code is then processed to produce piecewise-linear approximation of the branch with zero error as in [HK83]. The width of each line segment is calculated as the average width label of the skeletal pixels in the segment. The extracted segments of the same branch are stored as a chain of vector elements.

4.3.3. *Pruning and Analysis*

The extracted vector chains are further analyzed for constructing larger elements. There are four classes of vector chains, each described by the two end-points and the width of the line:

- Single point: (x_1, y_1, w_1).
- Single vector: (x_1, y_1, w_1), (x_2, y_2, w_2).
- Chain of n vectors: $\{(x_k, y_k, w_k) \mid k = 1,\ldots, n + 1\}$.
- Ring of n vectors: $\{(x_k, y_k, w_k) \mid k = 1,\ldots, n + 1\}$ where $x_1 = x_n$ and $y_1 = y_n$.

Vector elements are combined (pruned) from primitives having a common end-point and the same orientation. Small gaps between the lines are filled, and false branches are removed. The remaining vector chains are then classified as either “good” (linear) or “bad” (noise and non-linear). The good chains are stored by their coordinate differentials using a variable-length code.

4.4. **Hybrid Raster/Vector Image Representation and Modeling**

In a hybrid raster/vector storage system, both raster and vector representations of the images are encoded and stored [Wills99, FA+98a], see Figure 4-8. The raster representation provides an exact digitized replica of the original image. The vector representation contains semantic information extracted from the image. It benefits from vector editing capabilities and is suitable for further image processing and semantic analysis [KOG92]. The compressed file consists of the extracted line features and the compressed raster image.

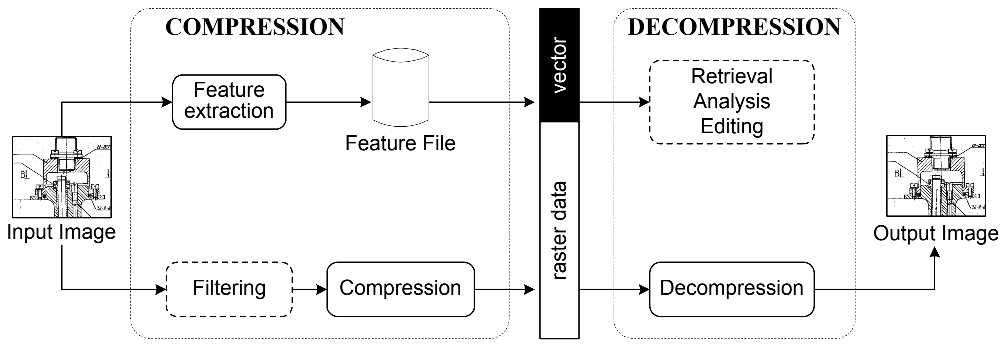


Figure 4-8. Block diagram of a typical hybrid compression system.

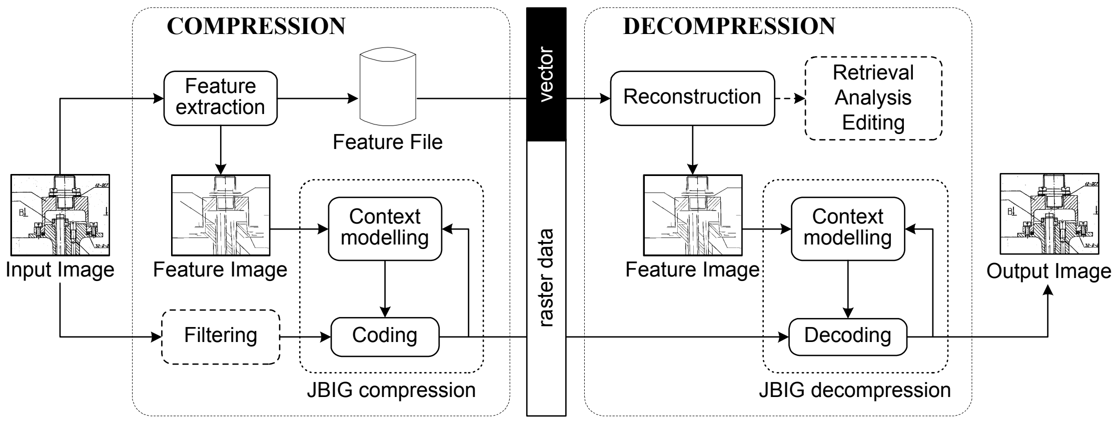


Figure 4-9. Block diagram of our hybrid compression system.

4.4.1. *Raster Representation*

The advantage of raster representation is that images can be easily digitized and stored compactly using the latest compression technology. Reproduction of the image is easy, and lossless compression guarantees that an exact replica of the original image can always be restored. Vector representation, on the other hand, allows better editing capabilities and resolution-independent scaling and reproduction. Complete raster-to-vector conversion, however, is not a realistic solution. The existing conversion systems are of high complexity and cannot reliably capture all possible vector features without human interaction. Either the file will be filled by a huge number of small vector elements, or some of the undetected information will be lost.

We consider here the storage problem of hybrid raster/vector systems. In an ideal situation, all linear features will be stored in vector format while the rest of the data remains in raster. In practice, only new data, drawn in CAD systems, will be stored in the vector format, while the remainder are kept in raster format because of the reasons mentioned above. The question is whether we can utilize the existence of the vector features for compressing the raster image more efficiently, see Figure 4-9.

4.4.2. *Vector Representation*

The vector representation may be obtained directly from the Hough transform (see Section 4.2) or raster-to-vector conversion (see Sec 4.3). The extracted line segments are stored in the form of $\{(x_1,y_1), (x_2,y_2)\}$, representing the end-points of the lines. A single coordinate value takes $\lceil \log_2 n \rceil$ bits where n is the dimension of the image. For example, a line in an image of 4096×4096 pixels takes $4 \times 12 = 48$ bits in total. A somewhat more compact representation could be achieved if the line segments are sorted according to their first coordinate x_1. Instead of storing the absolute value, we could store the difference between two subsequent coordinates x_1. Most of the differences are very small (about 40 % of them are in the range [0, 2]). An improvement of about 7 bits (from 12 to 5 bits) was estimated for the case when entropy coding was applied to these difference values [FAK99].

4.4.3. *Feature Image*

A *feature image* is an equal size raster approximation of the input image reconstructed from the extracted features. It represents the extracted semantic information in an easier form for modeling and filtering. The feature image is prepared by drawing the respective width (one-pixel if using HT) straight lines between the end-points of the line features. The Hough transform does not determine the width of the lines, and wide lines are represented by a bunch of collinear line segments, see Figure 4-10. The line segments may also deviate from their original direction and/or have one-pixel positional errors because of the quantization of the accumulation matrix. Therefore we do not utilize the feature image directly but process it first by subsequent operations of morphological *dilation* and *closing* [Hei94]. These operations make the lines one pixel thicker in all directions (dilation) and fill gaps between the line segments (closing). We apply a symmetric 3×3 structure element (*Block*) for the dilation, and a 3×3 cross structure element (*Cross*) for the closing, see Figure 4-11. The cross element is chosen to minimize the distortion in line intersections caused by closing.

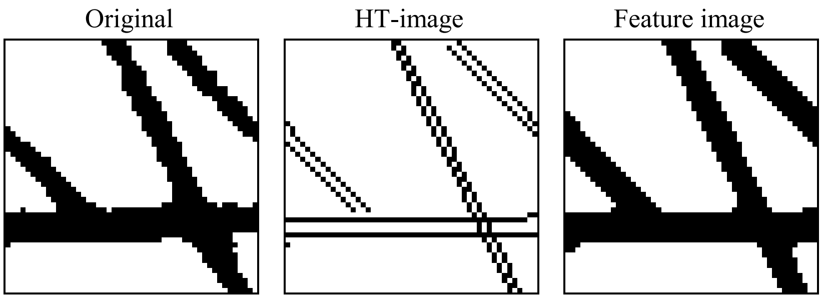


Figure 4-10. Illustration of the feature image for an image sample of size 50×50 pixels.

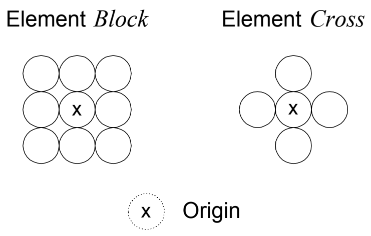


Figure 4-11. Structural elements *Block* and *Cross*.

4.4.4. *Hybrid Modeling*

Although the raster and vector data could be stored independently, the vector representation (feature image) can be used to improve compression of the raster image, see Figure 4-9. There are two basic approaches for utilizing the feature image: (1) lossless compression of the residual between the original and the feature image; (2) compression of the original image using the feature image as extra information. The first approach does not work in practice, because taking the residue destroys spatial dependencies near the borders of the extracted line features. The residual image is therefore not any easier to compress than the original one [WMB94]. On the other hand, the effectiveness of the second approach has been shown for textual images in [How97].

In [FAKK98a] we propose a new hybrid modeling method, in which the context is determined by combining the neighboring pixel values taken from both the input and feature images. We use 10 pixels from the original image to substitute into the three-line standard JBIG1 template, and five pixels from the feature image, see Figure 4-12. An important point is that in the feature image we can utilize even pixel locations that have not yet been processed, because the line features are already stored in the compressed file and are similarly available to the decoder.

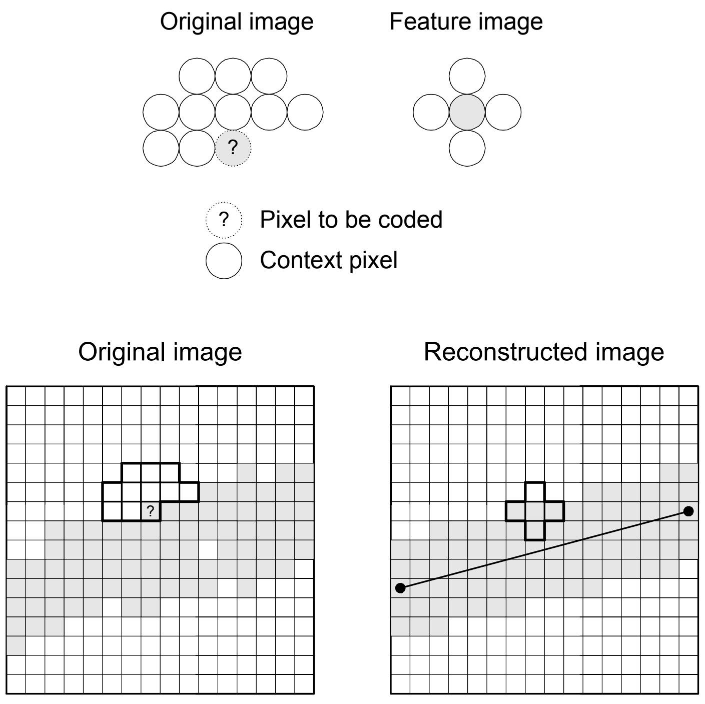


Figure 4-12. Illustration of the two-level context template.

A summary of experimental results for the hybrid compression of the line-images from Appendix D is shown in Table 4-1. “Simple RVC” stands for the raster-to-vector conversion as it has been explained above, whereas the “professional RVC” denotes the results obtained using one of the professional vectorizing systems available on the market. In our experiments, an improvement of up to 25 % is obtained. For a sample image from Figure 4-6, the amount of the raster and vector data in the compressed file is shown in Figure 4-13. The improvement is greater when more line segments are extracted. The amount of saving, however, is rather small and in all cases too small to compensate for the overhead required by the vector file. This is especially visible in the case of the simple raster-to-vector conversion; the method produces excessive feature files. The main reason for that is that the information of the extracted line features is mainly in all-black neighborhoods inside the line segments. These are the pixels that are already compressed well by JBIG1, and therefore only small improvements can be achieved. On the contrary, most of the information (output bits) originates from the boundaries of the objects. These areas are not well predicted by the local modeling of JBIG1, and global information is useful, especially if the input image is noisy. This emphasizes the importance of the exactness of the feature extraction. The vector elements do not provide reduction in the overall file size and their storage can be recommended argued only for their usage in hybrid editing and indexing tasks.

The speed of the methods discussed for Pentium-200 machines is summarized in Table 4-2. The HT-based feature extraction dominates the running time in the compression phase and makes it an order of magnitude slower. The method is therefore suitable only for applications where the compression operation can be made off-line. Simple RVC is fast, provides significant savings, but results in excessive vector files that cannot be directly used in CAD/CAM applications because of their coarse quality. The professional RVC performed on-demand can improve the compression of raster data by 1-10 %. Hybrid compression and decompression procedures are about 35 % slower than JBIG1 because of the additional processing of the vector features.

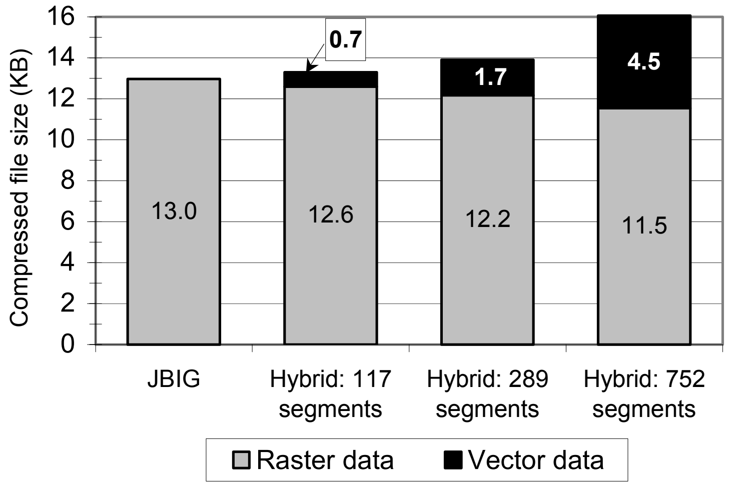


Figure 4-13. The amount of raster and vector data for the hybrid compression method using feature files with different level of details.

Table 4-1. Summary of the storage sizes for different hybrid compression methods (in bytes).

		Hybrid compression					
	JBIG1	Hough transform		Simple RVC		Professional RVC	
Image	raster	vector	raster	vector	raster	vector	raster
PLAN	5,098	2,370	4,578	7,932	3,889	8,077	4,556
HOUSE	15,688	13,398	13,961	26,640	12,109	10,495	14,786
CHAIR	52,384	16,710	50,140	134,143	38,385	56,695	48,328
MODULE	7,671	3,468	7,222	10,898	5,816	–	–
PLUS	17,609	5,268	17,132	34,885	13,204	–	–
BOLT	12,966	6,438	11,514	24,711	9,879	–	–
Total (I)	73,170	32,478	68,679	168,715	54,383	75,267	67,670
Total (I+II)	111,416	47,652	104,547	239,209	83,282	–	–

Table 4-2. Speed of different hybrid compression methods, kB/s for Pentium-200.

	Feature extraction	Compression/Decompression
JBIG1	N/A	196
HT-hybrid	0.676	149
SRVC-hybrid	180	149
PRVC-hybrid	*human interaction*	149

5. Image Enhancement and Noise Removal

The quality of document images may have faded during the document life cycle. Noise appears on the images because of such factors as low quality originals (e.g. old blue-prints), quantization errors in the digitization process, non-optimal light and contrast settings of the copying/scanning process, document transmission errors, paper defects, and dirty optical sensor systems. The noise degrades image quality and makes further image processing and analysis more difficult. Even though human eyes and modern OCR systems can tolerate some level of noise, it still introduces unnecessary details that weaken compression performance.

Noise appears in the images in two forms: *additive,* as randomly scattered noise pixels, and *content-dependent,* distorting the contours of printed objects (lines, characters) by making them ragged, see. Figure 5-1. Image enhancement is aimed at eliminating noise degradation and improving image compression performance, while still preserving image quality.

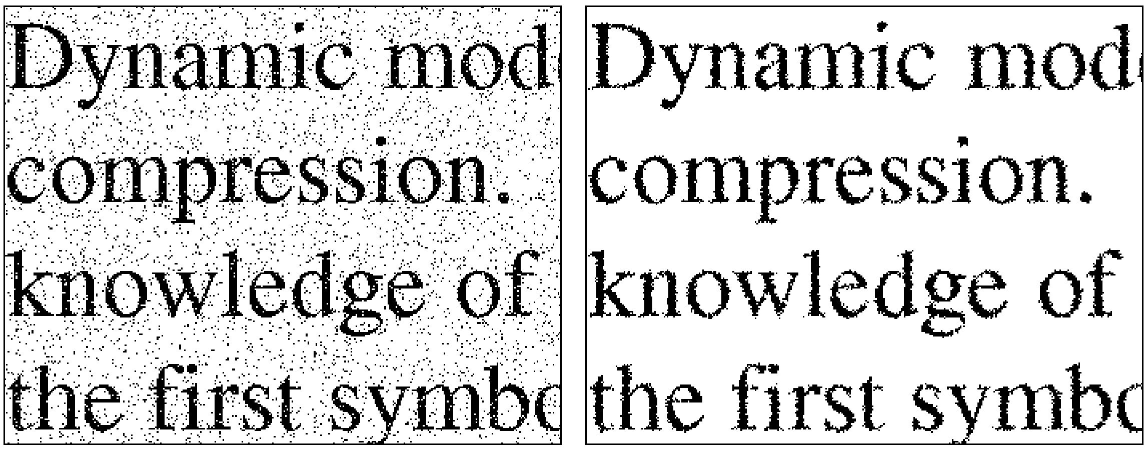


Figure 5-1. Illustration of the heavy additive noise (left) and content-dependent noise (right).

5.1. **Context-based Filtering**

The idea of context-based filtering is to apply statistical context modeling and measure the information content of the entire document image to achieve better selection of the noise pixels [DA97]. Although context modeling is commonly used in image compression, the primary aims of filtering and compression are not the same. This implies differences in the choice of the context template and in the way the statistics are collected. In compression, only the preceding pixels that are known to both the coder and decoder can be utilized in the context template. For filtering purposes, however, a directionally limited context template would not be accurate enough. As there are no limits for referring pixels in all directions, a symmetric filtering template can, therefore, be applied. The round shape of the template ensures even filtering in all directions.

We have chosen the 20-pixel filtering template, see Figure 5-2. It is a well-balanced trade-off between filtering performance, and reliability of the statistics. Using a larger context template, we could utilize spatial dependencies from a wider area, and in this way construct an even better statistical model. The number of contexts, on the other hand, increases exponentially with the number of pixels in the template. Therefore, further extension in the context template could lead to a context dilution problem, resulting in inaccurate statistics because of a lack of the context samples. The huge memory requirement is another limitation for using very large templates.

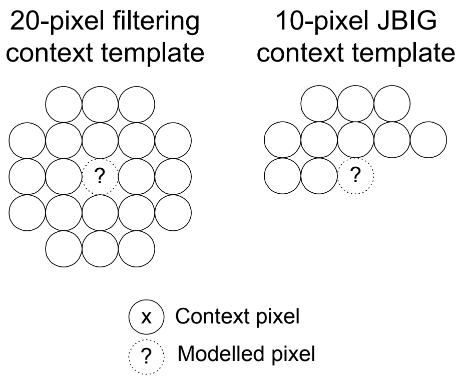

20-pixel filtering context template
10-pixel JBIG context template
?
?
x Context pixel
? Modelled pixel

Figure 5-2. The 20-pixel filtering context template (left) and 10-pixel three-line compression context template used in JBIG1 (right).

The second difference is that the compression is typically performed by a single pass over the image, and the statistics are adaptively determined during the compression. In the filtering, on-line adaptation to the statistics would make the result unreliable until the model adapts to the image. Instead, a two-pass scheme is applied to achieve uniform image filtering: one pass for collecting the statistics, and another one for the actual filtering.

Two context-based filtering methods, namely *Simple Context Filter* and *Gain-Loss Filter*, are proposed in [AF00a] for the enhancement of document images. The Simple Context Filter unconditionally changes the uncommon pixels in low entropy contexts, whereas the Gain-Loss Filter changes the pixels conditionally, depending on whether the gain in compression outweighs the loss of information. Both filters reduce irregularities in the image statistics caused by noise, and in this way, improve the compression without degradation of the image quality and OCR accuracy.

5.1.1. *Simple Context Filter*

A simple context filter is based on determining the statistical content of the image using context-based modeling, and flipping the pixels with low probability values, using the assumption that they are noise. The filtering process consist of two phases, each requires one pass over the image.

In the *analyzing phase*, context modeling with a 20-pixel filtering template is applied for the input image, and the number of black (n_B^C) and white (n_W^C) pixels for each context C and their respective probabilities are calculated:

$$p_W^C = \frac{n_W^C}{n_W^C + n_B^C}, \quad p_B^C = \frac{n_B^C}{n_W^C + n_B^C}. \tag{5.1}$$

After analysis, the contexts are categorized as *low information contexts* if the probability of either black or white pixel (p_B^C or p_W^C) does not exceed a predefined *threshold* value (e.g. 0.05). The probabilities are calculated on the basis of observed pixel frequencies. The less probable pixels in low information contexts are classified as *rare*, and most probable as *common* pixels.

The output image is generated in the *filtering phase* when all rare pixels in low entropy contexts are flipped. The *threshold* value is a trade-off between compression improvement and image degradation caused by filtering. The Simple Context Filter is illustrated in the Figure 5-3.

Statistics for the context and pixel shown in figure:

n_{BLACK} = 924

n_{WHITE} = 31255

p_{BLACK} = 0.0287

threshold = 0.05

Pixel changed to white.

Gain in compression = $= H(C)$ = 0.1879

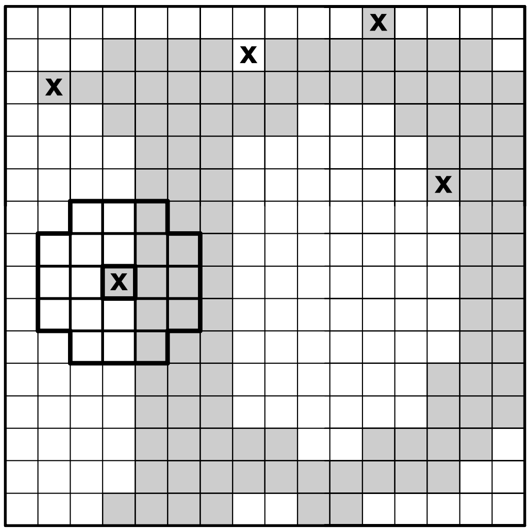


Figure 5-3. Examples of the Context filter. The original image is shown in gray, and changed pixels are marked with 'x'.

5.1.2. *Gain-Loss Filter*

In the previous filtering scheme, all pixels in the same context are processed in the same way, although resulting compression improvement may vary from pixel to pixel. Because of the difference in the context templates used for filtering and for compression, it is possible that the alteration of pixel value does not result in compression improvement. On the contrary, it may extend the code size.

To alleviate this problem, we have proposed a *Gain-Loss Filter* (GLF) in [AF00a]. Instead of using a simple probability threshold, as in CF, GLF takes into account the possible compression gain (*Gain*) as well as the error (*Loss*) caused by changing the pixel color during filtering.

Gain is defined by the impact on the code length caused by flipping the pixel x in context CC. It is composed of the direct effect of coding another pixel value and the effect of changing the context for all the pixels for which the flipped pixel appears in the context template [MF99]:

$$Gain(x) = Gain_0(x) + \sum_{y \mid x \in CC_y} Gain_y(x) \tag{5.2}$$

where $Gain_0$ denotes the direct effect of flipping x, $Gain_y$ denotes the impact on the code length caused by altering the contexts CC_y of all the pixels y for which x is the context pixel.

Gain$_0$ is calculated as the difference between entropy values (*H*) of the pixel *x* before and after its change:

$$\begin{aligned} Gain_0(x) &= H(x|CC) - H(\bar{x}|CC) = \\ &= -\log_2 p(x|CC) + \log_2(1 - p(x|CC)) \end{aligned} \tag{5.3}$$

where $\bar{x}$ is the flipped value of *x*. The context *CC* is obtained using the compression context template, see Figure 5-2, and the pixel probability is estimated as:

$$p(x|CC) = \begin{cases} p_W^{CC} = \dfrac{n_W^{CC} + \delta}{n_W^{CC} + n_B^{CC} + 2\delta}, & \text{if } x \text{ is } \textit{white} \\ p_B^{CC} = 1 - p_W^{CC}, & \text{if } x \text{ is } \textit{black} \end{cases} \tag{5.4}$$

where $\delta = 0.45$ as in [JBIG1], and n_B^{CC}, n_W^{CC} are the numbers of black and white pixels of the image in the context *CC*. Note that the probability is calculated on the basis of the statistics collected over the whole image, not the statistics at the moment, as in adaptive image compression. This is done to achieve equal filtering at the beginning and the end of the image. Thus, the learning cost and the possible compression improvement caused by local adaptation do not affect the filtering.

Gain$_y$ is calculated as the difference between entropy values of the pixel *y* before and after the pixel *x* has been flipped:

$$\begin{aligned} Gain_y(x) &= H(y|CC_y(x)) - H(y|CC_y(\bar{x})) \\ &= -\log_2 p(y|CC_y(x)) + \log_2 p(y|CC_y(\bar{x})) \end{aligned} \tag{5.5}$$

where $CC_y(x)$ and $CC_y(\bar{x})$ are the contexts of the pixel *y* before and after the pixel *x* was flipped, respectively.

Loss is defined as the amount of information that was lost when the pixel *x* was flipped. It is calculated as the entropy of the flipped pixel $\bar{x}$:

$$Loss(x) = H(\bar{x}|FC) = -\log_2(1 - p(x|FC)). \tag{5.6}$$

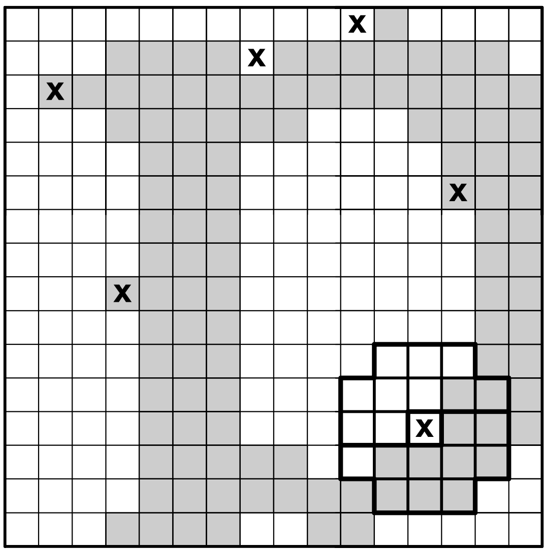
Here *FC* is the context obtained using the filtering context template, see Figure 5-2. The pixel probability is estimated either as p_B^{FC} or p_W^{FC}, as regards the color of *x*:

$$p(x|FC) = \begin{cases} p_W^{FC} = \dfrac{n_W^{FC}}{n_W^{FC} + n_B^{FC}}, & \text{if } x \text{ is } \textit{white} \\ p_B^{FC} = 1 - p_W^{FC}, & \text{if } x \text{ is } \textit{black} \end{cases} \tag{5.7}$$

Thus, changing the pixel value to the less probable one will result in the higher loss (values greater than 1), and vice versa. Changing one pixel value to another with equal probability (0.5), will deliver a unit of loss. The decision as to whether a pixel color should be changed is based on the following criterion:

$$\frac{Loss(x)}{Gain(x)} < Threshold \ . \tag{5.8}$$

The GLF requires two passes: analyzing and filtering. In the analyzing, the image statistics (n_B^{CC}, n_W^{CC}, n_B^{FC}, n_W^{FC}) for both context templates are calculated. After the statistics have been collected, the *Loss* is calculated for each context. The output image is generated in the second phase, where for each pixel *x*, the contexts *CC* and *FC* are obtained, *Gain* and *Loss* values are calculated, and the filtering criterion (5.7) is checked. If the threshold condition is met, the pixel is flipped and the statistics (n_B^{CC}, n_W^{CC}) for the altered compression contexts are updated. This update is performed because the encoder will process the already filtered image and these updated statistics will be the statistics used when the actual coding is carried out. The work of the GLF filter is illustrated in Figure 5-4.



Statistics for the context and pixel shown in figure:

$p(x|\text{FC})$ = 0.3750

$p(x|\text{CC})$ = 0.6255

$Loss(x)$ = 0.6781

$Gain(x)$ = 3.2378 - - 0.5013 = 2.7366

$Loss/Gain$ = 0.2478

$threshold$ = 0.5

Pixel changed to black

Figure 5-4. Examples of the Gain-Loss filter. Original image is shown in gray, and flipped pixels are marked with 'x'.

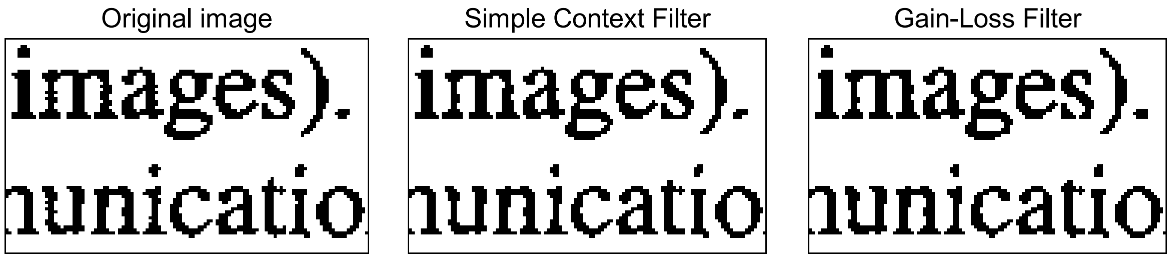


Figure 5-5. Example of the filtering with Simple Context Filter with threshold = 0.25, and Gain-Loss Filter with threshold = 0.5.

5.2. **Feature-based Filtering**

In [FAK99a] we have proposed a *feature-based filtering* technique for removing the quantization noise from digitized line drawings. The noise is a mixture of the additive and content-dependent noise. It can be visible as randomly scattered isolated pixels and jagged boundaries of the boundaries of the image objects (lines, symbols, etc). The proposed filtering technique is based on the semantic image modeling that utilizes the global spatial dependencies in the image. Line drawings consist mainly of straight-line elements, and global information can be gathered by extracting *line features*. The filtering is applied as a part of an image compression system, see Figure 5-6. The feature extraction and the filtering are considered as preprocessing steps before the compression. The noise removal improves the image quality and alleviates the loss in the compression ratio caused by noise.

We consider here two different approaches for the feature extraction. The first one is based on Hough Transform (see Section 4.2) as proposed in [FAKK98b]. The second one utilizes Raster-to-Vector conversion (see Section 4.3) as proposed in [FAK99a]. An equal size feature image is created from the extracted line segments to approximate the input image. We use the feature as a semantic model of the image (see Section 4.4.3).

The filtering is based on the noise removal procedure shown in Figure 5-7 (left). A *mismatch* image is constructed from the differences between the original and the feature image. Isolated mismatched pixels (and pixel groups of up to two pixels) are detected, and the corresponding pixel values in the original image are changed. This removes additive noise and smoothes the edges along the detected line segments. The quality of the filtering is by allowing only isolated groups of noise pixels to be changed. Objects that are not recognized by the feature extraction process are left untouched. The compression remains near-lossless because an uncontrolled loss of image quality cannot appear.

The noise removal procedure is successful if the feature image is accurate. The applied vectorizing method, however, does not always provide the exact width of the lines. The noise removal procedure is therefore iterated three times as shown in Figure 5-7 (right). In the first stage, the feature image is applied; in the second stage the feature image is *dilated*; and in the third stage, it is *eroded* before being input into the noise removal procedure. This compensates for most of the inaccuracies in the line width detection. See [Ser82, Hei94] for the details of the morphological dilation and erosion.

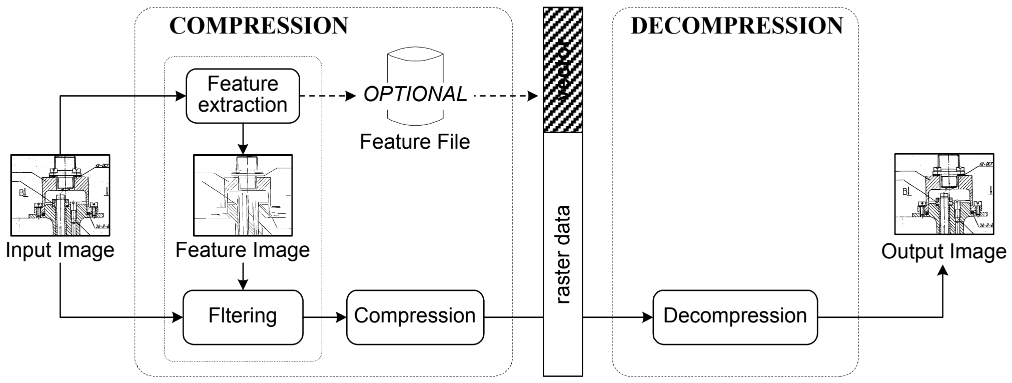


Figure 5-6. Block diagram of the three-stage compression method.

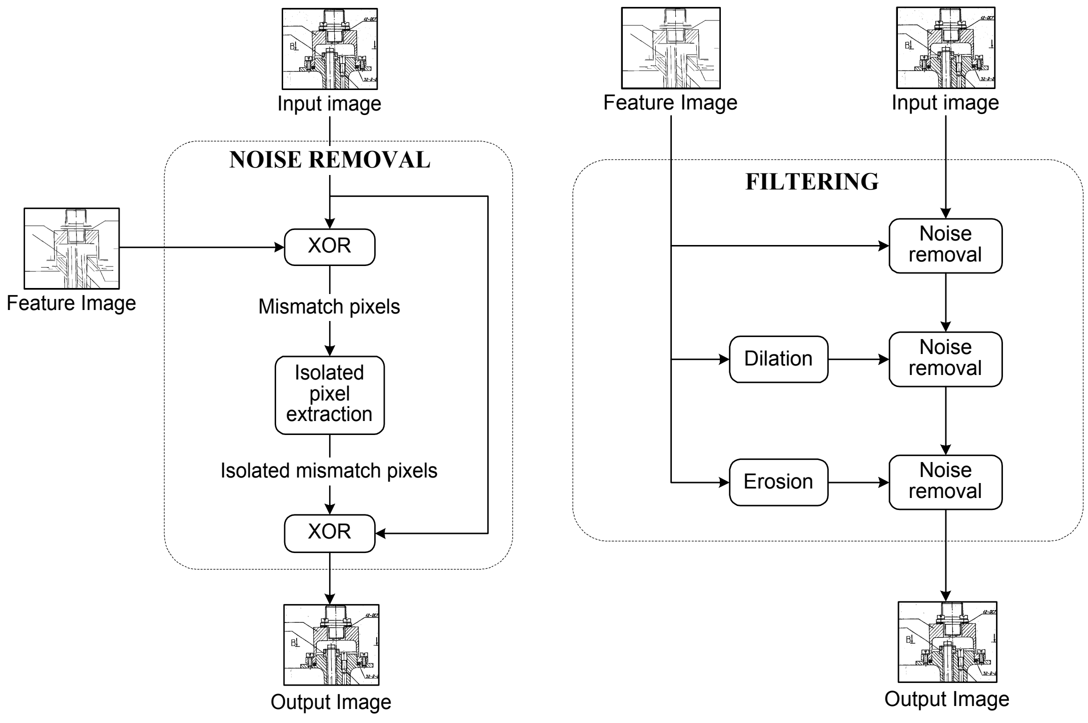


Figure 5-7. Block diagram of the noise removal procedure (left) and the entire three-stage filtering process (right)

The stepwise process for a small image sample is demonstrated in Figure 5-8 (using Hough transform) and Figure 5-9 (using RVC). Most of the noise is detected and removed in the first phase. However, in some cases there are too many mismatched pixels grouped together because of an incorrect estimation of the line width and therefore no pixels can be filtered. Even if these inaccuracies have a visually unpleasant appearance in the feature image, they do not necessarily prevent effective filtering. For example, the right-most diagonal line in the feature image in Figure 5-8 is too wide in some places and the pixels are therefore not filtered in the first two stages. The eroded version, however, gives a more accurate approximation of the line, and more noise pixels can be detected and filtered in the third stage.

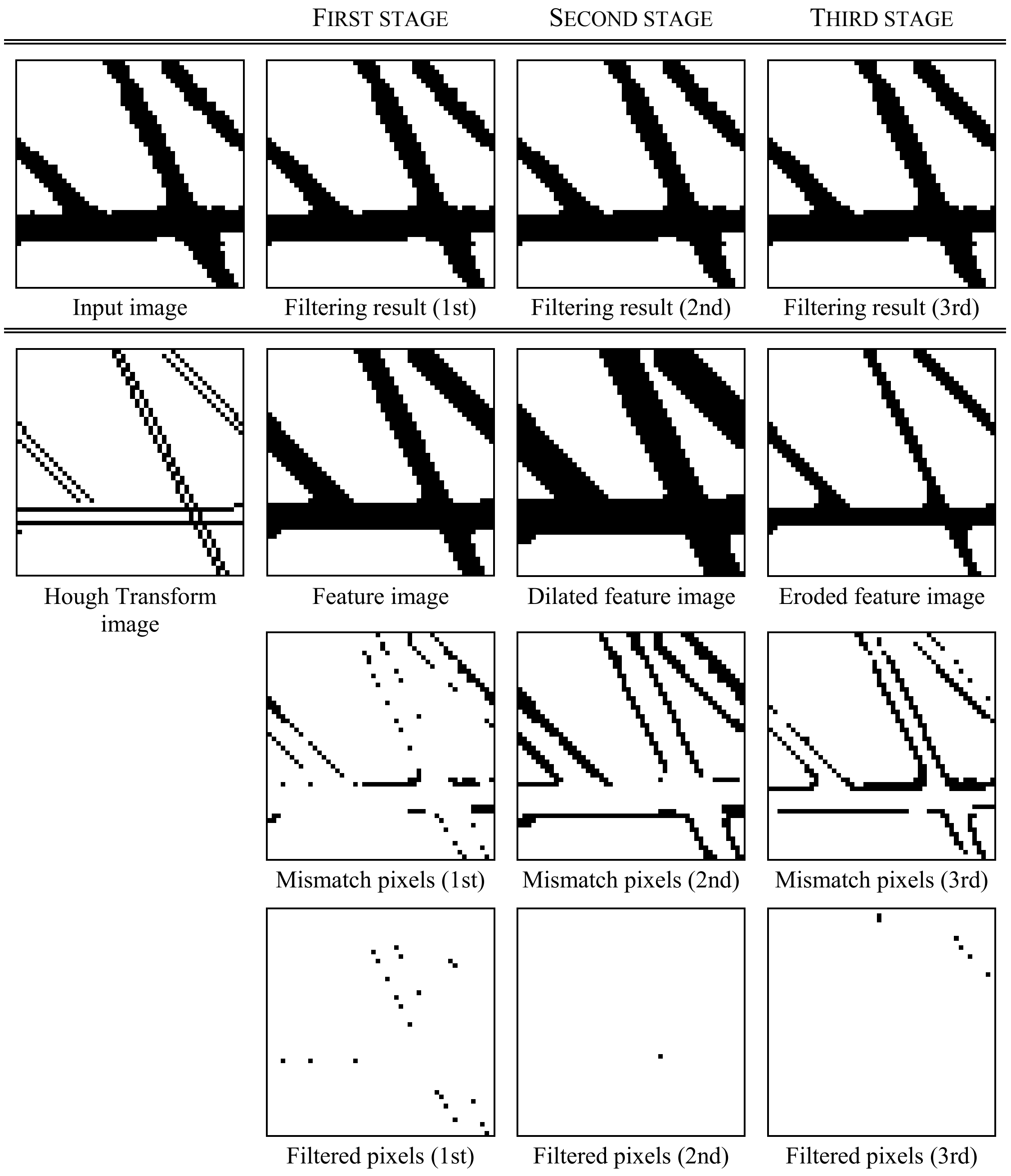


Figure 5-8. Illustration of the feature-based filtering using Hough transform

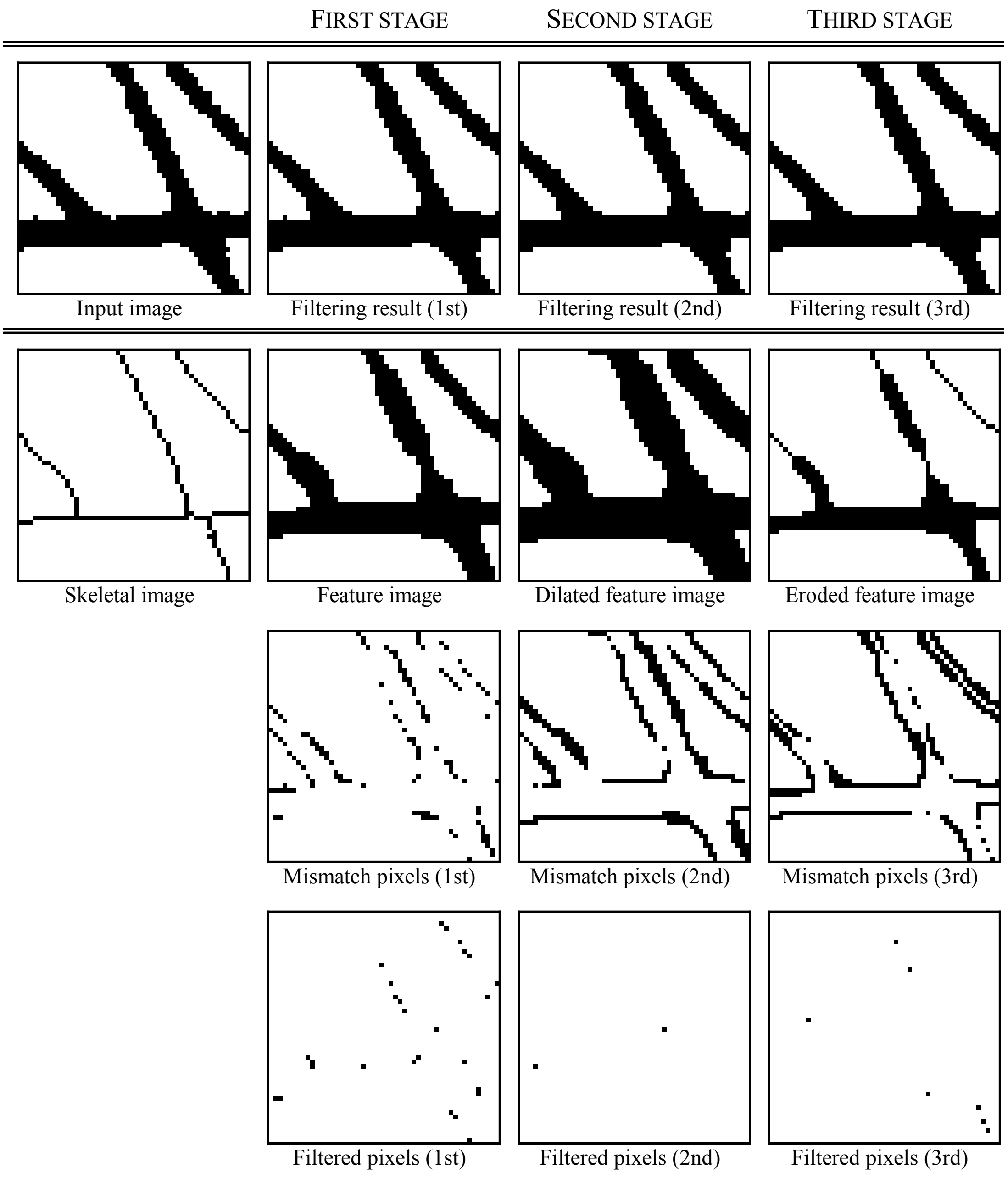


Figure 5-9. Illustration of the feature-based filtering using vectorizing algorithm.

6. Interactivity and Spatial Access

The actual image database may not be physically present at the viewing location, but is accessed through communication channels, which could be nothing more than a slow telephone connection. The compression reduces the amount of data to be transferred and makes the image retrieval faster. Next, we define important properties that must be taken into account in the design of the image archiving system.

The *Instant Preview* property enables the user to browse the archive without decompressing entire images. Preview represents a recognizable version of an image (*thumbnail*), using only a small portion of the compressed image data. It must be constructed and transmitted quickly enough to avoid inconvenient delays.

Fast Decompression: one might tolerate longer compression times if it can be performed off-line. On the other hand, the decompression process must be fast so that the system does not loose its interactivity because of decompression delays.

Spatial Access stands for direct access to an image fragment in the compressed file without having to retrieve and decompress the entire image. It enables efficient retrieval of the desired image fragments with high precision. Spatial access is the requirement for applications that deal with spatial data structures, e.g., digital spatial libraries and GIS.

When an image is accessed, the entire file is typically retrieved and decompressed into memory. However, memory resources sufficient to hold the entire decompressed image, and high-speed channels able to quickly transfer the entire compressed file, are not always available. At the same time, typical viewing devices have smaller size and resolution than the original raster image and thus, only a small fragment of the entire image may be viewed at a time. If spatial access is supported, an image may be interactively browsed on the viewing device. When the image is scrolled, a new part of data is retrieved and decompressed on the fly. In this way, spatial access eliminates time delays caused by image decompression and transfer. The thumbnail image may serve as a map to locate the desired part of the image at a higher scale.

6.1. **A Storage System for Interactive Access**

In [AF98] we propose a storage system that combines the compression methodologies presented in Section 2 with the properties that enable interactive access to the compressed images.

6.1.1. *Storage System Architecture*

The proposed storage system (see Figure 6-1) is based on JBIG1 with the following modifications:

1. Each image is segmented into separate *clusters* of $C \times C$ pixels. The clusters are compressed separately.
2. An *index table* of cluster pointers is constructed for locating the clusters in the compressed file.
3. Clusters are segmented into the blocks of $B \times B$ pixels, by the block modeling technique of [Fr94]. Block level codes form the *preview data*, which is stored at the beginning of the compressed file and used to build an image thumbnail.
4. Forward-adaptive statistical modeling is used to construct the initial probability model for the QM-coder. The model table is stored in the compressed file and is used for the coder re-initialization. The re-initialization reduces learning cost caused by small cluster sizes.

The structure of the compressed file is shown in Figure 6-2. Unlike [Fr94], the block and pixel level codes are not mixed, and the block level codes appear in the compressed file before the pixel level code. Thus, a thumbnail image can be constructed by reading the block level codes only. The pixel level data are stored sequentially, cluster by cluster. Any cluster can be reconstructed from the block level codes and the appropriate pixel level codes starting from the position given by the cluster index. The text header consists of an identification string and image parameters, such as image size, cluster size, block size, etc.

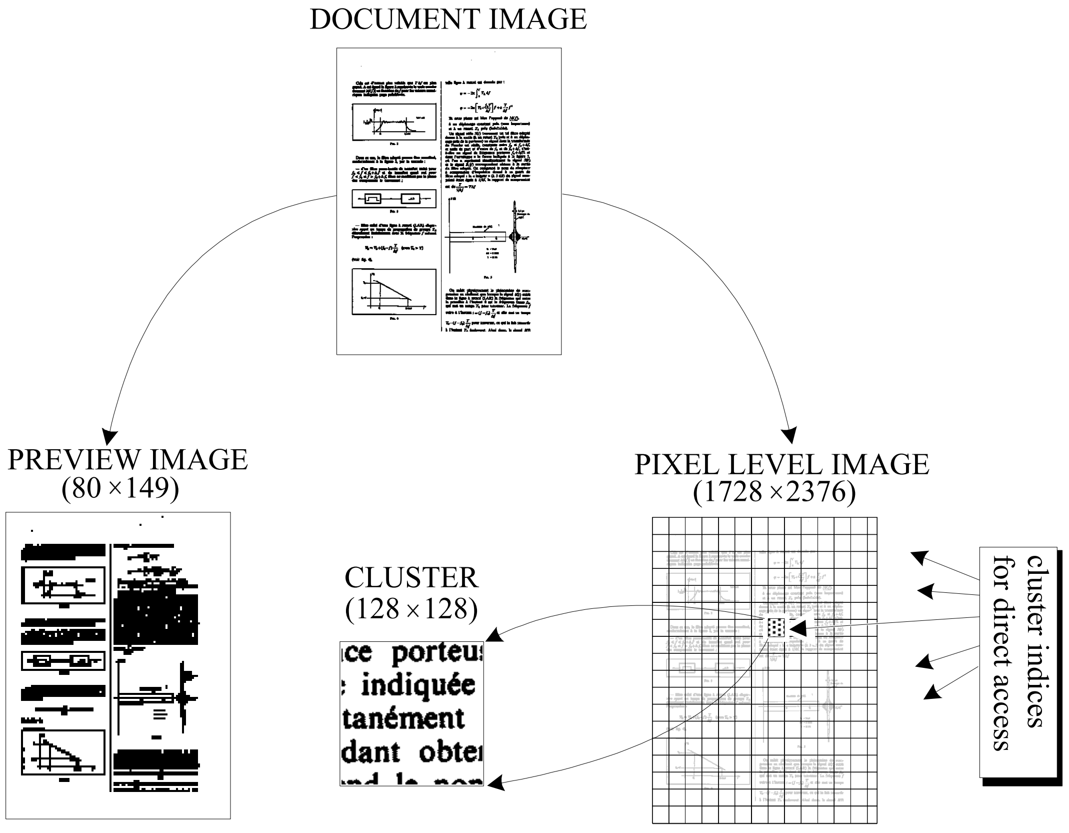

DOCUMENT IMAGE
PREVIEW IMAGE (80 × 149)
PIXEL LEVEL IMAGE (1728 × 2376)
CLUSTER (128 × 128)
cluster indices for direct access

Figure 6-1. Outline of the storage system.

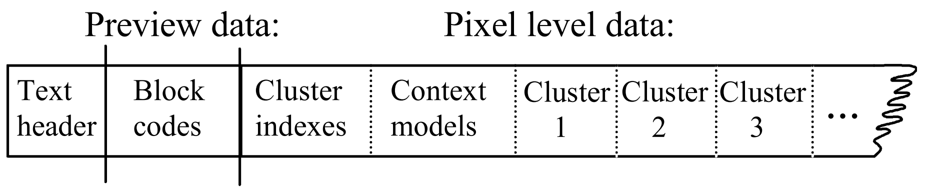


Figure 6-2. Organization of the compressed file.

6.1.2. *Compression Algorithm*

The compression algorithm is outlined in Figure 6-3. It includes forward-adaptive statistical modeling (see Section 2.7) to minimize learning cost when coding small clusters. The algorithm works in two phases, each making a pass over the input image. During the analyzing phase, the block types are determined, and the forward-adaptive statistical model for pixel level data is calculated. The header data is then stored, block types are compressed, and a context model table is stored in the compressed file. During the compression phase, the pixel level data are compressed. The QM-coder is used for compressing the block codes and pixel level data. When compressing pixel level data, each cluster is processed separately. The QM-coder is reinitialized and the model is restored each time the compression of a new cluster begins. The implementation details are discussed in the following subsections.

```
1. Analyze the image (analysis phase)
        1.1. Analyze block types and pixel statistics
        1.2. Construct initial model for pixel level compression
2. Write header and block level data
        2.1. Store text header
        2.2. Compress block codes using QM-coder
        2.3. Store dummy indexes
        2.4. Store pixel level model
3. Process each cluster (compression phase)
        3.1. Reinitialize QM-coder to initial model
        3.2. Compress the pixels with sequential JBIG1
        3.3. Record the starting positions of the next clusters
4. Terminate compression
        4.1. Replace the dummy indexes with the real ones
```

Figure 6-3. Main steps of the compression algorithm.

6.1.3. *Preview Data*

Two different techniques can be used to generate and encode the *thumbnail* (*preview*) of the image: (1) the *block modeling* scheme [FN93, Fr94] or (2) the *resolution reduction technique* of JBIG1. The block modeling works as follows. The image data is split into two separate levels: block level and pixel level codes. The block codes are obtained by dividing the clusters into smaller blocks of $B \times B$ pixels. Each block is classified either as an *all-white*, *all-black*, or *mixed* block. The block classifications are coded by two binary decisions shown in Figure 6-4. The all-white blocks are represented by a single 0-bit, all-black blocks by a bit sequence of 10, and mixed blocks by 11. The actual coding is performed by the standard QM-coder, using a second-order context model. The classification of the neighboring blocks to the left and above determines the context, yielding $2 \cdot 3^2 = 18$ different contexts for the block codes in total. The block types of the entire image are constructed during the model construction pass.

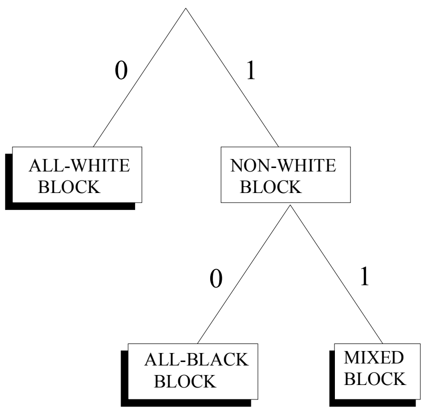


Figure 6-4. Decision tree for the block classification.

The block coding method has several advantages:

1. It is much simpler to implement than the progressive JBIG1 algorithm.
2. It requires only one pass over the data to generate a thumbnail image.
3. It does not significantly increase the bit rate because the pixels in uniform (all-white and all-black) blocks can be omitted without compression. Only the pixels of the mixed blocks must to be compressed. This fact mostly compensates the overhead due to the block codes.
4. The decrease in the pixel-level data also speeds up the decompression time by a factor of 2.5, on average.

The thumbnail image can be constructed from the block codes using a simple resolution reduction technique known as the *logical sum* method [MM87, EKY91]. Each pixel in the preview image represents a $B \times B$ block in the original image. The color of a pixel is white if the corresponding block type is all-white; and black, otherwise. This kind of preview is usually sufficient to identify the image. At the same time, the overhead remains marginal. A better quality of the preview can be obtained if the mixed blocks are exposed in gray.

The *resolution reduction technique* of JBIG1 can be used in the following way. The low-resolution thumbnail image can be a priori generated, independently compressed by a sequential JBIG1algorithm, and stored in the same file as the original compressed image. The comparative test data for these two techniques are presented in Section 7.4.

6.1.4. *Pixel Level Data*

The image is divided into fixed size clusters of $C \times C$ pixels. Each cluster is compressed separately. An *index table* is constructed from the pointers indicating where the data of each cluster is located in the compressed file. The index table is stored at the beginning of the compressed file. To restore any part of the image, only the clusters consisting of the desired pixels need to be decompressed. The cluster size is a compromise between compression efficiency and decoding delay; the smaller the clusters, the shorter the decoding delay but at the same time the overhead of the indices increases.

The cluster indices are coded by calculating the starting point of the cluster data relative to the previous cluster. The space requirement of the indices is known before the compression, and, thus, we can allocate enough space in the header. The actual indices, however, are not known until the entire image has been compressed. For this reason, the pointers can be stored at the end of the algorithm. The overhead cause by the indices remains rather small. However, very small cluster sizes will result in a relatively large number of clusters, and the compression of the cluster indices will be required to reduce the overhead. In the present method, we omit such compression schemes for simplicity.

6.2. **Spatial Access**

6.2.1. *Implementation*

Here we discuss three different choices available for implementing the image tiling, see Table 6-1 [AF99a]. In all cases, the clusters are compressed independently by using the QM-coder (as in JBIG1). The QM-coder is re-initialized each time when the compression of a new cluster starts. The difference is in the initial model used for re-initialization. These re-initialization options are:

- *zero-state* as in JBIG1 (*T-JBIG*);
- *forward adaptive model*, estimated for the image (*FA-JBIG*);
- *static model* estimated for a set of training images (*S-JBIG*).

Table 6-1**:** Implementation alternatives for spatial access (and JBIG1).

Method	Spatial access	Initial model	Passes
JBIG1	–	–	1
T-JBIG	+	–	1
S-JBIG	+	static	1
FA-JBIG	+	forward-adaptive	2

One problem with the straightforward combination of tiling and JBIG1 (*T-JBIG*) is the high learning cost. In JBIG1, the model is dynamically estimated during compression starting from scratch (*zero-state*). In principle, the adaptation is fast and the learning cost is restricted to the early stage of compression. However, the effect of the learning cost increases significantly when coding small clusters. A better initial model should, therefore, be applied to overcome the learning cost problem.

We propose the use of the forward-adaptive modeling technique, as described in Section 2.7. The method is a two-stage procedure consisting of (1) construction and storage of the initial model, and of (2) pixelwise compression of the clusters. The initial model is constructed of statistics gathered from the entire image. Once constructed, it is used for the re-initialization of the QM-coder's internal model. Otherwise, the coding is performed using the standard QM-coder routines. This technique alleviates the deterioration of the coding efficiency caused by tiling because of faster adaptation and smaller learning cost.

It should be also noted that the pixels of the neighboring cluster could not be used in the context template. The pixels outside the cluster are, therefore, assumed to be of the dominant image color (background color). After the cluster has been coded, the data buffer is filled with dummy bits to byte-align the cluster, and flushed to the code stream. Cluster indices are recorded and stored in the compressed file so as to indicate the starting points of the clusters in the compressed bit stream.

The forward-adaptive scheme requires two passes over the image even though the decompression can be performed with one pass only. A one-pass variant can be obtained using a static initial model, estimated off-line for a training image sequence. As a drawback, this technique would result in a less accurate initial model and, therefore, slightly higher learning cost.

The decompression is similar to the compression, except that a separate stage for constructing the initial model is not needed. Instead, the model is read from the compressed file.

6.2.2. *Analysis*

The forward-adaptive method improves the compression performance because the adaptation does not start from scratch but a pre-calculated model is used for the initialization. The re-initialization decreases learning cost and increases local adaptation further by pushing the models from slowly adaptive non-transient states back to the fast-attack states when the coding of a new cluster starts. These effects, for typical GIS images (see Appendix F), are shown in Figure 6-5.

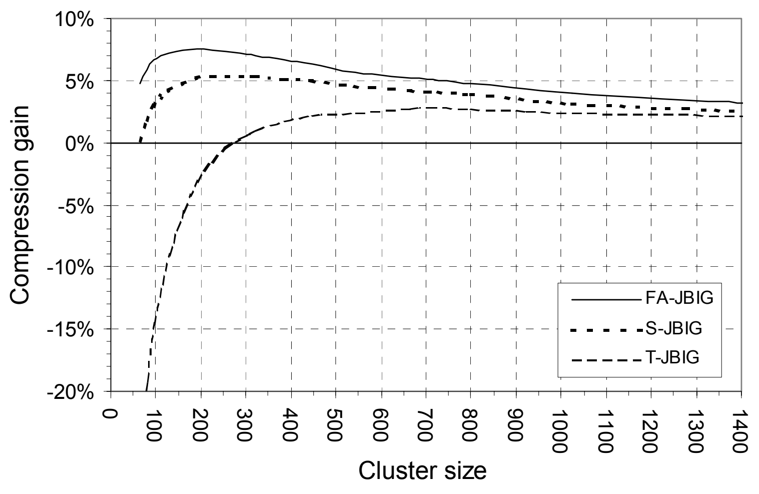


Figure 6-5. Illustration of the tiling and coder re-initialization effects on the compression as a function of cluster size (relative to JBIG1).

On the other hand, there are also several sources of deficiencies:

- overhead of the model table Ω_M;
- overhead of the cluster indices Ω_C;
- inefficient compression at cluster boundaries;
- inefficient disk access caused by fixed-size image partitioning.

We will measure the overhead by the number of extra bits relative to the JBIG1 compressed file size.

Model table overhead: The model table is stored in the compressed file using five bits per context. The total overhead for a *k*-pixel context template is thus:

$$\Omega_M = 5 \cdot 2^k \cdot \frac{1}{S_f} = 5 \cdot 2^k \cdot \frac{R}{X \cdot Y} , \tag{6.1}$$

where *X·Y* is the image size, S_f is the compressed file size, and *R* the compression ratio of JBIG1. The overhead is constant in respect to the cluster size. In the case of static initialization (S-M), the model table is not stored and it, therefore, causes no overhead.

Cluster overhead: Cluster indices can be stored compactly as the offset (in bytes) from the previous cluster location. In this case, the actual cluster index table will be reconstructed and held in memory. In the presented scheme, we use two bytes to hold an offset. It is enough to point clusters up to $2^{16} = 65536$ bytes (724×724 pixels). In the worst case, when no compression is achieved (theoretically possible), the cluster is stored as such without compression. This situation is indicated by a special cluster offset code #FFFF. Additional overhead originates from the dummy bits that must be added to the last code byte, which amounts to four bits per cluster, on average. The overhead is denoted as *cluster overhead* and it totals to 20 bits per cluster. The cluster overhead is calculated as:

$$\Omega_C = 20 \cdot N_C \cdot \frac{1}{S_f} = 20 \cdot \frac{R}{C^2} \ , \qquad (6.2)$$

where N_C is the number of clusters, and $C \times C$ is the cluster size.

Boundary overhead: Tiling the image also has the drawback that pixels outside the cluster cannot be used in the context template. The compression of pixels along cluster boundaries becomes less efficient and weakens the overall compression performance (this problem is referred to as *boundary overhead*).

To sum up, both the cluster overhead and the boundary overhead are inversely proportional to the cluster size. The boundary overhead is the dominant of these two. The model overhead, however, depends on the image size only, but it is relatively small for larger images. The overheads for typical GIS images (see Appendix F) are illustrated in Figure 6-6.

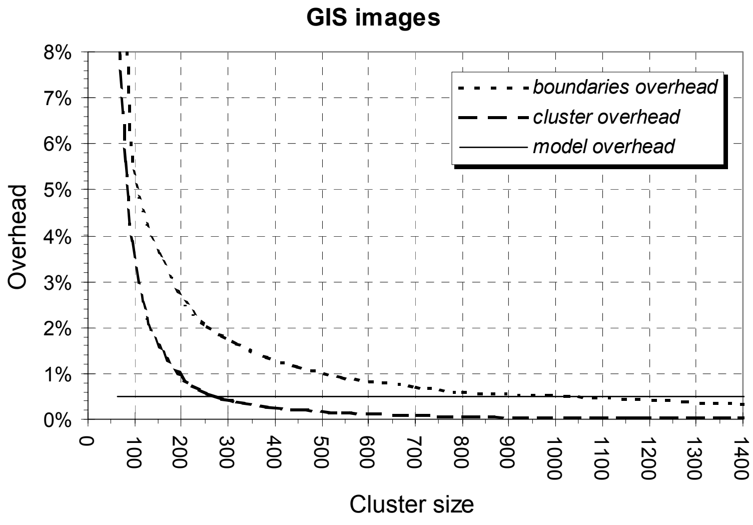


Figure 6-6. Overhead of the model, cluster and boundaries as a function of the cluster size.

Disk access inefficiency: The fixed image partition leads to regions whose compressed data may not fill entire data blocks of the database or disk, in which the compressed image is stored. If the compressed cluster data falls on the border of the physical data block, an additional data block may need to be accessed to retrieve the data. The disk access inefficiency increases when compressed clusters of the requested image fragment do not come together in a continuous stream but are broken into separate chunks, which are located in different data blocks, see Figure 6-7.

To avoid this problem, dynamic image clustering based on the compressed data has been studied in [PW96]. This strategy scans the image along an appropriate *space filling curve* [Jag90] and builds its regions according to the amount of compressed data filling one or more entire data blocks. This technique is not very applicable to context-based image compression, but the idea can be adapted for fixed-size image tiling, as follows. While the cluster content is compressed in raster-scan order, the clusters can be organized in the compressed file along the *Hilbert* curve [PW96]. In this case, the clusters that form the requested image fragment will most likely appear in the disk closer to each other, which will minimize disk access operations, see Figure 6-7.

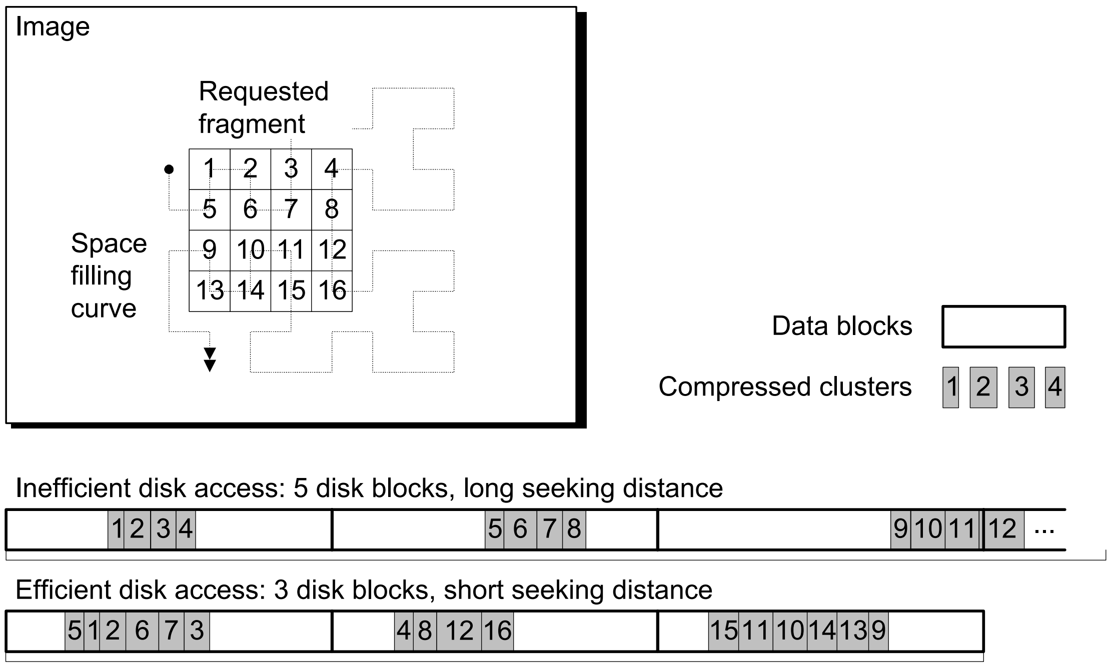


Figure 6-7. Illustration of the inefficiency of disk access.

7. Empirical Study

The performance of the proposed modeling techniques, filtering methods and storage system has been empirically evaluated. Various test sets were used, depending on the technique and its application. These sets include: the CCITT standard facsimile test set, digitized document and newspaper images, digitized textual documents, real-life line-drawing images of various categories, and cartographic and GIS images (see Appendix for test set details). The QM-coder has been used as an arithmetic-coding component. The compression results were compared with JBIG1. To report running times, we have used our high-level modular implementation of JBIG1. It takes about 10 s per 1 Mb of raw image data on a Pentium-200 machine, whereas the low-level hardware-optimized implementation may take about 2 s.

7.1. **Variable Size Context Modeling in Image Communications**

Here we study the use of variable-size context modeling for facsimile image communications [FA99]. In variable-size context modeling, the number of context pixels depends on the combination of the neighboring pixel values. The context is constructed by checking the pixel values in the positions given by the context tree instead of checking the pixel values in the fixed-size local template. Two modeling variants are evaluated: *Context tree* refers to the variant with a fixed order of the context pixels, and *Free tree* to the one with variable ordering. For these, we consider both the semi-adaptive and static approaches for constructing the tree. The static tree is generated off-line using a training image. The semi-adaptive tree is built on-line for the input image before the actual compression. The static tree, once constructed, can be used for the compression of multiple images. It eliminates the need for additional passes over the image required for semi-adaptive tree construction, and therefore makes the compression method suitable for facsimile communication. If the semi-adaptive approach is used, the cost of storing the tree must also be included in the splitting criterion (3.1). The additional cost is 2 bits per context for *Context tree*, and $2+\lceil \log(40) \rceil = 8$ bits for *Free tree* (with a 40-pixel search template). The rest of the parameter setup is given in Table 7-1.

Table 7-1: Parameter setup.

	Context tree	Free tree
k_{MIN}	6	2
k_{MAX}	24	24
Search template	–	40

The performance of the context tree is tested using two sets of A4-size images. The first set (*CCITT*) consists of the eight *CCITT* images scanned at 200 dpi (see Appendix A). The second set (*Newspaper*) consists of eight typical 300 dpi newspaper images containing variations of text and graphics (see Appendix B). For training, we use a separate newspaper image of the same size and resolution.

The comparative compression performance of the various context tree construction alternatives using a static approach is summarized in Table 7-2. The delayed pruning gives improvement in all cases.

Table 7-2: Effect of the delayed pruning in the static approach. The numbers give the amount of improvement in comparison to baseline JBIG1.

Test Set	*Context tree*		*Free tree*	
	normal	delayed	normal	delayed
Newspaper	8.5 %	10.9 %	11.1 %	14.5 %
CCITT	5.2 %	7.2 %	2.5 %	6.6 %

Table 7-3 shows that the static approach compares favorably with the semi-adaptive approach. It gives similar or better compression performance without the heavy computation in the compression phase. The actual running time of the static approach is about twice as long for the *Context tree* as for JBIG1, which takes about 30 s per page of document, or 1.8 times longer than JBIG1 for the *Free tree*. The semi-adaptive approach would require several minutes (*Context tree*), or several hours (*Free tree*), depending on the tree construction approach and the search template size (for *Free tree*). Of the two static variants, *Free tree* is preferred if applied to the images of the same type as the training image, cf. the *Newspaper* test set. For *CCITT* images, the *Context tree* variant gives slightly better compression rates.

Table 7-3: Comparison of the static and semi-adaptive approaches. The numbers give the amount of improvement in comparison to baseline JBIG1.

Test Set	*Semi-adaptive*		*Static*	
	Context tree	Free tree	Context tree	Free tree
Newspaper	8.6 %	13.4 %	10.9 %	14.5 %
CCITT	4.0 %	8.4 %	7.2 %	6.6 %

For experimental purposes, we adjust the number of contexts in the tree by using an additional *growth control parameter* (ω) in the node splitting criterion (3.1) as follows:

$$Gain(C, C_W, C_B) = l(C) - l(C_W) - l(C_B) - SplitCost - \omega, \tag{7.1}$$

Positive values of *w* will decrease the number of contexts in the tree and vice versa. The minimum number of contexts (with $\omega = \infty$) is $2^{k_{MIN}} = 32$. The maximum number of contexts (with $\omega = -\infty$) depends on the training image. For the *CCITT* images, it was 14,400 in the case of the *Context tree,* and 43,980 in the case of the *Free tree*, see Figure 7-1. The effect of the parameter, ω, on compression performance is shown in Figure 7-2. The *CCITT* images and the *Context tree* approach were used in this example. The optimal value for ω was found to be 0.1, although the default value (0) gave essentially the same results. The deep slope on the 'Context tree' curve in Figure 7-2 (the decrease in the compression ratio for very large numbers of contexts) is explained by the increased learning cost and context dilution effect.

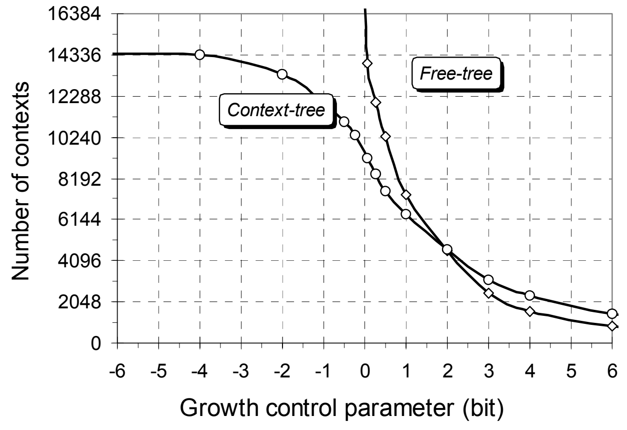


Figure 7-1. Number of contexts as a function of ω.

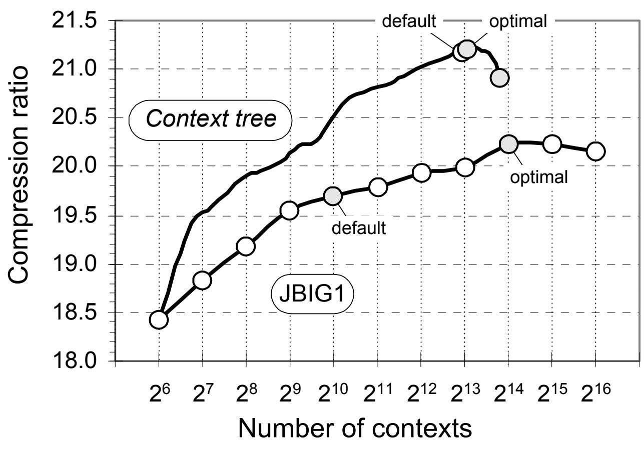


Figure 7-2. Compression ratio for the *Context tree* and JBIG1 methods (for *CCITT* images) as a function of the number of contexts. The number of contexts is controlled by the *growth control parameter* for *Context tree*, and by the context template size for JBIG1.

A summary of the test runs for the CCITT test set [ITU T.24] is given in Table 7-4. The results are for *Context tree* and *Free tree* techniques using semi-adaptive and static approaches. We performed the following tests to estimate the maximum, theoretically achievable improvement (shown in Table 7-4 as Max.) using variable-size modeling. The tree was constructed on-line as for the semi-adaptive variant, but neither the cost of storing the tree, nor the size of the tree was taken into consideration.

Table 7-4: Compression performance (bytes) for the CCITT test set. The methods are: sequential JBIG1 with standard 10-pixel context template ($JBIG_{10}$) and custom 14-pixel context templates ($JBIG_{14}$); Context tree and Free tree methods using static (static) and semi-adaptive (S-A) approaches, and theoretical maximum improvement (Max.).

Image	JBIG1		Context tree			Free tree		
	$JBIG_{10}$	$JBIG_{14}$	static	S-A	Max.	static	S-A	Max.
CCITT 1	14717	14618	14279	14402	13907	14465	14180	12889
CCITT 2	8500	8186	7678	8052	7445	7723	7664	7101
CCITT 3	21999	21263	20328	20694	19944	20870	19924	18847
CCITT 4	54300	52652	49393	50263	48591	48267	48012	42091
CCITT 5	25832	25239	24196	24587	23722	24213	23379	21694
CCITT 6	12561	12202	11287	11961	11156	11630	11200	10203
CCITT 7	56316	55116	53189	56670	52946	54480	53705	48660
CCITT 8	14238	13762	13021	13571	12906	12978	12935	12127
TOTAL	208463	203038	193371	200200	190617	194626	190999	173612
Improvement	–	2.6 %	7.2 %	4.0 %	8.6 %	6.6 %	8.4 %	16.7 %

Conclusion

The compression methods based on the context trees trained on a similar type image achieve a 14 % improvement over JBIG1 for a set of newspaper images. An improvement of about 7 % was obtained when the same context tree was applied to the CCITT test images. Most of the improvement originates from a selective context expansion. Larger context templates are utilized without overwhelming the learning cost. The compression takes about twice as long as if JBIG1 is used.

7.2. Context-based Filtering

The two context-based filtering methods described in Section 5.1 were applied to a set of digitized document images. The first method, the Simple Context Filter (CF) unconditionally flips the uncommon pixels in low entropy contexts. In the second method, the Gain-Loss Filter (GLF) flips the pixels conditionally, depending on whether the gain in compression outweighs the loss of information. We use two test sets in our evaluation. The first set consists of eight artificially generated document images, and has been used for estimation of the threshold parameters of the filtering methods. The test images originate from a document that has been typed using a text processing system and has been further transformed through printing, photocopying, faxing, and digitization (see Appendix C for details). The objectives of the first evaluation are to determine the effect of filtering on the compression performance and the OCR accuracy (*recognition error*) of the images after filtering.

The second large scale test set consists of real documents. The documents are taken from conference proceedings and contain text with a variety of fonts and offprint quality. The documents are digitized at resolutions of 300 and 400 dpi, resulting in 56 images of the size 2328×3028 and 3112×4038 pixels, respectively. These images are further referred to as *real document images*. This test set was used for the final evaluation of the filters.

To measure compression performance, we apply the sequential JBIG1 with the default (three-line ten-pixel) context template. We measure the improvement in compression, i.e. the difference in the file size before and after filtering, and compare it with the upper limit. The upper limit for compression is estimated by compressing the original noise-free document image, which has been generated from the *PostScript* file of the original document.

The Caere OmniPage 5.0 LE OCR software was applied, to recognize the textual content of the digitized images. This software offers good recognition rates (97.2 % on average) and is widely available [Has98]. The resulting textual files were compared with the original text. Recognition error was measured as the *edit distance* under the unit cost model, that is, the minimum number of edit operations (e.g. symbol changes, insertions and deletions) required for transforming a given text back to the original [SK83]. All spaces between words and paragraphs were counted as one *space* symbol.

Filtering performance

To estimate the upper limit of compression improvement by image filtering, we compress the *noiseless* document image, which is generated directly from the PostScript output of the text processing system. The size of the 300-dpi noiseless image has been found to be 58,773 bytes, and 36,763 bytes for the noiseless 200-dpi image. The upper limit for compression improvement is therefore estimated as 21.5 % for 300-dpi images and 22.4 % for 200-dpi fax images. The effect of the threshold parameter on the compression and recognition of document images is illustrated in Figure 7-3. For 300-dpi document images, the threshold value for GLF can be set to 0.5 without any noticeable effect on the recognition error rate. At this point, the method has almost achieved the estimated maximal improvement (20.6 % vs. 21.5 %). The simple context filter, on the other hand, weakens the OCR accuracy much sooner and it never reaches the maximal compression improvement. The best result remains around 10 % for the threshold value of 0.1.

For the fax image, CF always weakens the recognition accuracy, and the compression improvement remains rather small (below 7 %). GLF achieves an improvement of about 10 % (with a threshold value of 0.1) without producing defective recognition accuracy. Filtering with higher threshold values would achieve further compression, but it starts to have a significant impact on recognition accuracy. Text typed in 10-point *Times* and digitized at 200 dpi makes the letters too small and coarse to be recognized, and does not allow a distinct statistical description, which is necessary for the filtering, to be built.

The compression and the recognition error rates for the original and filtered images are summarized in Table 7-5. Typical recognition errors are illustrated in Table 2. They originate mostly from two distinct symbols adjacent to each other (such as Th is mistaken for either ’A or Ml, mn is often confused with nm, cl with d, and so on), or due to character similarity (like l, /, and i for example). To sum up, no new serious errors were found, with one exception, where digit 8 is interpreted as 9 in both the original and filtered versions of one document.

Table 7-5: Compression performance of JBIG1, and the recognition errors for the original (noisy) and filtered images using the GLF filter (with threshold = 0.5).

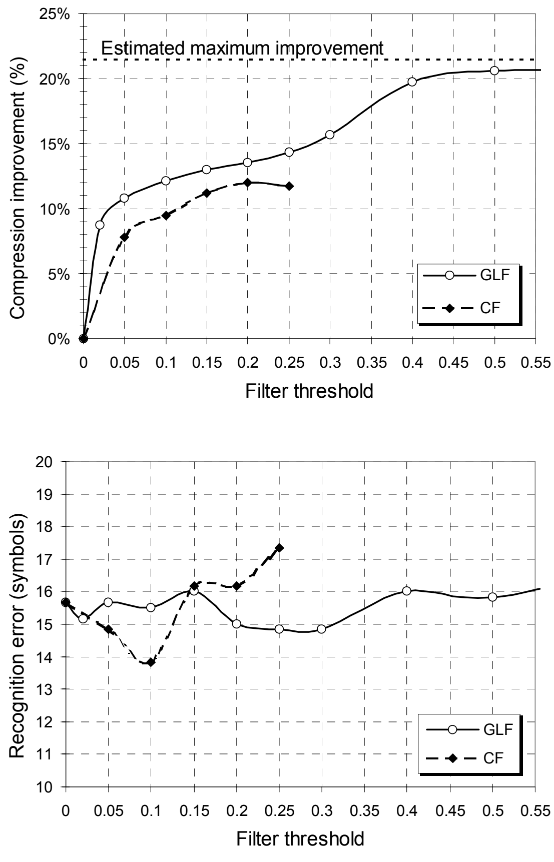
Image	Measured value	Laser print	Ink print	HiQ* copy	LoQ* copy	Dark copy	Bright copy	Fax (fine)
Original	Recognition error	4	7	9	13	28	33	9
	Compressed size	68,545	83,832	74,396	72,454	77,825	72,131	47,360
Filtered	Recognition error	4	9	12	7	24	39	8
	Compressed size	54,511	64,537	59,374	58,980	61,809	57,455	42,904

* HiQ and LoQ stand for High-quality and Low-quality copies made by using high and low resolution copiers, respectively

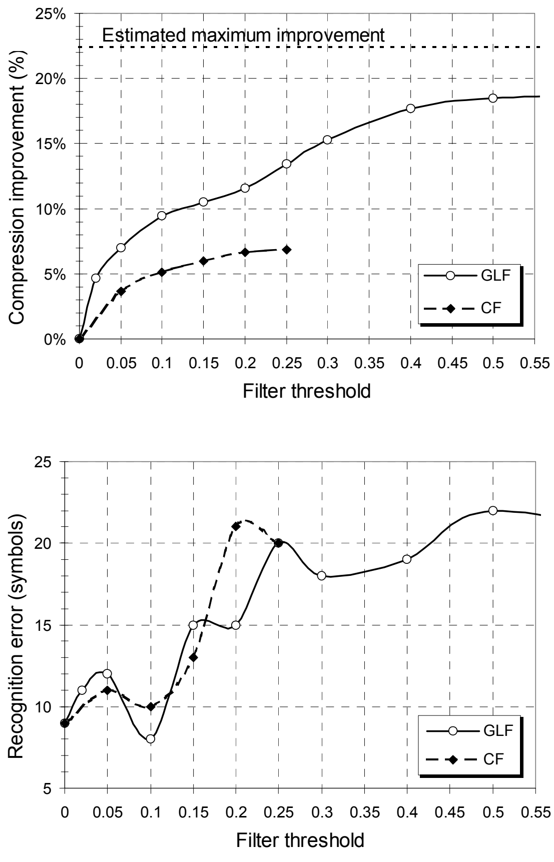
Table 7-6: Illustration of the recognition errors for two digitized images, using the Gain-Loss filter with threshold = 0.5. The total number of errors in the document is shown in parentheses. Mismatched symbols are typed in bold, and unrecognized symbols are marked as '∅'.

Font, size	Low-quality copy		Ink printing	
	Before filter (13)	After filter (7)	Before filter (7)	After filter (9)
Times, 12-pt.	unco**m**pressed unco**rn**pressed		and/or and**l**or	and/or and**l**or
	and/or and**l**or	and/or and**l**or		
Times, 10-pt.	unco**m**pressed unco**rn**pressed			ava**i**lable ava**f**lable
	2**8**,800 2**9**,800	2**8**,800 2**9**,800	**Th**e **'M**e	**Th**e **Al**e
	satel**l**ite satel**f**ite		and/or and**l**or	and/or and**l**or
	and/or and**l**or	and/or and**l**or		
	. ,			
Arial, 10-pt.	and**/**or and**l**or	and**/**or and**l**or	and**/**or and**l**or	and**/**or and**l**or
	"on-the-f**ly"** "on-the-f**i∅'**	"on-the-f**ly"** "on-the-f**i∅'**	**"**on-the-fly**"** **'**on-the-fly**'**	**"**on-the-fly**"** **'**on-the-fly
				lines **f**ines

The evaluation results for the filtering methods applied to the second test set (real document images) are shown in Figure 7-3. We have used the threshold values optimized for test set 1 (0.1 for CF and 0.3 for GLF) to ensure maximal compression improvement with minimal affect on the recognition error. The experiment shows approximately the same rate of compression improvement as was obtained using the generated images.



a) Average for 300-dpi document images



b) 200-dpi fax image

Figure 7-3. Compression improvement and recognition error rates of the Gain-Loss (GLF) and Simple Context filters (CF) as a function of the threshold. Compression improvement is reported as the difference of the compressed file size before and after filtering.

Traditional non-statistical filtering methods

Traditional non-statistical filters (such as median and morphological filters) are based on the shape, or quantitative analysis of a local neighborhood area using a set of predefined rules. They result in much smaller compression improvement and less accurate image recognition. We have tested the various filters reported in [TP80, B87, AG91, AKS90, Hei94, ZD96, DA97], including the median filter, self-dual operator with different rank, alternative sequential filters, and soft morphological filters. Among these, the best results for the 300-dpi test images (from set one) were achieved with the *median filter* (10 % improvement, recognition error of 19 symbols), and with the *self-dual rank operator* (11 %, 28 symbols). The median filter flips the pixel if the majority of the pixel colors in the neighborhood are of the opposite color. Its *soft* generalization, *self-dual rank operator* [Hei94], flips a pixel if the number of the opposite color pixels exceeds a predefined threshold value. For the fax image, only the median filter was able to keep the number of errors at a tolerable level (25 symbols), delivering 8.8 % improvement in compression.

Conclusion

The filtering method based on context-based statistical modeling of the image was proposed for enhancement of the document images for compression and recognition. Context-based filtering does not depend on the semantic interpretation of the image and is solely based on the statistical properties of the image. The method removes the digitization noise from the images and alleviates losses in the compression performance caused by noise. A 15-20 % improvement in compression performance has been achieved, while the image quality and OCR accuracy have been preserved.

7.3. **Feature-based Filtering**

Here we study the feature-based filtering method (outlined in Section 5.2) as a pre-processing stage in an image compression system, which uses either of the two standard compression components, JBIG1 or the older, but still widely used ITU Group 4 [ITU T.6]. The method is based on the flipping of isolated pixel groups found in the difference (mismatch) between the original image and one that is reconstructed from extracted vector features. Filtering examples are shown in Figure 7-4. In these examples, the pixel-level noise is mainly filtered out, but some of the roughness remains along the line boundaries. It consists of large groups of noise pixels that are not filtered by the proposed method. Symbols and other small and non-linear elements are not completely detected, and therefore parts of them may not have been processed.

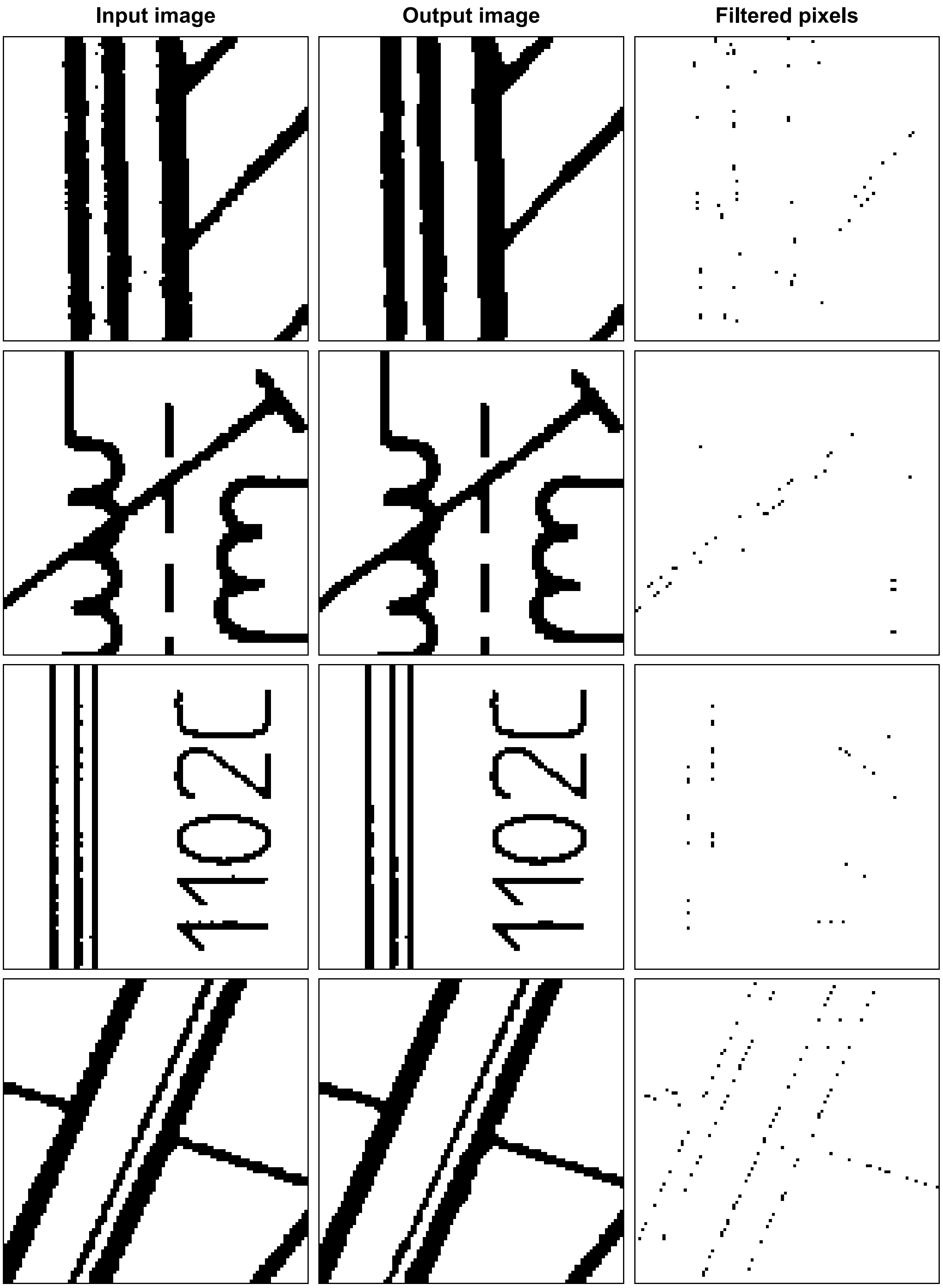


Figure 7-4. Feature-based filtering examples from left to right: sample of original image, the filtered image, and their difference.

In our evaluation, we first compare the two different feature extraction approaches studied in Chapter 4: Hough transform (HT) and raster-to-vector conversion (RVC). We use three typical line-drawing images from the line-drawing set as representatives: image Bolt (engineering drawing), Plan (architectural plan) and Power (electrical circuit); see Appendix D. Table 7-7 shows that the Hough transform gives a lower compression ratio than RVC and is an order of magnitude slower. Therefore, we will continue further only with RVC-based filtering.

In order to carry out thorough testing, we use a set of 28 test images, divided into four classes: *electrical circuits*, *engineering drawings*, *cartographic maps*, and *architectural and urban plans* (see Appendix D). The results are summarized in Table 7-8. The compressed vector file represents the result of the vectorizing when the chain-coded elements are compressed by ZIP (a commonly used file compression method). The corresponding compression ratios (in total) are 15:1 for the vector file, 33:1 for JBIG1, and 40:1 for the proposed method. At the same time, the quality of the decompressed images is visually the same as the original, given that only isolated groups of mismatched pixels are filtered. The quality is sometimes even better than the original, because the filtered pixels are mainly quantization noise near the borders of line segments. The preprocessing (vectorizing and filtering) slows down the entire compression process, which is now about 2.7 times slower than standard JBIG1 encoding.

Finally, we have considered existing filtering techniques adapted to the same near-lossless context. We apply the traditional *Median filter* and a combination of three known morphological filters: *opening*, *closing* and *annular filters*. The results of the filtering are fed to the noise removal process shown in Figure 5-7 in order to allow only isolated groups of noise pixels to be filtered. In this way, the compression method remains near-lossless. The compression improvement due to these filtering methods is summarized in Figure 7-5.

Conclusion

The proposed feature-based filtering technique removes additive and quantization noise from the original image, restores image quality, and in this way produces a better compression performance. For a set of test images, the method improves the compression ratio by about 17 % in comparison to JBIG1. One drawback of the method is that the compression phase is now more complex and the method must use several passes over the image. However, vectorizing can be performed quite quickly, and the vector features are not stored in the compressed file, so that the process is invisible in the decompression phase. The method can thus be considered as a preprocessing step to existing compression techniques, and standard decompression routines can be applied.

Table 7-7: JBIG1 compression results for images processed by feature-based filtering using a Hough transform and raster-to-vector conversions. The compression improvement is measured in comparison with the unfiltered image.

Input image	Original raster image	Without filtering	HT-based filtering	RVC-based filtering
Bolt	317,038	12,966	10,537	10,210
Power	512,199	17,609	16,271	14,581
Plan	484,561	5,098	4,319	3,978
TOTAL:	1,313,798	35,673	31,127	28,769
Improvement:	—	—	12.7 %	19.4 %

Table 7-8: Summary of the compression results (in bytes) for the line-drawing test set. The row TOTAL shows the results in total for the test set, and RATIO shows the compression ratio.

	Original raster image	Compressed vector file	ITU Group 4	Filtering + Group 4	JBIG1	Filtering + JBIG1
Circuits	6,092,892	268,953	220,430	193,702	150,119	122,799
Drawings	13,807,484	488,210	413,732	397,028	254,917	231,715
Maps	13,476,580	1,720,864	1,040,105	858,243	706,080	557,402
Plans	10,460,683	429,792	353,375	336,205	206,010	184,447
TOTAL:	43,837,639	2,907,819	2,027,642	1,785,178	1,317,126	1,096,363
RATIO:	—	15.1	21.6	24.6	33.3	40.0

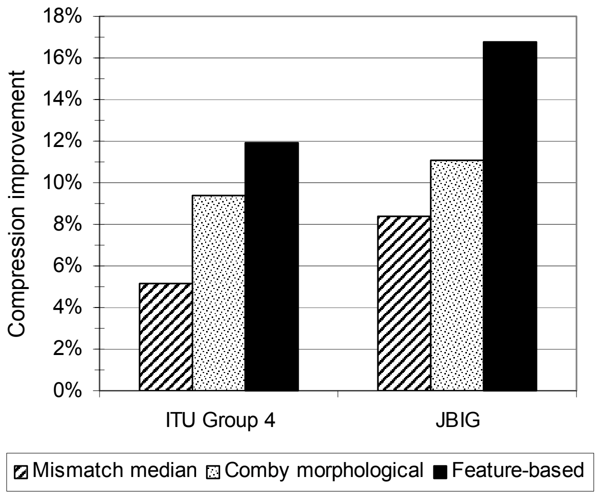


Figure 7-5. Comparison of the filtering methods discussed, used together with two compression standards. The figure shows the numbers as the relative reduction in file size when compressing the filtered images from the entire test set.

7.4. **On-line Image Processing and Spatial Access**

We evaluate here the document imaging storage system (DISS) for the interactive image browsing and retrieval system. The system architecture was outlined in Chapter 6. The system supports quick image decompression, instant preview, and spatial access through block coding and image tiling.

7.4.1. *Instant preview and fast decompression*

Two techniques, block coding and JBIG1 resolution reduction, are evaluated as regards the image preview using the standard CCITT test images and NLS topographic maps (component of Image$_0$), see Appendix for image details. In the case of block coding, the compressed file consists of block-level and pixel-level data. Block data represents the classification of the pixel blocks of a predefined size, and pixel data is the JBIG1-compressed stream of pixels belonging to mixed blocks. The thumbnail is generated directly from the block codes (see Section 6.1.3 for details). When the resolution reduction of JBIG1 (see Section 2.8) is applied, the original image and the generated thumbnail are independently compressed and stored together in the same file. Examples of the thumbnails generated using these two methods are shown in Figure 7-6 and Figure 7-7. The JBIG1 resolution reduction gives better quality thumbnails, whereas the block codes can speed up the image decompression.

For the block coding method, the block size is a trade-off between the compression ratio and running time (see Figure 7-8). A block size of 16×16 is a safe choice in the sense that the compression ratio is hardly compromised at all. The respective thumbnail images are sixteen times smaller than the original images in each dimension. Faster decoding (and a higher quality preview) could be achieved using a smaller block size (8×8) at the cost of a minor increase in the bit rate. For JBIG1 resolution reduction, the same size thumbnail images can be generated if the resolution reduction algorithm is applied sequentially to the original image four (16×16) or three (8×8) times, respectively.

The details of distribution of the code bits for block- and pixel-data are shown in Table 7-9. For block coding, the overhead of the block codes is mostly compensated for by the reduction of the pixel-level data that has to be compressed. For a block size of 16×16 pixels, this reduction is approximately equal to the size of block-data resulting in a zero net effect. In the case of JBIG1 resolution reduction, the data size of the compressed thumbnail image is approximately the same as for the block codes. Table 7-9 also shows comparative rates for the progressive compression of JBIG1.

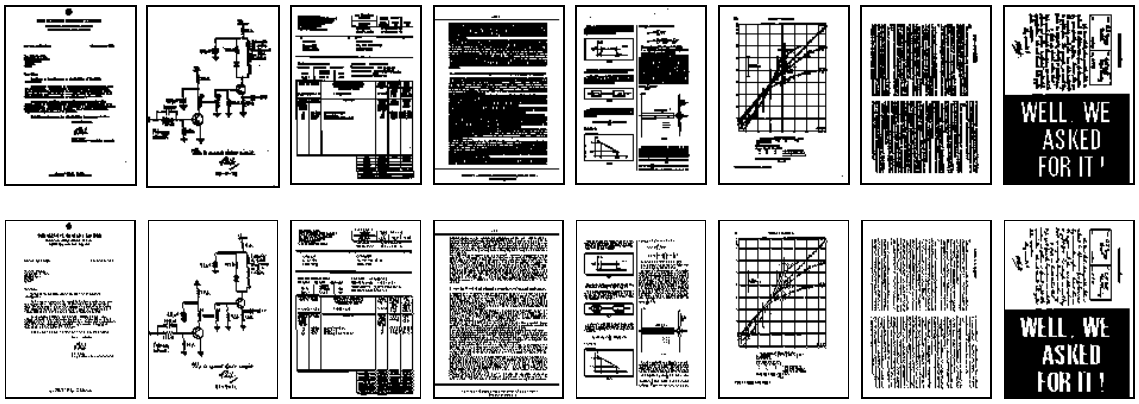


Figure 7-6. Thumbnail images for the CCITT test images (see Appendix A) generated with the block coding technique with 16×16 block size (upper row), and JBIG resolution reduction in the fourth generation (bottom row). Thumbnail images are 16 times smaller at each dimension.

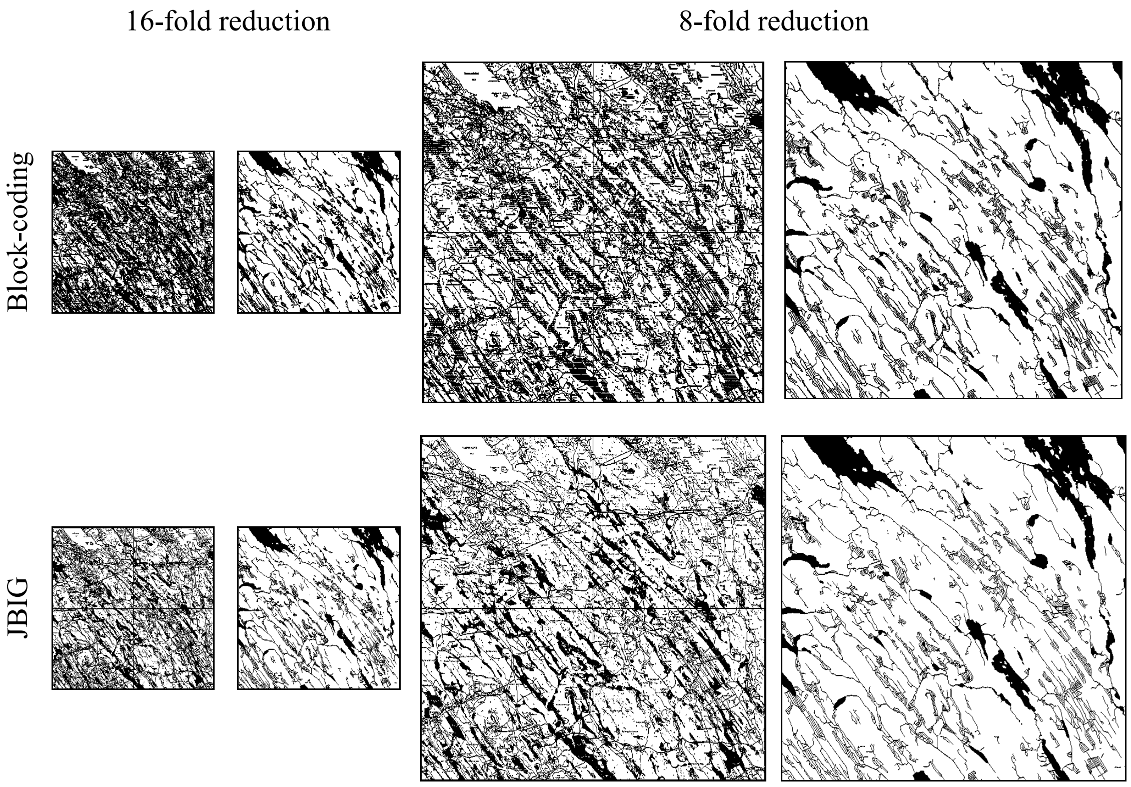


Figure 7-7. Thumbnail images for two NLS map images (see Appendix E) generated with block coding techniques (upper row), and with the JBIG resolution reduction algorithm (bottom row).

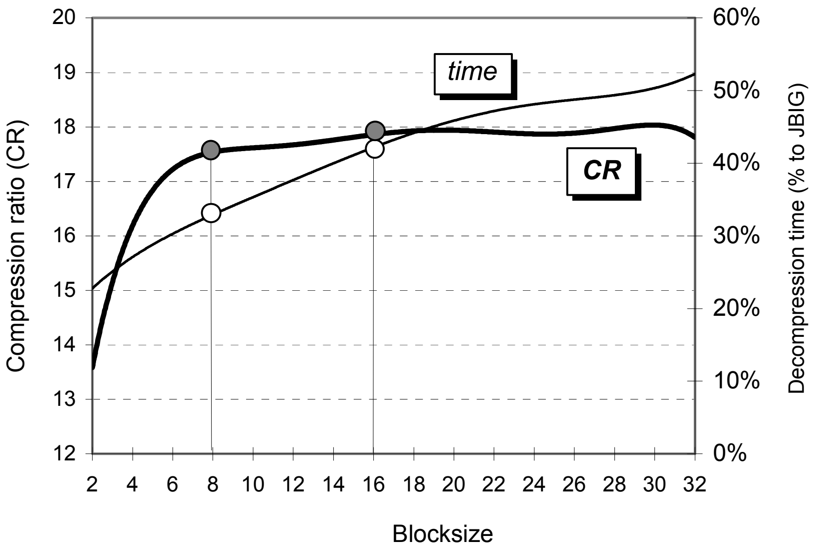


Figure 7-8 Compression ratio and decompression time as a function of block size.

Table 7-9: Distribution of code bits in the compressed image file using two compression techniques: DISS with block-coding and JBIG1 with thumbnail. All results are given in bytes and total results are also given in percentages relative to standard JBIG1 rates. Standard JBIG1 results are given in the JBIG1/'No preview' column.

	DISS (block coding + JBIG1)					
	Block size 8×8			Block size 16×16		
	Block data	Pixel data	Total	Block data	Pixel data	Total
Basic	29,099	339,381	368,480	8,223	349,152	357,375
Contour	33,624	305,148	338,772	7,865	311,345	319,210
Water	13,657	69,047	82,704	4,981	70,868	75,849
Field	5,065	17,937	23,002	2,011	18,738	20,749
Total	81,445	731,513	812,958	23,080	750,103	773,183
Relative to JBIG1	10.6 %	95.3 %	105.9 %	3.0 %	97.7 %	100.7 %

	JBIG1 (sequential mode)					JBIG1 prog-ressive mode
	No preview	8-fold image reduction		16-fold image reduction		
	Image	Thumbnail	Image + Thumbnail	Thumbnail	Image + Thumbnail	
Basic	357,612	27,001	384,613	9,513	367,125	438,879
Contour	316,781	33,818	350,599	10,663	327,444	359,597
Water	73,217	9,399	82,616	4,320	77,537	76,562
Field	20,076	2,799	22,875	1,380	21,456	20,425
Total	767,686	73,017	840,703	25,876	793,562	895,463
Per cent of JBIG1	100 %	9.5 %	109.5 %	3.4 %	103.4 %	116.6 %

For both methods discussed, the data required to reconstruct the preview image occupies only a few percent of the compressed file, enabling a quick preview. In particular, the preview data comprises 3 % of the original file size for a 16-fold, and 10 % for an 8-fold resolution reduction. The block coding offers benefits beyond the JBIG resolution reduction, because it speeds up the compression/decompression procedure. This is so because fewer pixels must be processed by the time-consuming context modeling and arithmetic coding procedures. On average, the decompression takes 30-50 % of the time required by JBIG1 (see Table 7-10). The method thus achieves decompression times comparable to the Group 4 standard. Compression takes longer than decompression because of the additional image analyzing procedures. Note that the method is expected to be even faster for higher resolution images, because the increase in resolution evidently results in an increased number of uniform blocks. On the contrary, if JBIG resolution reduction is applied, the compression operation becomes slower by the relative amount of data in the thumbnail image, which is 1/8 or 1/16 of the original image, plus the time required to generate the thumbnail.

Table 7-10: Comparison of running times for DISS with the block coding method (in percentage of time for sequential JBIG1).

	16×16 pixel block		8×8 pixel block	
Image	Compression	Decompression	Compression	Decompression
CCITT 1	29 %	19 %	21 %	13 %
CCITT 2	29 %	19 %	21 %	13 %
CCITT 3	43 %	31 %	36 %	25 %
CCITT 4	64 %	44 %	50 %	50 %
CCITT 5	43 %	31 %	36 %	25 %
CCITT 6	36 %	25 %	29 %	19 %
CCITT 7	57 %	38 %	43 %	44 %
CCITT 8	36 %	25 %	29 %	19 %
AVERAGE:	42 %	29 %	33 %	26 %
Basic	71 %	73 %	54 %	54 %
Contour	76 %	77 %	57 %	54 %
Water	34 %	27 %	26 %	17 %
Field	20 %	9 %	18 %	7 %
AVERAGE:	50 %	47 %	39 %	33 %

The exact running time depends on the implementation details. A high-level modular implementation of JBIG1 takes about 15 s per A4 document on a Pentium-200 machine, whereas the low-level hardware-optimized memory-consuming implementation may take about 3 s.

7.4.2. *Spatial access*

The effect of image tiling on compression performance is evaluated by compressing the set of map NLS images for five different domains (see Appendix E). The map image for each domain *X* consists of four binary component layers denoted as $Layer_X$, where *Layer* is the layer name, and *X* is the relative domain number from 0 to 4.

We evaluate the seven compression methods shown in Table 7-11. Sequential JBIG1 and the ITU Group 3 and Group 4 compression standards are the points of comparison. CT is the combination of context tree and baseline JBIG1. These four methods do not support spatial access, whereas the other methods do. T-JBIG is a combination of tiling and sequential JBIG1. FAM stands for compression based on forward-adaptive modeling (as in Section 6.2). CT-FAM stands for compression based on a combination of context tree and forward-adaptive modeling (as in Section 3.5), and CT-FAM^S is its static variant, where the context tree is optimized off-line for the training image (as outlined in Section 3.2.2). The two-stage bottom-up algorithm (see Section 3.4) has been used to construct the tree.

The overall effect of tiling and the variable-size context modeling are summarized in Figure 7-9. The benefits of a better initial model efficiently outweigh the overhead and learning costs for all cluster sizes, except the very small ones. In this case, the cluster and boundary overheads become too large to be compensated. The periodic coder re-initialization also improves local adaptation, so that even T-JBIG outperforms JBIG1 for large cluster sizes. The experiment shows that sequential JBIG1 can be applied with the tiling using a cluster size of about 350×350 pixels without sacrificing the compression performance. The corresponding number is 100×100 for the FAM, and 50×50 for the CT-FAM. Moreover, the CT-FAM improves the compression performance by 20 % if the cluster size is 200×200 or greater. The *maximum possible improvement* line shows the compression that would be achieved if the tiling were not applied (CT method).

Comparative compression results for the method discussed are summarized in Table 7-12. In this experiment, images 1 to 4 are used for the compression. Tiling the image to clusters of size 256×256 is assumed for the methods that support spatial access. For evaluating the static variant of the CT-FAM method, four context trees were trained separately off-line using $Image_0$ for each binary component. Experiments show that the proposed method improves the performance of JBIG1 by over 20 % for this cluster size. The static variant is also applicable because of similar types of images. In our example, the static variant was only 3.5 % worse than the semi-adaptive one.

Table 7-11: Compression methods and their properties.

Method	Tiling	Statistical model	Context tree	Passes
Group 3/4	–	–	–	1
JBIG1	–	backward-adaptive	–	1
CT	–	backward-adaptive	semi-adaptive	2*
T-JBIG	+	backward-adaptive	–	1
FAM	+	forward-adaptive	–	2
CT-FAM	+	forward-adaptive	semi-adaptive	2*
CT-FAM^S	+	trained	trained	1

* One stage is assumed for the bottom-up context tree construction. Add one more pass for the 2-stage bottom-up method.

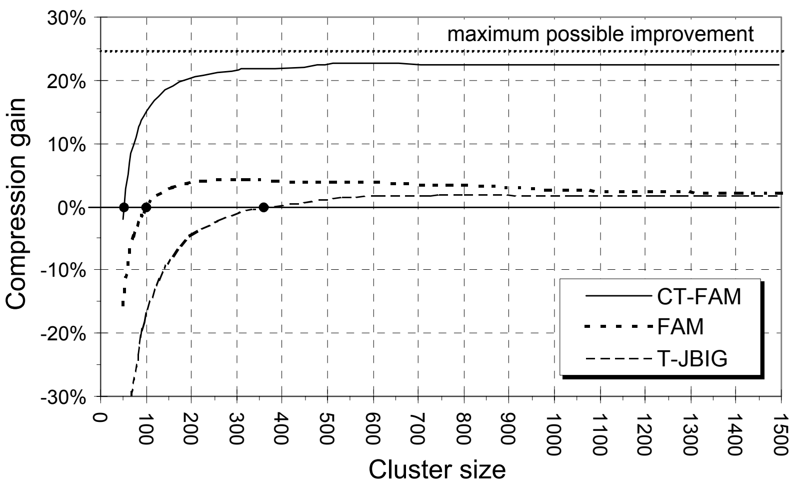


Figure 7-9. Compression gain for T-JBIG, FAM, CT-FAM as a function of the cluster size when compared to the sequential JBIG1.

Table 7-12: Bit rates per image type, and overall compression ratios. Cluster size of 256×256 pixels is assumed for the methods supporting tiling (spatial access). A two-stage bottom-up pruning approach is used for context tree construction.

Test images	Methods not supporting tiling				Methods supporting tiling (256×256 cluster size)			
	Group 3	Group 4	JBIG1	CT	T-JBIG	FAM	CT-FAM	CT-FAM^S
Basic$_{1-4}$	2,834,589	2,881,614	1,197,983	884,435	1,263,311	1,211,338	903,597	944,107
Contours$_{1-4}$	1,968,901	1,230,480	643,998	514,353	683,314	632,882	536,788	549,571
Water$_{1-4}$	1,122,591	548,124	270,703	206,282	280,031	249,697	207,829	210,636
Fields$_{1-4}$	233,415	64,530	29,127	25,030	35,914	33,412	28,558	33,412
TOTAL (16)	6,159,496	4,724,748	2,141,811	1,630,100	2,262,570	2,127,329	1,676,772	1,737,726
Compression ratio	**8.12**	**10.58**	**23.34**	**30.67**	**22.10**	**23.50**	**29.82**	**28.77**

Conclusion

The proposed storage system architecture meets the requirements of interactive image processing as outlined in Section 6. Quick image decompression, an instant preview option, and direct access to the compressed image were sufficiently supported by the block coding and image tiling. Block coding enables an image preview, and results in an increase in speed of the decompression time by a factor of 2.5.

Spatial access eliminates the need to decompress the entire image for accessing its fragment. The use of a context tree makes it possible to utilize larger context templates without greatly increasing the learning cost. The method alleviates the deterioration of the coding efficiency caused by tiling and achieves higher compression rates because of the improved pixel prediction.

Being applied to binary layers of large maps for four different domains, the proposed technique enables denser image tiling down to 50×50 pixels versus the 350×350 possible with JBIG1. In addition, the technique improves the compression performance of JBIG1 by over 20 % for clusters sized 200×200 pixels or larger.

We have also considered a static variant of the method, in which the model is generated using a training image. It gives faster one-pass compression and enables image tiling down to 100×100 pixels. The static variant can be applied if the images are of a similar type. Otherwise, the two-pass method should be used at the cost of higher compression times. The decompression times are similar in both cases.

8. Conclusions

In this thesis, we have studied compression of binary images in various document imaging applications. We have considered context-based statistical modeling and arithmetic coding. The JBIG1 standard coding implementation (QM-coder) has been adopted. Advanced modeling techniques, such as variable-size context modeling and forward-adaptive statistical modeling have been presented and evaluated.

The empirical study shows that these techniques improved compression performance by about 25 % in comparison to that achieved by JBIG1. A variant of the variable-size modeling has also been proposed for on-line image communications, improving compression up to 14 %. It is also possible to utilize the advantages of the proposed techniques when applied in conjunction with the recent image compression standard, JBIG2.

We have studied the use of statistical context-based modeling for removing the additive (random) and content-dependent (quantization) noise from digitized documents. Using the proposed filtering schemes, we have improved the compression performance of textual document images to the level of noise-free images. The quality of the images, as measured by the OCR accuracy, has not been affected by filtering.

For line-drawing images, we have studied global modeling. Two techniques for the extraction of linear features from an image have been evaluated: the Hough transform and raster-to-vector conversion. For the hybrid raster/vector compression, we utilized the extracted vector features in the compression of raster images. We have also introduced a new filtering technique that uses extracted vector information to remove the noise near the contours of printed objects. From the compression point of view, the new technique appears twice as effective as such traditional methods as morphological filtering.

Finally, we have considered direct access to the compressed images. A storage system that features instant previews, fast image decompression, and spatial access has been proposed. These properties enable the user to interactively browse the image archive and access the desired fragment of the image without transmission and decompression delays. A 2.5-fold increase in access time over JBIG1 has been achieved for image decompression, and only 2 % of the entire compressed data set must be retrieved to build an image thumbnail. The proposed technique enables denser image tiling down to 50×50 pixels versus the 350×350 possible with JBIG1, without sacrificing compression performance. It allows far more efficient image retrieval than when using standard JBIG1 compression.

References

[AB89] **Arcelli C., di Baja G.S.** (1989) A one-pass two-operation process to detect the skeletal pixels on the 4-distance transform. *IEEE Trans. on Pattern Analysis, Machine Intelligence*, **11** (4): 411-414.

[AF98] **Ageenko E.I., Fränti P.** (1998) Enhanced JBIG-based compression for satisfying objectives of engineering document management system, *Optical Engineering* **37** (5): 1530-1538.

a preliminary version: 'Storage system for document imaging applications' appears in *Proc. Picture Coding Symposium* (Berlin, Germany, 1997), 361-364.

[AF99a] **Ageenko E.I., Fränti P.** (1999) Forward-adaptive method for compressing large binary images, *Software Practice & Experience,* **29**(11): 943-952.

[**AF99b**] **Ageenko E.I., Kolesnikov A., Fränti P.** (1999) On-line demonstration of the Document Imaging Compression System supporting spatial access: http://cs.joensuu.fi/~ageson/research/edm_demo.html

[AF00a] **Ageenko E.I., Fränti P.** (2000) Context-based filtering of document images, *Pattern Recognition Letters* (to appear);

[AF00b] **Ageenko E.I., Fränti P.** (2000) Compression of large binary images in digital spatial libraries. *Computer & Graphics* **24** (1): 91-98;

[AKS90] **Algazi V.R., Kelly P.L., Estes R.R.** (1990) Compression of binary facsimile images by preprocessing and color shrinking. *IEEE Trans. on Communications* **38 (**9): 1592-1598.

[AN74] **Ascher R.N., Nagy G.** (1974) A means for achieving a high degree of compaction on scan-digitized printed text. *IEEE Trans. Computers*, **C-23** (November): 1174-1179.

[AT94] **Arps R.B., Truong T.K.** (1994) Comparison of international standards for lossless still image compression. *Proceedings of the IEEE* **82**: 889-899.

[Ber87] **Bernstein R.** (1987) Adaptive nonlinear filters for simultaneous removal of different kinds of noise in images. *Proc. IEEE Transaction on Circuits, Systems* **CAS-34 (**11): 1275-1291.

[Cap59] **Capon J.** (1959) A probabilistic model for run-length coding of pictures. *IRE Tran. Information Teory*, IT-5: 157-163.

[Col66] **Colomb S.W.** (1966) Run-length encodings. *IEEE Trans. Inform Theory*, IT-12: 399-401.

[DA97] **Dougherty E.R., Astola J. (eds)** (1997) Nonlinear Filters for Image Processing, SPIE Optical Engineering Press.

[DLDC93] **Dori D., Linag Y., Dowell J., Chai I.** (1993) Sparse-pixel recognition of primitives in engineering drawings. *Machine Vision and Applications* **6**: 69-82.

[EKY91] **Endoh T., Kato S., Yasuda Y.** (1991) Progressive coding scheme for binary images, *Electronic and Communications in Japan*, part 1, **74**(8): 1-17.

[ESRI94] **ESRI** (1994) GIS Approach to Digital Spatial Libraries. *ESRI White Paper Series*. Environmental System Research Institute, Inc., USA. http://www.esri.com/

[ESRI98] **ESRI** (1998) The role of Geographic Information Systems on the electronic battlefield. *ESRI White Paper Series*. Environmental System Research Institute, Inc., USA.

[FA99] **Fränti P., Ageenko E.I.** (1999) On the use of context tree for binary image compression. *Proc. IEEE International Conference on Image Processing* "*ICIP* '99" (Kobe, Japan), 27PP2.3.

[FAKK98a] **Fränti P., Ageenko E.I., Kälviäinen H., Kukkonen S.** (1998) Compression of line drawing images using Hough transform for exploiting global dependencies, *Proc. 4th Joint Conf. on Information Sciences JCIS'98* (RTP, NC, USA) **IV**: 433-436.

[FAKK98b] **Fränti P., Ageenko E.I., Kälviäinen H., Kukkonen S.** (1998) Lossless and near-lossless compression of line-drawing images using Hough transform. University of Joensuu, Department of Computer Science, Technical Report A-1998-6 (submitted for publication)

[FAK99] **Fränti P., Ageenko E.I., Kolesnikov A.N.** (1999) Vectorizing and feature-based filtering for line-drawing image compression. *Pattern Analysis and Applications*, **2** (4): 285-291;
a preliminary version: "Compression of line-drawing images using vectorizing and feature-based filtering" appears in *Proc. 8th Int. Conf. on Computer Graphics and Visualization* "*GraphiCon* 98" (Moscow, Russia, 1998), 219-224.

[FJ94] **Forchhammer S., Jensen K.S.** (1994) Data compression of scanned halftones images. *IEEE Trans. Communications,* 42: 1881-1893.

[FN93] **Fränti P., Nevalainen O.** (1993) A two-stage modeling method for compressing binary images by arithmetic coding. *The Computer Journal* **36** (7): 615-622.

[FN95] **Fränti P., Nevalainen O.** (1995) Compression of binary images by composite methods based on the block coding. *Journal of Visual Communication, Image Representation* **6** (4): 366-377.

[Fox+95] **Fox E.A., et al.** (1995) **(**Eds.) Digital Libraries. [*Special issue of*] *Communications of the ACM* **38** (4).

[Fr94] **Fränti P.** (1994) A fast and efficient compression method for binary images. *Signal Processing: Image Communication* **6** (1): 69-76.

[GW92] **Gonzalez R.C., Woods R.E.** (1992) Digital image processing. Addison-Wesley.

[Gray70] **Gray R.M.** (1970) Information rates of autoregressive processes. *IEEE Trans. Inform Theory* **IT-16**: 412-421.

[Haf+99] **Haffner P., LeCunn Y., Bottou L., Howard P., Vincent P., Riemers B.** (1999) Color document on the Web with DjVu. *Proc. IEEE International Conference on Image Processing* "*ICIP* '99" (Kobe, Japan), 25PS1.7.

[Har93]	**Harvey R.** (1993) (eds.). Preservation in Libraries: A Reader. London: Bowker-Saur.
[Haskel98]	**Haskell B.G. et al.** (1998) Image and video coding – emerging standards and beyond. *IEEE Trans. Circuits and Systems for Video Technology* **8** (7): 814-837.
[Haskin98]	**Haskins D.** (1998) Word for Word. *PC Magazine*, (1998)1: 20-2x.
[Hei94]	**Heijmans H.J.A.M.** (1994) Morphological image operators. Boston: Academic Press.
[How97]	**Howard P.G.** (1997) Text image compression using soft pattern matching. *The Computer Journal* **40** (2/3): 146-156.
[How+98]	**Howard P.G., Kossentini F., Martins B., Forchammer S., Rucklidge W. J., Ono F.** (1998) The emerging JBIG2 standard. *IEEE Trans. Circuits, Systems for Video Technology* **8** (7): 838-848.
[Hou62]	**Hough P.C.V.** (1962) Methods and means for recognizing complex patterns. U.S. Patent 3,069,654.
[Huf52]	**Huffman D.A.** (1952) A method for the construction of minimum redundancy codes. *Proc. of IEEE*, **40** (9): 1098-1101.
[HK83]	**Hung S.H.Y., Kasvand T.** (1983) Linear approximation of quantized thin lines. In: Haralick R.M. (eds) Pictorial Data Analysis. Berlin: Springer-Verlag, 15-28.
[Hun80]	**Hunter R., Robinson A.H.** (1980) International digital facsimile coding standards. *Proc. of IEEE*, **68** (7), 854-867.
[ITU T.4]	**ITU-T (CCITT) Recommendation T.4**. (1980)
[ITU T.6]	**ITU-T (CCITT) Recommendation T.6**. (1988) Facsimile coding schemes and coding control functions for Group 4 facsimile apparatus.
[ITU T.24]	**ITU-T (CCITT) Recommendation T.24.** (1994) Standardized digitized image set.
[ITU T.43]	**ITU-T (CCITT) Recommendation T.43.** (1997) Colour and gray-scale image representations using lossless coding scheme for facsimile.
[IW94]	**Inglis S., Witten I.** (1994) Compression-based template matching. *In Proc. IEEE Data Compression Conference* (Utah, USA), 106-115.
[Jag90]	**Jagadish H.V.** (1990) Linear clustering of objects with multiple attributes. *In Proc. ACM SIGMOD Int. Conf. on Management of Data*, 332-342.
[JBIG1]	**JBIG.** ISO/IEC International Standard 11544 (1993) *ISO/IEC/JTC1/SC29/WG9;* also ITU-T Recommendation T.82. Progressive Bi-level Image Compression.
[JBIG2]	**JBIG2 Working Draft**. http://www.jpeg.org/public/jbigpt2.htm
[JBIGWP]	**JBIG Alliance White Paper** (1996), InfoTrends Research Group, Inc., One Main Plaza, 4435 Main Street, Kansas City, MO 64111, USA. http://www.pdsimage.com/html/news/jbig/whtpaper.htm
[JKW98]	**Joung H., Kim S.P., Wong E.K.** (1998) Lossless grayscale image coding by using higher order context modeling and bit-plane decomposition. *Proc. 4th Joint Conf. on Information Sciences (JCIS '98)*, RTP, USA, **IV**: 254-257.

[JLG96] **Jann Lynn-George** (1996) Digitization: A literature review, summary of technical processes, applications, issues. Jonh A. Weir Memorial Law Library, Univ. of Alberta, Edmonton, Alberta, Canada, (http://www.library.ualberta.ca/library_html/libraries/law/pubs.html)

[JWC98] **Joung H., Wong E.K., Chen Y.** (1998) Document image compression using straight line extraction and block context model. *Proc. IEEE International Conference on image Processing (ICIP '98)*, Chicago, Illinois, USA

[KA94] **Koskinen L., Astola J.** (1994) Soft morphological filters: A robust morphological filtering method. *Journal of Electronic Imaging* 3: 60-70.

[Kas90] **Kasturi R., et al.** (1990) A system for interpretation of line drawings. *IEEE Trans. on Pattern Analysis, Machine Intelligence* **12**(10): 978-992.

[KBO96] **Kolesnikov A.N., Belekhov V.I., Chalenko I.O.** (1996) Vectorization of raster images. *Pattern Recognition, Image Analysis* **6** (4): 786-794.

[KHXO95] **Kälviäinen H., Hirvonen P., Xu L., Oja E.** (1995) Probabililistic, non-probabilistic Hough transforms: overview and comparisons. *Image, Vision Computing* **13**: 239-251.

[KJ80] **Kunt M., Johnsen O.** (1980) Block Coding: A Tutorial Review. *Proceedings of the IEEE*, **68** (7): 770-786.

[KOG92] **Kasturi R., O'Gourman L.** (1992) Document image analysis: A bibliography. *Machine Vision, Applications* **5**: 231-243.

[KT95] **Kolesnikov A.N., Trichina E.V** (1995) The parallel algorithm for the binary images thinning. *Optoelectronics, Instrumentation, Data Processing* **1995** (6): 7-13.

[Lea93] **Leavers V.F.** (1993) Survey: Which Hough Transform. *CVGIP Image Understanding* **58** (2): 250-264.

[Les92] **Lesk M.** (1992) Image formats for preservation and access: a report of the Technology Assessment Advisory Committee to the Commission on Preservation and Access, *reprinted* in [Har93].

[LR81] **Langdon G.G., Rissanen J.** (1981) Compression of black-white images with arithmetic coding. *IEEE Trans. Communications* **29** (6): 858-867.

[Lun90] **Lunin L.** (1990) An overview of electronic image information. *Optical Information systems* **10** (3): 114-130.

[Lyn90] **Lynn M.,** and the Technology Assessment Advisory Committee to the Commision on Preservation and Access. **(**1990) The relationship between digital and other media conversion processes: a structured glossary of technical terms. *Information Technology, Libraries* **9** (4): 309-336.

[MR92] **Markas T., Reif J.** (1992) Quad tree structures for image compression applications, *Information Processing & Management* **28** (6): 707-721.

[MF98] **Martins B., Forchhammer S.** (1998) Bi-level image compression with tree coding. *IEEE Trans. Image Processing* **7** (4): 517-528.

[MF99] **Martins B., Forchhammer S.** (1999) Lossless, near-lossless, and refinment coding of bi-level images. *IEEE Trans. Image Processing* **8** (5): 601-613.

[MM87] **Mintzer F.C., Mitchell J.L.** (1987) Line-preserving binary image reduction algorithm. ISO/IEC JTC1/SC2/WG8, no. 601.

[Mof91] **Moffat A.** (1991) Two-level context based compression of binary images. *IEEE Proc. Data Compression Conference* (Snowbird, Utah, USA), 382-391.

[MSY92] **Mori S., Suen C.Y., Yamamoto K.** (1992) Historical review of OCR research and development. *Proc. of IEEE*, 80: 1029-1058.

[NG96] **Nelson M., Gailly J.-L.** (1996) The data compression book. M&T Books, New York, NY, USA.

[NL90] **Nagasamy V., Langrana N.A.** (1990) Engineering drawing processing and vectorizing system. *Computer Vision, Graphics and Image Processing* **49**: 379-397.

[NLS] **NLS**: National Land Survey of Finland, Opastinsilta 12 C, P.O.Box 84, 00521 Helsinki, Finland. http://www.nls.fi/index_e.html.

[NW80] **Netravali A.N., Mounts F.W.** (1980) Ordering Techniques for Facsimile Coding: A Review. *Proceedings of the IEEE*, **68** (7): 796-807.

[OAS98] **O.A.S.** (1998) Cost Justification Paper. Open Archive Systems, Inc, 5 Jefferson Road, Windham, NH 03087, USA. Publications, Document CJ.DOC. http://www.openarchive.com/

[OO92] **Ogg H., Ogg M.** (1992) Optical Character Recognition: A Librarian's guide. Westport, Conn: Meckler.

[PMLA88] **Pennebaker W.B., Mitchell J.L., Langdon G.G., Arps, R.B** (1988) An overview of the basic principles of the Q-coder adaptive binary arithmetic coder. *IBM Journal of Research, Development* **32**(6): 717-726.

[PM88] **Pennebaker W.B., Mitchell J.L.** (1988) Probability estimation for the Q-coder. *IBM Journal of Research, Development* **32**(6): 737-759.

[PM93] **Pennebaker W.B., Mitchell J.L.** (1993) JPEG Still Image Data Compression Standard. Van Nostrand Reinhold.

[PW96] **Pajarola R., Widmayer P.** (1996) Spatial indexing into compressed raster images: how to answer range queries without decompression. *Proc. Int. Workshop on Multimedia DBMS* (Blue Mountain Lake, NY, USA), 94-100.

[Ric79] **Rice R.F.** (1979) Some practical universal noiseless coding techniques. *Proc. 1993 Data Compression Conference* (Snow Bird, Utah, USA), 351-360.

[Ris83] **Rissanen J.J.** (1983) A universal data compression system. *IEEE Trans. Inform. Theory* **IT-29** (Sept. 1983): 656-664.

[RL79] **Rissanen J.J., Langdon G.G.** (1979) Arithmetic coding. *IBM Journal of Research, Development* **23**: 146-162.

[RL81] **Rissanen J.J., Langdon G.G.** (1981) Universal modeling and coding. *IEEE Trans. Inform. Theory* **IT-27**: 12-23.

[RM92] **Rabbani M., Melnychuck P.W.** (1992) Conditioning context for the arithmetic coding of bit planes. *IEEE Trans. Signal Processing* **40**: 232-236.

[RM95] **Röösli M., Monagan G.** (1995) A high quality vectorizing combining local quality measures and global constraints. *IEEE Proc. 3rd Int. Conf. on Document Analysis, Recognition* (Montreal, Canada), 243-248.

[Saf93] **Saffady W.** (1993) Electronic Document Imaging Systems: Design, Evaluation, Implementation. Westport, Conn.:Meckler.

[Saf95] **Saffady, W.** (1995) Digital library concepts and technologies for the management of library collections: an analysis of methods, costs. *Library technology reports* **31** (3): 221-380.

[Sal97] **Salomon D.** (1997) Data compression: the complete reference. New York: Springer-Verlag.

[Sam89] **Samet H.** (1989) Applications of Spatial Data Structures: Computer Graphics, Image Processing, GIS. MA: Addison-Wesley, Reading.

[Sam90] **Samet H.** (1990) The Design and Analysis of Spatial Data Structures. MA: Addison-Wesley, Reading.

[Ser82] **Serra J.** (1982) Image Analysis and Mathematical morphology. London: Academic Press.

[SG91] **Schonfeld D., Goutsias J.** (1991) Optimal morphological pattern restoration from noisy binary images. *IEEE Trans. on Pattern Analysis, Machine Intelligence* **13** (1): 14-29.

[Sh48] **Shannon C.E.** (1948) A mathematical theory of communication. *Bell Syst. Tech Journal* **27**: 398-403.

[SK83] **Sankoff D., Kruskal J.B.** (1983) (eds.) Time Warps, string edits and macromolecules: the theory and practice of sequence comparison. MA: Addyson-Wesley, Reading.

[TWMG93] **Tisher P.E., Worley R.T., Maeder A.J., Goodwin M.** (1993) Context-based lossless image compression. *The computer Journal* **36**: 68-77.

[TP80] **Ting D., Prasada B.** (1980) Digital processing techniques for encoding of graphics. *Proc. of the IEEE* **68** (7): 757-769.

[TK99] **Tompkins D.A.D., Kossentini F.** (1999) A fast segmentation algorithm for bi-level image compression using JBIG2. *Proc. IEEE International Conference on Image Processing* "*ICIP* '99" (Kobe, Japan), 25PS1.4.

[Ur92] **Urban S.J.** (1992) Review of standards for electronic imaging for facsimile systems. *Journal of Electronic Imaging*, **1**(1): 5-21.

[VETK99] **Valliappan M., Evans B.L., Tompkins D.A.D., Kossentini F.** (1999) Lossy compression of stochastic halftones with JBIG2. *Proc. IEEE International Conference on image Processing* "*ICIP* '99" (Kobe, Japan), 25PS1.2.

[Vit97] **Vitter J.** (1987) Design and Analysis of dynamic Huffman codes. *Journal of Association for Computing Machinery*, 34:825-845.

[WW92] **Wao Y. Wu Y. J.-M.** (1992) Vector Run-Length Coding of Bi-level Images. *Proceedings Data Compression Conference*, Snowbird, Utah, USA, 289-298.

[WRA96] **Weinberger M.J., Rissanen J., Arps R.** (1996) Application of universal context modeling to lossless compression of gray-scale images. *IEEE Trans. Image Processing*, Apr, 1996.

[WD99] **Wenyin L.**, **Dori D** (1999) From raster to vectors: extracting visual information from line drawings, *Pattern Analysis & Applications*, (1999)2:10-21.

[Will91] **Williams R.** (1991) Adaptive data compression. Kluwer Academic Publishers.

[Will92] **Willis D.** (1992) Imaging: the information access tool of the nineties. *Proc. 13th National Online Meeting* (Medford, N.J., USA), 435-444.

[Wils96] **Wilson D.J.** (1996) How to modernize your paper engineering drawings. *Imaging World*, **5** (6): 52-53.

[Wils99] **Wilson D.J.** (1999) S.E.A. Your paper. Scan Edit Archive Initiative, White paper. http://www.sea-initiative.org/

[WMB94] **Witten I.H., Moffat A., Bell T.C.** (1994) Managing Gigabytes: Compressing and Indexing Documents and Images. Van Nostrand Reinhold, New York.

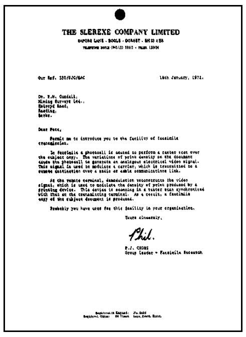
[WNC87] **Witten I., Near R., Cleary J.** (1987) Arithmetic coding for data compression. *Communications ACM*, **30**: 520-539.

[ZD96] **Zhang Q., Danskin J.M.** (1996) Bitmap reconstruction for document image compression. *SPIE Proc. Multimedia Storage, Archiving Systems* (Boston, MA, USA), Vol. 2916: 188-199.

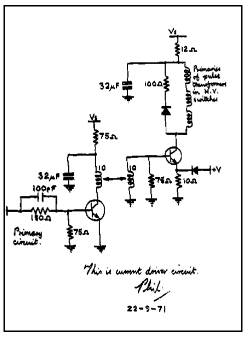
APPENDIX: THE TEST SETS.

A. CCITT Test Set – Facsimile Images

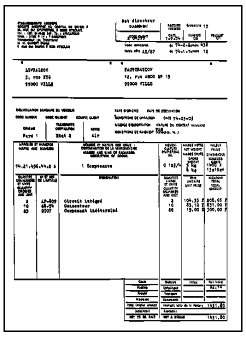
All images are standard CCITT documents of A4 format [ITU T.24], digitized at facsimile resolution of 200 dpi. Image size is 1728×2376 pixels.



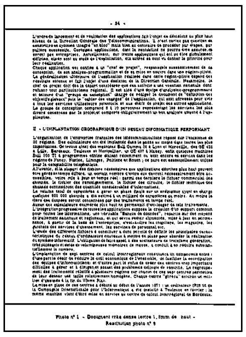
CCITT 1



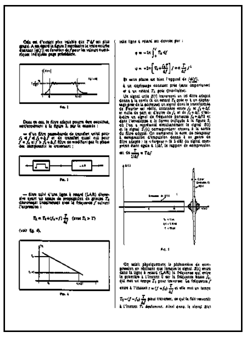
CCITT 2



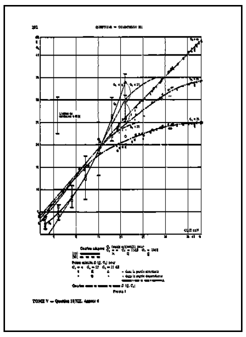
CCITT 3



CCITT 4

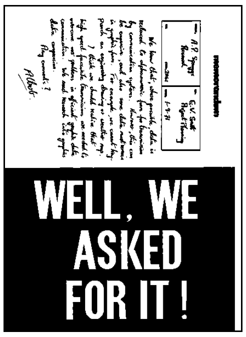


CCITT 5



CCITT 6

CCITT 7



CCITT 8

B. Newspaper Images

The set consists of eight newspaper images of A4 format, 2464 × 3497 pixels each:

Image 1

Image 2

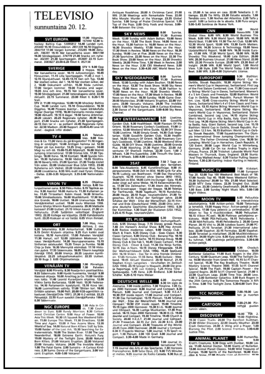


Image 3

Image 4

Image 5

Image 6

Image 7

Image 8

The combined image used for training (A4). It consists of various text types typical for the particular newspaper.

C. DIGITIZED DOCUMENT IMAGES

There are two test sets with document images that have been used in the evaluation. First set consists of eight A4 format document images originated from the text document, which has been typed in a word processing system. The document contains the same text parts formatted in two different fonts: *Times* and *Arial*, both in two different sizes: 10 and 12 points. The typefaces were chosen so that they represent the two commonly used families of fonts (with and without "serif" elements). The total number of symbols in the document is 4712 including spaces.

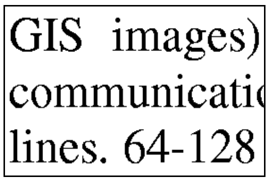
The printed document was a subject to further transformations such as photocopying and faxing. We include four different photocopies of the document: high quality (*HiQ*) copy (sophisticated copy machine with optimal copying parameter setup), bright and dark second copies, and low quality (*LoQ*) copy made using all-in-one desktop office machine. An ink-jet printing was also included in the set.

The resulting seven documents were then digitized on an office desktop scanner at the same resolution 300 dpi (*dot-per-inch*) referred further as *digitized images*. Only the fax image was electronically received at the resolution 200 dpi. The PostScript image was generated directly from the textual document and approximates noise free document image.

The second, larger scale, test set consists of real documents. The documents are taken from conference proceedings and contain text with a variety of fonts and offprint quality. The documents were digitized at resolutions of 300 and 400 dpi resulting in 56 images of the size 2328×3028 and 3112×4038 pixels, respectively.

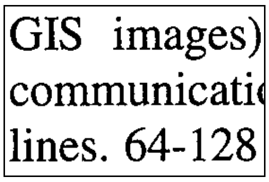
Samples of digitized documents (fragment of the text typed in font Times at 12 points):

PostScript image



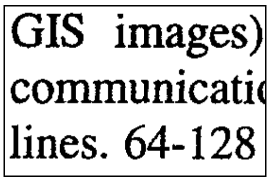
GIS images)
communicati
lines. 64-128

Laser printer



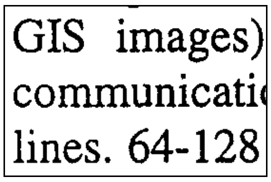
GIS images)
communicati
lines. 64-128

Ink-jet printer



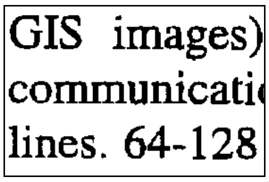
GIS images)
communicati
lines. 64-128

High quality copy



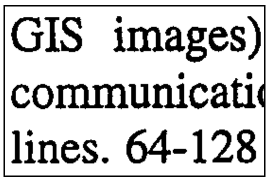
GIS images)
communicati
lines. 64-128

Low quality copy



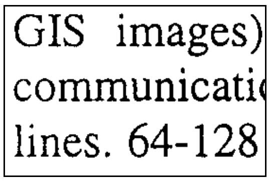
GIS images)
communicati
lines. 64-128

Dark copy



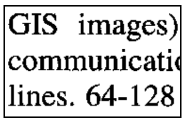
GIS images)
communicati
lines. 64-128

Bright copy



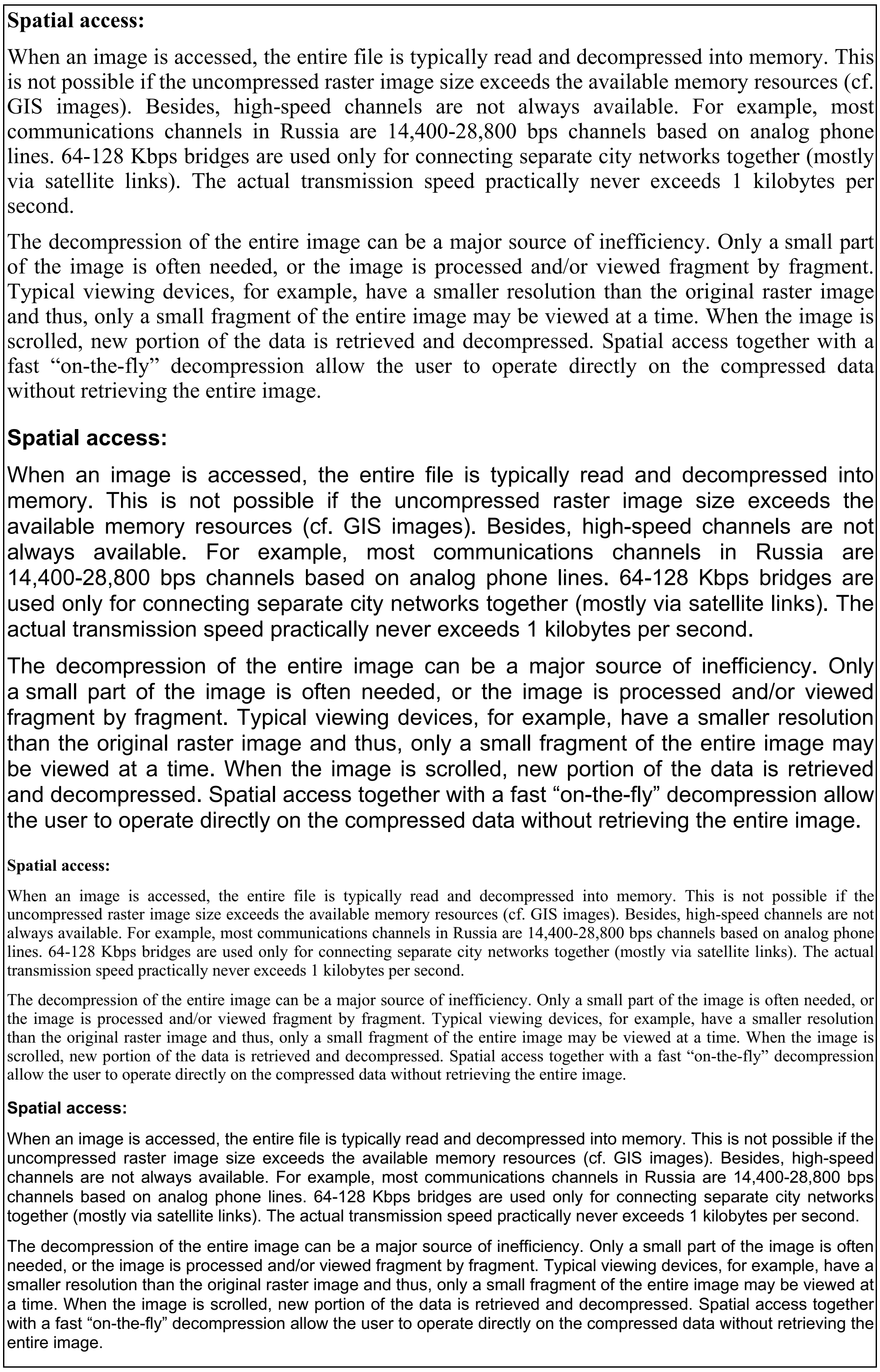
GIS images)
communicati
lines. 64-128

Fax (200 DPI)



GIS images)
communicati
lines. 64-128

Original text document (reduced PostScript image):



Spatial access:

When an image is accessed, the entire file is typically read and decompressed into memory. This is not possible if the uncompressed raster image size exceeds the available memory resources (cf. GIS images). Besides, high-speed channels are not always available. For example, most communications channels in Russia are 14,400-28,800 bps channels based on analog phone lines. 64-128 Kbps bridges are used only for connecting separate city networks together (mostly via satellite links). The actual transmission speed practically never exceeds 1 kilobytes per second.

The decompression of the entire image can be a major source of inefficiency. Only a small part of the image is often needed, or the image is processed and/or viewed fragment by fragment. Typical viewing devices, for example, have a smaller resolution than the original raster image and thus, only a small fragment of the entire image may be viewed at a time. When the image is scrolled, new portion of the data is retrieved and decompressed. Spatial access together with a fast "on-the-fly" decompression allow the user to operate directly on the compressed data without retrieving the entire image.

Spatial access:

When an image is accessed, the entire file is typically read and decompressed into memory. This is not possible if the uncompressed raster image size exceeds the available memory resources (cf. GIS images). Besides, high-speed channels are not always available. For example, most communications channels in Russia are 14,400-28,800 bps channels based on analog phone lines. 64-128 Kbps bridges are used only for connecting separate city networks together (mostly via satellite links). The actual transmission speed practically never exceeds 1 kilobytes per second.

The decompression of the entire image can be a major source of inefficiency. Only a small part of the image is often needed, or the image is processed and/or viewed fragment by fragment. Typical viewing devices, for example, have a smaller resolution than the original raster image and thus, only a small fragment of the entire image may be viewed at a time. When the image is scrolled, new portion of the data is retrieved and decompressed. Spatial access together with a fast "on-the-fly" decompression allow the user to operate directly on the compressed data without retrieving the entire image.

Spatial access:

When an image is accessed, the entire file is typically read and decompressed into memory. This is not possible if the uncompressed raster image size exceeds the available memory resources (cf. GIS images). Besides, high-speed channels are not always available. For example, most communications channels in Russia are 14,400-28,800 bps channels based on analog phone lines. 64-128 Kbps bridges are used only for connecting separate city networks together (mostly via satellite links). The actual transmission speed practically never exceeds 1 kilobytes per second.

The decompression of the entire image can be a major source of inefficiency. Only a small part of the image is often needed, or the image is processed and/or viewed fragment by fragment. Typical viewing devices, for example, have a smaller resolution than the original raster image and thus, only a small fragment of the entire image may be viewed at a time. When the image is scrolled, new portion of the data is retrieved and decompressed. Spatial access together with a fast "on-the-fly" decompression allow the user to operate directly on the compressed data without retrieving the entire image.

Spatial access:

When an image is accessed, the entire file is typically read and decompressed into memory. This is not possible if the uncompressed raster image size exceeds the available memory resources (cf. GIS images). Besides, high-speed channels are not always available. For example, most communications channels in Russia are 14,400-28,800 bps channels based on analog phone lines. 64-128 Kbps bridges are used only for connecting separate city networks together (mostly via satellite links). The actual transmission speed practically never exceeds 1 kilobytes per second.

The decompression of the entire image can be a major source of inefficiency. Only a small part of the image is often needed, or the image is processed and/or viewed fragment by fragment. Typical viewing devices, for example, have a smaller resolution than the original raster image and thus, only a small fragment of the entire image may be viewed at a time. When the image is scrolled, new portion of the data is retrieved and decompressed. Spatial access together with a fast "on-the-fly" decompression allow the user to operate directly on the compressed data without retrieving the entire image.

D. LINE-DRAWING IMAGES

The complete set:

The complete test set consists of 28 line-drawing images, divided into four classes: *electrical circuits*, *engineering drawings*, *cartographic maps*, and *architectural and urban plans*. The images are taken from real-life applications and amount in about 43 Mbytes in uncompressed form. The format of the images varies from A4 to A2.

Statistics of the test set:

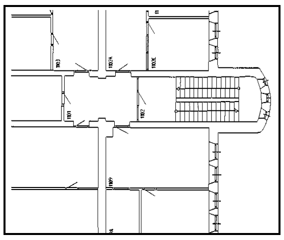
Image type	No. of images	Total size	Smallest	Largest
Circuits	6	5.8 Mbytes	1480 × 2053	5522 × 4039
Drawings	8	13.2 Mbytes	1765 × 1437	7296 × 4903
Maps	5	12.9 Mbytes	3100 × 3475	6608 × 4677
Plans	9	10.0 Mbytes	1253 × 970	5888 × 5888
TOTAL:	**28**	**42.8 Mbytes**	—	—

The mini-sets:

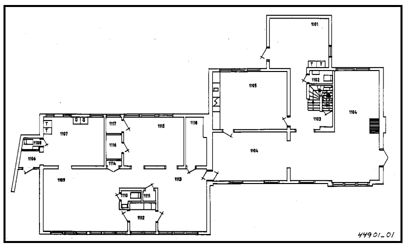
There are two line-drawing mini-sets A and B, each contains three images from the complete set:

A

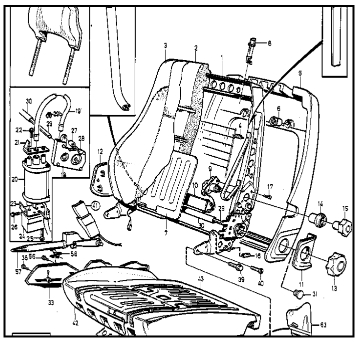
Plan (2167 × 1788)



House (4803 × 2873)

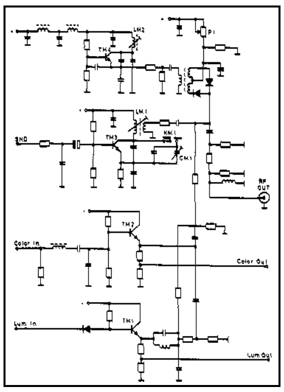


Chair (2842 × 2748)

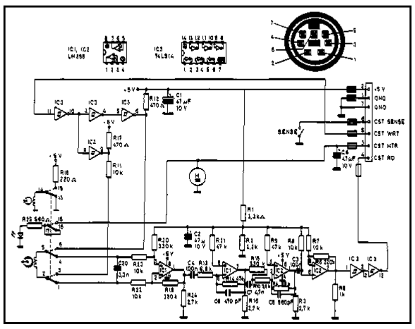


B

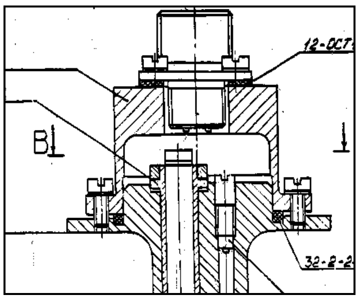
Module (1480 × 2053)



Power (2293 × 1787)



Bolt (1765 × 1437)

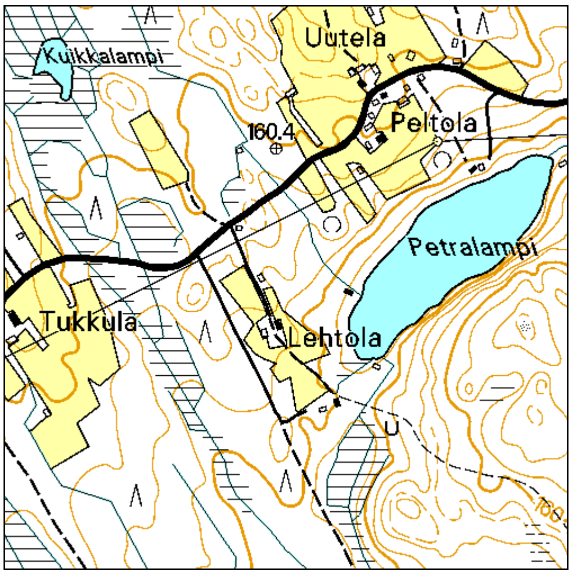


E. NLS Map Images

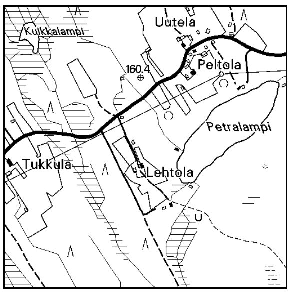
The test set includes five randomly chosen images from the “NLS Basic Map Series 1:20000” corresponding to the map sheets *No/No* 431306, 124101, 201401, 263112, and 431204. Each map image consists of four binary component layers, each has the size of 5000 × 5000 pixels. The images are denoted as $Image_X$, and the layers as $Layer_X$, where *Layer* is the layer name, and X is the relative domain number from 0 to 4 in a given order. The layer names are following:

- *Basic* – topographic image, supplemented with communications networks, buildings, protected sites, benchmarks and administrative boundaries;
- *Contours* – elevation lines;
- *Water* – lakes, rivers, swamps, water streams;
- *Fields*.

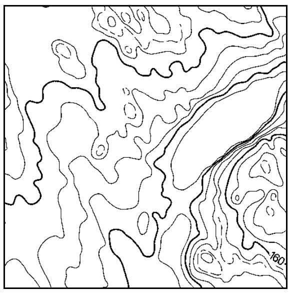
Sample fragment of a test image (left) and its four binary components (right):



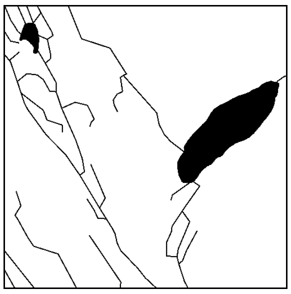
Sample test image (500 × 500)

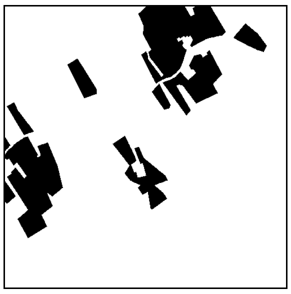


Basic



Contours

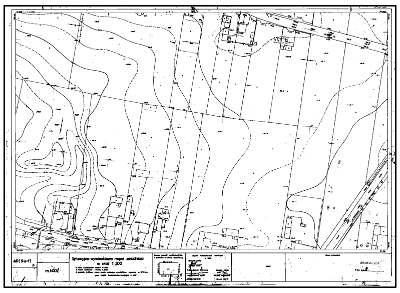

Water


Fields

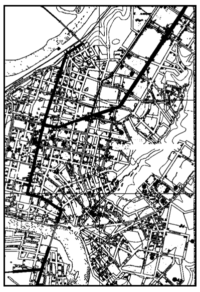
The image is re-printed with the permission of National Land Survey of Finland and available via web page: http://www.nls.fi/kartta/democd/aineisto/index_e.html.

F. **GIS** IMAGES

This mini-set consists of four images typical for GIS applications.


6608 × 4677

3425 × 4697


2368 × 3568

3322 × 5355