Department of Computer Science

The University of Waikato
Te Whare Wānanga o Waikato

Hamilton, New Zealand

Lossless Document Image Compression

Stuart J. Inglis

This thesis is submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy at The University of Waikato.

March 1999

Abstract

Document image compression reduces the storage requirements for digitised books or documents by using characters as the fundamental unit of compression. Compression gains can be achieved by identifying regions that contain text, isolating unique characters, and storing them in a codebook.

This thesis investigates several fundamental areas of the compression process. Algorithms for each area are tested on a corpus of images and the improvements tested for statistical significance. Methods for isolating characters from a bitmap are investigated along with techniques for determining reading order. We introduce the use of the docstrum to aid image compression and show that it improves upon previous methods. The Hough transform is shown to be an accurate method for determining page skew and gives robust results over a range of image resolutions. Compression is shown to improve when the skew of an image is determined automatically, and used to determine reading order.

If images can be segmented into regions of text and non-text, it is possible to change the compression model to reflect the region's contents. Instead of developing *ad hoc* methods to classify the regions, we introduce the use of machine learning schemes, and show both high predictive accuracy and compression improvements. Machine learning schemes are also applied to the problem of matching characters to determine similarity. Using the scheme's choice of parameters, the matching methods and their respective compression gains can be compared fairly. We investigate the changes in codebook size and compression gains when compressing multiple pages.

The compression system presented is complete, free of arbitrarily introduced parameters and achieves the best known lossless compression on a public corpus of images.

Acknowledgements

I have loved working in the Department of Computer Science at the University of Waikato. Many people have made the last few years extremely enjoyable.

I would like to continue the tradition of being indebted to my supervisor. Professor Ian Witten has provided everything that I asked for and often more than I expected. Over the last few years he has supported my thesis, along with my many forays into less thesis-oriented areas. He has supplied both academic and financial support allowing me to attend many conferences and to enjoy an extended period of study in Calgary, Canada. Thanks Ian.

There is a large group of people that I would like to thank. These people collectively are known as "the department." Thanks to Geoff Holmes for general inspiration and guidance over the years. Thanks to Ian Graham for scholarships, travel costs and the donation of the ubiquitous teal t-shirts to our department soccer team. Thanks to Lloyd Smith for the awesome music and the rendition of that classic Beatles song "write in C." Years later it still has me humming in the shower. Thanks to Sally-Jo Cunningham for the legendary tearoom parties that made the computer science department *the* place to be. The people who I would like to thank most are the people with whom I spent most of my time. These are the graduate students: Len Trigg, Mark Hall, Eibe Frank, Bill Teahan, Richard Littin, Gordon Paynter, Kirsten Thomson, Alvin Yeo, Stephen Donnelly and David McWha. Special thanks to the proof-readers of this thesis: Geoff Holmes, Len Trigg, Stuart Yeates, Margaret Wilson and Nicola Inglis.

I relied on freely available software for every aspect of my thesis. I owe a huge thank-you to the GNU project, along with all of the countless people who develop software—not for money, but for fun. In no particular order I directly used the following incomplete list

of software for this thesis: gcc, gdb, XFree86, netpbm, fvwm, make, bash, gnuplot, tex, latex, linux kernel, UnixStat, and a lot of other programs offered by the GNU project. I am releasing the software developed for this thesis under the GNU public license, in the hope that others will find it useful.

I have enjoyed the years of my Ph.D. study more than anything I have ever done, although there were many times when I thought the process would kill me. When the number of lines of code I have written for my thesis is counted it comes out at around 15,000 lines. When I performed the same operation in my `~/play` directory, where most of my non-thesis work was performed, I found I have written around 80,000 lines of code in over 50 sub-projects. These projects have included a character recognition system, image compression using machine learning, a chess playing program, random dot stereograms, a garage sale route analysis system, encryption code, battleship type games, sky television decoders, and my favourite—a stochastic photo-simulator whose sole purpose was to find the optimum position of fluorescent lights over a tropical fish tank! And who could forget the "multi player 2D scrolling moving the tank over the hills" game. I really think it needed a better name.

My love of school which developed into my love of University, is directly attributed to my parents. I have thanked them many times before, but it has to be said again. Thanks Mum and Dad. Thanks also to Dave for his support over the years.

There remains one person who needs to be thanked. Thanks to my wonderful wife Nicola for the years of putting up with me working late, working on weekends, or using thesis writing excuses to avoid watching movies like "They came to talk" or "The horse whisperer." She is an awesome proof-reader and has helped me keep working over the years. Nicola, I owe you one.

It looks like nine and a half months is the perfect amount of time to allocate to writing. Our first child, Caroline Grace, was born the day after I handed in this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Johann Gutenberg invented the modern-day printing press in Germany around 1450, and since then we have slowly become inundated with paper documents. Although pulp-based paper was manufactured in China as early as the second century A.D., and quality paper was widely available by the fifth century, the availability of printed material only started to increase after Gutenberg introduced the use of individual metal letters in the printing press. In less than fifty years following the first printed book, more than 1700 printing presses had been used in Europe, which had printed a total of 40,000 separate works [Joh66]. By 1975, almost four hundred years later, it has been estimated that approximately 50,000,000 works had been published [WMB94].

Today, many problems that arise with printed material are not related to the production aspects of papermaking, but with the physical storage requirements that such vast volumes of paper require. The storage requirements for digital images are several orders of magnitude smaller than for paper-based documents. Take for example, a 15.9 GB DVD disc. This can store approximately 16,000 uncompressed A4 300 dpi pages, and (including the case) requires a storage volume of 0.0001 cubic metres. When we compare this with normal 80 gsm paper, the storage volume would require 0.16 cubic metres, a physical storage increase of almost 1,600 times. This figure is impressive, but by using the image compression techniques presented in this thesis, it is possible to increase the amount that can be stored by

around 27 times, and store upwards of 430,000 pages on this same disc. The corresponding physical storage required compared with paper would be reduced by around 43,000 times. A library with 2,000,000 books each with 200 double-sided pages could reduce their storage requirements to a mere 0.2 cubic metres by digitising and compressing the data. Actually, the savings are higher than this, as this calculation ignores the additional savings due to a reduction in book bindings, shelving and aisle space—and in practice, books are smaller than A4 size.

Large potential gains like these drive the quest for increasingly better document image compression. This thesis investigates the various stages of the compression process, and in doing so, develops a freely available compression system that achieves the best known compression results.[1]

1.1 Motivation

Bitmapped documents are large, with a single digitised page requiring over 1Mb of storage if stored naïvely. Because images often contain large expanses of white areas as well as repeated characters, the image can be stored more efficiently. There have been many approaches to reducing the storage requirements of digital images. Capon was one of the first to introduce the compression of general images by run-length encoding [Cap59]. Run-length encoding replaces long contiguous lengths of same-colour pixels with a pair of numbers that represents the colour and length. When compressing 300 dpi images, run length encoding reduces the storage requirements by around five times. This approach works by analysing pixels. The first document specific compression method was designed by Ascher and Nagy [AN74], which extracted characters from an image and instead of storing multiple copies of a character, used an index to point to a previous example of the same character.

[1] The document image compression software that accompanies this thesis is free under the terms of the terms of the GNU public license, and can be downloaded from `http://www.cs.waikato.ac.nz/~singlis`.

Ascher and Nagy estimated compression ratios of around 16 to 1.

Today, this basic methodology remains unchanged, although the details (and therefore the compression ratios) have altered dramatically. We call the compression of bitmap images that have been scanned from paper-based documents *document image compression.* The process consists of several stages. The original image is first processed to extract connected regions of pixels, called *components*, that are used to approximate individual characters. The components are sorted, arranged into *reading order*, and processed sequentially. As each component is processed, a decision must be made: has the component been compressed previously, or is it a hitherto unseen character? The component, or the index to the previously compressed component, is compressed along with information that relates to its position on the page. When document images are compressed in this fashion, the DjVu system compresses the images with a 26.5 to 1 ratio [ATT99]. This thesis investigates parameters that affect the compression results, and extends the document image compression process in a variety of ways.

1.2 Objectives

Many assumptions have been made during the development of document image compression systems; about image resolution, whether the image is skewed or not, how many pages are compressed at a time, and the quality of the scanned image. Moreover, these systems do not discriminate between compressing areas of text or areas of graphics. This thesis examines these assumptions for lossless document image compression in finer detail. Lossless compression reproduces the original image exactly, unlike lossy compression which trades off higher compression with slight reproduction errors. The objectives of this thesis are:

- to investigate methods for decreasing the compressed file size during lossless document image compression;
- to explore methods for compressing document images that minimise the number of parameter choices and assumptions that are imposed;
- to design a robust method for detecting skew in document images, and the use of skew in reading order determination algorithms;
- to design an automated method for detecting the differences between text and non-text zones in a document, and to investigate various methods for encoding the different zone types;
- to use machine learning techniques to combine template matching methods to achieve better results than are possible using a single method;
- to investigate the best method for matching components in the codebook;
- to investigate the compression of multiple pages, and the influence that equivalence class parameters have on compression performance.

1.3 Thesis statement

The objectives can be summarized cohesively by our thesis statement. Because our guiding focus is the development of an automated compression system we make two claims:

> An automated and robust system for the lossless compression of document images can be designed that achieves good compression, is tolerant to document skew, and is able to perform well and adapt over multiple pages in a document. Best performance is obtained when parameters are selected automatically using machine learning techniques.

An automated document image compression system does not rely on manual intervention to achieve good results. While most compression schemes fulfill this requirement, Chapter 8 briefly describes how manually specified information can be used to improve results. We investigate how similar gains can be achieved automatically. By robust, we imply that the compression performance of the system is not unstable, and that the compression ratios will be consistent when small perturbations are applied on the input images. Specific examples include page skew, which occurs when documents are not perfectly aligned with scanning equipment, and noise, which is dependent on the cleanliness of the scanning surface and environmental conditions relating to the quality of the original document.

Machine learning techniques are used to automatically determine thresholds that are used by various areas of the compression process. Instead of fine tuning parameters on the test images, and evaluating the results on these same images, we show that when the learning schemes are trained on independent data, the classifications they produce increase compression. In this way, the compression process is seen as the validation stage, because the classification parameters are independent of the image corpus.

1.4 Thesis structure

Chapter 2 introduces document image compression, and places this thesis in its historical context. The compression process is partitioned into six areas: component extraction, determining reading order, skew detection, template matching, component codebook and separating text from graphics. A *base* compression system is presented, and the parameters that are needed in each of these six areas are described. The compression performance of the base system is compared with a variety of compression methods, including the best commercial compression system. The base system is updated to include the advancements made in each chapter. The compression results of the final system are shown in Chapter 9.

Various issues surrounding the extraction of components from images are pursued in Chapter 3. Section 3.1 shows that there are significant differences between nested 4-connected and nested 8-connected extraction, and that using nested 8-connected extraction gives better compression. Section 3.2 shows similar results between 4-connected and 8-connected extraction of non-nested components. The best results from these two sections are compared in Section 3.3, where we show that although there is a significant difference between the number of components and number of classes generated, there is no significant compression difference between 8-connected nested and 8-connected non-nested extraction. Section 3.4 shows that there is no significant difference in the mean compressed file size when a component's maximum width or height is bounded.

After components are extracted from the image they are sorted into reading order. Chapter 4 describes methods used for determining reading order, and Section 4.2 introduces a new method that is based on using components' nearest neighbours. Section 4.3 compares the new method with five others and shows that it achieves the highest compression in three situations: when using the original test images, when the images have been skewed by a random amount, and when the images have been rotated by 90 degrees. Section 4.4 compares encoding a component's index using a gamma encoder and encoding using a PPMD model with various orders. It is shown that the second order PPMD model is significantly better than the gamma encoder. A method of encoding components that represent noise is introduced in Section 4.4.1, and we find no significant improvement on the test images. Section 4.4.2 investigates whether the automatic insertion of spaces into the index stream would help compression. We show that the improvements are not significant during compression, due to the small number of components on the average page.

Chapter 5 describes the effect that scanning an image with skew has on the compression process. Section 5.2 introduces our design criteria for a skew detection system; its implementation is introduced using the Hough transform in Section 5.3. Section 5.4 shows

that it is possible to get compression gains when the skew is determined manually. The Hough transform is used to automatically determine the skew of images in Section 5.5, and we show that the method is robust over a range of resolutions, and that it gives significant improvements in compression.

Chapter 6 describes common template matching methods and a new template matching method based on cross-entropy is introduced in Section 6.2. Using the machine learning scheme C5.0, the features generated by each of the template matching methods, including the new one, are combined in Section 6.3. A large database is needed to train the machine learning scheme, and is described in Section 6.4. The classification results for each of the methods are shown in Section 6.5: we find that the combined method gives the best template matching accuracy. These results are applied to a simple lossy compression scheme in Section 6.6, and the lossless scheme in Section 6.7. These last two sections explore the relationship between template matching accuracy and compressed file size, and suggest reasons why the template matching method with the highest classification accuracy does not give the best compression.

Chapter 7 introduces issues involved with the codebook and component compression. Section 7.1 shows the compression gains that can be achieved when multiple pages from the same source are compressed consecutively, without re-initialising the codebook between pages. The section shows that both the maximum number of equivalence classes and the maximum number of examples in each class can be limited without degrading compression performance. A variety of different contexts used for pairwise component compression are introduced in Section 7.2, and the best context is determined. Section 7.3 introduces the use of ternary contexts, and shows that they give significant compression gains over binary contexts. The decision to update the global context information after pairwise compression of each component is investigated in Section 7.4, and we show that this gives no significant benefit. Aligning components based on their centroids or centres is investigated in

Section 7.5, where we show that it is better to encode the necessary extra information to match using the centroid. By matching on the centroid, significant compression gains can be achieved. When matching a component against the codebook, there are a variety of possible matching options. Section 7.6 shows that the best strategy is to compare each component with all previous components to find the best match, as opposed to comparing with the average for each class.

Many document images contain pictures that do not compress well using component-based techniques. Chapter 8 introduces methods for analysing regions in the image to determine whether they are composed of text or non-text. A variety of features that are calculated for each region on the page are defined in Section 8.2. When the zones are determined manually, Section 8.3 shows that significant compression gains can be achieved. Section 8.4 shows that when the docstrum is used to cluster the components into zones, we no longer achieve compression gains. Instead of attempting to cluster the page into zones and then automatically classify them, Section 8.5 classifies the individual components using machine learning to be either text or non-text. Using this simpler approach, large compression gains are achieved.

1.5 Image corpus

Previous document image compression results have been obtained using the eight standard CCITT facsimile test images [HR80], or with hard-to-obtain private image collections. This thesis uses a large publicly available database of 979 bi-level images, published by the University of Washington (UW), as the corpus for all experiments [UW93]. A subset of fifty randomly chosen images was picked for use in this thesis. The images are shown in Appendix A.

The images in the UW English Document Image Database are scanned at 300 dpi from

126 different journals. Many have small non-zero skew angles, and in some cases text is oriented vertically instead of horizontally. Each has been manually segmented into zones; there are a total of 13,831 zones in the database, an average of 14 per image. For instance, a particular image may be divided into six text zones, one halftone zone, and one table zone. The corpus contains a total of twelve zone classes. Text zones, which are by far the most frequent (88% of all zones), represent paragraphs, bibliography sections, page numbers and document titles; other zone types include line drawings, halftones, math, and rules.

1.6 Machine learning schemes

Machine learning has been defined as "the acquisition of structural descriptions from examples" by Witten and Frank [WF99], and in this thesis we use two of the best performing machine learning schemes to automatically derive classification rules and parameters that can be used to generate predictions. Witten and Frank stress that the performance of schemes is not as important as the *knowledge* that is acquired, and that "people generally use machine learning to gain knowledge, not predictions."

There are several terms that we will use in subsequent chapters. A *feature* is a measurement or a calculation that is present for all objects. Chapter 8.2 uses features such as *area* and *aspect ratio* to determine the classification of regions in an page. An *instance* equates with a particular object and may have many corresponding features. In the context above, each instance represents a particular region in the page—many features are calculated for each region. Each instance has an added feature: the classification of the instance. The machine learning schemes use the features to predict the classification of each instance.

Throughout this thesis, certain parameter values in the compression system will need to be specified which directly affect the compression ratios. Instead of compressing the images with each parameter value and choosing the best, the machine learning schemes analyse

the data independently of the compression process. The results that are generated are then used for compression. This training and testing division reduces the likelihood of spurious results, as the machine learning decisions are not based on the test images.

Two machine learning schemes are used in this thesis. The first scheme is is a Naïve Bayesian classifier [LIT92]. The second scheme is C5.0 [RUL99], which generates classification decision trees. These schemes are described briefly in the two following sections, but for a complete treatment consult [WF99]. C5.0 was obtained from the Rulequest web site [RUL99], and Naïve Bayes is a component in the freely available WEKA software from the University of Waikato [WEK99].

1.6.1 Naïve Bayes

Naïve Bayes is a statistical technique which implements the conditional probability rule, otherwise known as Bayes' rule. Naïve Bayes is so-named because it assumes that events are conditionally independent, and that the probabilities of two events can simply be multiplied to calculate the probability that both events will occur. Probabilities for nominal features are typically estimated using a Laplace estimate based on counts (the Laplace estimator initialises counts to 1 instead of zero, and avoids the problems caused by observed frequencies being zero). The probabilities for numeric features are also easy to calculate using a Gaussian estimator. A typical Naïve Bayes implementation assumes that features are normally distributed with respect to the class, and uses the Gaussian probability density function to give a probability for any numeric value.

Although many datasets do not have independent features, Naïve Bayes still performs admirably, and often outperforms more complex schemes [WF99].

1.6.2 C5.0

C5.0 is the commercial successor to the C4.5 decision tree method. Decision tree methods create a leaf node that predicts a distribution of class values, and recursively evaluates whether to convert the leaf into a decision node. Each possible split of the current node is evaluated using an information gain measure, and if the information gain when converting the leaf into a decision node is greater than the information gain of leaving it as a leaf, it is split and the process continues recursively. The information gain measure is an entropy calculation that represents the number of correct and incorrect classifications in each class. Numeric features are sorted and points across the range are evaluated as possible split points.

Often decision trees over-specialise, or overtrain, with respect to the training data. C4.5 attempts to combat this process by allowing pruning of the decision tree. Pruning removes branches in the decision tree that appear to be overly specific. Two different types of pruning are performed: sub-tree pruning and sub-tree raising. Sub-tree pruning replaces internal nodes with leaf nodes, while sub-tree raising replaces a node with a node that appears below it. Decision tree pruning uses heuristics to estimate the validation error rate at each node. Although pruning uses an *ad hoc* measure to estimate the error, it appears to work well in practice [WF99]. The details of the commercial C5.0 system are not clear, although the decision tree generation process is assumed to be similar to C4.5.

1.7 Compression

Image compression techniques will be explained in the text, but there are several general terms and concepts that need defining. Compression is the reduction of repetition. Shannon [Sha48] showed that the degree of uncertainty in a message could be defined as the *entropy*

of the message, and is the foundation of Information Theory. If a message is defined to be a sequence of n events, each with a probability $p(i)$, then the entropy of the message is $-\sum_i \log_2 p(i)$.

The compression process is therefore divided into two parts: the method of encoding the probabilities and the modelling of the message stream to accurately predict the probabilities. Various models to predict probabilities of events are used throughout this thesis. We use a conditioning context made up of pixel colours to encode a component's bitmap (Section 2.4.9), we use the tree-based gamma encoder to build models for encoding numbers, (Section 2.4.5) and we use PPM to predict the symbol indices (Section 4.4). Section 1.7.1 discusses the entropy coder we will be using, and Section 1.7.2 discusses the PPM model.

1.7.1 Entropy coding

The entropy coder converts the probabilities into a sequence of bits that can be transmitted. Huffman coding [Huf52] is one solution, where each probability is converted into a static bit sequence. Huffman coding is analogous to Morse code.

Instead of encoding each probability independently, arithmetic coding combines the probabilities for each event to form the probability of the entire sequence [RL79]. This combined probability can then be encoded efficiently. The implementation used in this thesis is by Moffat *et al.* [MNW95]. Their implementation incrementally encodes the probabilities by continually rescaling the probability range and using integer arithmetic.

1.7.2 PPM

Prediction by partial match (PPM) models a sequence of symbols using a context which corresponds with a sequence of previously encoded symbols, and is the best performing

compressor for text [Tea98]. If a context of a specific length has been encoded previously, it is used to predict the next symbol. Otherwise, PPM iteratively *escapes* down to a smaller length context. If no context of any size matches, PPM encodes the symbol using an order -1 model which allocates a fixed probability to each symbol.

The variations of PPM have different methods of calculating the escape probability. The specific version we use is PPM with method D escape estimation, or PPMD [How93]. Method C calculates the escape probability as $e = \frac{d}{n+d}$, where d is the number of distinct symbols that have followed the context, and n is the total number of symbols that have followed the context. Method D is a minor modification of method C—instead of incrementing the symbol and escape counts using the Laplace estimator (1), the Dirichlet estimator ($\frac{1}{2}$) is used [Tea98].

1.8 Significance tests

All tests for statistical significance throughout this thesis are two-tailed paired t-tests, that have been applied at the 0.01 (99%) confidence level. When analysing the fifty test images at this significance level, the critical value is $t = 2.68$. This means any t value greater than 2.68 indicates a significant result. The symbol $\surd$ is shown to indicate a significant difference, while the symbol $\times$ denotes that there is no significant difference. The $\surd$ symbol does not imply an order between the results, rather that the distributions are significantly different. The mean values for the distributions must be compared to indicate relative performance. Throughout this thesis, the phrase "significant" is used consistently to mean "statistically significant at the 0.01 confidence level."

Chapter 2

Document image compression

When documents are machine printed in languages with a small alphabet, most of the characters on a page occur multiple times. In the original digital document, each character is simply an index into a table that represents the character set of the language. The ASCII table is ubiquitous, while the Unicode character table is poised to become the new standard. When the document is printed, it retains the same form as was used to store it electronically. Human readers recognise each letter not as groups of black pixels, but as characters. Unfortunately, when a page is digitised all the index information is lost, and each character is rendered as a bitmap of pixels.

So long as the character is stored in an electronic document format, the memory requirements for each character are static. They do not change when the character changes font size or style. Nor do the memory requirements depend on the resolution or size of the output device. But once a document is rendered, memory usage increases dramatically. A single character may require 8 bits to store internally, but after it is rendered it may require a bitmap of 600 pixels. More problems arise when the resolution of the device increases. When a document is printed at 600 dpi, it contains four times as many pixels as the same document scanned at 300 dpi. As technological advances are made, scanning and printing resolution increase, which in turn drives up storage requirements.

To store document images efficiently, methods must be developed to effectively reverse

engineer the printing process. Ideally, it should be possible to compress a printed document into the same space as the original document. In practice, the internal document format will not be compressed, so we should be able to do better.

Instead of storing the hundreds of pixels that are required for each character on a page, we only need store a single representative sample, and reference this sample via an index. When we require multiple types and styles of each character, we only need a single instance of the style to be able to faithfully reproduce the document. When an image that has been compressed using this technique is decompressed, the result will closely match the original image but the entire process will have introduced loss. Characters that are bold may be matched against non-bold characters, broken or noisy characters may be smoothed, or matched against noisy characters.

Lossless compression is important because non-textual information such as halftone images, drawings, fingerprints and coffee stains, need to be compressed without introducing loss.

The compression process described above is lossy. To complete the process and achieve lossless compression there are two options. First, the difference between the lossy image and the original image can be encoded after all the components have been transmitted. Second, the difference between each component and the corresponding matched component in the codebook can be encoded, making the process lossless. Our focus in this thesis is the latter method because it is extensible over multiple pages and is flexible when matching against the codebook.

2.1 Previous work

One of the first methods to reduce the storage requirements of images was introduced by Capon [Cap59], who suggested replacing long runs of pixels with a count that represented

the length of the run. A contiguous run of pixels could be replaced by a bit describing the colour, followed by information about the length of the run. This technique came to be known as *run-length encoding*. Run-length encoding of the UW database achieves a total compression ratio of 4.5 to 1.

The business world was revolutionised by the introduction of an international standard for facsimile encoding and transmission. The so-called *Group 3* and *Group 4* fax standards achieved significant improvement over run-length encoding. Group 3 compression can achieve compression ratios of 6.4 to 1, while the more advanced Group 4 standard achieves 13.5 to 1 compression on the UW corpus. Both standards use a two-dimensional coding scheme that incorporates horizontal run-length encoding and vertical difference encoding of pixels [HR80].

The Joint Bi-level Image Experts Group standard (JBIG, also called JBIG1) introduced impressive technology that increased the compression ratios to 19.4 to 1 [JBI93]. The JBIG standard was not as widely accepted by the mainstream as the original Group 3 and 4 standards, as it was hard to compete with the huge installed base of facsimile machines, with a perceived small increase in compression. JBIG combines a 10-pixel context (see Figure 2.5) with a table driven arithmetic coder (the QM-Coder [JBI93]) to achieve quick compression with high compression ratios.

Moffat [Mof91] described a simple context based scheme and gave results for many different sized templates. He also introduced a two-stage context that used a smaller subset of the full context until the full context was determined to be a good estimator. Moffat showed that a full context of 22 pixels with a subset of 10 pixels provided excellent compression, with results on the UW corpus of 21.1 to 1 compression.

The MGTIC system was introduced as a freely available document image compression scheme [IW95, IW96, WBE+94], and achieves 22.1 to 1 compression on the test images.

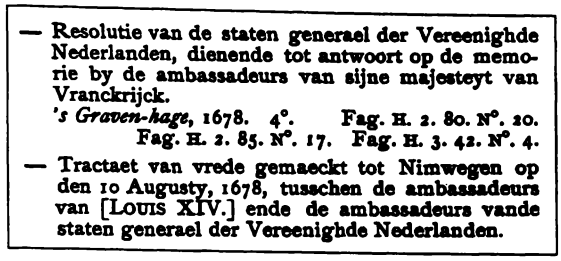


Figure 2.1: An index page from the Trinity College library in Dublin, Ireland

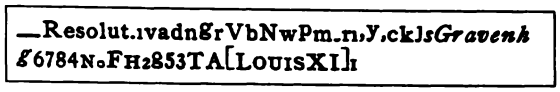


Figure 2.2: The components that make up the codebook

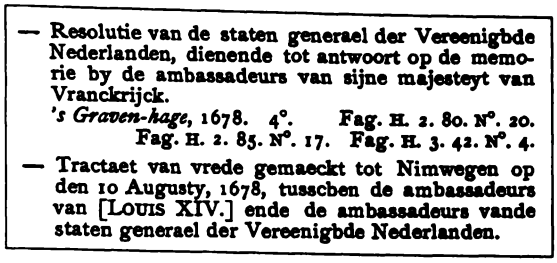
Zhang [Zha97] extended MGTIC, but against the spirit of the original software, did not make his modifications publicly available.

The commercial DjVu document compression system from AT&T Labs–Research [ATT99] extends the work of Howard [How96] to include the compression of grey-scale and colour images. Using the DjVu compressor (version 1.1.5) in lossless mode with the default parameters achieves 26.5 to 1 compression on the UW images.

2.2 Component-based compression

Component-based compression methods evolved from pixel-based methods. Instead of using pixels they use the character as the fundamental unit of compression. A codebook, or library, of characters is generated that allows an index to be compressed instead of the actual character in bitmap form.

The compression process is illustrated in a series of figures. Figure 2.1 shows the original image taken from an index page from the Trinity College library in Dublin, Ireland. The components have been extracted from the image and are shown in Figure 2.2. When the image is reconstructed using the codebook, the result looks like Figure 2.3. Because



— Resolutie van de staten generael der Vereenigbde Nederlanden, dienende tot antwoort op de memorie by de ambassadeurs van sijne majesteyt van Vranckrijck.
's Graven-hage, 1678. 4°. Fag. H. 2. 80. N°. 20.
Fag. H. 2. 85. N°. 17. Fag. H. 3. 42. N°. 4.

— Tractaet van vrede gemaeckt tot Nimwegen op den 10 Augusty, 1678, tusschen de ambassadeurs van [LOUIS XIV.] ende de ambassadeurs vande staten generael der Vereenigbde Nederlanden.

Figure 2.3: The lossy reconstruction of the image

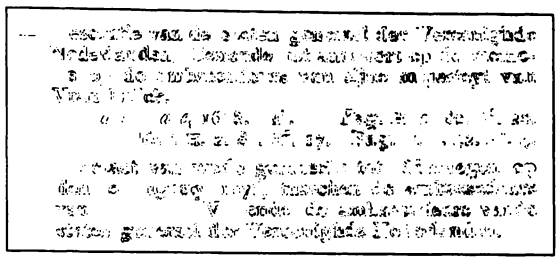


Figure 2.4: The residue, or difference, between the original and the lossy image

the codebook only stores a single example of each character, the reconstruction process is lossy. The difference, or *residue*, between the original and reconstructed image is shown in Figure 2.4.

There are two alternative approaches to codebook creation. The codebook can be pre-calculated and encoded statically at the beginning of the compression process, or it can be created adaptively and updated continually during compression.

2.2.1 Static codebook

Static codebook methods first cluster all the characters into homogeneous classes, such that each class contains a representative sample of a unique character. The codebook is encoded first, and each subsequent character is encoded as an index into the codebook. Static techniques are lossy by design, and are often combined with a second encoding stage to ensure lossless compression [WBH+92, WBE+94].

Systems that use static codebook creation suffer from one main problem: it is difficult to

modify the codebook as pages are processed. To compress multiple pages the system must either form a codebook based on the set of pages, or encode creation and deletion operators in the index stream. There are no known systems that use this approach for multiple pages.

For static codebook systems to achieve lossless compression the residue must be encoded after the individual components have been compressed. The advantage of this method is that decompression can be performed in a two-stage process, where an image can be decompressed into either a lossy or lossless form. A decompressor may stop to achieve a fast preview of the image, or continue for the full lossless decompressed image. The lossy version of the image displays quickly because it often corresponds to only quarter or half of the total compressed file size.

Another benefit of the static codebook is that there is a unique index that corresponds to each character in the codebook. As the index values do not change over time, models that require context, such as PPM [CW84], can be used to compress the index stream.

2.2.2 Adaptive codebook

Adaptive codebook techniques do not require the codebook to be transmitted before the components are compressed. Instead, it is created dynamically as unique characters are processed. The addition or deletion of components in the codebook is incremental, and is performed after each component is processed [MRA84].

Adaptive methods can easily become lossless methods, by encoding each component with respect to its match without loss.

Unlike static codebook methods, adaptive systems can match against any previously encoded character. One main drawback of this method is that components are not consistently matched with the same character, and the index value is not constant.

2.2.3 Hybrid codebook

The static and adaptive codebook methods can be easily combined. Instead of matching components adaptively with respect to a previously matched component, equivalence classes can be formed dynamically with matches performed with respect to an average character formed from each class.

This novel method allows the codebook to be recalculated as the document is processed, giving a consistent index value to all characters. The hybrid method is incorporated into our base compression system presented in Section 2.4.

2.3 Historical development

The Pattern Matching and Substitution (PMS) technique introduced by Ascher and Nagy [AN74] was a wholly lossy technique, and achieved very good compression ratios. Ascher and Nagy took the approach of extracting the characters from the bitmap and forming equivalence classes, where each class contained a single representative of each character. The scheme would first encode the entire codebook, and then encode the codebook index of each character in the original image. Extra information such as the relative position on the page was encoded. Once the stream of indices had been encoded, the image could be decompressed into a *lossy* form of the original image. Ascher and Nagy achieved around 16 to 1 compression on 200 dpi images. The PMS system encoded integers using fixed length codes, and they proposed extensions such as Huffman coding indices relative to their frequency.

Pratt *et al.* introduced the Combined Symbol Matching (CSM) scheme which extended the work of Ascher and Nagy [PCC+80] to allow lossless encoding of images. They compressed the page in two stages, the first stage following the PMS approach and the second

compressing the *residue*, which is the difference between the lossy and original image. Compression ratios of 25 to 1 were achieved on 200 dpi images.

Mohiuddin [Moh82] introduced a method of lossless image compression that was symbol based, but created the codebook dynamically. Mohiuddin's system worked by encoding the symbols sequentially. Each symbol could be encoded in two different ways, as previously unseen with all the pixels in the character being transmitted, or as matching a previous character in which case the index information was encoded. It is here that Mohiuddin's adaptive scheme differs from the static techniques. As well as encoding the index information, the difference between the actual character and the matched character is encoded, which guarantees lossless compression of the image. This approach differs from the hybrid scheme as equivalence classes are not formed.

Witten *et al.* introduced the *Textual Image Compression* (TIC) system which updated Pratt's CSM to include the use of PPM [CW84] to code component indices, arithmetic coding, gamma coding of integers, and two-stage contexts for image compression [WBH+92]. TIC extended the residue coding scheme to use *clairvoyant* contexts to compress the residue *with respect to* the lossy image. Clairvoyant contexts will be introduced in Section 2.4.9. This system was written by myself and was released as the MGTIC system [WMB94]. MGTIC extended the TIC system by using two-stage contexts for residue compression. Both the normal and clairvoyant contexts were two-stage [Mof91].

Mark and Shieber developed a commercial document image compression system [MS94] which appears similar to the approach of Witten *et al.* They extended TIC to include more sophisticated template matching which gives fewer errors on smaller characters, while storing the components in a compressed form to decrease compression time. Chapter 6 challenges this notion that better template matching increases lossless compression ratios.

Howard introduced Soft Pattern Matching (SPM) which used a similar adaptive model to

Mohiuddin's, but extended it to differentiate between the two different types of matches, updated the choice of contexts, used the QM-Coder, and introduced a novel extension for lossy compression. Lossy compression can be achieved by either pre-processing the original input image, or altering the encoding process, so that certain (possibly easy to predict) pixels are not encoded [How96]. Lossy SPM can be combined with the residue encoding stage to replicate MGTIC's two-level progressive encoding. SPM achieved lossless compression ratios between 10 and 20 percent better than MGTIC.

Kia described a component-based compression system that processed images in a compressed domain [Kia97]. No implementation of Kia's system is available, and few results are given.

Zhang extended the code of MGTIC by modifying the template matching, reading order, and character positions encoding [Zha97]. Like Kia's method, a public implementation is not available, and although Zhang tests on a few images, no tests for significance are performed. Zhang uses the text-based CCITT images along with some private images.

2.4 Base system

This section defines the base document image compression system that is used throughout this thesis. This will provide the baseline for improvements which are introduced in each of the following chapters. The base system achieves compression ratios approximately 20% higher than the JBIG standard, and gives a compression ratio of 22.2 to 1 on the UW image corpus. It has the following specifications:

- All components are encoded
- 4-connected, nested, component extraction without size restrictions
- No skew detection

- Left to right, top to bottom reading order based on the bottom of the characters
- All numbers encoded using the tree-based gamma encoder
- Component indices encoded using the tree-based gamma encoder
- Probabilities estimated using a count-based Laplace estimator
- Code unknown components using a JBIG 10-pixel context
- Code matched components using a combined 4/7 pixel clairvoyant context
- Update the JBIG counts after pairwise encoding
- The components are matched using the XOR method with a 21% threshold
- Codebook matches against the average component seen so far in each class.

The base system reflects current practice in document image compression with the exception of the hybrid codebook matching, which has not been described before and is novel to this thesis.

2.4.1 All components are encoded

In the base system every component extracted from the image is compressed. Chapter 8 distinguishes text from non-text zones, and in doing so, reduces the total number of components compressed. The components that are classified as non-text are not extracted and remain in the image, which is then encoded using the two-level 22/10 coder described in Section 2.4.9.

2.4.2 Connectivity, nesting and size restrictions

Components are extracted using 4-way connectivity. This means that a pixel is defined to be *connected* if it has an immediate neighbour in the horizontal or vertical directions that is the same colour. As an example, if two components were adjacent via a pair of diagonal

pixels, they would be unconnected using 4-way connectivity (but they would be connected using 8-way connectivity). In the base system it is assumed the page is composed of dark printing on a light background, and black components are extracted [WMB94].

The extraction process allows nesting, so that © and Θ are each treated as two characters. If nesting was not allowed, each of these would be extracted as a single character.

There are no size constraints for the components. The minimum size for a component is a single pixel, the maximum size is limited by the size of the original image.

2.4.3 Skew detection and reading order

When documents are scanned using a typical flatbed scanner, they are often skewed. In the base system it is assumed that the page is unskewed.

The components are sorted into an approximation to English reading order: left to right, and top to bottom. A horizontal profile of the components' centroids is formed, smoothed using a 3 mm wide rectangular low-pass filter, and quantised into two values: high and low. The mean position between the peaks is assigned as the inter-line breaks. For each line, the components are sorted based on their x-centroid position.

This method works well for horizontal text, but if the text is skewed, characters from the lines above and below the current line become merged. This issue is discussed in Chapter 5.

2.4.4 Arithmetic coder

The entropy coder used in this thesis is the low-precision arithmetic coder introduced by Moffat *et al.* [MNW95]. The default parameters for the arithmetic coder are used, which encode the probabilities using 32 bits of precision.

2.4.5 Coding numbers

The document image compression system process requires the ability to encode arbitrary integer numbers in a variety of situations. For this purpose, a gamma encoder that contains probability information at each of the internal nodes is used [How96]. A gamma encoder consists of three parts. First, the sign of the number is encoded as a single bit. Second, the number of bits required to store the number is unary encoded. Third, the actual number is encoded. As an example, $+19$ is encoded as 0 00001 10011.

The tree-based gamma encoder extends the normal gamma encoding process by adaptively updating counts that represent each binary digit. The encoder is implemented by creating a binary tree that splits, based on the bit that is coded. The root node corresponds to the decision to encode a positive or negative number, and includes the probability estimate at that level. In our implementation, frequency counts are stored in each node. After each bit is encoded, the tree is followed either left or right depending on the value of the bit. The sequence of encoding the bit, updating the counts, and following the leaf nodes is followed until the number is completely encoded. Using a tree-based gamma encoder allows it to adapt to some distributions (eg. bi-modal), while still requiring $O(\log N)$ bits to encode in the limit. Using a Laplace estimator the counts are initialised to 1, incremented by 1 each time a node is used, and normalised whenever the sum of the counts is greater than 255.

The gamma coder is used to encode the difference in position between subsequent components. The difference is measured from the bottom right of one component to the bottom left of the next. The first component's position is encoded with respect to the top left of the image.

2.4.6 Symbol matching and encoding

Once the components are sorted into reading order, they are processed sequentially starting at the top of the document. For each component a decision must be made that determines whether the component is compressed as an *orphan* (that is, without reference to another component), or whether it *matches* a component which has been seen before. The matching decision is encoded before each component's bitmap.

2.4.7 Template matching

To determine whether two components match, they are aligned by their centroids and the number of pixels that differ between them is measured. If this number is less than or equal to 21% of the area of the component, they are deemed to match. When components are matched, the area is determined by aligning them and bounding their perimeters to form a new rectangle. The 21% threshold used for the base system is taken from Howard [How96].

To avoid spurious matches, a *screening* policy is used before matches take place. Two components are only matched if the difference between both their width and height is less than or equal to 2 pixels. In practice this reduces the number of comparisons between components of different point sizes. This policy has been used in several systems before, and is justified in Section 6.5.

2.4.8 Codebook

The codebook stores equivalence classes representing each encoded component. When a component is matched against the codebook, the unknown component is matched against the *average* representative in each equivalence class. The average component is calculated by aligning each component in the equivalence class with their centroids and counting the

number of black pixels in each position. If the number of black pixels is greater than half the total number of components (eg. that position is mostly black) the pixel is set to black, otherwise the pixel is set to white.

When matching against the average components, each equivalence class is compared (if it passes the screening test), and the class with the best match is used. This index to the equivalence class is then encoded.

Components are added into the codebook in two ways: either as a new equivalence class, or as a new element in an equivalence class. This distinction corresponds with the decision to encode a component as an orphan or a match. If a component does not find a match in the codebook, it is encoded and added as a new class. If it matches with a representative component in the codebook, the component is encoded (see Section 2.4.9) and is added to the equivalence class with which it matched.

2.4.9 Image coding

Images are compressed using the binary context model introduced by Langdon and Rissanen [LR81]. A *context* which corresponds to a bit-mask is applied at each position in the image to be encoded. The context is formed by using the pixel values from the image, and for each context, a count is recorded and used to predict black (c_b) and white (c_w) pixels in that context. The number of bits required to encode a particular black pixel is $-\log_2 \frac{c_b}{c_b+c_w}$ bits. The Laplace estimator is used: the counts are initialised to 1 and incremented by 1 each time the context is used. The results are scaled to half their original size whenever $c_b + c_w$ becomes greater than 255.

If a component does not match a previously encoded component, it is compressed using the JBIG 10-pixel context shown in Figure 2.5. The width and height of the component are compressed using the gamma encoder.

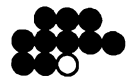


Figure 2.5: The JBIG 10-pixel context

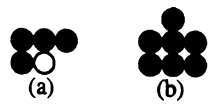


Figure 2.6: The 4-pixel context and the 7-pixel clairvoyant context

If a component is matched in the codebook, the index of the match is encoded, as well as the difference in width and height, and difference in centroid position. The bitmap of the component is *pairwise* compressed using a 4-pixel context (Figure 2.6a) and a 7-pixel clairvoyant context (Figure 2.6b). We denote this context combination as a 4/7 context. In pairwise compression, one component is being compressed with respect to another. For each pixel in the component to be compressed, two contexts are created: one from the component and the other from the matched component in the codebook. The two contexts are joined to form a single combined context that contains pixel information from both components. Each pixel is compressed using this combined context.

Clairvoyant contexts are formed with pixels whose values occur after the position of the pixel to be encoded. The use of clairvoyant contexts would not allow decompression, as information is required that is not known until later in the decompression process. Fortunately, when one component is being compressed with respect to another, the matched component is fully known by both encoder and decoder, so there are no restrictions on which pixels can be used. After a component is pairwise compressed, the context probabilities for the JBIG 10-pixel context are also updated using the recently encoded component.

The base system presented here encodes all of the components in the image. However, when components are classified as non-text in Chapter 8 they are not extracted from the image,

and the remaining image is compressed using the two-level context method introduced by Moffat [Mof91]. Instead of using a single layer like the JBIG 10-pixel context, it is arranged hierarchically with a large 22-pixel context that contains a smaller 10-pixel subset. Initially, the smaller 10-pixel context is used, but once it has been used C times, the larger one is used to predict the probability. Moffat uses $C = 2$ as the threshold for changing size, although we use $C = 5$, as informal results have shown slightly higher compression ratios.

2.5 Compression performance

The base system is compared with a variety of other schemes in Table 2.1, which shows the relative performance of compression schemes on the UW database. It compresses images almost five times more than run-length encoding, and almost twice as well as Group 4 encoding. It compresses slightly better than the MGTIC system, but not as well as the DjVu system.

We define the *total* compression as the sum of the original file sizes divided by the sum of the compressed file sizes. The *average* compression of the corpus is the mean compression ratio of the files.

As the individual areas of the document image compression process are examined in this thesis, the compression results gradually improve until we achieve a total compression of 27.0 to 1 on the UW images, which is 20% better than the starting point described in this chapter. This result is significantly better than the other systems shown.

Method	Total Compression	Average Compression
RLE	4.5	5.3
Group 3	6.4	7.3
Group 4	13.5	17.0
JBIG	19.4	25.6
Two-level 22/10	21.1	27.7
MGTIC	22.1	28.2
Base system	22.2	28.4
DjVu/JBIG-2	26.5	33.7
This thesis	27.0	33.7

Table 2.1: Compression ratios for various methods using the UW image corpus

Chapter 3

Component extraction

The character is the basic unit for written communication and therefore ought to be the ideal basis for the development of a document image compression system. Compression gains would be achieved by storing a single example of each character, and each character of the same type would be compressed by storing a pointer into a character set.

However, the specifications of an arbitrary character are difficult to define, and it is both conceptually and practically simpler to operate with a different fundamental unit: the *component*. We define a component to be a connected series of (usually black) pixels, which often correspond to a single character. The correspondence is not exact because some characters are comprised of multiple components, while other letters merge to form a single component. Characters such as these are known as compound characters, examples of which include i j ; : é Ξ ≡ as well as the ligatures ff fl ffl.

The first step in the document image compression process is to extract and order the components in an image. Once the components have been extracted from the image they will be used for further processing. This chapter describes the extraction and ordering processes.

There are three sources of variation between different component extractors, which we will express as three parameters. The first is whether the extraction process will retain nested characters as a single component or as multiple components. Nested characters are those

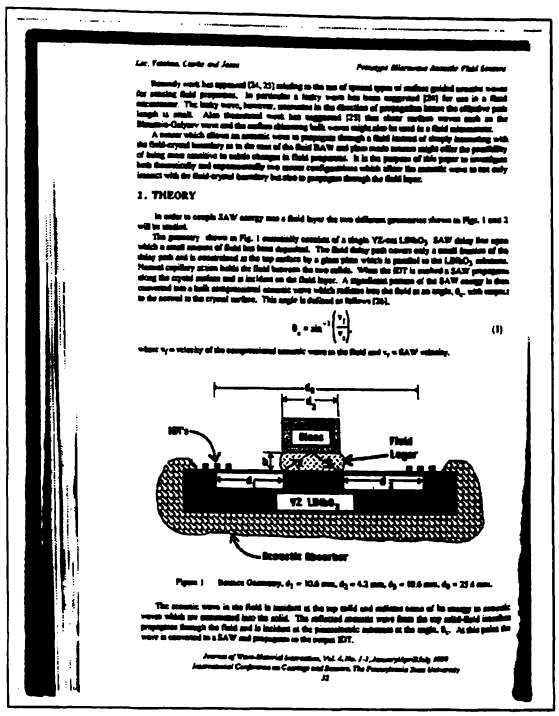


Figure 3.1: Image A034

where one component appears inside another. If components are not allowed to be nested, symbols such as © and Θ will be extracted as a single component, but special handling will be necessary to avoid creating huge components that consist of large bounding borders around text or tabular regions.

The second parameter determines the connectivity of the extraction process. A component is defined to be *8-connected* when each black pixel is connected to another black pixel in one of its surrounding eight positions. An alternative is to allow a pixel to only have four immediate neighbours, to the left and right, above and below. A component that is connected using only these four directions is defined as *4-connected.*

The third parameter determines the maximum size of a component. Each component may be bounded by a maximum width and height. When the bounds are applied, a component is broken into a number of smaller ones. Bounding is used when large components are processed and memory must be used efficiently.

	4-connected	*8-connected*
Nested	3029	2974
Non-nested	2694	2656

Table 3.1: Number of components for image A034

The choices for each of these three parameters are evaluated by implementing each method and using them in the base system presented in the previous chapter. The performance of each method is measured by its lossless compression ratio across multiple images from the University of Washington image corpus [UW93].

A typical image used in this chapter is shown in Figure 3.1. This image is 2592 pixels wide by 3300 high, and 1288017 (15%) of its pixels are black. Table 3.1 shows the number of components that are extracted from the image in Figure 3.1 using four different strategies. The components are extracted in nested, non-nested, 4-connected and 8-connected forms. Fewer components are 8-connected than 4-connected because components which would have been 8-connected due to diagonal pixels, are split and become multiple 4-connected regions.

Figure 3.2 shows an example image containing five letters. Two characters in this figure, *c* and *a*, are merged together by a diagonal pixel to form a single component. If these components were extracted using 4-connectivity this link would be broken, and the characters would be resolved as two separate components.

To find the best selection of parameters, we first compare nested 4-connectivity with nested 8-connectivity in Section 3.1. Section 3.2 performs a similar experiment, comparing the two forms of connectivity using non-nested components. Nested and non-nested extraction are compared in Section 3.3. The last parameter setting is described in Section 3.4, where several component bounding options are investigated.

	4-connected			*8-connected*		
	Components	Classes	File size	Components	Classes	File size
A006	5942	231	69207	5799	232	69008
A034	3029	389	62968	2974	387	62881
A03J	4981	548	51000	3978	536	49501
A04D	5944	572	77865	5411	577	76720
A05E	3715	303	59832	3520	296	59299
C03I	5125	506	49755	4671	507	49317
C048	3345	235	38913	3253	221	38563
D03D	3314	338	44015	2868	351	43646
D03M	728	128	19895	702	123	19843
D048	2031	144	26761	2026	145	26743
D04G	4235	443	58927	3858	448	58567
D05N	1895	191	28689	1885	194	28664
D06C	2589	324	35279	2570	324	36657
D06N	2691	446	51632	2610	452	51785
E009	4098	344	28996	2358	346	26543
E00H	3567	364	38611	3354	362	37937
E00M	3344	256	41712	3223	256	41413
E01J	8501	478	75780	8007	463	74778
E037	8383	845	82371	7668	853	81392
E04I	4179	478	43779	3853	494	43482
H00C	1238	194	16230	1216	200	16260
H019	3894	295	38970	3342	305	37318
H041	2731	279	42972	2559	274	42696
H04D	2489	233	28769	2375	219	28530
H04F	5471	290	61301	5165	264	60095
I00J	3703	427	71614	3674	428	71602
I00L	4170	399	75933	4139	398	75840
I037	2615	218	43052	2577	222	43141
J03M	4817	401	51213	4631	417	51061
J04A	5649	558	48792	4774	567	46868
J04K	6977	451	72078	6080	443	70926
K009	2929	278	40672	2682	271	40093
N01A	1391	101	18697	1391	101	18697
N027	1384	183	17194	1377	180	17181
N02A	476	138	6501	472	131	6485
N02K	8446	301	56612	8130	259	56286
N03K	2098	164	25108	2072	161	24965
N048	2431	124	36987	2397	124	36963
N05I	2970	150	41121	2957	150	41111
S021	4470	346	52385	4415	334	52315
S03G	3858	511	58664	3423	481	57556
S03R	3566	321	38797	2812	269	36118
S044	3652	185	45956	3629	188	45974
S047	5239	324	70004	5112	337	69769
S048	5120	170	67911	5007	183	68167
S04H	5436	186	63831	5379	184	63670
V004	22478	254	62344	22463	258	62404
V00C	3361	201	39269	3343	206	39311
V00G	5128	269	56987	5110	266	56913
V00N	4301	363	57775	4258	362	57746
Mean	4282	318	47875	4031	315	47456

Table 3.2: Nested compression figures

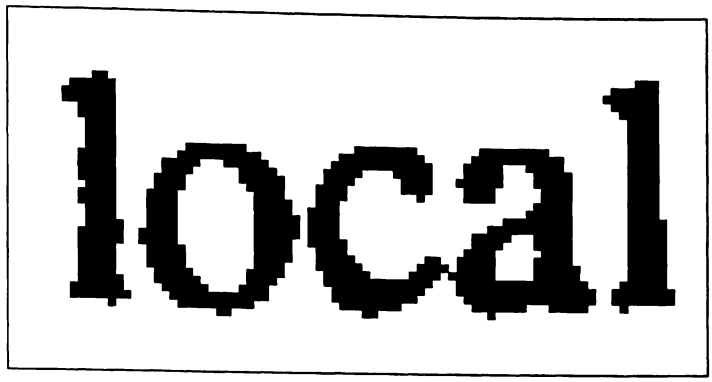


Figure 3.2: Two letters merged using 8-connectivity

	4-connected	*8-connected*	Significant
Components	4282	4031	$\surd\,(t = 5.21)$
Classes	318	315	$\times\,(t = 1.38)$
File size	47875	47456	$\surd\,(t = 4.17)$

Table 3.3: Compression figures using nested extraction

3.1 Nested components

Extracting components is often achieved as a two-stage process. The first stage consists of a boundary tracing procedure which determines the borders of the component. This allows a bitmap of the correct size to be allocated. The tracing process is initiated by scanning left to right, top to bottom, through the bitmap until a black pixel is encountered. This is used as the *seed* pixel, and the boundary tracing process begins from this point. The tracing process follows the border of the component until it arrives back at the seed pixel. The second stage extracts the component from the image by performing a flood-fill operation to isolate and remove the component from the image [FV82]. After the second stage is completed and the component is extracted from the image, the extraction process returns to the first stage and repeats until all components have been extracted [WMB94, pp. 263–268].

Use of nested component extraction began with Johnsen *et al.* [JSC83], and was continued by Witten *et al.* [WBE+94], Inglis and Witten [IW96] and most recently by Zhang [Zha97].

Nested component extraction is implemented, both in 4-connected and 8-connected forms. When the test images are compressed using the two forms of connectivity, the results shown in Table 3.2 are obtained. The columns are summarised and compared for statistical significance in Table 3.3. For 4-connected extraction, the number of components is significantly larger than the number of components for 8-connected extraction (average of 4282 vs. 4031), as is the file size (average of 47875 vs. 47456). The number of classes (average of 318 vs. 315) for each method is not significantly different.

3.2 Non-nested components

An alternative to nested extraction is to allow a component to contain multiple components nested within each other. The extraction process from the previous section can be modified slightly to allow this variation. Non-nested extraction begins with the boundary tracing procedure, but instead of following the tracing with a flood fill operation, all pixels that appear within the traced perimeter are treated as belonging to the component.

Non-nested extraction is faster than nested extraction because a flood-fill operation to extract the component is not required. The region inscribed by the perimeter is extracted as a single component.

Non-nested extraction for document image compression has been used by Pratt *et al.* with CSM (combined symbol matching) [PCC+80], followed by Mohiuddin *et al.* [MRA84], Holt and Xydeas [HX86], Holt with CSIS (combined size-independent strategy) [Hol88] and more recently by Howard with SPM (soft pattern matching) [How96].

In an attempt to overcome the nesting problems due to table borders and outlines, Howard [How96] limits the width and height of extracted components, but does not evaluate compression performance without these restrictions. We address this issue in Section 3.4.

	4-connected			*8-connected*		
	Components	Classes	File size	Components	Classes	File size
A006	5836	231	69061	5737	231	68922
A034	2694	369	62621	2656	370	62572
A03J	4500	525	50209	3644	519	48920
A04D	4796	496	76418	4596	499	76180
A05E	3535	294	59494	3401	295	59080
C03I	4755	481	49569	4395	481	49160
C048	3336	233	38882	3247	219	38542
D03D	1245	102	41708	1001	114	41567
D03M	55	37	21543	49	35	21531
D048	1979	125	26635	1974	126	26625
D04G	3812	421	58473	3563	422	58124
D05N	1895	191	28689	1885	194	28664
D06C	2588	324	35275	2570	324	36657
D06N	2420	407	51590	2344	417	51821
E009	4108	351	29049	2316	339	26477
E00H	3561	363	38467	3351	361	37925
E00M	3344	256	41712	3223	256	41413
E01J	5131	404	73859	4940	400	73253
E037	7196	756	81070	6676	752	80185
E04I	3803	440	44055	3498	465	43814
H00C	1235	193	16215	1214	199	16247
H019	3893	295	38965	3342	305	37318
H041	1736	229	46543	1573	226	46324
H04D	2462	229	28708	2348	215	28469
H04F	5454	288	61284	5158	264	60086
I00J	3572	418	71462	3551	419	71447
I00L	4101	397	75651	4082	396	75637
I037	495	92	50436	483	93	50423
J03M	4802	398	51201	4616	414	51046
J04A	4561	488	47511	3911	491	45879
J04K	5434	362	69663	4934	354	68391
K009	2641	260	40070	2549	249	39678
N01A	1391	101	18697	1391	101	18697
N027	1384	183	17194	1377	180	17181
N02A	476	138	6501	472	131	6485
N02K	1765	268	53496	1546	241	53403
N03K	2095	165	25093	2070	162	24946
N048	2426	124	36978	2395	124	36956
N05I	2970	150	41121	2957	150	41111
S021	4433	338	52355	4375	324	52230
S03G	3448	437	57591	3196	415	56878
S03R	3536	327	39050	2786	276	36403
S044	3652	185	45956	3629	188	45974
S047	5235	323	69995	5109	336	69762
S048	5118	170	67906	5006	183	68165
S04H	5435	186	63829	5379	184	63671
V004	22408	233	62152	22393	237	62203
V00C	3361	201	39269	3343	206	39311
V00G	5124	268	56976	5108	265	56907
V00N	4305	361	57831	4262	361	57787
Mean	3791	292	47762	3592	290	47409

Table 3.4: Non-nested compression figures

	4-connected	*8-connected*	Significant
Components	3791	3592	$\surd\ (t = 4.56)$
Classes	292	290	$\times\ (t = 1.25)$
File size	47762	47409	$\surd\ (t = 3.66)$

Table 3.5: Compression figures using non-nested extraction

Non-nested component extraction is implemented in both forms of connectivity. Table 3.4 shows the results of using non-nested extraction on the test images. The columns are summarised in Table 3.5. The results form a similar pattern to those in the Tables 3.2 and 3.3, where the figures for 4-connected extraction are often larger than for 8-connected extraction. In this experiment, the number of components for 4-connected extraction is significantly larger than 8-connected (average of 3791 vs. 3592), as is the file size (average of 47762 vs. 47409). There is no significant difference between the the number of classes in the library (average of 292 vs. 290).

3.3 Nested vs. non-nested

The preceding two sections have shown that 8-connectivity gives significantly better compression than 4-connectivity. This section compares nested with non-nested extraction. The compression figures for 8-connected extraction have been taken from the last two sections and are compared for significant differences and shown in Table 3.6. This table is summarised in Table 3.7 and shows that there is no significant difference in the file size between nested and non-nested extraction (average of 47456 vs. 47409). However, there is a significant difference between the number of components (average of 4031 vs. 3592) and the number of classes in the library (average of 315 vs. 290). We have shown there is no significant difference in file size between extraction using nesting or non-nesting. The decision to allow nested components cannot be decided by examining file size, although in practice, a small library may be favoured over a larger library when multiple pages are processed.

	8-nested			*8-non-nested*		
	Components	Classes	File size	Components	Classes	File size
A006	5799	232	69008	5737	231	68922
A034	2974	387	62881	2656	370	62572
A03J	3978	536	49501	3644	519	48920
A04D	5411	577	76720	4596	499	76180
A05E	3520	296	59299	3401	295	59080
C03I	4671	507	49317	4395	481	49160
C048	3253	221	38563	3247	219	38542
D03D	2868	351	43646	1001	114	41567
D03M	702	123	19843	49	35	21531
D048	2026	145	26743	1974	126	26625
D04G	3858	448	58567	3563	422	58124
D05N	1885	194	28664	1885	194	28664
D06C	2570	324	36657	2570	324	36657
D06N	2610	452	51785	2344	417	51821
E009	2358	346	26543	2316	339	26477
E00H	3354	362	37937	3351	361	37925
E00M	3223	256	41413	3223	256	41413
E01J	8007	463	74778	4940	400	73253
E037	7668	853	81392	6676	752	80185
E04I	3853	494	43482	3498	465	43814
H00C	1216	200	16260	1214	199	16247
H019	3342	305	37318	3342	305	37318
H041	2559	274	42696	1573	226	46324
H04D	2375	219	28530	2348	215	28469
H04F	5165	264	60095	5158	264	60086
I00J	3674	428	71602	3551	419	71447
I00L	4139	398	75840	4082	396	75637
I037	2577	222	43141	483	93	50423
J03M	4631	417	51061	4616	414	51046
J04A	4774	567	46868	3911	491	45879
J04K	6080	443	70926	4934	354	68391
K009	2682	271	40093	2549	249	39678
N01A	1391	101	18697	1391	101	18697
N027	1377	180	17181	1377	180	17181
N02A	472	131	6485	472	131	6485
N02K	8130	259	56286	1546	241	53403
N03K	2072	161	24965	2070	162	24946
N048	2397	124	36963	2395	124	36956
N05I	2957	150	41111	2957	150	41111
S021	4415	334	52315	4375	324	52230
S03G	3423	481	57556	3196	415	56878
S03R	2812	269	36118	2786	276	36403
S044	3629	188	45974	3629	188	45974
S047	5112	337	69769	5109	336	69762
S048	5007	183	68167	5006	183	68165
S04H	5379	184	63670	5379	184	63671
V004	22463	258	62404	22393	237	62203
V00C	3343	206	39311	3343	206	39311
V00G	5110	266	56913	5108	265	56907
V00N	4258	362	57746	4262	361	57787
Mean	4031	315	47456	3592	290	47409

Table 3.6: Figures for 8-connected extraction

	8-nested	*8-non-nested*	Significant
Components	4031	3592	$\surd (t = 2.88)$
Classes	315	290	$\surd (t = 3.96)$
File size	47456	47409	$\times (t = 0.24)$

Table 3.7: Comparing 8-connected extraction methods

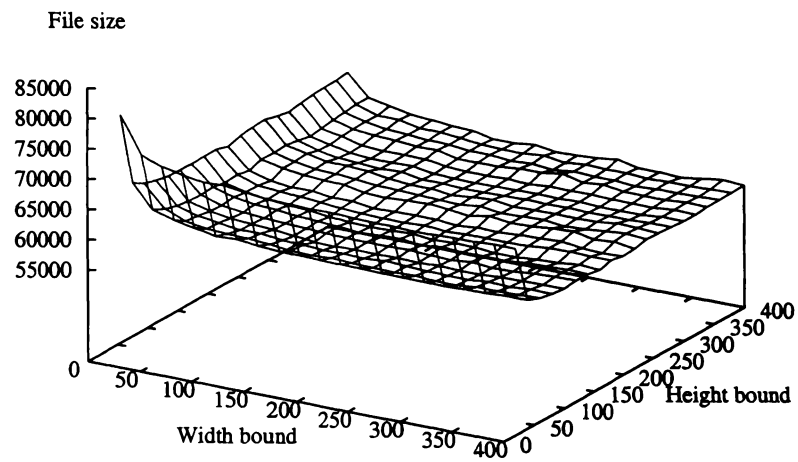


Figure 3.3: Bounding width and height on the image A034

3.4 Bounding component size

To minimise memory requirements in a document image compression system, or to facilitate matching between large components, it may be useful to decompose large components into a number of smaller sub-components. For example, if a component was to be truncated, or bounded, to a width of 400 and a height of 650, a 2000 by 1000 pixel component would be decomposed into ten (five horizontal by two vertical) sub-components. Sub-components which contain no black pixels can be discarded. In this section we use 8-connected nested extraction to investigate the effects of bounding component size.

This section treats the possible width and height restrictions as two parameters, and inves-

tigates the relationship between the parameter settings and compression performance. To examine the effect that width and height restrictions have on the compressed file size, both width and height parameters were enumerated over the range 20 to 400 pixels in increments of 20 pixels. Because of the large execution time for this experiment, a single test image A034 was selected, and all 400 parameter settings were used to compress it. The surface described by the width and height bounds and the compressed file size is shown in Figure 3.3. Although the global minimum point on this graph is reached when the width is bounded to 300 pixels and the height is bounded to 160 pixels, there is clearly a large plateau. This plateau occurs when the width bound is greater than 160 and the height bound is greater than 100. When the thresholds that achieved the global minimum are applied to the test image, the compressed file size reduces from 62968 (without bounding) to 58472 (with bounding). This is a reduction of 7%.

To investigate how well the global maximum will generalise, the same threshold was used to compress each of the test images. The compression results are shown in Table 3.8, and are summarised in Table 3.9. When the non-bounded images are compared with the bounded images, there is a significant difference between the number of components (average of 4031 vs. 4078) and the number of classes (average of 315 vs. 339). There is no significant difference between the mean file size using bounded components and non-bounded components (average of 47456 vs. 47159), although the mean size for the bounded components is slightly smaller. This shows that bounding does not provide a uniform improvement. Figure 3.4 shows the compression ratios for the 50 test images. The compression ratios range from 0.82 (compression) to 1.08 (expansion). These combine to give no significant overall improvement.

	Non-bounded			*Bounded to w=300, h=160*		
	Components	Classes	File size	Components	Classes	File size
A006	5799	232	69008	5845	257	70831
A034	2974	387	62881	3071	447	58472
A03J	3978	536	49501	4044	568	51081
A04D	5411	577	76720	5476	614	73045
A05E	3520	296	59299	3596	327	56294
C03I	4671	507	49317	4731	537	51105
C048	3253	221	38563	3306	239	39953
D03D	2868	351	43646	2999	414	41810
D03M	702	123	19843	798	165	16296
D048	2026	145	26743	2052	159	27129
D04G	3858	448	58567	3964	499	55110
D05N	1885	194	28664	1929	218	24276
D06C	2570	324	36657	2584	333	35763
D06N	2610	452	51785	2678	490	52910
E009	2358	346	26543	2381	363	27334
E00H	3354	362	37937	3375	374	38452
E00M	3223	256	41413	3241	265	41808
E01J	8007	463	74778	8053	488	76783
E037	7668	853	81392	7732	889	82478
E04I	3853	494	43482	3934	531	38465
H00C	1216	200	16260	1236	210	17052
H019	3342	305	37318	3368	321	35851
H041	2559	274	42696	2727	360	46174
H04D	2375	219	28530	2400	235	27980
H04F	5165	264	60095	5195	279	60804
I00J	3674	428	71602	3753	471	68541
I00L	4139	398	75840	4234	446	72133
I037	2577	222	43141	2659	261	40135
J03M	4631	417	51061	4660	434	51820
J04A	4774	567	46868	4833	599	49603
J04K	6080	443	70926	6180	492	66920
K009	2682	271	40093	2771	307	42079
N01A	1391	101	18697	1391	101	18697
N027	1377	180	17181	1377	180	17181
N02A	472	131	6485	476	139	6554
N02K	8130	259	56286	8161	282	59904
N03K	2072	161	24965	2072	161	24965
N048	2397	124	36963	2397	124	36963
N05I	2957	150	41111	2957	150	41111
S021	4415	334	52315	4436	346	53065
S03G	3423	481	57556	3462	507	58529
S03R	2812	269	36118	2841	284	37075
S044	3629	188	45974	3630	189	45985
S047	5112	337	69769	5149	355	69970
S048	5007	183	68167	5030	195	66742
S04H	5379	184	63670	5425	208	64418
V004	22463	258	62404	22478	265	63205
V00C	3343	206	39311	3404	232	39911
V00G	5110	266	56913	5113	269	56930
V00N	4258	362	57746	4273	374	58232
Mean	4031	315	47456	4078	339	47158

Table 3.8: Bounding the test images to $w = 300$ and $h = 160$

	Non-bounded	*Bounded*	Significant
Components	4031	4078	$\surd\,(t = 8.71)$
Classes	315	339	$\surd\,(t = 9.22)$
File size	47456	47158	$\times\,(t = 0.98)$

Table 3.9: Bounding the test images to $w = 300$ and $h = 160$

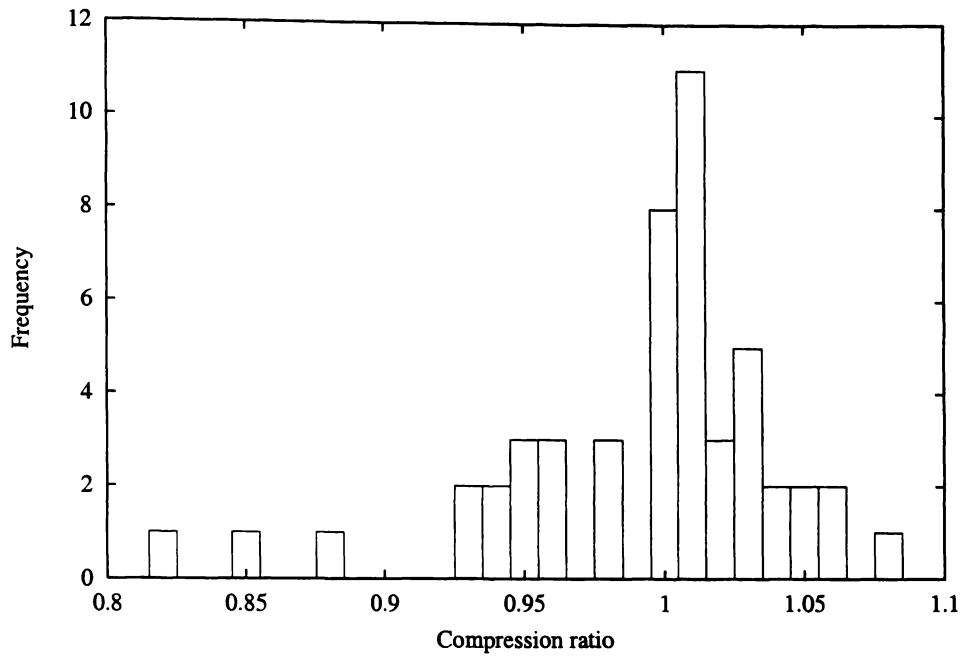


Figure 3.4: Histogram of the bounded compression ratios

3.4.1 Bounding component height

Howard [How96] limits the height of a component to 5% of the page height for efficiency reasons. This section considers a range of height bounds and evaluates them on the test images. The height bound parameter was varied between 5 and 100% in 5% increments. For the test images, this increment step corresponds to 165 pixels. Each of the test images was compressed using each of the twenty thresholds (5 to 100%). The average expansion ratio for each threshold is shown in Figure 3.5. The figure shows that when the height bound percentage is set to Howard's 5%, this corresponds to an average increase in the compressed file size of almost 2%.

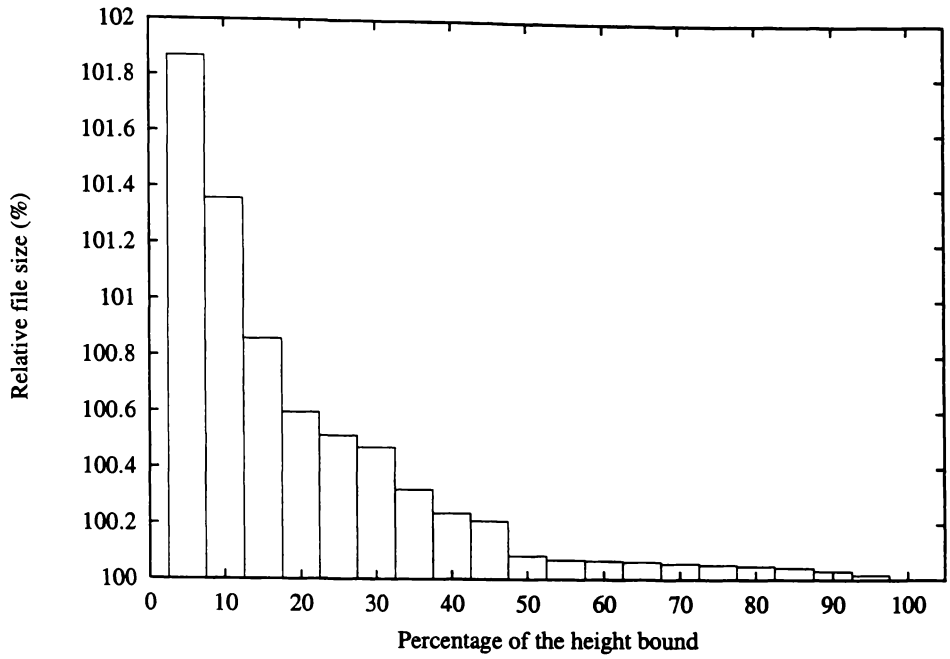


Figure 3.5: Varying the bounding height percentage

Each pair of height thresholds is examined for statistical significance with the results shown in Table 3.10. The significant changes in the threshold parameter occur at the 5→10%, 10→15%, 15→20%, 20→25%, 30→35%, 45→50%, 75→80% and 95→100% boundaries. Setting the maximum height of a component to be 50% of the height of the document would incur less than a 0.1% compression penalty.

3.5 Summary

This chapter compares the standard methods for extracting components from an image and investigates the parameter settings that affect their performance. The use of 8-connected components provides significantly better compression than 4-connected components. For both nested and non-nested extraction, the compressed file size improvement over using 4-connected extraction is 1%.

Threshold	Significant
5% → 10%	√ ($t = 5.50$)
10% → 15%	√ ($t = 3.85$)
15% → 20%	√ ($t = 5.13$)
20% → 25%	√ ($t = 3.38$)
25% → 30%	× ($t = 2.58$)
30% → 35%	√ ($t = 3.43$)
35% → 40%	× ($t = 1.27$)
40% → 45%	× ($t = 1.89$)
45% → 50%	√ ($t = 3.12$)
50% → 55%	× ($t = 1.62$)
55% → 60%	× ($t = 1.29$)
60% → 65%	× ($t = 1.24$)
65% → 70%	× ($t = 0.99$)
70% → 75%	× ($t = 1.16$)
75% → 80%	√ ($t = 2.81$)
80% → 85%	× ($t = 1.39$)
85% → 90%	× ($t = 2.66$)
90% → 95%	× ($t = 2.65$)
95% → 100%	√ ($t = 3.89$)

Table 3.10: Changes in height thresholds and their significance

When nested extraction was compared with non-nested extraction, no significant compression difference was observed between the two methods.

Section 3.4 investigated the relationship between width and height bounds and the compressed file size. A finely grained experiment was performed on a test image to find the best choice for the width and height bounds. On the test image the minimum file size occurs when the width is bounded greater than 160 while the height can be bounded above 100 pixels. When the height and width bounds that produce the local minimum for the test image (width bound of 300, height bound of 160) were applied to the test images we found no significant difference from the images without bounding. However, the single training example did not turn out be a significant predictor of the performance for the test corpus.

The effects of limiting component height on the compressed file size were examined. In the test images, a height limitation of 5% of the page height increases the average file size by 2%. The height bound that maximised compression was found to be a bound that corre-

sponded to the height of the page, which is effectively using no bounding at all. Bounding the height to more than 50% of the page height had little influence. The difference between 50% and no bounding was less than 0.1%.

Chapter 4

Determining reading order

When we read a document, the order in which each character is examined depends on the language in which the document is written. In English, we read paragraphs with our eyes moving left to right, top to bottom. This fact alone does not give enough information to replicate the reading process. We also need to know in which order columns, headers, footers and footnotes are read. Still more information is required when the text is not horizontally aligned. If a document is skewed by a few degrees, the left-to-right reading process is adjusted to match the overall skew of the document. In this case our eyes track deviations from normal horizontal writing without conscious thought.

In document image compression, it is advantageous to sort the components into an approximate reading order after they have been extracted from an image. This facilitates further processing that takes advantage of the sequential structure of the text. If this step is not performed, components will be processed in the order that they have been extracted from the image. In our implementation, this would process the tallest characters on each line first, and the periods and commas last.

In this chapter, Section 4.1 describes methods that have previously been used for determining the reading order of documents. Section 4.2 introduces the use of a page layout technique that determines page structure and column position, as well as the position of lines in a paragraph. Section 4.3 compares the reading order methods on a variety of test

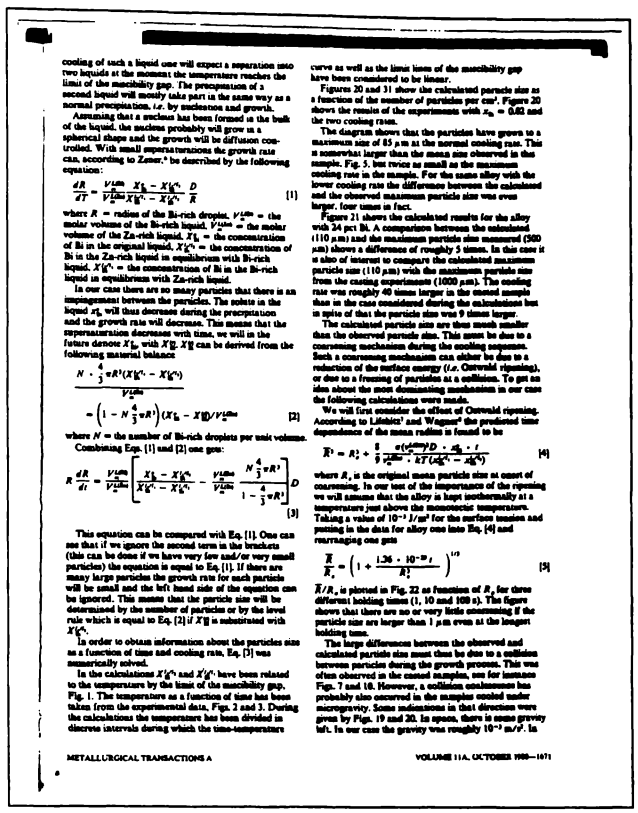


Figure 4.1: Image H04F

images under three different page transformations.

Section 4.4 discusses methods of compressing component indices (which are identifiers that correspond to a character class in the codebook), and compares the gamma encoding method with PPMD [How93]. The compression ratios for each method are compared with the best that can be achieved on the ground truth text data for the images. This section also discusses the issue of whether the automatic determination of spaces aids compression. The results for this chapter are summarised in Section 4.5.

4.1 Reading order methods

Previous document image compression systems have assumed a single-column document structure, and have generally been based either on a profile of pixel positions or a localised

tracking method that follows from one component to the next.

Witten *et al.* project the centroid of each component into a vertical profile [WBE+94]. The gaps between the peaks in the profile correspond to rows in the image where no characters are present, and the actual peaks correspond to the position of the line itself. The profile is smoothed using a 1 mm rectangular filter. The regions in the profile that are non-zero correspond to the position of a line. Witten *et al.* assume the skew of the document has been corrected, and give no experimental results justifying this method.

Howard [How96] compresses the images in horizontal stripes, where only 5 percent of the image height is processed at any one time. The components are first extracted and stored in the order they were processed. An attempt is made to approximate the location of the baseline for each character by averaging the baselines of the next few components to be processed. The next stage selects all components whose top scan line is at least as high as the approximate baseline position, and sorts them according to their leftmost pixel. After the components in the current line are extracted, the process iterates until all components have been processed. Howard reports that this method for determining reading order incurs a loss of compression efficiency for vertical text[1] of less than 10 percent. We will examine this claim in Section 4.3.

Zhang [Zha97] calculates reading order in a similar way to Howard. The components are initially sorted by the bottom of the component. The initial baseline is set to be the bottom of the first component. The rest of the components are examined and if a component's top vertical position is higher than the baseline, it is assumed to be on the current line. If not, this component starts a new line. After all lines are identified, components are sorted by their horizontal position.

[1] We have defined vertical text to be lines of horizontal text rotated 90 degrees.

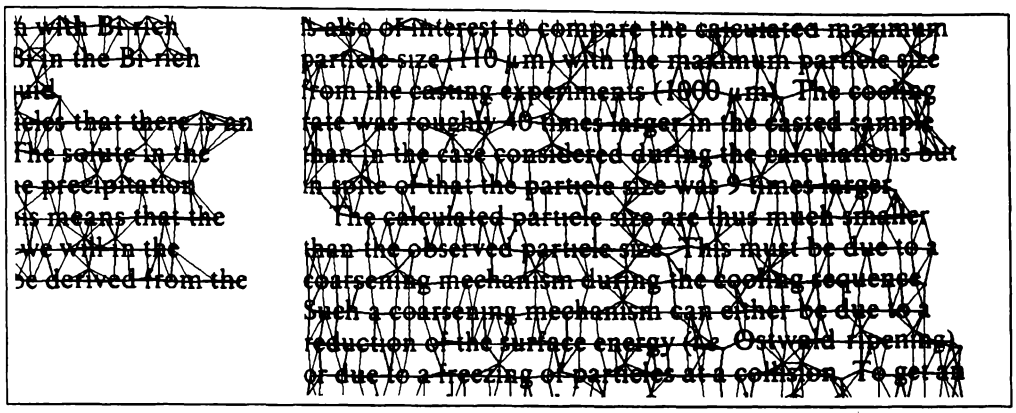


Figure 4.2: Docstrum showing the 5 nearest neighbours

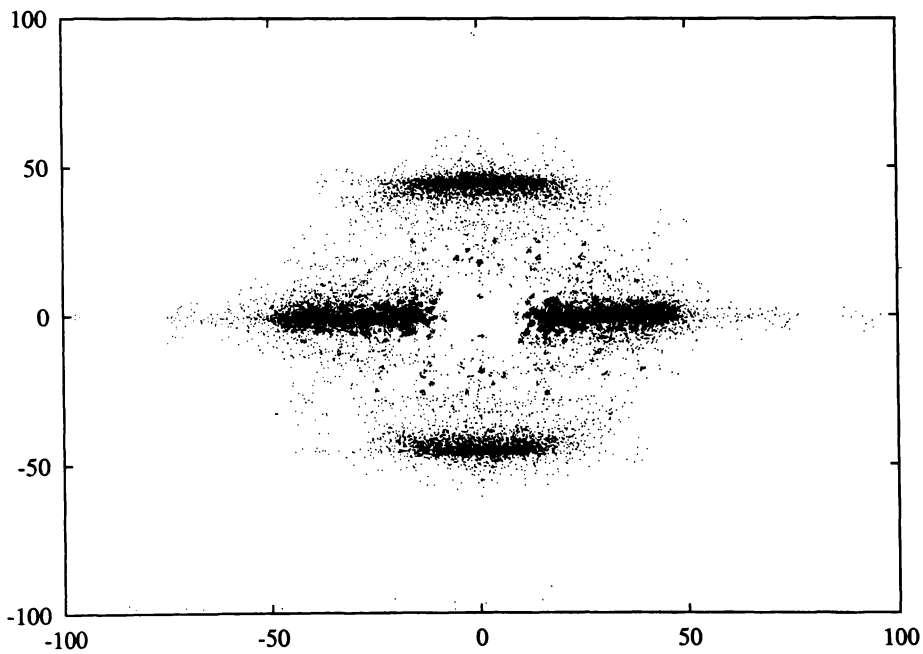


Figure 4.3: Docstrum plot

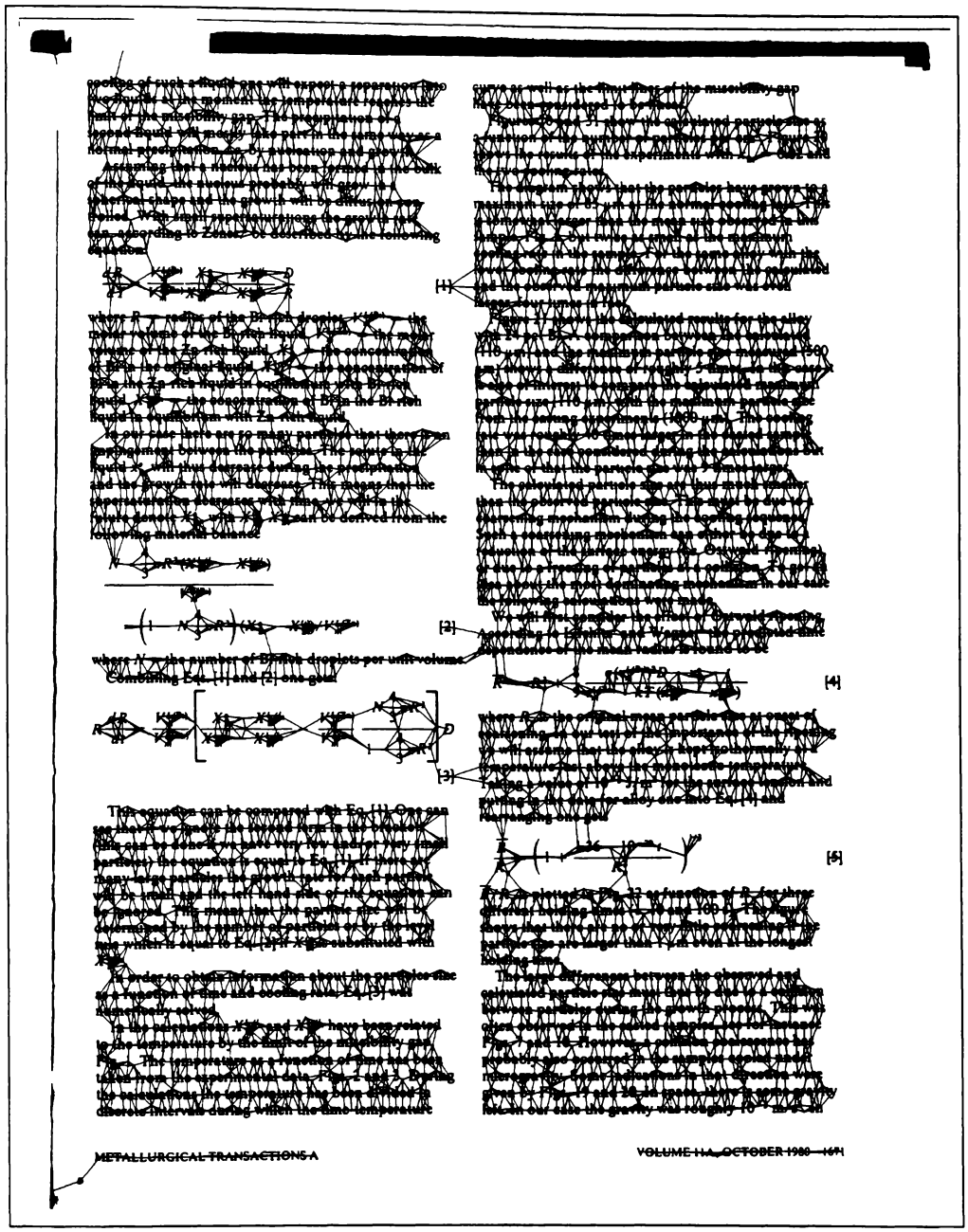


Figure 4.4: Docstrum connectivity

4.2 Determining reading order using the docstrum

O'Gorman introduced the *docstrum* (from document spectrum) as a bottom-up page layout technique for finding groups in text [O'G93]. The docstrum is a visualisation of the distance from each component to each of its k nearest neighbours. The distance is measured between the centroids of each component. Figure 4.2 shows the connections between the $k = 5$ nearest neighbours of a zoomed region of Figure 4.1. For $k = 5$, each component is most often connected to letters in the same word and occasionally connected to a word above or below in another line. The figure also shows that the nearest neighbours have not spanned the column boundaries. In this specific example this is because the column separation is several times larger than the inter-character space (distance between adjacent characters). If the column separation were reduced, the docstrum would not cleanly separate the columns. Human readers would also find such text difficult to read.

The relative distances to each of the nearest neighbours in Figure 4.1 is shown in Figure 4.3. This figure is a visualisation of each of the nearest neighbours, assuming that the centroid of the current component is at the origin $(0, 0)$. The figure is not necessarily symmetric, although most docstrums appear to produce an approximately symmetric plot. The white space in the centre of the figure corresponds to the minimum distance between the centroids of two components. The vertical distance to the clusters above and below the origin correspond to the inter-line spacing (distance between adjacent lines), while the horizontal distance to clusters along the x-axis correspond to inter-character spacing. These figures are roughly 50 pixels and 25 pixels respectively.

To determine reading order using the docstrum, the k nearest neighbour connections for the entire image are calculated. This is shown, with $k = 5$, in Figure 4.4 for the test image of Figure 4.1. The docstrum plot is then used to calculate the inter-character and inter-line spacing, averaged over the whole page. The inter-character distance is calculated by

integrating the distances between ± 30 degrees of horizontal. The inter-line distance is calculated by integrating the distances between ± 30 degrees of vertical.

Connections between nearest neighbours that exceed three times the average inter-line distance are broken. This helps to isolate clusters that are joined by a tenuous connection, and removes noise from the borders of text zones. A transitive closure is then applied to group the connected regions. Each of the regions is compressed in turn. The order in which the regions are compressed is determined by vertically sorting them on the centre of the region. The reading order for each of the regions is determined by Howard's method, and each line compressed in turn. When a new region is to be compressed, the context for PPM is not reinitialised but continues from the previous region.

The benefit of the docstrum method is shown in Figures 4.5 and 4.6. Figure 4.5 highlights how lines are determined by Howard's method. The right-angle symbol denotes the predicted position of the end of the line. If the beginning of the column was correctly determined, a left-angle symbol would prefix each line in the right column. When the same image is processed with the docstrum method described above, the lines are determined much more accurately. Figure 4.6 highlights the position of the lines as determined by the docstrum method. In this case the column is handled correctly.

4.3 Comparing reading order methods

In this section, we compare the three reading order methods discussed in Section 4.1 as well as the docstrum method introduced in Section 4.2. Two extra methods are also included in the comparison: an unsorted method, where the components are kept in the order in which they were extracted; and a random ordering method, where the component ordering has been randomly shuffled.

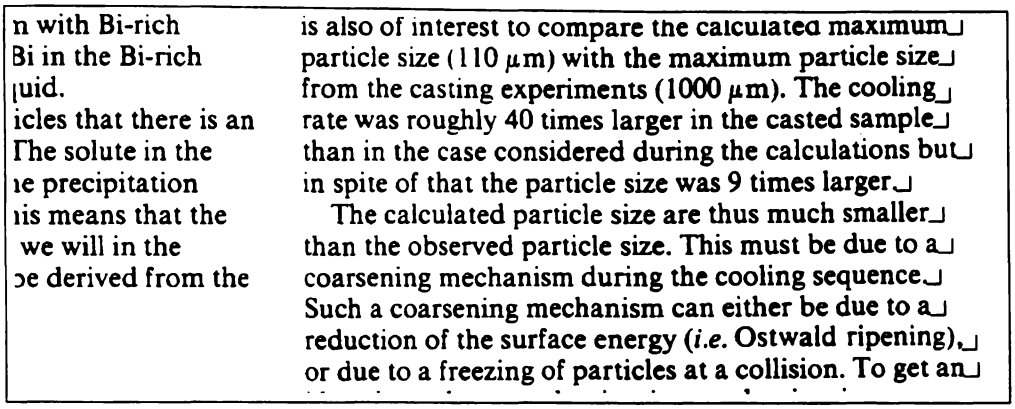


Figure 4.5: Line position as determined by Howard's method

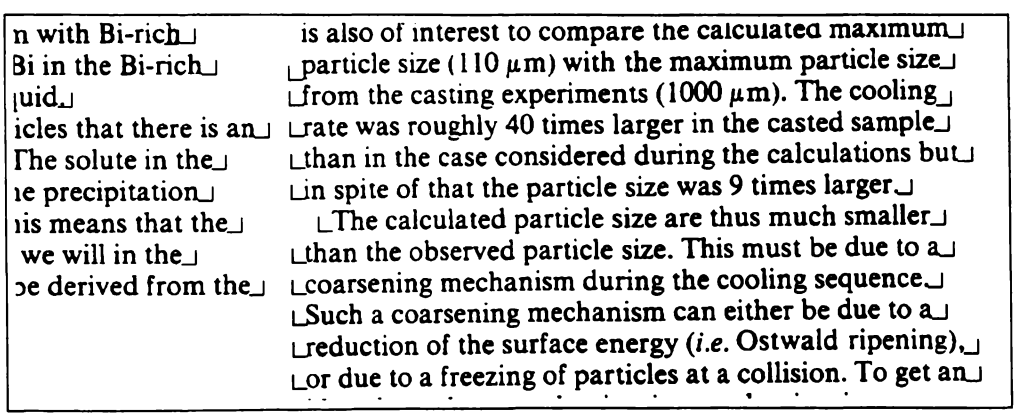


Figure 4.6: Line position as determined by the docstrum method

Each of these six methods for determining reading order is applied to the 50 test images. To assess the robustness of the schemes, the original images were distorted in two different ways. The first distortion skews images by a random amount uniformly distributed between -10 and 10 degrees. The second distortion rotates images 90 degrees anti-clockwise.

The effects of different reading order determination methods on the compressed file size are shown in Tables 4.1, 4.2 and 4.3. Table 4.1 shows the results for the original (undistorted) test images, Table 4.2 shows the results for the images after they have been skewed, and Table 4.3 shows the results for the images after they have been rotated by 90 degrees.

The results for the original images shown in Table 4.1 are summarised and tested for significance in Table 4.4. Each of the methods is compared against the docstrum method (average of 46469 bytes), and it is shown to be significantly better than all other methods. The compressed file size for the docstrum method is significantly smaller than Howard's method, Zhang's method, the Profile method, the Extraction method and the Random method.

To examine each method's performance under less ideal conditions, each of the images is skewed by a random amount. Table 4.2 shows the compressed file size for each method on the skewed images, along with the angle (in degrees) by which each image is skewed. This information is summarised in Table 4.5. For skewed images, the mean file size for the docstrum method (average of 56149 bytes) is significantly smaller than all others.

Table 4.3 shows compression results for each of the methods when each page has been rotated anti-clockwise by 90 degrees. This information is summarised in Table 4.6, and shows a slightly different result from the two previous experiments. Here, the docstrum method has no significant difference when compared with Howard's method (average of 48273 bytes) or Zhang's method (average of 48277 bytes). However, the differences between the docstrum method and the other three methods are significant. The docstrum, Howard's and Zhang's methods incur a 3% expansion when compressing the rotated images compared

	Random	Extraction	Profile	Zhang	Howard	Docstrum
A006	78640	70042	66708	66751	66506	66134
A034	68709	64268	62743	62867	63084	62769
A03J	55774	50422	48468	48563	48596	48488
A04D	87940	79232	76636	77023	76971	76730
A05E	65853	60784	59157	59318	59257	59054
C03I	57698	51434	49070	49038	49258	48974
C048	45591	40178	39234	38589	38427	38466
D03D	48600	44683	43761	43952	43764	43554
D03M	21059	20111	19886	19732	19799	19793
D048	30203	27436	26680	26539	26523	26552
D04G	65073	59700	58124	58086	58100	58103
D05N	32300	29265	29380	28375	28316	28312
D06C	41868	38050	36572	36535	36446	36467
D06N	56318	52097	50643	50764	50801	50714
E009	31389	27914	26808	26899	26638	26507
E00H	45013	39363	37908	37846	38134	37968
E00M	47728	42616	41116	41187	41160	40835
E01J	91302	76639	74051	74170	74798	73587
E037	93582	81823	79343	79047	79446	78657
E04I	50541	44511	43460	43366	43513	43263
H00C	18446	16584	16256	16087	16074	16066
H019	43679	38775	37998	37316	37429	37150
H041	47396	43553	42494	42594	42683	42392
H04D	32489	28899	28469	27873	27918	27805
H04F	70213	62036	58885	58785	58744	58755
I00J	78687	73175	71187	70844	70903	70870
I00L	81353	76311	73051	73244	73377	73308
I037	48291	44084	43203	42888	43156	42942
J03M	59456	53396	53252	50707	50760	50786
J04A	54560	46768	45608	45440	45648	45595
J04K	81375	71292	69195	68748	68683	68702
K009	45009	41319	40239	39877	40091	39745
N01A	21731	19580	18747	18635	18655	18645
N027	19710	18020	17266	17291	17294	17231
N02A	7304	6722	6491	6513	6536	6462
N02K	71085	55199	55561	54227	54736	54468
N03K	28517	25748	24976	24964	25080	24971
N048	40281	36483	36945	35286	35226	35147
N05I	46867	42595	42559	41164	41005	40998
S021	60154	53776	51917	50931	51136	50533
S03G	63893	58481	56727	56641	57111	56569
S03R	41337	36524	35863	35513	35471	35563
S044	53181	47381	45695	45837	45702	45424
S047	78161	69317	66521	66511	66664	66502
S048	76730	69454	66163	66272	66326	65635
S04H	72472	63703	59830	59587	60056	59778
V004	116039	58756	61150	56949	56588	55640
V00C	46880	40940	39339	39332	39223	39262
V00G	67980	60119	56394	56493	56447	56176
V00N	64897	57610	55556	55423	55613	55381
Mean	55067	48343	46946	46612	46677	46469

Table 4.1: File sizes for the original test images

	Skew	Random	Extraction	Profile	Zhang	Howard	Docstrum
A006	−0.96	84326	76581	76630	72671	72400	72038
A034	−9.79	87182	84031	83392	82588	82104	81850
A03J	1.20	57831	52888	52733	50971	50892	50836
A04D	7.85	103969	97287	96624	94738	93897	93338
A05E	4.71	76101	72053	71691	70021	69791	69625
C03I	−1.02	60781	54355	53964	51974	52123	51812
C048	−3.14	52404	47916	48197	45563	45586	45320
D03D	2.60	52366	48820	47951	47909	47502	47299
D03M	4.36	25341	24604	24650	24255	24222	24237
D048	−9.70	40186	37711	37460	36725	36535	36440
D04G	−7.85	79866	75949	75489	74259	73614	73271
D05N	1.90	35672	33519	33161	32295	32132	32125
D06C	−3.07	48055	44648	45340	42892	42900	42911
D06N	−8.49	71284	68573	68624	67296	66855	66660
E009	0.33	32088	28964	28957	27797	27736	27603
E00H	8.58	57725	54121	53196	52350	51469	51351
E00M	3.43	57147	53649	52893	51197	51001	50844
E01J	7.73	110368	100093	98225	96306	95834	94599
E037	−3.48	101637	92657	91982	88980	88631	87895
E04I	−0.75	51874	46650	46240	45180	45057	44815
H00C	−7.91	22766	21490	21428	20829	20725	20737
H019	2.94	49952	46314	45718	44068	43962	44032
H041	9.50	78151	75501	74439	74326	73798	73530
H04D	−5.11	38504	35988	35846	34549	34345	34226
H04F	−4.19	83286	77662	77546	73895	73510	73508
I00J	−8.70	93324	90026	89252	87804	87334	87156
I00L	0.39	83633	78609	78664	75435	75249	75160
I037	1.03	51992	48165	48269	46460	46706	46713
J03M	−8.39	73330	68950	67805	66559	65828	65819
J04A	5.98	61736	54847	54969	53189	53052	53172
J04K	1.01	84481	75400	75255	72744	72529	72509
K009	0.65	47344	43910	43750	42146	42341	42113
N01A	6.19	26124	24370	24257	23523	23454	23486
N027	−7.79	24098	22457	22342	21927	21788	21596
N02A	−1.50	7798	7218	7229	7028	7035	6988
N02K	0.89	72296	57568	58298	56468	56609	56416
N03K	1.19	30575	27949	27373	26771	26760	26760
N048	5.36	45975	43025	42807	41968	41741	41545
N05I	−6.51	55664	52184	51621	50630	50324	50334
S021	−4.44	70226	64990	64688	61872	61616	61068
S03G	5.66	76140	72330	71637	70203	69700	69358
S03R	−4.36	49214	45941	45657	44049	43780	43797
S044	7.46	66382	62521	62059	60275	59875	59952
S047	−7.41	96826	91146	90490	87798	86945	86976
S048	−2.85	88047	81793	82088	78225	77963	78035
S04H	−2.22	80983	73014	73472	69368	69421	69197
V004	−8.83	133869	86882	85443	83765	84506	82366
V00C	−9.42	67721	63962	63519	62146	61536	61503
V00G	−4.49	80666	74605	74680	70819	70452	70420
V00N	−2.31	72825	67500	67876	64148	64166	64113
Mean		64003	58628	58318	56579	56347	56149

Table 4.2: File sizes for the skewed test images

	Random	Extraction	Profile	Zhang	Howard	Docstrum
A006	78680	71849	69560	68546	68497	68501
A034	70341	66363	65508	64987	65005	64871
A03J	55919	50810	50232	49515	49373	49398
A04D	88569	82515	79993	80111	79460	79010
A05E	66282	62053	61047	60302	60211	60195
C03I	57929	51748	50723	50030	49950	49834
C048	45663	41677	40009	39906	40110	39621
D03D	48262	44076	44130	43568	43476	43437
D03M	21352	20603	20353	20353	20357	20400
D048	30348	27855	27485	27039	27092	27524
D04G	65376	60457	59958	58984	58729	58986
D05N	32234	30291	29590	29464	29412	29130
D06C	42001	40366	38390	38895	38952	37980
D06N	56156	53981	52598	52429	52575	52292
E009	31463	29228	28172	28224	28000	27605
E00H	45246	41477	39709	39846	39838	39567
E00M	47969	44121	42890	42146	42149	42430
E01J	91377	79114	76252	75936	76692	76471
E037	93742	84571	81885	81329	81725	80975
E04I	50604	45164	44620	44097	44025	43853
H00C	18628	17410	17025	16949	16954	16858
H019	43778	40201	39203	38488	38423	38372
H041	47270	43383	42846	42727	42580	42756
H04D	32447	30119	29294	29130	28951	29002
H04F	70876	65195	62070	61840	61757	61929
I00J	78701	75151	73771	72984	73019	72957
I00L	81644	77592	75655	75039	75087	75063
I037	47937	45780	43876	44364	44298	44094
J03M	59319	54425	52882	52220	52403	52186
J04A	54828	49367	47544	47594	47431	47235
J04K	81431	74006	71609	71532	71159	71051
K009	45026	42233	41167	40874	40850	40590
N01A	21502	19972	19692	19455	19407	19456
N027	19814	18110	17943	17751	17738	17932
N02A	7267	6678	6687	6650	6649	6580
N02K	70977	58044	55423	57007	57427	56517
N03K	28612	26259	25721	25193	25133	25125
N048	40380	37220	36735	36310	36388	36430
N05I	47142	42897	42878	41610	41495	42033
S021	60552	54921	52687	52816	52669	52444
S03G	63654	59636	58045	58061	57988	57799
S03R	41337	38035	36131	36657	36611	36640
S044	52939	49483	47958	47276	47198	47210
S047	78179	73014	69695	69871	69937	69699
S048	76125	70669	67080	67313	67521	67676
S04H	72026	66327	62961	62970	62728	62912
V004	117382	69788	61517	65799	66631	60991
V00C	46991	43855	41955	41918	41968	41660
V00G	67690	62686	60359	59615	59536	60174
V00N	65043	60749	58561	58122	58086	57975
Mean	55180	50030	48441	48277	48273	48069

Table 4.3: File sizes for the test images rotated 90 degrees anti-clockwise

Docstrum	Howard	Zhang	Profile	Extraction	Random
46468.60	46677.40	46612.40	46945.70	48343.40	55067.10
	$\surd$ ($t = 5.60$)	$\surd$ ($t = 3.89$)	$\surd$ ($t = 3.76$)	$\surd$ ($t = 13.40$)	$\surd$ ($t = 7.29$)

Table 4.4: Comparing reading order methods using the original images

Docstrum	Howard	Zhang	Profile	Extraction	Random
56149.10	56346.60	56579.10	58317.50	58627.70	64002.60
	$\surd$ ($t = 3.83$)	$\surd$ ($t = 7.34$)	$\surd$ ($t = 13.25$)	$\surd$ ($t = 14.27$)	$\surd$ ($t = 7.75$)

Table 4.5: Comparing reading order methods using the skewed images

Docstrum	Howard	Zhang	Profile	Extraction	Random
48068.50	48273.00	48276.80	48441.50	50030.50	55180.20
	$\times$ ($t = 1.72$)	$\times$ ($t = 2.01$)	$\surd$ ($t = 6.02$)	$\surd$ ($t = 10.20$)	$\surd$ ($t = 6.49$)

Table 4.6: Comparing reading order methods using the rotated images

with the original undistorted images.

4.4 Symbol encoding

Whenever a component is matched against a previously processed component, the symbol that corresponds with that character's class must be encoded. The symbols are encoded using the tree-based gamma encoder as described in Howard [How96]. This is a normal gamma encoder, except that each binary digit is encoded as a branch of a binary tree. We treat 0's as a left branch and 1's as a right branch. At the internal nodes of the tree, probability information is updated about the left and right choices at that position. This has the added benefit over a normal gamma encoder that the probabilities adapt over time to capture first-order information.

This section compares gamma encoding with the best text compression system, prediction by partial match (PPM) [BCW90]. As explained in Section 1.7.2, PPM models the text using a context which corresponds with a sequence of recently encoded symbols. The compression gains that can be achieved on the symbol sequence component of the output

PPMD2	Gamma	PPMD0	PPMD1	PPMD3	PPMD4
4.91	5.14	5.67	4.93	4.91	4.92
	$\surd\,(t = 4.67)$	$\surd\,(t = 17.93)$	$\times\,(t = 1.67)$	$\times\,(t = 0.11)$	$\times\,(t = 1.00)$

Table 4.7: Bits per symbol for PPMD and gamma encoding (zero)

stream are shown in Sections 4.4.1 and 4.4.2. The results that are achieved when comparing the total compressed file sizes are shown in Section 4.4.3.

The index that corresponds to each component class can be coded in one of two ways. A previously unseen character can be assigned a fixed symbol number, or it can be assigned the next symbol index in the sequence. In practice we use the symbol index α as the fixed symbol. This can be illustrated by the encoding of the string "Hello". In the first case, each previously unseen symbol is encoded with the symbol α. The output stream that corresponds to this string is $\alpha, \alpha, \alpha, 3, \alpha$. This shows there are four new symbols *H*, *e*, *l*, *o* and one previously seen symbol. If the symbol indices were encoded using the next available symbol the output stream would be $1, 2, 3, 3, 4$.

Table 4.7 shows the results for gamma encoding and PPMD with orders from zero to four. In this table, the unknown symbol is encoded with the symbol index 0. The best result in this table is achieved by PPMD2. PPMD2 achieves significantly better compression than the gamma encoder (average of 5.14 bits per symbol, $t = 4.67$), as well as PPMD0 (average of 5.67 bits per symbol, $t = 17.93$). There were no significant differences between PPMD2 and the first, third or fourth order PPMD models.

Table 4.8 shows the results for gamma encoding and PPMD with orders from zero to four, but this time the unknown symbol is encoded as the next unused symbol in the sequence. In this table, gamma encoding has the best performance of 5.70 bits per symbol, although this is 16% larger than Table 4.7.

Gamma	PPMD0	PPMD1	PPMD2	PPMD3	PPMD4
5.70	6.58	5.85	5.82	5.81	5.82
	$\surd$ ($t = 16.45$)	$\times$ ($t = 2.23$)	$\times$ ($t = 1.53$)	$\times$ ($t = 1.32$)	$\times$ ($t = 1.38$)

Table 4.8: Bits per symbol for PPMD and gamma encoding (next)

PPMD2	Gamma	PPMD0	PPMD1	PPMD3	PPMD4
4.12	4.52	5.25	4.18	4.10	4.10
	$\surd$ ($t = 9.68$)	$\surd$ ($t = 26.84$)	$\surd$ ($t = 5.49$)	$\surd$ ($t = 6.39$)	$\surd$ ($t = 5.87$)

Table 4.9: Bits per symbol for PPMD and gamma encoding (noise)

4.4.1 Moving noisy components

When constructing the docstrum, noise on the page (or fragments of dithered images) can alter the reading order or join two separate paragraphs. If we move components that are smaller than a certain threshold to the end of the compressed sequence, PPMD will be able to predict the text more accurately. When the noise components are compressed, their symbol numbers should be similar (or indeed the same), so PPMD should be able to compress them well.

We have measured the area of small components and defined *noise* to be components that have an area less than 0.1 mm^2. For 300 dpi images, these noise components contain less than 20 black pixels. Table 4.9 shows that by processing noise components in this manner, the size of the symbol sequence is decreased by around 16%. This time PPMD3 and PPMD4 had the best compression, but PPMD2 was very close—within 0.5%. PPMD2 (average of 4.12 bits per symbol) is significantly better than gamma encoding (average of 4.52 bits per symbol, $t = 9.68$), PPMD0 (average of 5.25 bits per symbol, $t = 26.84$) and PPMD1 (average of 4.18 bits per symbol, $t = 5.49$). Both PPMD3 (average of 4.10 bits per symbol, $t = 6.39$) and PPMD4 (average of 4.10 bits per symbol, $t = 5.87$) were significantly better than PPMD2, although the relative difference is around 0.5%.

4.4.2 Encoding spaces

To investigate whether automatically inserting space symbols into the symbol stream can improve compression, the ground truth ASCII text that corresponds to each page is compressed with and without the spaces present. Table 4.10 shows the results for compressing each of the 50 text files with PPMD2. While the original text files compressed to a total of 499048 bits (average of 3.75 bits per symbol), removing the spaces reduced this to 468040 bits (average of 4.16 bits per symbol). The removal of the spaces reduced the compressed file size.

This confirms a well known fact: text with spaces compresses more than text without spaces [Tea98]. To illustrate this point, consider a version of Jane Austin's *Emma*, consisting of 887902 bytes of ASCII text, compressed by PPMD5. With spaces left intact, it reduces to 1647952 bits (1.86 bits per symbol). With spaces omitted, it reduces to 1710224 bits (2.37 bits per symbol). This corresponds to a 4% *increase* in file size when spaces are removed. However, when the same text file is compressed with PPMD2, the results are reversed: with spaces it is reduces to 2355736 bits (2.65 bits per symbol), without spaces it is further reduced to 2262344 bits (3.11 bits per symbol). Although the number of bits per symbol is lower when spaces are encoded than when they are not encoded, it is not enough of an improvement to outweigh the smaller number of symbols.

To judge whether the above results apply to the 50 images in the UW corpus, all of the corresponding text files are concatenated together and the resulting file is compressed with and without spaces. When this concatenated file is compressed using PPMD2 it reduces to 440416 bits (3.12 bits per symbol), and without spaces it decreases further to 422688 bits (3.57 bits per symbol). When the same file is compressed using PPMD5 it reduces to 345264 bits (2.45 bits per symbol) but when spaces are removed it increases to 349088 bits (2.95 bits per symbol).

	With Spaces	Without Spaces
Bits per symbol	3.75	4.16
Total bits	499048	468040
Total symbols	141071	118508

Table 4.10: Compressing the ground truth text with and without spaces

Similar results are observed for a large plain English text file (Emma) and the text files in our image corpus. In Section 4.4 we have shown that PPMD2 gives the best compression (due to the size of the input text), and this section has shown that it is not beneficial to encode space information. For space information to be beneficial, the number of the components in an image must be larger than the number of components in the UW corpus. Although Kia [Kia97] suggested that insertion of spaces can help compression, we have shown that for single pages in the absence of training information, space insertion does not give compression gains.

4.4.3 Evaluating the docstrum enhancements

In Sections 4.4.1 and 4.4.2 two enhancements to the docstrum method were described: the use of PPMD2 to encode the symbol indices and the movement of image noise to the end of the PPMD symbol stream. Both enhancements decrease the size of symbol sequence. This section evaluates these enhancements as a part of the entire compression process, and compares the compressed file sizes.

When encoding image noise, we assume that the noise will be randomly distributed and we know by definition that the components will be very small. In this case, the encoding of hundreds of small components, many consisting of just one or two pixels, can be streamlined by modelling the offset positions more accurately. A typical component's height and width is encoded along with its horizontal and vertical position. If we code all the noise as single pixels, we need only code the relative offsets between them. The relative offset

between pixels is coded in a row major, column minor order. Each offset is coded with a tree-based gamma encoder. Better encoding orders are possible—the noise pixels could be encoded using a minimum distance function determined by the Travelling Salesperson Problem [Sed92]. The time complexity when there is a lot of noise (large N) make experiments on the images infeasible.

Table 4.11 shows the results of successive improvements to the docstrum reading order method. The first column represents the normal docstrum with the gamma encoder. The next column represents the addition of PPMD2 and the last column represents the use of noise moving (re-ordering small components under 0.1 mm^2) and PPMD2. The table shows consistent improvement by using PPMD2. When PPMD2 and noise moving are combined, the result is not always an improvement. Images such as V004 contain large halftone areas and have been expanded significantly using our naïve method of encoding the noise.

Table 4.12 compares the original docstrum figures with the enhancements and tests for significant differences. The table shows that the docstrum method using the gamma encoder (average of 46468 bytes) requires significantly more space than when PPMD2 is used in its place. There is no significant improvement when both PPMD2 and noise moving are used.

4.5 Summary

This chapter compares the standard methods for determining reading order of an image. Six different methods were described and implemented. We found that there was little difference between the methods of Zhang and Howard but that both of these methods were significant improvements over using the profile method. The docstrum was introduced as a novel method for document image compression and was shown to achieve significantly better compression than the other methods.

	Docstrum	PPMD2	PPMD2 with noise moving
A006	66134	65495	65442
A034	62769	62529	62901
A03J	48488	48303	48072
A04D	76730	76519	76276
A05E	59054	58465	58228
C03I	48974	48454	48037
C048	38466	38132	38309
D03D	43554	43630	43893
D03M	19793	19715	19699
D048	26552	26362	26354
D04G	58103	57903	57418
D05N	28312	28150	27968
D06C	36467	36296	36251
D06N	50714	50602	50484
E009	26507	26474	26458
E00H	37968	37833	37350
E00M	40835	40701	40400
E01J	73587	73548	73061
E037	78657	78678	79270
E04I	43263	43092	42622
H00C	16066	16023	15986
H019	37150	37299	37302
H041	42392	42237	42101
H04D	27805	27664	27758
H04F	58755	58579	58555
I00J	70870	70674	70519
I00L	73308	72652	72411
I037	42942	42614	42543
J03M	50786	49986	48996
J04A	45595	45722	45308
J04K	68702	68988	67161
K009	39745	39759	39817
N01A	18645	18606	18605
N027	17231	17295	17339
N02A	6462	6488	6530
N02K	54468	54486	52864
N03K	24971	25119	24846
N048	35147	34985	34786
N05I	40998	40856	40777
S021	50533	50127	50028
S03G	56569	56663	56544
S03R	35563	35511	35532
S044	45424	45060	45057
S047	66502	66032	65872
S048	65635	65423	65345
S04H	59778	58975	59170
V004	55640	55262	61688
V00C	39262	38987	39136
V00G	56176	55570	55732
V00N	55381	54825	54917
Mean	46469	46267	46234

Table 4.11: Enhancing the docstrum with PPMD and noise moving

Docstrum	PPMD2	PPMD2 with noise moving
46468.60	46267.00 $\surd$ ($t = 5.70$)	46234.40 $\times$ ($t = 1.64$)

Table 4.12: Comparing the docstrum enhancements for significance

When the test images were both skewed by random angles and rotated 90 degrees anti-clockwise, the docstrum method outperformed or equalled the compression of the other methods. The docstrum, Howard, and Zhang methods increased the file size by around 3% if the image was rotated 90 degrees.

Two methods for encoding symbol indices were discussed: PPMD with a second order model was shown to give the best compression. Two approaches to encoding previously unseen symbols were investigated and the procedure of assigning a single value to the unseen case was shown to be best.

To increase the performance of the symbol encoding method, noise components were taken out of the normal symbol stream and were encoded last. This had a noticeable impact on the performance of the PPMD method and achieved a 16% compression gain on the symbol sequence. When the method was evaluated on the image corpus, the improvements were not shown to be significant. This is because the symbol sequence is a small fraction of the total compressed file size.

Chapter 5

Skew detection

It is difficult to orient documents accurately on a flatbed scanner. Whenever a page is scanned from a book, the user has to contend with the book's spine, trying to press the page flat whilst making sure it is aligned correctly with the glass scanning face. As the number of pages in the document increases, so do the chances that some of them will be misaligned.

Although it is difficult to scan a document perfectly, the assumption that lines of text are horizontal underlies most document image compression systems. In this chapter we introduce methods of detecting skew in English documents and compare their accuracy. The aim is to accurately determine the amount of skew so that reading order can be reliably inferred and the size of the component offsets is minimised.

Baird [Bai87] claims that a reasonably careful user can place a document on a scanner with an error of less than 3 degrees. The distribution of skew angles in the UW database verifies this claim and our own results confirm this. Figure 5.1 shows the document A06M, which has a skew of 3 degrees. Although this image contains a partially visible page, the skew angle corresponds to the page that is wholly visible.

Throughout this thesis, positive skew is defined to move the page through an anti-clockwise rotation.[1]

[1] This contrasts with the UW database, which refers to this as a negative skew.

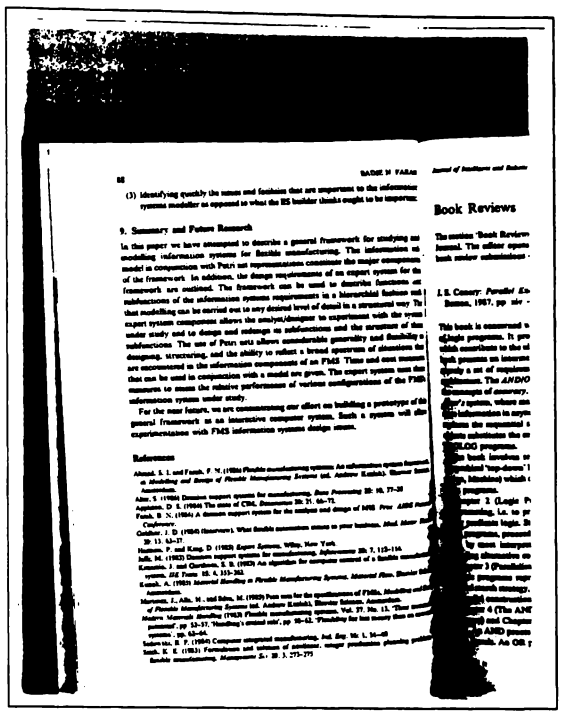


Figure 5.1: Image A06M demonstrating a skew angle of 3 degrees

The University of Washington database [UW93] contains 979 images that have had their skew calculated manually. The skew angle distribution, shown in Figure 5.2, closely approximates a normal distribution. The maximum skew angle is 3 degrees, the minimum is -1.8 degrees, and the mean is 0 degrees with a standard deviation of 0.5 degrees.

Skew detection is usually one of the first stages in a document analysis system, and it needs to be as accurate as possible. If a document is skewed by more than about 0.3 degrees, neighbouring lines interfere with each other when projected horizontally [Phi68, Bai87]. Even such a small skew interferes with standard reading-order determination methods, which assume the page is horizontal [WBE+94, WMB94].

Once a page's skew is known, line-finding and reading-order procedures can use it to either rotate the components before processing, or incorporate the skew information into the process. When skewed documents are processed by a document compression system that ignores skew, the line finding stages group the wrong components together, thereby incur-

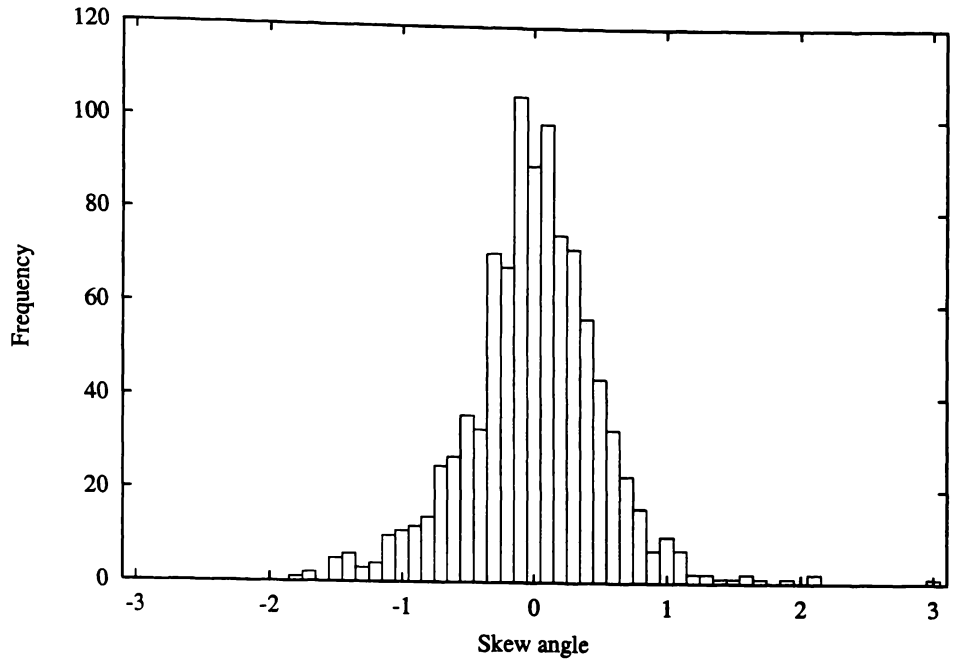


Figure 5.2: Distribution of skew angles in the UW database

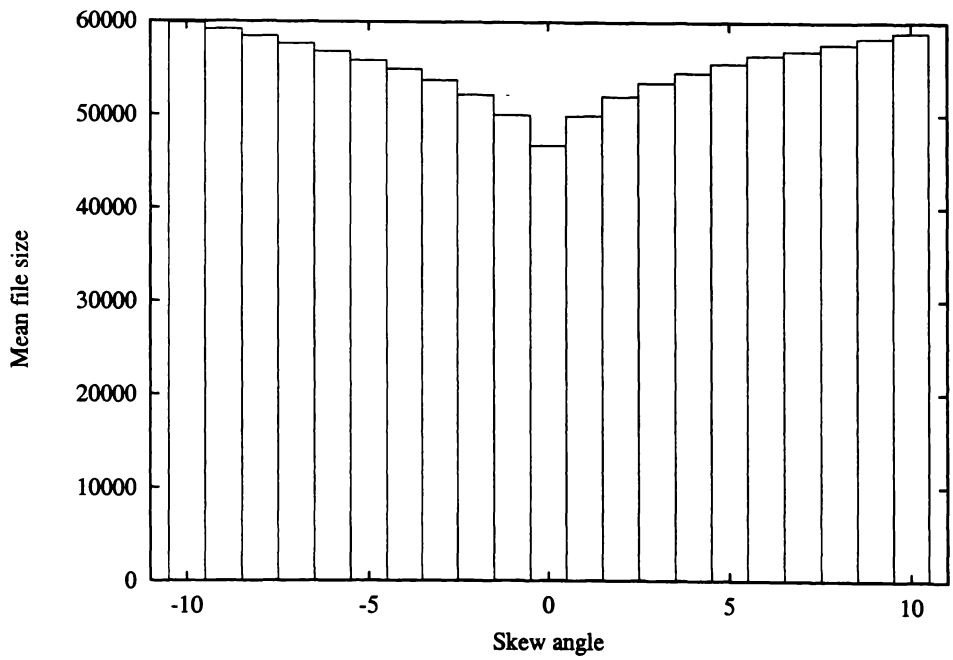


Figure 5.3: Skew angle vs. average file size over the test images

ring compression penalties in both the symbol sequence and the component position offsets.

Figure 5.3 shows the quantitative effect of skew when images are rotated and then compressed while ignoring skew information. Each ordinate gives the mean compressed file size of the fifty test images after they are rotated. Each image was rotated around its centre by the specified amount, keeping width and height constant. Figure 5.3 shows that mean file size is smallest when the image is not skewed, and increases by 15% when the page is skewed by 3 degrees.

The following sections survey previous methods of detecting skew in document images and compare their accuracy. We introduce guidelines for skew detection algorithms, develop a scheme based on these guidelines, and show that it is significantly better than other methods.

Section 5.1 introduces previous work on skew detection. Skew detection has not been used in a document image compression system before, and we are the first to evaluate skew techniques by measuring the size of the compressed output. Design criteria for skew detection procedures are discussed in Section 5.2. We approach the design in a manner that seems intuitive and is consistent with how we read paper documents. The Hough transform is described in Section 5.3, along with its application to document skew detection.

Before we can evaluate the utility of skew detection in a document compression system, the best case scenario must be determined. When using the UW database, we assume that the best case is when skew has been measured manually. Section 5.4 evaluates the compression process by comparing pages where the skew is known and with pages that are assumed to have no skew.

Section 5.5 evaluates the Hough transform as a method for skew detection using three accuracy measures. The section also considers whether it is best to perform the transform on pixels or components. Methods to reduce the time and space complexity of the transform are discussed, along with an analysis of how its accuracy and robustness change as the

resolution of the images is altered. Section 5.5.4 describes the hierarchical Hough transform, which iteratively approximates the transform and is an order of magnitude faster. The section concludes by evaluating the compression using the predicted skew angle, and comparing this with the use of manually measured skew angles. Surprisingly, the predicted skew values are more accurate than the manually measured skew angles, and when applied to the compression process, the mean file size is reduced.

Section 5.6 calculates a variety of features that can be generated to determine skew from the Hough accumulator array and evaluates their relative accuracy. The coding of component offsets is discussed in Section 5.7.

5.1 Background

Many methods for determining the orientation of scanned documents have been published over the last 15 years. Almost half of them use pixels as the basic unit for analysis while the others use some form of component. Of those that use components, the fiducial point of a component has been chosen to be its centroid [SOF+92, Sah93, Ing95, YJ96], its lower middle point [Bai87], its lowest point as determined by Group 4 encoding [Spi92], and its lower left corner [LTW94, YTS95]. At least five schemes have used some form of the projection profile [Bai87, AH90, SOF+92, Spi92, Smi95] and seven have used the Hough transform [SG89, NSF+90, HFD90, Yan93, LTW94, Ing95, YJ96]. Bones [BGCS90] combines the projection profile with autocorrelation to calculate skew angle. Other solutions have involved line fitting [PZ92, SOF+92, CH94, Smi95, YTS95], morphological transforms [CH94, CHP95], neural networks [RB95] and Fourier transforms [Pos86]. Few methods are designed to process grey-scale images [Pos86, Yan93, CHP95] and still fewer work with upside-down text [SG89].

The earliest well-documented skew detection approach was developed by Trincklin [Tri84]

with a system called Redresseur. Redresseur calculates a least squares fitting through vertical white run-lengths, and is claimed to work well for various forms of text, including tidy hand-print. Baird [Bai87] notes that no analysis of accuracy or speed is presented and that Redresseur requires a single column of text and a noise-free left margin.

Hashizume *et al.* [HYR86] note that characters are separated by small distances along the direction of a line. They find the single nearest neighbour and connect each pair of characters with a straight line segment. A histogram of the line segment orientations shows that the peak generally corresponds to the skew angle. Hashizume *et al.* evaluate their method on 13 sample images and find the average error to be 1.5 degrees and the maximum error 4.1 degrees. Baird [Bai87] comments that this method fails for document regions with wide character spacing—as occurs in newspapers.

Postl [Pos86] introduces two methods of skew detection. Unlike most other schemes, these can be applied to continuous tone images. The first method simulates skew for a range of angles and computes the density of scan-lines for each skew angle. Then, for each pair of neighbouring lines, the difference between their intensities is calculated. These differences are added together to give a measure of straightness for the simulated angle. In an experiment consisting of a single image, the skew was determined with an error of 0.06 degrees. The second method applies a 2D Fourier transform to the image and processes the power spectrum coefficients. For all angles an alignment measure, or *premium*, is calculated by integrating the power spectrum along a vector corresponding to the angle. The peak in the histogram formed by the premium corresponds to the skew angle of the image. No accuracy or speed figures are given for this method.

Rastogi and Srihari [RS86] introduce the Hough transform for skew detection. They perform the transform in the pixel domain. Because of the abundance of clustered pixels, the peaks in the Hough accumulators are not easily visible due to the blurred appearance of

the accumulator. The authors attempt to overcome this problem by investigating the low-high-low transitions of the accumulator. There is no discussion of either run-time speed or accuracy.

Baird [Bai87] introduces a projection profile using the bottom centre of a character's bounding box as the fiducial point. For each angle θ, a profile is projected into an accumulator, which is partitioned into m bins. Baird defines $c_i(\theta)$ to be the number of characters projected into the ith bin at angle θ and chooses the bin size to be one third of the height of a 6-point letter x. These accumulator values are combined to form an alignment measure, $A(\theta)$.

Any super-linear function of the accumulator has been shown to find the correct maximum: Baird uses $A(\theta) = \sum_{i=1}^{m} c_i(\theta)^2$. Because exhaustively generating $A(\theta)$ to find its peak is expensive, Baird uses iterative sampling and progresses from a coarse (0.7 degrees) to fine (0.03 degrees) angular resolution. Experimental results are given on fifty pages from a variety of sources, all with a skew angle within 15 degrees of horizontal.

Fletcher and Kasturi [FK88] use the Hough transform, but unlike Rastogi and Srihari they base the transform on characters rather than individual pixels. Fletcher and Kasturi use knowledge of character size and scanning resolution to reduce each character down to a single point, and then use these points as the domain for the Hough transform.

Srihari and Govindaraju [SG89] continue Rastogi and Srihari's work by investigating methods for handling the discretisation problems in the accumulator array. They describe several methods of accumulator analysis, including anti-aliasing and area normalising. No experimental results are presented.

Hinds *et al.* [HFD90] develop the use of the Hough transform, by applying a vertical run-length encoding of the characters to reduce pixels to a single point. Instead of incrementing the accumulator by unity, their approach increments the accumulator for each vertical run

in the component by the size of the run-length. The image resolution was reduced from 300 dpi to 75 dpi to decrease execution times. To eliminate the introduction of noise by margins or columns of black pixels, only run lengths between 1 and 25 pixels (corresponding to 0.3mm and 8.5mm) were transformed. At 75 dpi this assumes that all text is at most a 24 point font. To determine the skew, the single maximum point in the accumulator is found. To further decrease execution times, Hinds *et al.* decrease the range of the Hough transform to ± 15 degrees. Experimental results were given for 13 test images, two of which were scanned in landscape mode. For each image, the authors simply report whether the skew was correct or not.

Nakano *et al.* [NSF+90] attempts to calculate the baseline of components using a method similar to Section 4.1, and performs an operation analogous to the Hough transform. Instead of using an accumulator array, they present a sharpness array and four possible features that may be used to evaluate possible skew angles. Of these four features, two are selected as good indicators. Evaluation is performed on two of the eight CCITT fax test images.

Akiyama and Hagita [AH90] introduce a skew detection method based on features. A number of features such as horizontal and vertical pixel profiles, crossing counts, and component bounding box sizes are combined into an alignment measure, and the peak is selected as the skew angle.

The docstrum (described in Section 4.2) was introduced in Story *et al.* [SOF+92], and is a plot of inter-component distances with measurements being taken between the components' centroids. Figure 4.3 shows the docstrum of an example image with $k = 5$. This value for k usually picks two characters on either side of the component, as well as one above or below. The docstrum estimates the skew by integrating over the angle in the polar plot to find the most common angle between components. This initial estimate of the skew is extended by fitting a least-squares line through components which have been identified as

being on the same line [O'G93]. Neither Story *et al.* [SOF+92] nor O'Gorman [O'G93] present quantitative results. The docstrum method of skew determination is similar to the slope histogram but restricts attention to the nearest neighbours of each component. The overall skew is approximated by the average of local skew values. This means that it is insensitive to global effects such as vertically aligned characters. Other interesting features such as inter-line and inter-character spacing can be extracted from the docstrum plot.

Spitz [Spi92] extends Baird [Bai87] but uses fiducial points derived from the CCITT Group 4 compressed codes. Instead of using the lower centre point of a component's bounding box, Spitz uses a black pixel (usually) towards the bottom of the character, that has white to the right and beneath it. No connected component analysis is required. The exact position of this point is derived from the modified READ (relative element address designate) or G4 coding [HR80]. Spitz presents a fast system, and gives results on three images.

Saheed [Sah93] calculates the most common angle between each pair of components by generating a histogram of the angle between each component and every other one. The angle is measured between the components' centroids. It is quantised into bins of the required resolution and plotted as a histogram of frequencies. Slope histograms sometimes give 90 degree errors on tall portrait images that contain many vertically aligned components with small horizontal displacements. In this case two peaks appear at around 90 and 0 degrees. Saheed uses 31 test images, including the eight CCITT fax test images. No measured orientations for the images are provided, although the results show that the skew calculated for 5 of the 31 images is 90 degrees out.

Yan [Yan93] uses a vertical, pixel-based, cross-correlation between pairs of lines to determine the optimal skew. Only two results are given, one of them being in error by two degrees. This is another approach that uses continuous instead of bilevel images.

Le *et al.* [LTW94] use the Hough transform on a simplified binary image. In this system

the transform is performed only on the bottom row of pixels within each component.

Smith [Smi95] introduces a new scheme and compares it with Baird [Bai87] on over 400 document pages. High accuracy is demonstrated up to 15 degrees of skew. The process works by grouping characters into an approximate reading order. When an entire line is found, the baseline for the row is calculated. The document skew is found by averaging the individual lines. The character-by-character line finding process is derived from Pratt *et al.* [PCC+80].

Inglis [Ing95] presents a method using the hierarchical Hough transform over four levels. The fiducial point is the centroid of each component, and the method is compared against Saheed [Sah93] and the docstrum [O'G93] over fifty test images from the UW database [UW93]. The Hough transform is found to be the most accurate method.

Yu and Jain [YJ96] use the hierarchical Hough transform based on centroids. The authors use a rough-to-fine strategy over two levels. This paper differs from Inglis [Ing95] by using non-integer increments to the Hough accumulator array and anti-aliasing the array.

A common theme across most of this previous work is the lack of experimentation on multiple images, no error measures, and image corpuses that are not publicly available. This chapter addresses these issues.

5.2 Skew detector design

The skew of a document reflects the skew of the individual lines of text that it contains. We list three constraints that appear to subjectively describe how a human determines skew:

1. Do not need to be able to resolve characters
2. Individual skew measures for each line are calculated
3. The individual line orientations are combined to give the global skew of the page.

Item 1 suggests either a character/component based approach, or a form of resolution reduction. The former approach could use the process outlined in Chapter 3. Resolution reduction (for example, reducing from 300 dpi to 150 dpi) can be achieved either by reducing to greyscale and then thresholding, or by the JBIG1 resolution reduction method [JBI93]. Once the image is reduced to point sources, item 2 suggests using a projection profile or Hough transform to determine the skew of each line. Item 3 integrates the resulting measurements by calculating features based on the profile or accumulator to determine the final orientation.

Based on these criteria and the results of Inglis [Ing95] and Yu and Jain [YJ96], the following sections use a component-based Hough transform. Initially, the greatest value in the accumulator is used to determine the skew angle, although alternative methods are introduced in Section 5.6.

Although we have assumed that skew will be uniform, the Hough transform can be modified to find curves, so that text close to a book's spine can be modelled.

5.3 Hough transform

The Hough transformation maps positions in the (x, y) coordinate system to a parametrically defined coordinate system [Hou62]. For each position (x, y), all lines that pass through this point have the form

$$y = mx + c \tag{5.1}$$

where we view x and y as constants, and m the gradient and c the y-intercept of the line. Each line that passes through (x, y) is defined by a position in (m, c) space. Multiple points arranged in a line in (x, y) space correspond to *the same point* in (m, c) space.

Equation 5.1 is undefined for vertical lines. This obstacle can be overcome by rearranging with respect to the *normal of the line* where ρ and θ are the distance and angle to the line from the origin. That is,

$$\rho = x \cos\theta + y \sin\theta. \tag{5.2}$$

This mapping is similar to the linear case but now M collinear points map to M sinusoidal curves that intersect at (ρ, θ). Conversely, an intersection point of curves in polar space represents a series of lines passing through that point.

5.3.1 Skew detection

The Hough transform can be used for skew detection by mapping N source locations to their corresponding positions in Hough space. For skew detection, the source locations are either positions of pixels or some fiducial point derived from a component. Each time a point is mapped into Hough space, an *accumulator* array records the frequency of usage for that location. The accumulator can be viewed as a histogram in Hough space, and finding the maximum value in this array corresponds to the "most co-linear" angle for the image. In the context of skew detection, we assume that co-linearity corresponds with the skew of

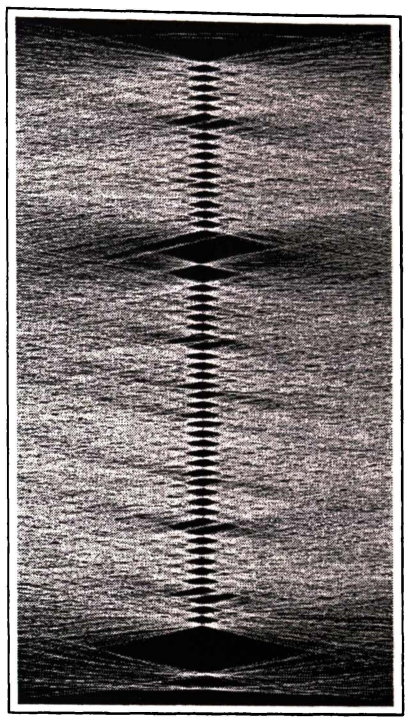


Figure 5.4: Hough accumulator array for image A06M (using components)

the image.

The time complexity of the Hough transform is $O(NB)$ where N is the number of objects and B the number of bins (quantising θ) required to achieve the desired resolution. Figure 5.4 is a typical Hough transform that has been performed with $N = 4089$ and $B = 180$ ($180/B = 1$ degree bins). Black values in the figure indicate low accumulator values, while white values indicate high values.

5.4 Compression when the skew angle is known

To determine a lower bound for the compression that can be achieved for skewed images, we assume that the smallest file will be generated when the manually measured skew angle is used to determine reading order. The manual skew angles from the UW database are used as ground truth data.

To compare image compression across known and unknown skew, each of the fifty images was skewed by a random amount, shown in the *applied* column of Table 5.1. When the original skew of the page is combined with the randomly applied skew, the resulting skew angle is shown in the *new skew* column. The *unknown* column assumes that the skew is not known, and therefore that the text is horizontal. In the *known* column, the skew of the image is known, and the coordinates of the components are rotated by the skew angle and extracted and sorted into reading order using the new positions.

Table 5.1 shows that the mean compressed file size when the skew angle is known gives about 1% improvement on average, the resulting file being smaller in 43 of the 50 cases.

These figures are disappointing. If the manually measured skew angle really does represent a lower bound, the best compression improvement we can achieve is only 1%. Nevertheless, Table 5.2 compares these two columns and shows that results with known skew are significantly better ($t = 5.51$) than with unknown skew.

The next section investigates how well we can automatically determine skew, and in Section 5.5.5 we will see whether we can improve on these figures using an automated method.

5.5 Automatically determining page skew

Before using a skew detection method in image compression, we first evaluate the accuracy of the Hough transform method. The first experiment examines the input transformation to point sources for the Hough transform. With black and white images there are two options: components or pixels. When components are used, the Hough transform is performed on the components' centroids, otherwise it is performed on individual pixels. These two options are investigated in Section 5.5.1 along with a further modification: incrementing the accumulator with the logarithm of the area of each component. The logarithm transform is

Image	Skew	Applied	New skew	Unknown	Known
A006	−0.45	0.2	−0.25	66969	66926
A034	0.21	4.0	4.21	73039	72694
A03J	−0.81	8.0	7.19	57990	58041
A04D	0.38	7.4	7.78	93034	92677
A05E	0.00	6.8	6.80	71992	71082
C03I	0.56	2.9	3.46	55401	55066
C048	0.11	5.2	5.31	48293	47720
D03D	0.65	6.6	7.25	51621	51813
D03M	0.30	−0.3	0.00	20242	20242
D048	0.04	9.7	9.74	37673	37185
D04G	0.41	1.4	1.81	63010	62934
D05N	−0.29	0.1	−0.19	28399	28386
D06C	0.57	−4.7	−4.13	44477	44042
D06N	−0.32	−6.0	−6.32	63671	63283
E009	0.38	9.0	9.38	35951	35406
E00H	−0.12	−3.2	−3.32	45443	45105
E00M	0.06	2.3	2.36	48450	48409
E01J	0.05	−2.3	−2.25	84175	83595
E037	−0.13	−2.6	−2.73	86719	86085
E04I	−0.04	9.2	9.16	53988	53996
H00C	0.12	4.0	4.12	19379	19377
H019	−0.28	−5.4	−5.68	47051	46979
H041	0.51	−10.0	−9.49	67439	67454
H04D	0.06	2.9	2.96	32564	32384
H04F	0.35	−9.4	−9.05	81168	79904
I00J	0.18	−4.0	−3.82	80763	80240
I00L	1.01	4.1	5.11	85517	84976
I037	0.39	−7.0	−6.61	53384	52930
J03M	0.75	2.8	3.55	57933	57566
J04A	0.28	−4.0	−3.72	50772	51026
J04K	0.89	−3.5	−2.61	77261	77554
K009	0.39	−7.0	−6.61	52401	52038
N01A	−0.36	−10.0	−10.36	24918	24642
N027	0.26	−5.4	−5.14	20941	20868
N02A	0.31	−9.7	−9.39	8773	8772
N02K	−0.47	6.8	6.33	63256	63978
N03K	0.00	7.5	7.50	30673	30398
N048	−0.33	5.5	5.17	41453	41067
N05I	−0.34	3.4	3.06	47650	47074
S021	1.00	−2.8	−1.80	57751	57596
S03G	0.08	5.2	5.28	69012	68362
S03R	−0.06	−5.2	−5.26	44857	44319
S044	−0.14	7.3	7.16	59565	58602
S047	−0.40	−9.5	−9.90	89327	88172
S048	0.79	−1.2	−0.41	72122	72042
S04H	0.74	6.3	7.04	78543	77316
V004	0.10	−2.7	−2.60	69883	70474
V00C	0.14	−8.9	−8.76	60659	60111
V00G	−0.29	−6.0	−6.29	72281	71118
V00N	−0.49	4.7	4.21	68334	67516
Mean				56323	55991

Table 5.1: Compressed file sizes for unknown and known skew angles

Unknown Skew	Known Skew
56323	55991
	$\surd(t = 5.51)$

Table 5.2: Comparing compression using unknown versus known skew angles

used because it returns small values when the area of the component is small, while giving large regions little more weight than that of typical characters. Informal experiments that incremented the accumulator by the value of the area and the square root of the area gave poor results.

In the following sections, each method is evaluated on the fifty test images. The error of the Hough transform is determined by measuring the root mean squared (RMS) error, maximum absolute error (MAE), and the correlation coefficient (CC), when measured with respect to the manually determined skew angles.

The search space for the Hough accumulator is between -45 and 45 degrees in 0.1 degree increments. It seems impossible to find the correct skew of a page that has been rotated 90 degrees without performing some form of character recognition. To determine whether a tall and narrow column is upright or whether it is in fact a wide and short row of text involves recognising the individual characters and making a higher level judgement. In our experiments we will find the most likely skew that is in the range -45 to 45 degrees.

The skew angle will be determined by finding the accumulator cell with the greatest value. Alternative approaches for determining the skew angle based on the accumulator array will be investigated in Section 5.6.

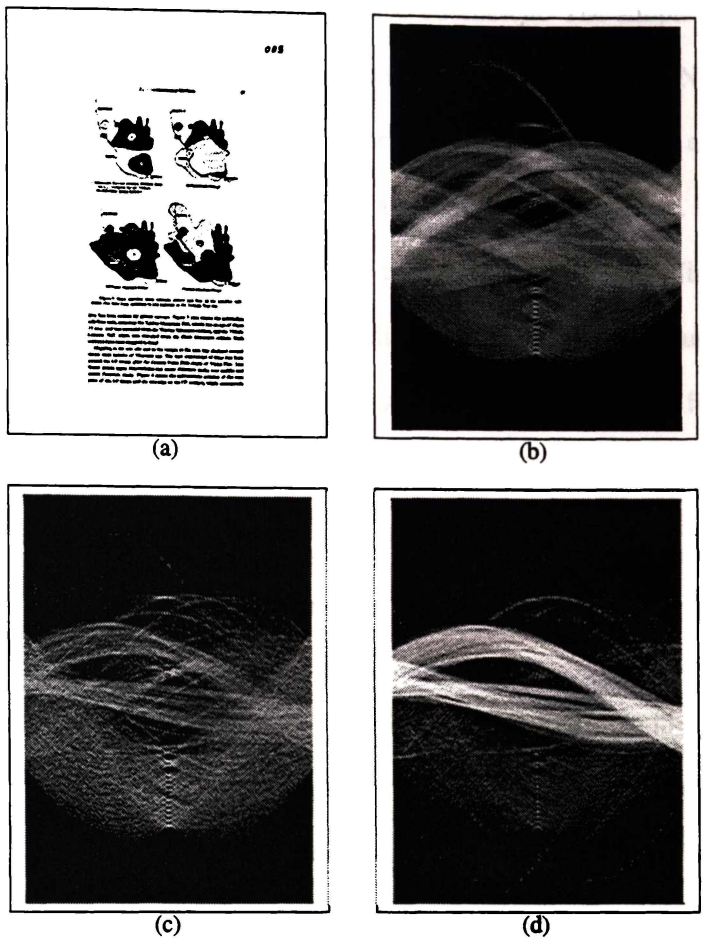


Figure 5.5: Hough transformations using the N02K image. (a) The original image (b) Hough transform using black pixels (c) Hough transform using components, incrementing by $\log(area)$ (d) Hough transform using components.

5.5.1 Using components, component area, or pixels

This section evaluates the best domain for the Hough transform. The three alternatives include using black pixels as points to be transformed into Hough space (Figure 5.5b), using the centre of the components but updating the accumulator by the logarithm of the components' area (Figure 5.5c), and using the centre of the components as a point (Figure 5.5d).

The skew angles predicted using each of the three domains are shown in Table 5.3. The table shows the manually measured skew angle along with the predicted skew angles. Table 5.3 shows that the $\log(area)$ column has the lowest maximum absolute error, the lowest

root mean squared error and the highest correlation coefficient. The simple use of the components performed quite well, but large mistakes were made on images N02K and V004. These two images contain large areas of halftone pictures which generate a lot of co-linear points in Hough space, forming a spurious peak at an incorrect skew angle. The last column of the table shows the results when using pixels instead of components. Nine values in this column have large (greater than five degrees) differences from the measured skew angle. When these images are examined (eg. images A03J or J04K), they are found to contain large black borders, typically seen when photocopying close to the spine or edge of a book.

5.5.2 Quantising the Hough parameters

The Hough accumulator is a quantised version of the Hough space. In the previous section the parameter θ was quantised to 0.1 degree resolution, while the parameter ρ was not quantised beyond pixel resolution. This means that a Hough transform for an image with $w = 2592$ and $h = 3300$, between -45 and 45 degrees in 0.1 degree intervals requires a array of $(45 + 45 + 1)/0.1$ by $\sqrt{(w^2 + h^2)}$, which is 910 by 4197.

Quantising ρ has two benefits. First, the reduced precision makes more points co-linear. Second, it decreases memory requirements for the array.

Table 5.4 shows similar results to Table 5.3, with two different quantisations of ρ. The $\log(area)$ method continues to give the best performance, although the accuracy of the other two methods increases slightly. When ρ is reduced to $1/4$ of its value, the error increases slightly and the correlation coefficient decreases slightly. This trend continues when ρ is reduced to $1/8$ of its value.

Image	Skew	Components	log(*area*)	Pixels
A006	−0.5	0.3	−0.3	−0.5
A034	0.2	0.7	0.7	0.7
A03J	−0.8	1.1	1.1	10.0
A04D	0.4	0.3	0.8	1.0
A05E	0.0	0.3	0.3	−11.0
C03I	0.6	0.3	0.5	0.5
C048	0.1	0.3	0.3	0.5
D03D	0.7	0.8	0.8	8.3
D03M	0.3	0.5	0.5	−7.4
D048	0.0	0.3	0.3	0.5
D04G	0.4	0.6	0.6	−7.4
D05N	−0.3	0.3	−0.2	1.2
D06C	0.6	0.9	0.9	0.9
D06N	−0.3	−0.1	−0.1	−0.5
E009	0.4	0.8	0.8	1.1
E00H	−0.1	0.2	0.2	0.3
E00M	0.1	0.3	0.3	0.3
E01J	0.1	0.3	0.4	−1.1
E037	−0.1	0.1	0.1	1.5
E04I	−0.0	0.3	0.3	0.2
H00C	0.1	0.4	0.2	0.2
H019	−0.3	0.3	0.1	0.1
H041	0.5	1.0	1.0	0.1
H04D	0.1	0.3	0.2	0.0
H04F	0.3	0.8	0.8	0.9
I00J	0.2	0.5	0.5	8.3
I00L	1.0	1.2	1.2	8.3
I037	0.4	0.6	0.6	−7.4
J03M	0.8	0.3	0.9	1.1
J04A	0.3	−4.7	0.6	−0.1
J04K	0.9	1.1	1.1	11.9
K009	0.4	0.3	0.5	−1.1
N01A	−0.4	−0.1	−0.1	−0.1
N027	0.3	0.6	0.6	0.6
N02A	0.3	0.6	0.9	0.8
N02K	−0.5	19.7	−0.1	0.0
N03K	0.0	0.5	0.5	0.6
N048	−0.3	0.5	0.2	0.2
N05I	−0.3	0.4	0.1	0.3
S021	1.0	1.4	1.5	1.6
S03G	0.1	0.5	0.5	0.5
S03R	−0.1	0.3	0.3	0.4
S044	−0.1	0.3	0.3	0.3
S047	−0.4	−0.1	−0.1	−0.4
S048	0.8	1.1	1.1	1.1
S04H	0.7	1.1	1.1	0.5
V004	0.1	45.8	0.4	0.6
V00C	0.1	0.3	0.3	0.4
V00G	−0.3	0.0	0.0	0.0
V00N	−0.5	−0.5	0.0	0.4
MAE		1.76	0.34	1.98
RMS		7.11	0.43	3.83
CC		−0.06	0.80	0.14

Table 5.3: Calculating the Hough transform for components, area and pixels

Image	Skew	Components	log($area$)	Pixels	Components	log($area$)	Pixels
		reducing ρ by 4			reducing ρ by 8		
A006	−0.5	−0.4	−0.3	−0.5	−0.2	−0.2	0.3
A034	0.2	0.6	0.6	0.7	0.6	0.7	0.6
A03J	−0.8	1.1	1.1	10.0	1.0	1.1	10.2
A04D	0.4	2.0	0.8	1.0	1.6	0.9	1.0
A05E	0.0	0.3	0.4	−11.8	0.0	0.0	−11.7
C03I	0.6	0.4	0.4	0.4	0.4	0.5	0.3
C048	0.1	0.4	0.4	−45.0	0.6	0.6	−44.9
D03D	0.7	0.9	0.9	8.6	0.9	1.1	9.0
D03M	0.3	0.6	0.6	−8.1	0.6	0.6	−8.0
D048	0.0	0.3	0.3	0.8	0.3	0.3	0.8
D04G	0.4	0.5	0.5	−7.4	0.6	0.4	−8.1
D05N	−0.3	0.3	−0.1	1.1	0.3	−0.2	−1.0
D06C	0.6	0.8	0.9	1.0	0.9	0.9	0.9
D06N	−0.3	0.0	0.0	0.3	0.0	0.0	0.2
E009	0.4	0.7	0.7	0.2	0.9	0.9	0.0
E00H	−0.1	0.4	0.3	0.2	0.2	0.2	0.3
E00M	0.1	0.3	0.3	0.4	0.4	0.4	0.3
E01J	0.1	0.3	0.3	−0.8	1.0	0.3	−1.0
E037	−0.1	0.2	0.1	1.5	1.6	0.0	1.4
E04I	−0.0	0.7	0.4	0.4	0.7	0.3	0.6
H00C	0.1	0.3	0.4	0.2	0.4	0.1	0.4
H019	−0.3	0.2	0.1	0.5	0.3	0.1	0.1
H041	0.5	1.0	1.0	0.2	1.1	1.1	0.0
H04D	0.1	0.6	0.4	−0.1	0.4	0.4	0.0
H04F	0.3	0.8	0.8	0.9	0.7	0.7	0.5
I00J	0.2	0.3	0.3	8.3	0.6	0.6	8.4
I00L	1.0	1.4	1.4	8.7	1.6	1.6	8.3
I037	0.4	0.3	0.7	−7.4	0.3	0.5	−7.3
J03M	0.8	0.4	0.9	1.1	0.4	1.1	0.4
J04A	0.3	−4.2	−0.1	0.0	−4.5	−0.1	0.2
J04K	0.9	6.0	1.2	12.2	6.1	0.9	12.2
K009	0.4	0.3	0.6	−1.0	0.3	0.6	−0.9
N01A	−0.4	−0.1	−0.1	−0.2	−0.3	−0.2	−0.1
N027	0.3	0.5	0.5	0.5	0.6	0.6	0.5
N02A	0.3	0.8	0.9	0.8	1.0	0.9	0.9
N02K	−0.5	−39.3	0.0	−0.1	−39.2	−0.2	6.9
N03K	0.0	0.6	0.6	0.5	0.4	0.4	0.4
N048	−0.3	0.6	0.2	0.1	0.8	0.3	0.2
N05I	−0.3	0.4	0.2	0.4	0.5	0.0	0.2
S021	1.0	1.6	1.6	1.4	2.0	1.5	0.7
S03G	0.1	0.5	0.5	0.5	0.4	0.4	0.7
S03R	−0.1	0.3	0.4	0.3	0.3	0.3	0.8
S044	−0.1	0.3	0.3	0.3	0.0	0.0	0.1
S047	−0.4	−0.3	−0.3	−0.2	−0.4	−0.4	−0.2
S048	0.8	1.1	1.1	1.4	1.1	1.1	1.0
S04H	0.7	1.3	1.3	0.5	1.2	1.2	0.7
V004	0.1	0.6	0.6	0.5	0.5	0.6	0.4
V00C	0.1	0.2	0.3	0.4	0.4	0.4	0.4
V00G	−0.3	−0.1	−0.1	0.0	0.0	0.0	0.0
V00N	−0.5	−0.3	−0.3	−0.3	−0.2	−0.2	−0.6
MAE		1.35	0.37	2.91	1.42	0.35	3.05
RMS		5.60	0.45	7.50	5.60	0.44	7.58
CC		0.28	0.78	0.08	0.28	0.78	0.05

Table 5.4: Reducing the scale of ρ by 4 and 8

5.5.3 JBIG1 resolution reduction

When reading, it is not necessary to resolve individual characters on a page to determine skew. Looking from a distance we can accurately determine skew even though the text is unreadable. This suggests testing each method's performance on lower resolution images. Each of the images is incrementally halved in size. The original images are 300 dpi, and the new sizes are 150 dpi and 75 dpi. The JBIG1 resolution reduction method is used because it gives results that are visually superior to naïve averaging methods [JBI93].

Table 5.5 shows that the $\log(area)$ technique performs consistently well, and results continue to improve right down to 37.5 dpi. For example, at 150 dpi the maximum absolute error is 0.35 and decreases to 0.34 at 75 dpi. Table 5.6 shows that the maximum absolute error continues to decrease to 0.30 at 37.5 dpi. At this resolution most characters have been reduced to a few pixels, and the effect of large noisy areas has been ameliorated. Further reduction causes dramatic deterioration in the results. Tables 5.7, 5.8 and 5.9 show the error rates at resolutions of 18.75, 9.375 and 4.6875 dpi. At these low resolutions the results are extremely poor.

5.5.4 Hierarchical Hough transform

Although the use of the Hough transform on components has been shown to be accurate (Section 5.5.1) and performs consistently across different resolutions (Sections 5.5.2 and 5.5.3), it is rather slow. A common method of decreasing its computational cost is to generate the accumulators at a low resolution, and "zoom in" on the regions of interest at higher resolutions. We originally proposed the use of the hierarchical Hough transform for skew detection in Inglis [Ing95], and a similar method was independently published by Yu and Jain [YJ96]. Both of these use a rough to fine strategy of calculating the Hough transform, although Yu and Jain investigated methods for reducing noise in the accumulator array.

Image	Skew	Components	log(*area*) 150 dpi	Pixels	Components	log(*area*) 75 dpi	Pixels
A006	−0.5	0.3	−0.2	−0.5	0.3	−0.2	−0.4
A034	0.2	0.7	0.7	1.0	0.7	0.7	0.0
A03J	−0.8	1.1	1.1	10.0	1.1	1.1	8.3
A04D	0.4	1.0	1.0	0.2	1.0	1.0	1.0
A05E	0.0	0.3	0.3	−11.0	0.3	0.3	8.3
C03I	0.6	0.3	0.4	0.6	0.3	0.6	0.4
C048	0.1	0.4	0.4	0.5	0.4	0.4	−45.0
D03D	0.7	0.8	0.8	8.3	0.2	1.0	8.3
D03M	0.3	0.4	0.4	−7.4	0.3	0.3	8.3
D048	0.0	0.3	0.3	0.3	0.1	0.1	0.2
D04G	0.4	0.6	0.6	−7.4	0.4	0.4	−7.4
D05N	−0.3	0.5	−0.1	1.1	0.3	−0.1	−0.7
D06C	0.6	0.8	1.2	0.8	0.8	0.9	0.8
D06N	−0.3	−0.1	−0.1	−0.5	0.4	−0.1	0.4
E009	0.4	0.3	0.8	1.1	0.2	0.8	0.4
E00H	−0.1	0.4	0.2	0.3	0.4	0.2	0.2
E00M	0.1	0.3	0.3	0.7	0.2	0.1	0.4
E01J	0.1	0.4	0.4	−0.5	0.4	0.3	0.4
E037	−0.1	0.1	0.1	1.4	−0.1	0.1	1.5
E04I	−0.0	0.3	0.3	0.3	0.3	0.4	0.2
H00C	0.1	0.4	0.2	0.4	0.2	0.3	0.4
H019	−0.3	0.3	0.1	0.1	0.3	0.1	0.1
H041	0.5	0.9	0.9	0.1	1.0	1.0	0.0
H04D	0.1	0.5	0.2	0.0	0.6	0.4	−0.2
H04F	0.3	0.8	0.8	0.9	0.7	0.7	0.9
I00J	0.2	0.3	0.6	8.3	0.6	0.6	−7.4
I00L	1.0	0.9	0.9	8.3	1.5	1.5	8.3
I037	0.4	0.5	0.6	−7.4	0.4	0.7	−7.4
J03M	0.8	0.4	0.9	0.4	0.3	0.9	1.0
J04A	0.3	−2.7	0.7	0.0	−2.5	0.3	0.0
J04K	0.9	1.2	1.2	11.9	2.4	1.2	8.3
K009	0.4	0.4	0.4	20.9	0.2	0.6	19.1
N01A	−0.4	−0.1	−0.1	−0.1	−0.2	−0.2	−0.1
N027	0.3	0.4	0.7	0.6	0.4	0.4	0.4
N02A	0.3	0.9	0.9	0.8	0.8	0.8	0.8
N02K	−0.5	−45.0	−0.1	−0.1	−41.3	0.0	0.1
N03K	0.0	0.4	0.4	0.6	0.4	0.4	0.6
N048	−0.3	0.4	0.2	0.2	0.4	0.1	0.2
N05I	−0.3	0.1	0.1	0.3	0.1	0.1	0.0
S021	1.0	1.5	1.5	1.4	1.4	1.5	0.1
S03G	0.1	0.4	0.4	0.5	0.4	0.4	0.4
S03R	−0.1	0.4	0.4	0.4	0.4	0.4	0.4
S044	−0.1	0.1	0.4	0.3	0.0	0.0	0.0
S047	−0.4	−0.4	−0.3	−0.3	0.1	0.3	−0.2
S048	0.8	1.3	1.3	1.1	1.4	1.4	1.1
S04H	0.7	1.2	1.1	0.5	1.2	1.2	0.4
V004	0.1	0.4	0.4	0.4	0.4	0.3	0.4
V00C	0.1	0.3	0.3	0.3	0.2	0.4	0.4
V00G	−0.3	−0.1	−0.1	0.0	0.4	−0.1	0.4
V00N	−0.5	−0.4	−0.4	0.4	0.3	−0.1	0.4
MAE		1.30	0.35	2.34	1.28	0.34	3.01
RMS		6.33	0.44	4.79	5.81	0.44	7.69
CC		0.24	0.78	0.16	0.25	0.79	0.09

Table 5.5: JBIG1 resolution reduction to 150 and 75 dpi

Measure	Components	log($area$)	Pixels
MAE	0.40	0.30	5.73
RMS	0.53	0.36	13.44
CC	0.27	0.75	0.27

Table 5.6: JBIG1 resolution reduction to 37.5 dpi

Measure	Components	log($area$)	Pixels
MAE	1.32	8.38	13.86
RMS	5.92	17.03	24.11
CC	0.09	0.10	0.04

Table 5.7: JBIG1 resolution reduction to 18.75 dpi

Measure	Components	log($area$)	Pixels
MAE	5.22	20.18	22.76
RMS	11.78	26.07	31.50
CC	0.16	−0.04	−0.03

Table 5.8: JBIG1 resolution reduction to 9.375 dpi

Measure	Components	log($area$)	Pixels
MAE	4.60	18.37	32.61
RMS	10.17	24.98	37.95
CC	0.08	−0.02	0.14

Table 5.9: JBIG1 resolution reduction to 4.6875 dpi

Table 5.10 shows the results of calculating the Hough transform with a skew resolution of 1 degree. The $\log(area)$ column obtains the skew angle to within 2 degrees. Centering the next hierarchical step on the predicted angle, covering 2 degrees on either side of the predicted angle, and recalculating the Hough transform to 0.1 degree accuracy gives a higher resolution result. The number of components is constant, so the complexity is determined by the number of bins required for the desired resolution. At 0.1 degree accuracy, $(45 + 45 + 1)/0.1 = 910$ bins are needed. With the two stage approach, the first stage requires $(45 + 45 + 1) = 91$ bins, and the second requires $(2 + 2 + 1)/0.1 = 50$ bins, for a total of 141 bins—a speedup factor of $910/141 = 6.5$ times. The resolving power of the Hough transform is reduced as the bin size is increased. If the bin size is too coarse, the regular peaks in the accumulator that correspond to lines of components will not be visible.

5.5.5 Compressing using the predicted skew angle

Table 5.11 shows the results when the Hough transform is used to predict the skew of the image, which is then passed to the compression process. A 2.3% average compression gain is achieved over using the manually measured values, and a gain of almost 3% compared to the situation where the skew is unknown. For each individual image the Hough transform prediction of the skew achieved a smaller file size than the manually measured skew with two exceptions: images D03M and D05N, where the difference was marginal (less than 0.05%).

Table 5.12 compares the two columns for statistical significance and shows that the Hough transform gives significantly lower compressed file sizes than manually measured known values ($t = 5.54$). A large improvement can be seen on image A03J which contains a partial segment of the previous page—manual calculation of the skew may have taken this partial page into consideration. When the skew of the page is checked using a collection of

Image	Skew	Components	$\log(area)$	Pixels
A006	−0.5	0.0	−1.0	−1.0
A034	0.2	0.0	0.0	0.0
A03J	−0.8	−2.0	0.0	10.0
A04D	0.4	0.0	0.0	1.0
A05E	0.0	0.0	0.0	−12.0
C03I	0.6	0.0	0.0	0.0
C048	0.1	0.0	0.0	−44.0
D03D	0.7	0.0	0.0	8.0
D03M	0.3	0.0	0.0	−8.0
D048	0.0	0.0	0.0	0.0
D04G	0.4	0.0	0.0	−8.0
D05N	−0.3	0.0	0.0	0.0
D06C	0.6	0.0	0.0	0.0
D06N	−0.3	0.0	0.0	−1.0
E009	0.4	0.0	0.0	0.0
E00H	−0.1	0.0	0.0	0.0
E00M	0.1	0.0	0.0	0.0
E01J	0.1	0.0	0.0	−1.0
E037	−0.1	0.0	0.0	1.0
E04I	−0.0	0.0	0.0	0.0
H00C	0.1	0.0	0.0	0.0
H019	−0.3	0.0	0.0	0.0
H041	0.5	0.0	0.0	0.0
H04D	0.1	0.0	0.0	−1.0
H04F	0.3	0.0	0.0	0.0
I00J	0.2	0.0	0.0	8.0
I00L	1.0	1.0	1.0	8.0
I037	0.4	0.0	0.0	7.0
J03M	0.8	0.0	0.0	0.0
J04A	0.3	−1.0	0.0	0.0
J04K	0.9	7.0	1.0	12.0
K009	0.4	0.0	0.0	−1.0
N01A	−0.4	0.0	0.0	−1.0
N027	0.3	0.0	0.0	0.0
N02A	0.3	0.0	0.0	0.0
N02K	−0.5	−41.0	−1.0	4.0
N03K	0.0	0.0	0.0	0.0
N048	−0.3	0.0	0.0	0.0
N05I	−0.3	0.0	0.0	0.0
S021	1.0	1.0	1.0	1.0
S03G	0.1	0.0	0.0	0.0
S03R	−0.1	0.0	0.0	0.0
S044	−0.1	0.0	0.0	0.0
S047	−0.4	−1.0	−1.0	−1.0
S048	0.8	1.0	1.0	1.0
S04H	0.7	1.0	1.0	0.0
V004	0.1	0.0	0.0	0.0
V00C	0.1	0.0	0.0	0.0
V00G	−0.3	0.0	0.0	0.0
V00N	−0.5	−1.0	−1.0	0.0
MAE		1.22	0.28	2.88
RMS		5.81	0.35	7.36
CC		0.30	0.70	0.09

Table 5.10: Hough transform using with a skew resolution of 1 degree

baselines at the beginning and ends of lines, it is obvious that the Hough transform method has determined the skew more accurately than the "ground truth" data.

This result is explained by the method used for manual calculation of the skew angle. Whereas a collection of end points is used to determine the overall skew of the document manually, the Hough transform induces a complete pairwise enumeration of all components on the page. Figure 5.3 showed that even a small skew angle increases compression significantly, and it follows that reduced precision due to fewer data points leads to manual skew angles which do not capture the skew of the page as accurately as the Hough transform.

5.6 Using features

When determining skew using the Hough transform, the angle corresponding to the cell in the accumulator that contains the greatest value is often used. However, more accurate results may be obtained with different features. Each feature generates a value for the skew and these values are compared with the manually determined skew angles using the maximum absolute error (MAE), the root mean squared error (RMS), and the correlation coefficient (CC) are shown in Table 5.13.

Features that can be easily derived from the accumulator include:

(F_1) Sum of the accumulator values for each column

(F_2) Sum of the squares of the accumulator values for each column

(F_3) Sum of the accumulator edges (differences between vertically adjacent cells) for each column

(F_4) Sum of the squares of the accumulator edges for each column

($F_{5...9}$) The sum of the greatest 1, 3, 10, 50 and 200 values in each column

Image	New Skew	Hough skew	Known	Hough
A006	−0.25	−0.2	66926	66904
A034	4.21	4.7	72694	71538
A03J	7.19	9.1	58041	53649
A04D	7.78	8.4	92677	89770
A05E	6.80	7.2	71082	68470
C03I	3.46	3.8	55066	54534
C048	5.31	5.6	47720	46855
D03D	7.25	7.5	51813	49332
D03M	0.00	0.3	20242	20244
D048	9.74	10.1	37185	35167
D04G	1.81	2.0	62934	62645
D05N	−0.19	0.0	28386	28399
D06C	−4.13	−3.9	44042	41133
D06N	−6.32	−6.1	63283	61673
E009	9.38	9.9	35406	34730
E00H	−3.32	−3.0	45105	44975
E00M	2.36	2.6	48409	47980
E01J	−2.25	−2.0	83595	83535
E037	−2.73	−2.5	86085	85624
E04I	9.16	9.6	53996	48699
H00C	4.12	4.3	19377	18959
H019	−5.68	−5.3	46979	46651
H041	−9.49	−9.1	67454	58313
H04D	2.96	3.1	32384	32020
H04F	−9.05	−8.7	79904	78032
I00J	−3.82	−3.5	80240	78696
I00L	5.11	5.4	84976	83320
I037	−6.61	−6.4	52930	51652
J03M	3.55	3.8	57566	57083
J04A	−3.72	−3.5	51026	50333
J04K	−2.61	−2.4	77554	76293
K009	−6.61	−6.6	52038	51078
N01A	−10.36	−10.2	24642	24628
N027	−5.14	−4.9	20868	20807
N02A	−9.39	−9.0	8772	8319
N02K	6.33	6.9	63978	63879
N03K	7.50	8.1	30398	30013
N048	5.17	5.8	41067	40570
N05I	3.06	3.6	47074	46844
S021	−1.80	−1.3	57596	56983
S03G	5.28	5.8	68362	66942
S03R	−5.26	−4.9	44319	42829
S044	7.16	7.7	58602	58433
S047	−9.90	−10.0	88172	84819
S048	−0.41	−0.1	72042	68927
S04H	7.04	7.5	77316	75072
V004	−2.60	−2.3	70474	70122
V00C	−8.76	−8.7	60111	59797
V00G	−6.29	−6.4	71118	70661
V00N	4.21	4.3	67516	67011
Mean			55991	54699

Table 5.11: Using the predicted skew angle for compression

Known	Hough
55991	54699
	$\surd\,(t = 5.54)$

Table 5.12: Comparing compression using known versus predicted skew angles

($F_{10...14}$) The sum of the smallest 1, 3, 10, 50 and 200 values in each column

(F_{15}) The sum of the power spectrum coefficients for each column

(F_{16}) The sum of the power spectrum coefficients using edges for each column

Table 5.13 shows that the usual choice of using the maximum point in the accumulator array, F_5, gives a good result. It is only marginally improved upon by the power spectrum measure F_{15}, which has a much greater computational cost. Features $F_{10}...F_{14}$ give a large correlation coefficient as they chose the skew to be -45 degrees. There is a column of values that are all approximately 0 and a column of values all approximately -45, we can see that differences in this magnitude are highly correlated. The use of the top 200 values (roughly 5% of the points in the column), Feature F_9, gave a slightly larger (better) correlation coefficient, but a slightly larger (worse) RMS and MAE error measures.

5.7 Offset encoding

Although we are using a simple (xd, yd) encoding of component offsets, the methods above that calculate document skew can be incorporated into any offset encoding method.

Howard [How96] and Zhang [Zha97] present updated methods for encoding symbol offsets. Howard's method involves keeping a moving average of the baseline position of the characters in a line. The positions of new characters are encoded with respect to this moving average. This technique avoids the incremental double handling of positional errors.

Feature	MAE	RMS	CC
F_1	0.35	0.43	0.81
F_2	0.34	0.43	0.80
F_3	13.96	20.55	0.30
F_4	0.35	0.45	0.79
F_5	0.34	0.43	0.80
F_6	0.32	0.42	0.78
F_7	0.34	0.43	0.80
F_8	0.35	0.44	0.80
F_9	0.36	0.44	0.82
F_{10}	45.13	45.14	1.00
F_{11}	45.13	45.14	1.00
F_{12}	45.13	45.14	1.00
F_{13}	45.13	45.14	1.00
F_{14}	45.13	45.14	1.00
F_{15}	0.34	0.43	0.80
F_{16}	0.35	0.45	0.79

Table 5.13: Each feature compared with the manually determined skew values, with their maximum absolute error (MAE), root mean squared (RMS) error and correlation coefficient (CC) measures

Zhang's method is more complex and involves the automatic determination of space position and counting the number of words. This method effectively implements a paragraph layout algorithm that would more commonly reside inside a word processor. Zhang's technique justifies the words automatically and encodes the differences between the predicted position and the true position of the words and components. The system updates itself to incrementally refine its prediction as each component is encoded.

Both of these mechanisms are improved when the skew angle is known. In our implementation the coordinates of each component are rotated to straighten each line, but the skew angle is not incorporated into the prediction of the next character's position. Calculating the skew angle accurately has one clear advantage—there are improvements due to a more accurate determination of the reading order for multiple columns and complex documents.

5.8 Summary

This chapter has shown that the Hough transform is an accurate method of skew determination. Incrementing the accumulator by an amount that is proportional to the logarithm of the component's area increases the accuracy of the transform. To decrease the computational cost, the transform can be calculated progressively, quickly focusing on the true skew angle. The method is shown to be robust and accurate at resolutions down to 37.5 dpi. Below this resolution the results are extremely poor.

Skewing pages by even a small amount can dramatically impact on the compressed file size. Using the Hough transform to determine the skew of rotated images gives better compression results than manually measured skew values: compressed file size can be reduced by 3% as opposed to 0.7%.

The common method for choosing the representative skew angle from the Hough accumulator, finding the greatest value, works well, although it can be improved slightly by using the top 200 (5%) of the values or by using the power spectrum measure.

Chapter 6

Template matching

A critical requirement in a document image compression system is the ability to match components accurately. This is actually a process of pattern comparison, and is often referred to as *template matching*. Template matching in document image compression differs from the matching required in an optical character recognition (OCR) system because there is no need to identify the actual character to which any particular component corresponds. Nevertheless, it is important to match each new occurrence of a component consistently with the class to which it belongs, because compression deteriorates if the matching operation introduces errors. Although the characters are encoded without loss, when there are large errors between a component and its match, the difference requires more bits to encode.

Matching procedures generally work by aligning the two images in the position where it is assumed that the best match will be made, and then examining an *error map*, which is defined as the bitwise exclusive-OR of the images. The most frequently used method for template matching counts the non-zero pixels in the error map and expresses this number as a percentage of the total area. Howard [How96] uses 21% as a threshold: if two components have an error of less than or equal to this value then they are assumed to match. When searching for a match in the codebook, the search process need not stop after finding the first match. A better result may be obtained by searching further and seeking a closer match. Matches can be ranked by the percentage returned by the XOR function, which reflects the

quality of the match. For some other methods, the result is binary: either *match* or *differ*, and such methods cannot be used to find a closer match.

This chapter compares a collection of well known template matching methods and evaluates their matching accuracy and performance in our compression system. Because each method has one or two parameters that are used to determine the match, the machine learning scheme C5.0 is used to automatically optimise each of the thresholds. This allows the methods to be compared fairly, each parameter being chosen to give the best template matching results on the training data. While the best methods, by definition, give the best matching accuracy, we will see—surprisingly—that this does not necessarily yield the best compression performance. C5.0 was used instead of Naïve Bayes in these experiments, because the decision trees can easily be transformed into native code and embedded in the document image compression system.

To illustrate how varying a threshold for template matching affects compression, Figure 6.1 shows the classification accuracy on a database of characters, the compressed file size and the number of classes, as the value of the XOR threshold is varied. Each data point is averaged over the fifty test images. We can see how classification accuracy begins at the baseline of 83.3% and peaks at around 88%. With further increases in the threshold it quickly decreases to 100% minus the baseline accuracy. The compressed file size follows a more interesting pattern. Although compression is quite good around the 17% threshold, the actual optimum occurs when the threshold is 10%, the mean file size being around 4% lower. The bottom line in the figure shows the number of equivalence classes formed during compression. Ideally, the number of equivalence classes corresponds to the number of unique characters and symbols on the page. When the threshold is very low, few characters match, so a large number of classes are generated. In contrast, when the threshold is high, most characters match and there are few equivalence classes. At a threshold of 100% we might expect a single equivalence class, but the components are not matched if their sizes

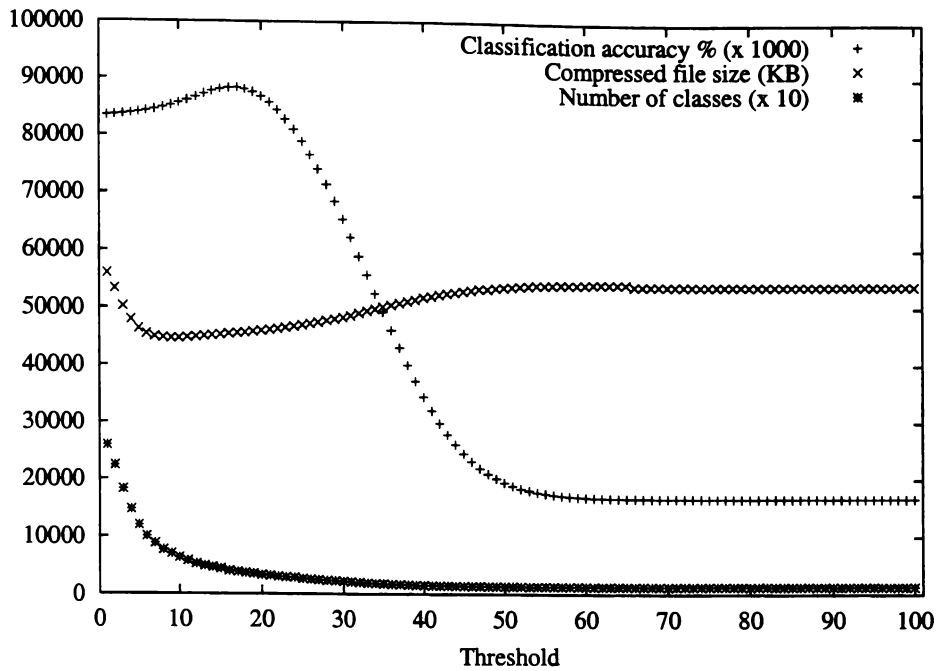


Figure 6.1: How the XOR threshold affects classification accuracy, compressed file size and the number of classes when averaged over the fifty test images

are disparate (see Chapter 2.4.7). This is why the number of classes does not decrease when the threshold increases over 50%—when the threshold is high the number of classes corresponds to the various sizes of the characters, not their content. The number of classes must be combined with the classification accuracy to visualize the contents of the codebook. After the peak of 17% the number of errors in each class grows quickly.

Five template matching methods are introduced in Section 6.1, one being subsumed by another. We introduce a template matching method that does not require an error map and is based on calculating the entropy of one component with respect to another. Our entropy based method is the fifth method. These methods are merged into a sixth that encompasses all of the features of the others. Section 6.4 introduces the character database used to train the machine learning scheme which determines the thresholds used for each template matching method.

Thresholds for each method are determined in Section 6.5 using the machine learning scheme C5.0. The classification accuracy and false positive rate for each template matching method is shown. This section shows that the CSIS method [Hol88] is the worst for individual character comparisons, while the best results are given by the new method that combines the decisions of the other five.

Lossy compression systems using each method are evaluated in Section 6.6, where the differences between the component and its corresponding match are not encoded. When ranked according to performance, this produces the same list as in Section 6.5. Again the best result is achieved by combining the matching methods, and this method also formed the smallest number of equivalence classes.

Section 6.7 uses each template matching method for lossless compression. Unlike the two previous sections, the best result is obtained by the CSIS matching method and the worst by the combined method. These interesting findings are discussed further.

6.1 Template matching methods

Following work by Inglis and Witten [IW94], we divide methods for template matching into two categories: local and global matching. Local methods perform analysis on a small section of the error map (or sometimes the actual image), and return a boolean *match* or *differ* result. Global methods examine the whole error map and generate a metric corresponding to the quality of the match. While local methods can only be used to make a single judgement, global ones can rank a series of components, making it possible to find the best match.

Template matching was first applied to compression by Ascher and Nagy [AN74], who used the XOR method to determine whether components matched or not. This was extended by Pratt *et al.* [PCC+80], with a system that weights error pixels by the number of their

neighbours in the error map. This has the effect of decreasing the likelihood of a match when errors occur in large clusters. A match is found if the weighted sum exceeds a predetermined threshold. This method is referred to as the *weighted XOR* (WXOR) method.

By distinguishing black-to-white errors from white-to-black ones, Holt and Xydeas [HX86] further improve accuracy. In this method referred to as the *weighted and-not* (WAN) method, two weighted error maps are created and the total weight of each one is combined to form a single value.

The *pattern matching and substitution* (PMS) method of Johnsen *et al.* [JSC83] rejects a match between components if a cell in the error map has two or more neighbours that are set, at least two of which are not connected to each other. A further heuristic attempts to detect matching errors when the corresponding pixel in either of the original bitmaps is completely surrounded by pixels of the same colour. Because the PMS method is resolution dependent, Holt *et al.* [Hol88] introduced the CSIS (combined size-independent strategy) method which alters the threshold for accepting or rejecting the matches depending on the width and height of the components. When the width or height of the component is twelve pixels or less the PMS heuristics are used. If the component is larger, a match is rejected if there is a pixel in the error map with four of more neighbours. As the CSIS method includes the rules of the PMS method, the latter is not evaluated in our experiments. To allow the closest matches to be determined when using the CSIS method, if a match succeeds, the number of pixels in the error map is returned.

The *clairvoyant template matching* (CTM) method is presented in the next section and the combined system that incorporates the other methods is presented in Section 6.3.

6.2 Clairvoyant template matching

Following Mohiuddin *et al.* [MRA84, Moh82], the amount of uncertainty or entropy between components is used as the criterion for the matching process. Given a component M consisting of binary pixels M_{ij}, the conditional information in M given the dictionary pattern L is defined as

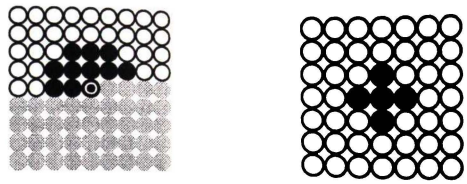
$$I(M|L) = -\log_2 P(M|L) = -\sum_i \sum_j \log_2 P(M_{ij}|L)$$

The pattern M is accepted as a match to that class L which minimizes $I(M|L)$, so long as $I(M|L) < T_L$, where T_L is a pre-defined threshold for acceptance to pattern class L.

In order to employ this method of classification it is necessary to estimate the probabilities $P(M_{ij}|L)$. The aim is to find, for each unknown component M, the dictionary image L_k that best models it—meaning that the amount of information required to specify M given L_k is minimised. This distance metric is called the *cross-entropy* between pairs of images, and can be approximated by compressing one component relative to another. The entropy model we use is the context-based compression model proposed by Langdon and Rissanen [LR81] and further developed by Moffat [Mof91].

6.2.1 Cross-entropy

To calculate the cross-entropy between two components, we use the pairwise compression of components introduced in Chapter 2. A clairvoyant context can be used to estimate the information content of each pixel, because it is not necessary to be able to compress and decompress the component. We use the same contexts as Inglis and Witten [IW94], and these are shown in Figure 6.2(a) and Figure 6.2(b). Larger contexts may be more suitable



(a) Conventional (b) Clairvoyant

Figure 6.2: Two different types of contexts

for larger components, or for higher digitisation resolutions.

It is normal to use an adaptive model, accumulating the counts as compression proceeds, otherwise the model must somehow be transmitted to the decoder. However, when obtaining an entropy figure there is no need to transmit the model, so a static one can be used. Like the clairvoyant context, this provides a lower bound on the amount of compression that could possibly be achieved in practice. Informal experiments indicated that the static model gives a slight performance increase over an adaptive model.

The compression-based template matching procedure begins by building the model of one component with respect to the other. It would be much faster to calculate a single model and use it for all matches. However, this is found to degrade performance very seriously. Suppose the two components to be matched are L and M. For any particular pixel $L_{i,j}$ in the first, there is a corresponding pixel $M_{i+\delta x,j+\delta y}$ in the second, where $(\delta x, \delta y)$ is the displacement due to registration. Figure 6.3 shows two different components with the 5-pixel clairvoyant context superimposed. The values of the five pixels under the context in the first component L, taken together, determine a particular context. The value of the central context pixel in the other component M gives a color, black or white, and the corresponding count for that context value is incremented. The process is repeated for every pixel to build a complete model. In fact, the components are assumed to be surrounded by white space. Once they have been registered, the maximum extent in both the horizontal and vertical directions is used as the area over which the entropy is calculated.

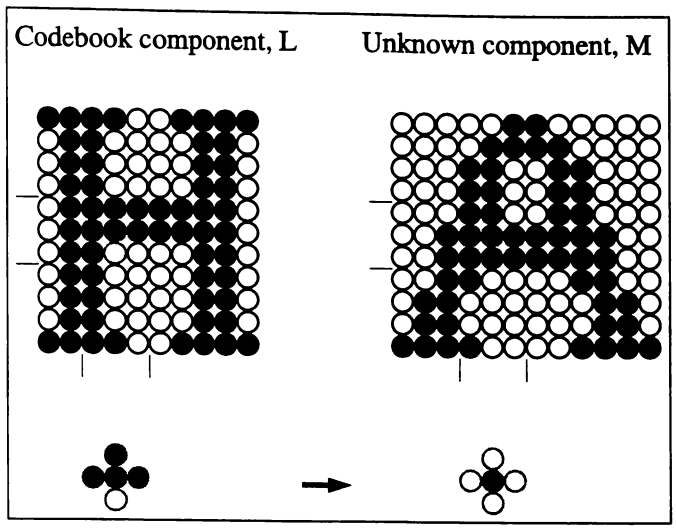


Figure 6.3: Using a five-pixel context to build a model with respect to another component

Once the model has been created, a second pass is taken over the components and the counts in the models are used to calculate the entropy of the second component M with respect to the first one L. The process is not symmetric; the entropy depends on whether the calculation is done for component M with respect to L, or for L with respect to M. Both are calculated and the maximum is taken as the final cross-entropy measure.

To determine the thresholds for acceptance, two features were generated. These are: the number of bits per pixel, B, and the total number of bits required to encode the component, T. The actual values of these two parameters are determined by the machine learning scheme in Section 6.5.

In the case where clairvoyant template matching (CTM) decides that two components match, the parameter T is returned instead of returning a binary decision. This enables a search to find the best match. In informal experiments, the use of T increases compression performance slightly, so this is included in our CTM implementation.

6.3 Combining multiple methods

If the template matching methods capture somewhat different features from the characters, it may be worth combining them together to form a new matching method. We do this using C5.0, but instead of analyzing the parameters individually for each method, all the parameters are combined into a decision tree where the internal nodes have binary decisions and the leaves assign a classification. The results of this combined tree are shown in Section 6.5.6, and the decision tree is shown in Figure 6.6.

6.4 Character database

In order to test the template matching methods, a large collection of characters is required, along with their correct classification. These characters are obtained from the Bell Labs Image Defect Model 0 [Bai93, UW93], a collection of 8,565,750 characters and their classifications. All of the characters are in the Computer Modern Roman font [Knu86]. Each of 94 different characters is generated at five type sizes: 4, 6, 8, 10 and 12 point. Each character is manipulated using ten parameters with a parameter space of 18225 different deformations of each character.

A subset of this database is used as training data, and the template matching methods are evaluated on the UW images. Only 8, 10 and 12 point characters are used, and the parameters are limited to: *skew* in $[-3, 0, 3]$, *yoff*=0, *blur* in $[0.5, 1, 1.5]$, *jitter*=0, *xoffset*=0, *sensitivity*=0, *threshold* in $[0.2, 0.25, 0.3, 0.35, 0.4]$, *xscale* in $[0.9, 1.0, 1.1]$, and *yscale*=0. In some cases the image is completely eroded, so that particular parameter combination is not used. For example, 305 8-point characters are eroded and therefore ignored, due to combinations of these parameters. The parameter values shown generate 135 deformations of each character. Three examples are shown in Figure 6.4. The training dataset contains

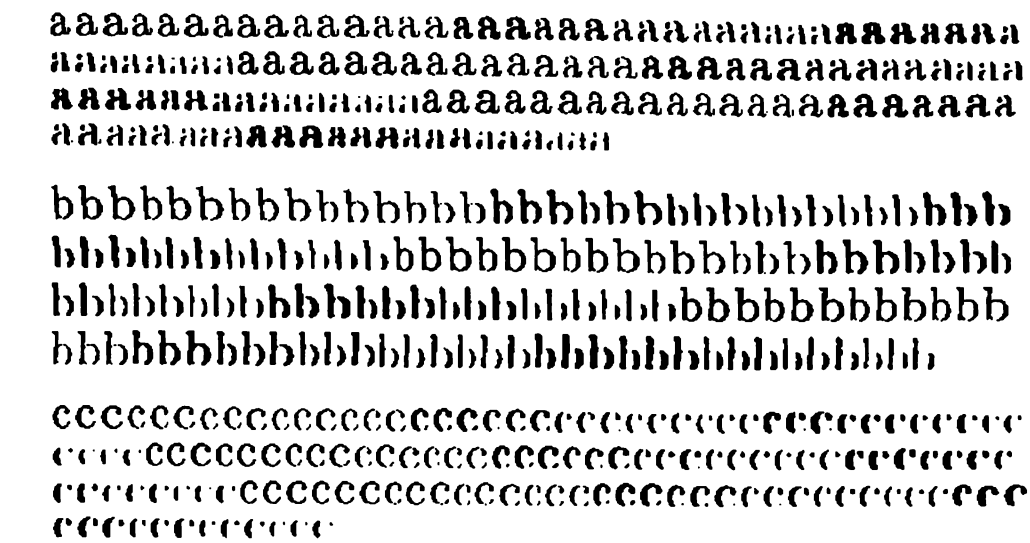


Figure 6.4: Demonstrating the 135 deformations that are applied to each character in the training data. This example shows (enlarged) 8-point characters.

12385 8-point characters, 12549 10-point characters and 12518 12-point characters to form a total of 37452 characters.

To use this data to train the machine learning schemes, pairs of characters are matched using the template matching methods from Section 6.1, giving features for each of the methods along with the classification of either *match* or *differ*. The data is generated by matching each character with a single randomly chosen example of the same letter, along with five randomly chosen examples which are different letters. For example, if a particular character is a *t*, it is matched with another *t* (the classification is *match*) and with five other characters that are not *t*'s (the classification is *differ*). In this way, the 37452 characters are converted into a training file containing 224712 instances.

The template matching methods XOR, WXOR, WAN, and CSIS have a single parameter. This value is combined with a threshold to make a single classification. The CTM method generates two values, B and T. When the classification feature is combined with these, the training file contains a total of six features generated for each instance.

Classified as →	*Match*	*Differ*
Match	22267	15185
Differ	10944	176316
Accuracy	88.4%	
False positives	4.9%	
Threshold	$\leq 17\%$	

Table 6.1: Confusion matrix for XOR

6.5 Classification results

For each of the template matching methods, the corresponding parameter (or parameters for CTM) is analyzed by C5.0 in order to determine the threshold that has the highest classification accuracy. For the schemes that have a single parameter (XOR, WXOR, WAN and CSIS), C5.0 used the default settings. To simplify the decision tree generated for CTM, and reduce the risk of over-training, the minimum leaf size was set to 200 instances. This means that small branches in the decision tree that might capture a few instances are ignored.

It is interesting to note that when the difference in width and height between two components is included as a feature in the training file, C5.0 generates this simple rule: if the difference in width or height is greater than 2 pixels then the result is *differ*. This is the screening policy previously adopted by Witten *et al.* [WBE+94] and Howard [How96], and is the policy adopted in Section 2.4.7.

For the combined method, C5.0 used the entire dataset with the same minimum leaf size as the CTM method. The results for each of the methods follow.

6.5.1 XOR method

Best results for the XOR method are achieved when the matching threshold is set to 17%, and are shown in Table 6.1. For this threshold 22267 component pairs are correctly classi-

Classified as →	*Match*	*Differ*
Match	22478	14974
Differ	7854	179406
Accuracy	89.8%	
False positives	3.4%	
Threshold	$\leq$ 58%	

Table 6.2: Confusion matrix for WXOR

fied as *match*, and 176316 components are correctly classified as *differ*. The errors that are made by the XOR method include 10944 pairs of characters that are matched when they should not have been and 15185 that are not matched when they should have been. The total accuracy for the XOR method is 88.4%, with a 4.9% false positive rate.

6.5.2 WXOR method

Best results for the WXOR method are achieved when the matching threshold is set to 58%, and are shown in Table 6.2. For this threshold 22478 component pairs are correctly classified as *match*, and 179406 components are correctly classified as *differ*. The errors include 7854 pairs of characters that are matched when they should not have been and 14974 that are not matched when they should have been. The total accuracy for the WXOR method is 89.8%, with a 3.4% false positive rate.

The accuracy rate for the WXOR method is slightly higher than for the XOR method, and on examination of some of the errors made, it appears that WXOR successfully rejects components with large clusters of pixels in the error map.

6.5.3 WAN method

Best results for the WAN method are achieved when the matching threshold is set to 76%, and are shown in Table 6.3. This may seem like a high percentage, but it stems from the

Classified as →	*Match*	*Differ*
Match	28746	8706
Differ	7561	179699
Accuracy	92.8%	
False positives	3.4%	
Threshold	≤ 76%	

Table 6.3: Confusion matrix for WAN

Classified as →	*Match*	*Differ*
Match	6649	30803
Differ	2821	186978
Accuracy	86.2%	
False positives	1.2%	
Threshold	–	

Table 6.4: Confusion matrix for CSIS

fact that two error maps are being added and the total area has not been doubled. For this threshold of 76%, 28746 component pairs are correctly classified as *match*, and 179699 components are correctly classified as *differ*. The errors that are made by the WAN method include 7561 pairs of characters that are matched when they should not have been and 8706 that are not matched when they should have been. The total accuracy for the WAN method is quite high at 92.8%, with a 3.4% false positive rate.

Visually, the WAN method does a good job. Some of the errors that were made were hard to identify manually.

6.5.4 CSIS method

The results from C5.0 for the CSIS method are shown in Table 6.4. The CSIS method is a local method and does not need a threshold to determine whether two characters match. For this method, 6649 component pairs are correctly classified as *match*, and 186978 components are correctly classified as *differ*. The errors that are made by the CSIS method include

Classified as →	*Match*	*Differ*
Match	24697	12755
Differ	5176	182084
Accuracy	92.1%	
False positives	2.3%	
Threshold	*see Figure 6.5*	

Table 6.5: Confusion matrix for CTM

2821 pairs of characters that are matched when they should not have been and 30803 that are not matched when they should have been. The total accuracy for the CSIS method is 86.2% with a 1.2% false positive rate.

The error rate of the CSIS method is the lowest of any of the methods, as it is the most conservative with matches. The total accuracy is low, but the false positive rate is the best of any of the methods.

6.5.5 CTM method

The results from C5.0 for the CTM method are shown in Table 6.5. Unlike the previous methods, CTM calculates two features, and Figure 6.5 shows how they are combined in a decision tree to determine the final decision. Using this decision tree, 24697 component pairs are correctly classified as *match*, and 182084 components are correctly classified as *differ*. The errors that are made by the CTM method include 5176 pairs of characters that are matched when they should not have been and 12755 that are not matched when they should have been. The total accuracy for the method is 92.1%, with a false positive rate of 2.3%.

The CTM method compares well with the WAN method, which has a slightly better accuracy, but a worse false positive rate.

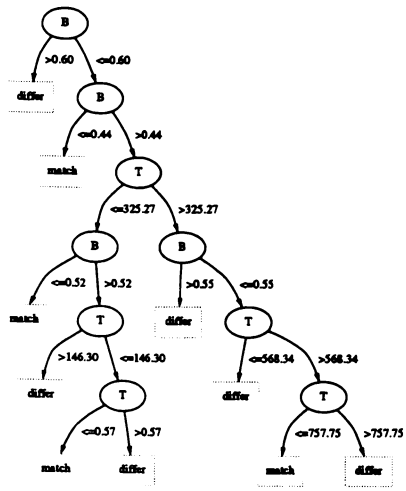


Figure 6.5: Decision tree generated for CTM

Classified as →	*Match*	*Differ*
Match	30497	6955
Differ	5046	182214
Accuracy	94.7%	
False positives	2.2%	
Threshold	*see Figure 6.6*	

Table 6.6: Confusion matrix for the combined method

6.5.6 Combined method

The results from C5.0 for the combined method are shown in Table 6.6. Unlike the previous methods, the combined method has six features, one for each of the first four methods and two for the CTM. Figure 6.6 shows how the features are combined in a tree to determine the final decision. Using this decision tree, 30497 component pairs are correctly classified as *match*, and 182214 are correctly classified as *differ*. The errors that are made by the combined method include 5046 pairs of characters are matched when they should not have been and 6955 that are not matched when they should have been. The total accuracy is 94.7%, with a false positive rate of 2.2%.

The combined method has the highest accuracy of the template matching methods pre-

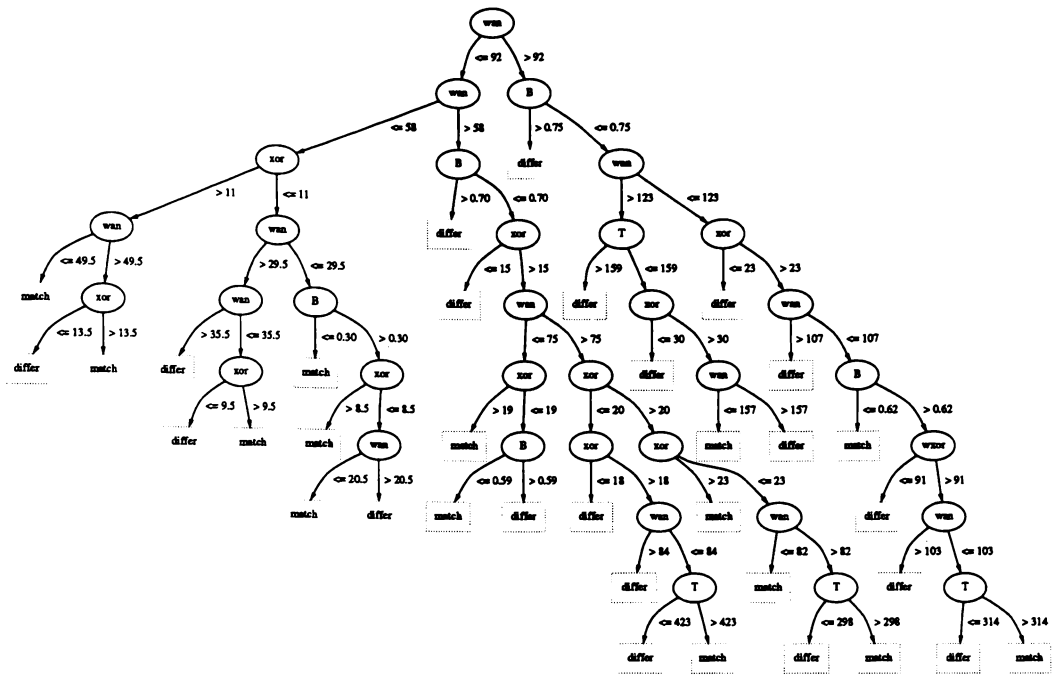


Figure 6.6: Decision tree generated using the combined features

sented, and a good false positive rate.

6.5.7 Discussion

The results for each of the classification methods are given with respect to the training data. By itself, the classification accuracy on the training data is not a reliable indicator of future performance, so the fifty test images are compressed with each of these methods to evaluate their performance. Sections 6.6 and 6.7 show the results.

The decision to include five negative examples for each positive example in the training data will affect the false positive rate, but the methods may be fairly compared as the same training data is used for each.

6.6 Results applied to lossy compression

The results from Section 6.5 show the classification accuracy as applied to the training character data. Now the thresholds that have been determined by C5.0 are used to compress the fifty test images. In this section images are compressed in a lossy mode, so that the differences between the component and the matched component in the codebook are not encoded. We expect the best template matching result of the previous section to give the best compression and the smallest codebook.

Table 6.7 shows the results for each of the methods; their performance ranks in the same order as their previously determined template matching accuracy. The combined method has a mean file size of 20577, which is 9% lower than the mean file size for the XOR method of 22479. Table 6.8 shows that the combined template method has a significantly smaller file size than any other method. Table 6.9 shows that the combined method also produces a significantly smaller codebook than each of the other five methods.

Evaluating the results using lossy compression confirms that the training data generalizes to the testing images, and a subjective visual inspection of the several of the lossy images produced showed that the combined images looked superior. No substitution errors were obvious, although the apparent density (or thickness) of some characters waxed and waned throughout the lines.

6.7 Results applied to lossless compression

Each of the template matching methods is now used, with the thresholds determined in Section 6.5, as a component of the lossless document image compression process.

Table 6.10 shows the lossless compression results for each method. Compared with Sec-

Image	XOR	WXOR	WAN	CSIS	CTM	Combined
A006	22780	21928	21525	23775	21570	21202
A034	42785	41947	41525	45374	41180	40549
A03J	31069	29374	28582	31283	28808	28011
A04D	44877	42919	41902	44724	42448	41294
A05E	29449	28500	28057	31790	28264	27761
C03I	28433	26365	25661	27760	25817	24839
C048	14894	14234	13844	15212	13914	13477
D03D	37456	36579	36085	37304	36317	35902
D03M	14866	14549	14411	15299	14444	14411
D048	9763	9228	9180	10518	9018	8976
D04G	35375	34138	33631	36462	33811	32917
D05N	14926	14295	14186	15128	14146	14020
D06C	15550	15267	14813	16369	14995	14670
D06N	29521	28789	28350	31556	28348	27821
E009	14033	13415	13022	14165	12912	12563
E00H	15680	14977	14402	16282	14549	14074
E00M	13755	13379	13136	14557	13027	12881
E01J	37672	35646	35132	37227	34670	33873
E037	56256	52599	51416	53713	51753	50468
E04I	30613	28854	28105	29658	28339	27604
H00C	9147	8671	8439	9839	8522	8175
H019	15330	14257	13891	15706	13788	13381
H041	31119	29886	29403	30399	29148	28805
H04D	11686	11161	10911	11924	10899	10747
H04F	17133	16133	15596	18366	15440	15231
I00J	38459	37616	36808	44742	37320	36389
I00L	35166	34457	33379	40922	34094	32852
I037	22979	22645	22384	24271	23242	22884
J03M	21770	20507	19942	23463	20035	19459
J04A	34454	32266	31196	32877	31407	30533
J04K	43005	40981	40535	42030	40677	39827
K009	20317	19677	19224	20933	19189	18795
N01A	5177	4792	4644	8090	4659	4592
N027	6017	5315	5079	9610	4978	4844
N02A	3200	2897	2656	4894	2612	2519
N02K	43215	41587	41341	43368	40370	39835
N03K	7166	6731	6472	9899	6478	6310
N048	8725	8427	8075	14686	8444	7931
N05I	9053	8928	8583	14555	8879	8696
S021	16764	15540	15040	17484	15138	14758
S03G	29844	28872	28119	31498	28313	27503
S03R	13314	12352	11885	14552	11830	11425
S044	11547	11066	10673	12416	10781	10578
S047	22610	21688	21116	24709	21304	20171
S048	18385	17771	17446	20420	17705	17243
S04H	16143	15322	14900	16924	14906	14630
V004	25526	23148	24205	23583	21400	20453
V00C	11951	11520	11306	12718	11930	11511
V00G	15633	14848	14584	16230	14540	14371
V00N	19372	18459	17689	20784	17753	17078
Mean	22479	21490	21050	23601	21082	20577

Table 6.7: Lossy file sizes for each of the matching methods

Combined	XOR	WXOR	WAN	CSIS	CTM
20577	22479	21490	21050	23601	21082
	$\surd(t = 11.13)$	$\surd(t = 11.07)$	$\surd(t = 5.67)$	$\surd(t = 13.06)$	$\surd(t = 10.71)$

Table 6.8: Average file size for lossy compression

Combined	XOR	WXOR	WAN	CSIS	CTM
234	395	284	252	385	265
	$\surd(t = 9.62)$	$\surd(t = 10.49)$	$\surd(t = 9.65)$	$\surd(t = 14.05)$	$\surd(t = 9.45)$

Table 6.9: Average number of equivalence classes for lossy compression

tion 6.6, the ranked order of each of the methods has almost been reversed! The CSIS template method has the lowest mean file size of 45490, while the combined method has a mean size file of 48574, almost 7% higher. Table 6.11 shows that CSIS is significantly better than all other methods with the exception of WXOR. The WXOR method has a mean file size of 45778 which is less than 1% higher. Although the XOR method has a very similar mean file size of 45807, the variation is not as high and the CSIS method is consistently better.

This surprising result, that the performance ranking for the methods is almost completely reversed between lossy and lossless compression, deserves some further discussion. The differences between lossy and lossless compression are quite minor. For lossless compression, the component is encoded with respect to the matched component in the codebook. At this stage we are comparing against an average representative of each equivalence class. There are two possibilities that may explain this behaviour. First, as the accuracy of the template matching methods increase, the differences between the characters within an equivalence class differ enough to affect the *average* character. Second, the process of encoding a component with respect to another component is biased towards the types of errors made by different template matching methods.

Figure 6.12 investigates the first possibility by performing matches against all characters to

find the best match, rather than matching against the representative component of each class. The details of this matching method are discussed in Section 7.6. This gives similar results to Figure 6.11, which leads us to believe that the bias of the component compression process outweighs the benefits of increased template matching accuracy. Recall from Chapter 2 that the component is compressed using a combined context which includes a 4-pixel context from the component to be compressed and a 7-pixel clairvoyant context from the matched component. The CSIS method clusters errors into small rectangular regions, often 2 pixels by 2 pixels in size. Errors of this size fit well within a context arrangement of this type, while the larger errors generated by the CTM method do not necessarily conform to such strict requirements. This explains why there is no significant difference between the CSIS and WXOR schemes, because the WXOR method also penalises large clusters of errors.

6.8 Summary

This chapter introduces a division between training the template matching methods and their evaluation on real-world images. The effect that the matching threshold has on matching accuracy, file size and codebook size is demonstrated and we show that appreciably different results are obtained depending on the value of the threshold. Six template matching methods are compared, their thresholds determined automatically using the machine learning scheme C5.0.

We develop a new template matching method based on cross-entropy, and introduce a new method for matching that combines the results of all the methods. This combined method has the highest template matching accuracy and gives significant improvements when applied to lossy compression.

When the template matching methods are applied in a lossless compression system, however, the best matching method does not guarantee the best compression: the CSIS method

Image	XOR	WXOR	WAN	CSIS	CTM	Combined
A006	64723	65327	65425	64043	65315	66781
A034	62211	53159	62409	61941	61564	62924
A03J	48076	45857	48620	47649	49190	50141
A04D	75938	67849	77227	75658	77307	78949
A05E	58255	58307	58679	58016	59148	60768
C03I	47840	48575	48815	47794	48680	51080
C048	37529	37682	38477	37221	39225	40653
D03D	43420	43559	43603	43402	43456	43492
D03M	19748	19796	19842	19754	19643	19879
D048	25915	26295	26475	25860	26539	29362
D04G	57247	57642	57832	57176	58571	59466
D05N	28001	28059	28147	27891	28599	30029
D06C	35671	35176	36024	34910	35819	37121
D06N	49942	50153	50486	49792	50440	52187
E009	26469	26599	26921	26209	27038	27787
E00H	36995	37482	37806	36671	38744	40316
E00M	40233	40357	40543	39449	40769	42202
E01J	71876	73244	73558	71516	75243	76829
E037	77577	78971	79022	77792	79781	81830
E04I	42434	43069	43130	42547	43195	43777
H00C	15847	15844	16040	15793	15907	16540
H019	36516	36850	37376	36046	39687	41676
H041	42498	42774	42793	42403	42643	43081
H04D	27182	27442	27558	26989	28446	28866
H04F	57355	58148	58378	56766	60482	62589
I00J	70325	70265	70508	70551	71406	74692
I00L	72648	72460	72966	72271	73038	76796
I037	42266	42606	42865	41456	42765	43453
J03M	49752	50070	50536	49394	50488	52366
J04A	45435	46003	46430	45473	45787	46882
J04K	67449	68739	68643	67541	68572	70466
K009	39184	39282	39794	38518	40500	42386
N01A	18500	18407	18559	18685	18803	20371
N027	17247	17207	17241	17489	17320	17542
N02A	6457	6468	6494	6729	6444	6485
N02K	53619	54179	54504	54158	53309	54064
N03K	24798	24853	24864	24813	25039	25479
N048	34845	34756	35021	35096	35411	38830
N05I	40705	40703	40771	40737	40922	41170
S021	49546	49792	49928	48557	50739	53394
S03G	56163	56260	56607	54956	57695	59924
S03R	35208	35351	36384	34496	36612	38734
S044	44512	45003	45280	43801	45762	48857
S047	65409	65638	67147	64549	68759	72598
S048	64570	65329	65749	63353	67536	69524
S04H	58767	59348	59675	57583	61145	62566
V004	55316	58159	58405	54577	58065	61446
V00C	38601	38938	39311	38227	40835	42276
V00G	55001	55990	56109	54599	57784	59738
V00N	54537	54897	55575	53584	55976	60336
Mean	45807	45778	46491	45490	46923	48574

Table 6.10: Compressed file sizes for each of the matching methods

CSIS	XOR	WXOR	WAN	CTM	Combined
45490	45807	45778	46491	46923	48574
	$\surd\,(t = 5.27)$	$\times\,(t = 1.05)$	$\surd\,(t = 8.75)$	$\surd\,(t = 7.82)$	$\surd\,(t = 10.78)$

Table 6.11: Average file size for lossless compression

CSIS	XOR	WXOR	WAN	CTM	All
42442	43584	44070	43734	44678	44859
	$\times\,(t = 1.38)$	$\surd\,(t = 3.11)$	$\surd\,(t = 3.20)$	$\surd\,(t = 7.74)$	$\surd\,(t = 8.29)$

Table 6.12: Average file size for lossless compression using the best possible match

compresses significantly better than all others with the exception of WXOR. We have shown that the method of lossless encoding by compressing one component with respect to another is biased towards certain types of error that are present in the CSIS and WXOR methods, but not prevalent in the other template matching methods.

Chapter 7

Component codebook

The component codebook is the core of a document image compression system: it stores equivalence classes that represent each type of character in an image. A typical document page contains around 2000 characters and between 100 and 200 distinct characters.

Two separate methods for generating a component codebook have evolved since Ascher and Nagy first introduced a method for the compression of document images [AN74]. In their technique, a codebook of components was built as a pre-processing stage. Subsequently, each component was encoded with respect to its best match. We call this a *static* codebook. This procedure of building in a pre-processing stage was continued by Pratt *et al.* [PCC+80], Johnsen, Segen and Cash [JSC83], Holt and Xydeas [HX86], Holt [Hol88] and Witten *et al.* [WBH+92, WBE+94]. Our previous system, introduced in Witten, Moffat and Bell [WMB94] and extended in Inglis and Witten [IW95, IW96], used this approach. Most recently Mark and Shieber [MS94], Zhang and Danskin [ZD96], Zhang [Zha97], and Kia [Kia97] have continued using a static codebook. Because the codebook contains component equivalence classes, the value of the index for each component is correlated with the appearance of the character. When equivalence class information is available, it is possible to compress the index stream with a higher-order model such as PPM, which uses the index values of previous characters for context information.

An alternative approach was introduced by Mohiuddin, Rissanen and Arps [MRA84], who

built the codebook incrementally as components were compressed. As each component is processed it is compared against the codebook. If a match is found, the component is compressed *with respect to* the matched component, and then added to the codebook. This approach guaranteed lossless compression, and did not require an expensive building stage. This approach uses an *adaptive* codebook and was largely ignored until Howard [How96] reintroduced the basic process with refinements that allow lossy compression. Unlike static codebook generation, higher-order information is unavailable because the indices are encoded with respect to the best matching component, not the best equivalence class.

Previous adaptive techniques have not formed equivalence classes: instead they transmit index information that corresponds to the order in which components have been seen. The technique introduced in Chapter 2 is a novel method which extends the dynamic methods to include equivalence class information, so that the codebook index represents the *average* component in that equivalence class. Using this technique, it is possible to use methods such as PPM for encoding the symbols (see Section 4.4).

This chapter discusses issues that are involved when compressing multiple pages from the same source. Section 7.1 introduces a testing corpus from *Jefferson the Virginian* and describes how the file size and number of equivalence classes are affected as multiple pages are encoded.

Section 7.2 compares alternative contexts that are used when one component is compressed with respect to another. We show that there is an appreciable range in the compressed file sizes, and that compression can be improved by choosing an appropriate context.

Contexts that are used for black and white image compression are binary: each position in the context stores whether the pixel is black or white. Because of the small size of the characters being compressed, for a significant number of cases a portion of the context lies outside the image. Section 7.3 shows that compression can be significantly increased by

introducing a third state that represents whether the pixel is wholly inside the image.

Recall from Chapter 2 that a component can be encoded in one of two ways: individually or pairwise. Individual encoding uses context information from the JBIG1 10-pixel context and does not use a clairvoyant context. Pairwise encoding compresses one component with respect to another. In previous experiments, after pairwise encoding the component was used to update the probability information for the individual contexts. The individual counts are updated because it is assumed that valuable statistical information is otherwise lost. Section 7.4 shows that, in fact, no significant improvement is achieved by updating the individual context.

Section 7.5 compares two methods of aligning components for pairwise encoding. The current method aligns two components by their centroids. This section evaluates whether it is better to align the components based on their centres. When aligning components using centroids, the position of the centroid must be encoded before the component. Alternatively, when matching using centres, the position of the centre is determined using the width and height of the component. Although more information needs to be encoded when components are matched by their centroids, the method is shown to be significantly better.

When a component is matched against the codebook there are several matching strategies that can be used. Section 7.6 compares the current method of matching against the average component with three other methods; matching against the first equivalence class where a match is successful, the best single character overall, and the first matching single character overall.

[XXVIII]

An End and a Beginning

1783–1784

JEFFERSON played his final role in the drama of the American Revolution as a member of the Continental Congress; and his six months' service in that feeble body, ending with his foreign appointment soon after he became forty-one, constituted the closing scene of a notable legislative career. He afterwards presided over the Senate of the United States, but never again was he a member of a representative assembly or any other sort of deliberative body. So far as lawmaking was concerned this was the end. At the last this experienced performer played a stellar part, but the contemporary audience was small and apathetic, the cast as a whole was weak, and the setting was unimpressive in the extreme. During much of the time he operated in a vacuum, for Congress as a working body was often non-existent. At the outset he did not even know where he should go to find it.

When he accepted election as one of Virginia's delegates early in June, 1783, he supposed that in the autumn he would have quarters in Philadelphia, where he could find more kindred philosophical spirits than on any other American spot. He soon learned, however, that the discredited Congress of the Confederation had fled to Princeton, a pleasant but physically inadequate village and not then an intellectual center of wide renown. Nobody knew where the legislature would sit after that. A group of mutinous soldiers who were seeking back pay had precipitated the flight. The authorities of the Commonwealth of Pennsylvania had not provided the necessary safeguards, and there were those who viewed the departure of the delegates without regret. A disgruntled officer said: "The grand Sanhedrin of the Nation, with their solemnity and emptiness, have removed to Princeton, and left a state where their wisdom has long been questioned, their virtue suspected, and their dignity a jest." Even after Congress had been three months gone Dr. Benjamin Rush said that it was "abused, laughed at and cursed in every company." The derided body, being determined to preserve its independence and a modicum of dignity, would not accept the proffers of hospitality which numerous citizens of Philadelphia now held out.

(a)

On personal grounds Jefferson would have liked to serve there; he could have lodged pleasantly with Mrs. House and Mrs. Trist, as he had done briefly in the previous winter, and he could have remained near Patsy, whom he had decided to leave in the metropolis in any case. Father and daughter, riding in a two-horse phaeton, reached Philadelphia toward the end of October, and Jefferson went on to Princeton. Congress was still there, but on the day he arrived it adjourned to meet in Annapolis about three weeks later; so, after taking his seat, he came straight back. There was little time for relaxation at Princeton, but he managed to get a shave.

Back in Philadelphia after a meaningless trip, he wrote the Governor of Virginia about the future residence of this wandering Congress. In this matter he was admittedly a localist, though no more so than everybody else. He wanted the permanent seat of Congress to be as near his own State as possible, and he preferred Georgetown to Annapolis, hoping that the commercial benefits would accrue to the Potomac rather than Chesapeake Bay. He went on record at this early date as favoring a Potomac site, and before he ceased being a legislator he analyzed sectional opinions, estimated distances, and considered the whole problem with characteristic thoroughness and realism. He was fully informed on this subject long before he had any occasion to bargain with Alexander Hamilton.

Before leaving Philadelphia for Annapolis the last week in November he made more lasting arrangements for his daughter. It was probably on the suggestion of warm-hearted Elizabeth Trist, who unfortunately

(b)

was soon to leave town herself, that Mrs. Thomas Hopkinson, mother of Francis Hopkinson, took the eleven-year-old girl into her home. She was a widow and rather too old for Martha to regard her as a mother, but this deeply religious lady was obviously of the kindly sort. The circumstances facilitated the friendship of Jefferson with Francis Hopkinson, who was not only a fellow signer of the Declaration but also a musician and author of some note. Separated from her father, Patsy danced out the old year at the home of this gentleman with the young Hopkinsons and the children of David Rittenhouse, whose attainments in astronomy Jefferson so much admired, while her host played on the pianoforte.

This was a circle in which the mathematical and music-loving Virginian would himself have liked to linger, and he took comfort in the thought that Patsy would be more "improved" in it than she could have been at Annapolis with him. He had already planned a serious course of reading for her – not because of any advanced ideas about the education of girls and women generally but because of his anticipation of her future responsibilities. Writing to Marbois, who had kindly undertaken to find a French tutor for her, he said that he had to look forward to the family she would have someday. His comments were not complimentary to his countrymen. "The chance that in marriage she will draw a blockhead I calculate at about fourteen to one, and of course that the education of her family will probably rest on her own ideas and education without assistance. With the best poets and prosewriters I shall therefore combine a certain extent of reading in the graver sciences." The latter would have to wait until she rejoined him, however, and in the meantime he hoped that she would acquire in Philadelphia more taste and proficiency in the arts than she had been able to in the forest of remote Albemarle. Besides English and French, the rigorous schedule he worked out for her after he went to Annapolis called for three hours every day in music, and three every other day in dancing and drawing. However, the Frenchman who had been engaged to teach her drawing, and paid a guinea as a retainer, claimed that she had no capacity for the subject, and he could not be induced to continue. Jefferson insisted that the unwilling tutor keep the guinea for his pains.

He did not reproach Martha but he did continuously exhort her to

(c)

strive to be good and accomplished. He insisted on her writing regularly to him and to her aunts, warning her about her spelling – despite the inconsistency of his own. Following a suggestion from Mrs. Trist, he urged her to be specially careful of her personal appearance; he hoped that from the time of rising she would be so neatly dressed that no gentleman could discover a pin amiss. He was a moralistic parent, urging her to obey the inner monitor, the conscience God had implanted in every breast, but Patsy remembered his kindness best. Her father's wanderings were something of a trial to her then and thereafter, but she was happier in Philadelphia than Mrs. Trist had expected. She had inherited his sanguine temperament and she probably concluded that he had made the best arrangements for her a homeless widower could.

Madison, who had now become Jefferson's favorite political companion, journeyed to Annapolis with him but, being no longer a delegate, soon proceeded homeward, stopping on the way at Gunston Hall to find out how George Mason stood on various public questions of the moment. The latter's heterodoxy from the Madisonian and Jeffersonian point of view, which was strongly unionist, lay chiefly in his being "too little impressed with either the necessity or the proper means of preserving the Confederacy." A good many others were now heterodox, judging from the extreme dilatoriness of the delegates in getting to Annapolis. It was more than two weeks beyond the appointed date before enough of them were on hand for a meeting at the State House. Twenty were then present, including four Virginians, but only seven commonwealths had two or more delegates and thus it was impossible to attend to important matters of business, which required the vote of nine states. It was possible, however, to give an audience to the Commander in Chief, who wanted to lay down his sword.

Several weeks earlier Jefferson, who was deeply solicitous for the weary veteran, saw Washington for the first time in seven years, and noted with pleasure that the General had more health in his face than he had ever seen in it before. Jefferson was chairman of the committee that made arrangements for the public audience two days before Christmas, though the report on details of ceremony was largely in the hand of Elbridge Gerry. Congress gave "an elegant public dinner" the day before; some two hundred guests were plentifully supplied

(d)

Figure 7.1: Four of the twenty-one pages from *Jefferson the Virginian*

7.1 Compressing multiple pages

When compressing an image, the codebook contains a large number of the most common characters on a page. Although a fraction of the equivalence classes are singletons that only occur once, most classes are seen repeatedly. If multiple pages are compressed with a codebook that grows continuously between pages, it is expected that each new page can be stored more efficiently.

The performance of the document image compression process on multiple pages is evaluated on the last chapter of *Jefferson the Virginian*, which is the first volume of Dumas Malone's *Jefferson and his time* [Mal77]. The last chapter comprises twenty-one pages, each of which was scanned at 300 dpi. The first four pages are shown in Figure 7.1.

Figure 7.2 shows that when each of the images is compressed individually, the total file size for the corpus is 522552 bytes, or 24883 bytes per image. When the images are compressed sequentially, building on the codebook from the previous pages, the total file size is 464811 bytes, or 22134 bytes per image. By combining the images into a single compressed stream the saving is 11%.

A more dramatic result is seen in Figure 7.3, which shows how the number of equivalence classes grows as more images are processed. When the images are compressed individually the number of equivalence classes is the combined total of the twenty-one pages, or 2669 classes. Each page therefore contains an average of 127 equivalence classes. When the images are combined, only 590 equivalence classes are generated. The bottom line in the figure shows that once the first image had been processed, approximately the same number of new classes were added for each subsequent image. The exception to this is the last page of the Jefferson corpus, which contained about half the text of the other pages. A similar effect with words has been described by Zobel *et al.* [ZMWSD95], where they show the number of distinct words continues to increase and is best modelled as kN^{β}.

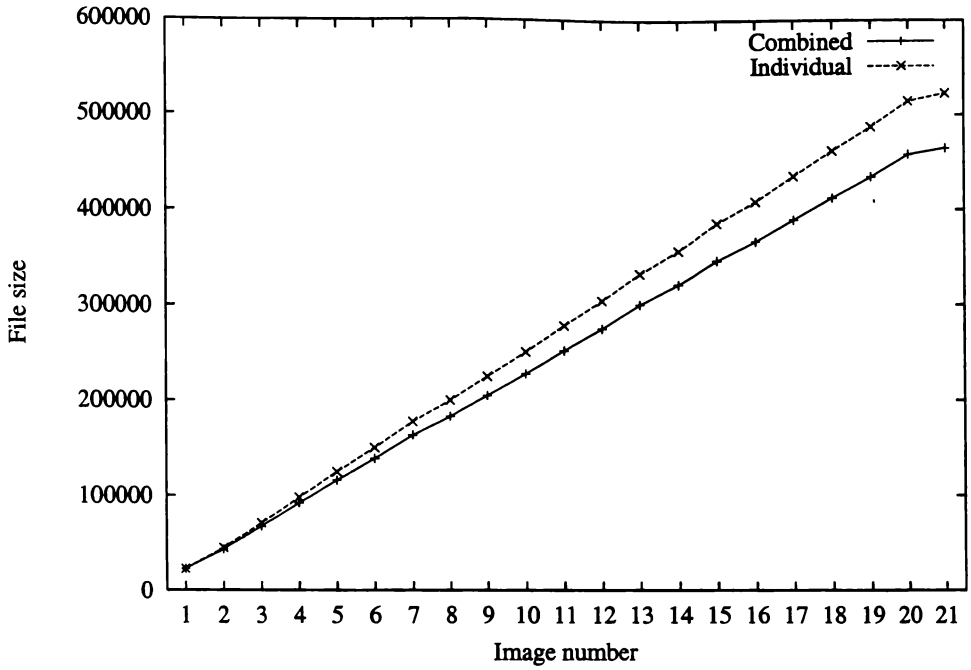


Figure 7.2: How compressing the pages individually or combined affects the cumulative file size

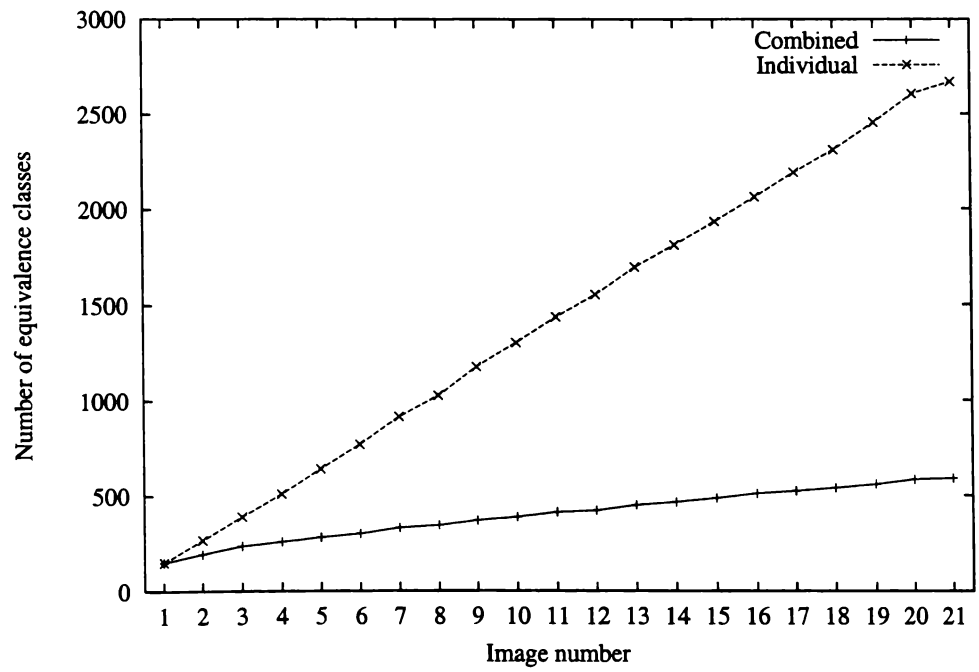


Figure 7.3: How compressing the pages individually or combined affects the cumulative number of equivalence classes

When compressing the images together, after the final image is encoded the codebook contains 41772 components, with the equivalence class that represents the letter *e* containing 2428 examples. It is highly unlikely that this many examples of the same letter are needed to accurately determine the average representative for the class. Figure 7.4 shows that when a limit is imposed on the number of components that can be stored in each equivalence class, the total combined file size is barely affected. Components are pruned from the equivalence class using a first-in-first-out method. Although the results for significance *t*-tests are not presented, the only significant difference occurred when the number of examples was limited to 40. The file size for the other values varied from the overall file size by less than 0.1%. Examining Figure 7.4 emphasises this point: there is barely a discernable difference between any of the parameter values. This result gives two benefits. First, the compression process is faster because average components need not be recalculated using a large number of examples. Second, the total storage requirements are significantly reduced.

The maximum number of equivalence classes that can exist in the codebook can also be limited. Figure 7.5 shows the compression results when the limit varies from 50 up to 500 in increments of 50. The final column shows the value for no limit. To limit the maximum number of classes, they are pruned using a least-recently-used method. Although significance tests show that each parameter choice is significantly worse than having no limit on the number of equivalence classes, the relative difference between a limit of 350 equivalence classes and the unbounded case is only 0.6%.

7.2 Contexts for component compression

This section compares the context that is used for encoding one component with respect to another, otherwise known as pairwise encoding. Pairwise encoding uses a context that is formed by merging contexts from the two images. Our previous results have used the 4/7

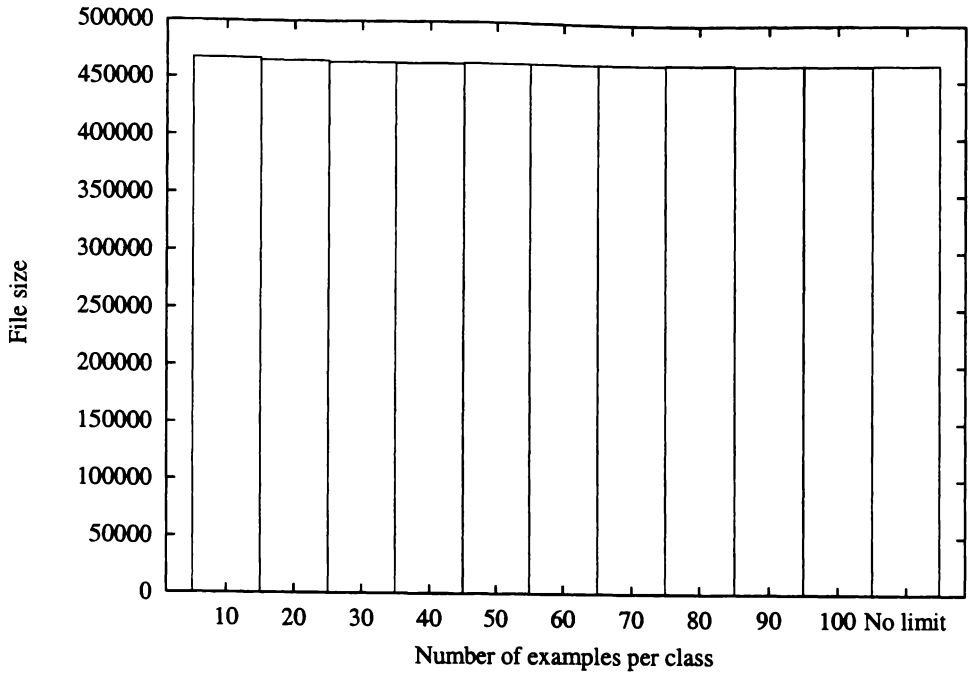


Figure 7.4: How limiting the maximum number of components in each equivalence class affects the total file size

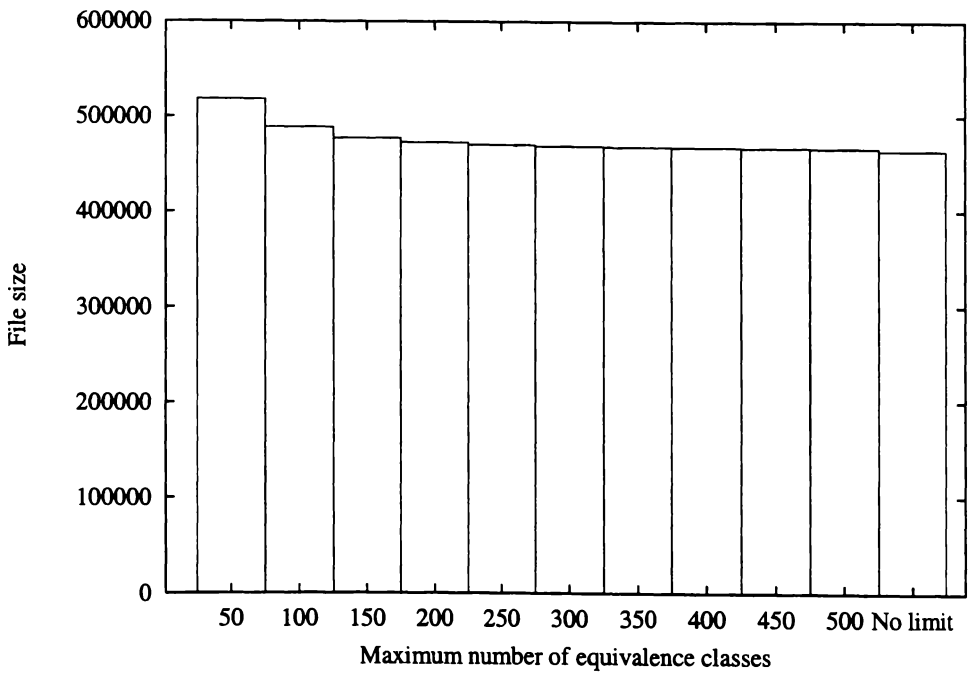


Figure 7.5: How limiting the maximum number of equivalence classes affects the total file size

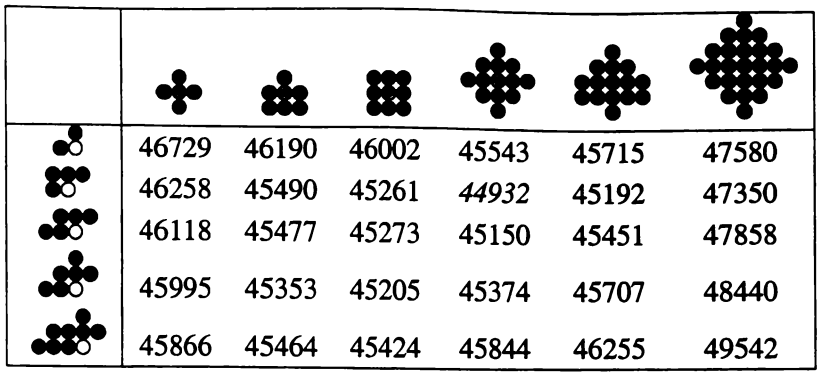
						
	46729	46190	46002	45543	45715	47580
	46258	45490	45261	*44932*	45192	47350
	46118	45477	45273	45150	45451	47858
	45995	45353	45205	45374	45707	48440
	45866	45464	45424	45844	46255	49542

Table 7.1: Using binary contexts for component compression

CSIS 4/7	Binary 4/13
45490	44932
	$\surd\,(t = 8.00)$

Table 7.2: Comparing the default context with the best context

pixel context used by Howard's SPM method [How96].

When encoding one component with respect to another, they are first aligned by their centroids, and two contexts are created for each pixel in the component to be encoded. The first consists of 4 pixels in the component to be encoded and the second 7 pixels in the matched component. Because the matched component is known to both encoder and decoder, we can use a clairvoyant context centred around the pixel to be encoded. This 4/7 context is shown at the cell position in the second row (4-pixel context) and second column (7-pixel context) of Table 7.1. The black and grey solid circles represent pixel positions that are used in the context. The white and grey circles represent the position of the current pixel that is to be encoded, which in effect is the corresponding alignment point between the two contexts.

Table 7.1 shows the results for compressing the fifty UW images using each combination of the contexts. For example, when using the 4/7 context the mean compressed file size is 45490 bytes. This corresponds to the results for the CSIS method in Table 6.10. The table shows that the best method on the test images is the 4/13 method, with a mean file size of 44932 bytes (shown in italics). This is 1.2% smaller than our previous 4/7 method.

	45840	45273	45177	45532	45940	49663
	45296	44637	*44487*	45148	45659	50033
	45255	44671	44625	45638	46210	51590
	45221	44699	44780	46138	46792	51984
	45376	45025	45262	47143	47999	53710

Table 7.3: Using ternary contexts for component compression

Binary 4/13	Ternary 4/9
44932	44487
	$\surd\,(t = 14.97)$

Table 7.4: Comparing the best binary context with the best ternary context

Table 7.2 shows that although the difference is small, it is significant. The method that has the greatest file size is the 8/25 method with a mean size of 49542, 10% higher than the best method.

7.3 Ternary contexts

The mean width of components in the Jefferson images is 17 pixels, and the mean height 22 pixels. If we assume that a component is matched with another of the same size, and that they are aligned by their centres, then for around 20% of their area the contexts are not wholly inside the image to be encoded. This result assumes we are using the 4/7 context pair. Using this particular combination of contexts, when the context is positioned in the single pixel perimeter around the image, part of the it will be outside the image. When the contexts are not inside the image, and the only pixel values are black or white, it is not clear which colour should be used.

Normal contexts are made up of binary values that dictate whether a pixel is black or white.

By extending the contexts to include a third state that represents whether the pixel is outside the image to be encoded, the modelling process does not need to assume the pixel colour outside the region. We call these *ternary* contexts.

The result of compressing the UW test images with the various context combinations is shown in Table 7.3. The table shows that unlike the binary contexts in Table 7.1, the best is the 4/9 context with a mean file size of 44487 bytes (shown in italics). This is 1% better than the binary contexts, and Table 7.4 shows that the improvement is significant. The worst case for the ternary context is again the 8/25 context, but now it is 21% worse than the best ternary context.

7.4 Updating after pairwise encoding

Two types of context are used during the compression process. The first is the standard JBIG1 context that is used to compress previously unseen components. The context, shown in the previous section, is made up of a combination of normal and clairvoyant contexts and is used to pairwise compress the components. Each component that is compressed is encoded using one of two methods. The first method compresses the component as a previously unseen character using statistical information from the JBIG1 context. The second method compresses the component with respect to another component using a secondary context.

When a component is compressed using the second method, we also update the JBIG1 context probability counts for that component, so that possibly helpful statistical information is not lost. Table 7.5 compares the difference between updating the JBIG1 context and not updating it after pairwise encoding. It requires an average of 44487 bytes to encode an image when the JBIG1 context information is updated, and an average of 44502 bytes to encode an image without updating. These results are almost identical, and Table 7.6 confirms that

Image	With updates	No updates
A006	62793	62719
A034	61186	61233
A03J	46742	46671
A04D	73989	74052
A05E	56685	56708
C03I	46906	46835
C048	36140	36198
D03D	43076	43074
D03M	19754	19747
D048	25401	25423
D04G	56009	55971
D05N	27497	27517
D06C	34138	34149
D06N	48986	49067
E009	25685	25728
E00H	35489	35505
E00M	38217	38255
E01J	69434	69400
E037	76395	76352
E04I	42014	42014
H00C	15761	15792
H019	34890	34913
H041	42096	42090
H04D	26317	26319
H04F	54292	54258
I00J	69095	69118
I00L	70458	70462
I037	40679	40808
J03M	48147	48163
J04A	44884	44888
J04K	66295	66104
K009	37871	37899
N01A	18485	18513
N027	17363	17406
N02A	6831	6848
N02K	53662	53765
N03K	24427	24486
N048	34587	34630
N05I	39926	39975
S021	47246	47259
S03G	53821	53858
S03R	33870	33911
S044	42577	42626
S047	62541	62564
S048	61658	61711
S04H	55804	55780
V004	52789	52766
V00C	36940	36970
V00G	52615	52654
V00N	51904	51954
Mean	44487	44502

Table 7.5: Global context updates

With updates	No updates
44487	44502
	$\times$ ($t = 2.08$)

Table 7.6: Comparing global context updates for significance

there is no significant difference between the two methods.

7.5 Aligning using the centroid and the centre

When compressing a component with respect to another component, they must first be aligned at some reference point. The centroid is commonly used [IW94, WBH+92, Kia97, Zha97], although Howard uses the component centres for alignment [How96]. This section compares these two methods and tests the results for statistical significance.

When components are matched using their centroids, there is the extra cost of encoding the difference between the centroid of the known match and the unknown component that is being transmitted. When the components are aligned on their centres, this extra information does not need to be transmitted because the decoder knows the width and height of the component before the pixels are decoded. Table 7.7 shows the results of the two methods. Although matching using centroids involves sending more information, the gain from better alignment is worthwhile. The centroid alignment method has a mean file size of 44497, which is 2% smaller than the mean file size of 45394 using centre alignment. Table 7.8 shows that the improvement is significant.

Image	Centroids	Centres
A006	62793	64958
A034	61186	61750
A03J	46742	47137
A04D	73989	75572
A05E	56685	57953
C03I	46906	47055
C048	36140	37127
D03D	43076	42783
D03M	19754	19851
D048	25401	25963
D04G	56009	56684
D05N	27497	27942
D06C	34138	34820
D06N	48986	50073
E009	25685	25820
E00H	35489	36273
E00M	38217	39217
E01J	69434	70446
E037	76395	76125
E04I	42014	41903
H00C	15761	15884
H019	34890	35369
H041	42096	42157
H04D	26317	26777
H04F	54292	57439
I00J	69095	70603
I00L	70458	72364
I037	40679	41388
J03M	48147	49090
J04A	44884	44547
J04K	66295	66704
K009	37871	38676
N01A	18485	19610
N027	17363	17897
N02A	6831	6951
N02K	53662	52772
N03K	24427	25266
N048	34587	36077
N05I	39926	42065
S021	47246	48689
S03G	53821	55022
S03R	33870	35484
S044	42577	44418
S047	62541	64783
S048	61658	64419
S04H	55804	58258
V004	52789	52282
V00C	36940	37586
V00G	52615	54670
V00N	51904	53016
Mean	44487	45394

Table 7.7: Component alignment

Centroids	Centres
44487	45394
	$\surd (t = 7.34)$

Table 7.8: Comparing component alignment using centroids and centres

7.6 Matching strategies

Components can be matched against the codebook in a variety of ways. In previous experiments an unknown component is matched against the average components from each equivalence class, and the best match is picked. This is called *best average* matching. An alternative is to stop the search process once a match has occurred against the average from each equivalence class; this is called *first average* matching. Matching is said to succeed if it satisfies the thresholds determined using machine learning in Chapter 6.

Instead of transmitting the index of the equivalence class, an index can be transmitted which selects a particular component in an equivalence class. In this case the matching process examines *all* components in each equivalence class to find the best match. This is called *best single* matching. The corresponding approach when the search stops after the first match is found is called *first single* matching.

For these experiments, both *single* methods encoded the index as $index = E \times \Phi + C$, where E is the equivalence class, C is the component index in the equivalence class and Φ is the scaling factor to map the two parameters into a single value. Φ was set to 256, and the number of values in each equivalence class was limited to $E - 1$. Informal experiments have shown that the choice of Φ does not greatly affect the mean file size. The alternative to this seemingly inelegant solution is to encode the two components E and C separately, but this mechanism was used because the index values of similar characters will be clustered, and therefore captured by the tree-based gamma encoder.

Image	Best average	First average	Best single	First single
A006	62793	63560	58450	67259
A034	61186	61412	60595	62671
A03J	46742	47262	46053	48600
A04D	73989	74725	71905	77278
A05E	56685	57155	54449	58631
C03I	46906	47353	46205	49368
C048	36140	36429	33605	37978
D03D	43076	43429	43609	44482
D03M	19754	19820	19583	20137
D048	25401	25776	24075	26611
D04G	56009	56368	54909	58333
D05N	27497	27725	26358	28607
D06C	34138	34276	32403	35496
D06N	48986	49336	47807	50297
E009	25685	25894	24863	26824
E00H	35489	36003	33641	37245
E00M	38217	38752	34777	40403
E01J	69434	70842	65909	74352
E037	76395	76989	74872	79955
E04I	42014	42313	41801	44222
H00C	15761	15832	15656	16225
H019	34890	35572	33022	36739
H041	42096	42309	41672	43429
H04D	26317	26857	25135	28421
H04F	54292	55798	49942	57959
I00J	69095	69392	68221	70349
I00L	70458	70900	68644	71874
I037	40679	41115	38655	42366
J03M	48147	48732	46528	50753
J04A	44884	45170	45215	46923
J04K	66295	66789	64120	69500
K009	37871	38406	36289	39281
N01A	18485	18789	18284	19066
N027	17363	17472	17378	17644
N02A	6831	6833	6866	6893
N02K	53662	53799	53786	55827
N03K	24427	24743	24398	25251
N048	34587	34944	34236	35271
N05I	39926	40532	39746	41131
S021	47246	48738	43699	50426
S03G	53821	55155	51941	56624
S03R	33870	34561	32287	35551
S044	42577	43461	38403	45601
S047	62541	63700	59197	67604
S048	61658	63128	57087	65918
S04H	55804	57278	52252	59916
V004	52789	54666	55166	63863
V00C	36940	37877	34761	39957
V00G	52615	53365	47498	56264
V00N	51904	52827	48576	55373
Mean	44487	45083	42891	46815

Table 7.9: Four methods for matching against components in the codebook

Best average	First average	Best single	First single
44487	45083	42891	46815
	$\surd(t = 9.42)$	$\surd(t = 7.25)$	$\surd(t = 9.35)$

Table 7.10: Comparing the codebook matching methods

Table 7.9 shows the results for each of the four methods. The *best average* method has a mean file size of 44487, which is 1.5% smaller than the *first average* method. A similar pattern is shown for the individual characters, where the *best single* method has a mean file size of 42891—8% smaller than the *first single* method with a mean size of 46815. The table shows that matching against the best single character achieves a mean file size 3.6% smaller than using the representative match from each class. Table 7.10 shows that all of these results are statistically significant. It is worth the extra effort of finding a better match.

7.7 Summary

This chapter introduces the use of document image compression techniques for compressing multiple pages from the same data source by building a codebook incrementally. Limiting the maximum number of examples in an equivalence class and the maximum number of equivalence classes is investigated, and we see that serious restrictions can be placed on both parameters without adversely affecting the mean file size, while dramatically decreasing run time.

A variety of contexts for pairwise encoding are introduced and their relative performances are shown. Ternary contexts are introduced which capture image boundary information, and are shown to improve over binary contexts. The QM-Coder has the update procedure built into the coder, and has different properties from the probability estimation and arithmetic coder that is used in this thesis. It is unknown whether the use of ternary contexts generalise to other coders.

The extra processing of updating the JBIG1 context every time a component is compressed with respect to another component is shown to have no compression advantages. Centroids are shown to be significantly better for alignment than components' centres.

Several matching strategies are introduced and it is shown that matching against the single best component gives significant compression gains over previously used methods.

Chapter 8

Separating text from graphics

Pages are composed of many zones, each of which may contain text, line drawings, halftone images, mathematical formulae and tables. A page *zone* is defined as a region on the page that contains homogeneous contents: for document images most zones contain text. Pages often contain rectangular sections of text which correspond to tables or columns, although some are ragged due to text flowing around images.

Although the component-based method of document image compression works well for text, there are better ways of compressing halftone images. If a document image compression system can determine the classification of a zone in the image, it can model the area more appropriately. For example, if a certain zone is known to contain only text, the system can compress it using a component based compression model. If it contains halftone images and line drawings, the image is compressed using the method in Section 2.4.9.

We will develop a procedure for determining the classification of zones that works in several stages. The first stage consists of a segmentation phase that attempts to group components into homogeneous sections. At this stage, the actual classifications of the zones do not need to be known: the segmentation only needs to cluster similar components together. Once the zone boundaries are determined, they are analysed and classified into categories.

The UW database contains 979 images that have been segmented manually into 13831

Zone type	Frequency
text	12216
math	511
drawing	461
ruling	314
halftone	154
table	134
logo	15
map	14
advertisement	8
not-clear	2
announcement	1
seal	1
Total	13831

Table 8.1: Zone classifications and their frequency in the UW database

different zones. The zones have been classified by the operators into the twelve categories shown in Table 8.1.

This chapter focuses on determining the difference between text zones and non-text zones. The *text* classification is extended to include *math*. The other ten classifications are grouped together and labelled as *non-text*.

Section 8.1 introduces page segmentation and classification and describes previous approaches. Most of the methods for classification have involved *ad hoc* rules. The aim of this chapter is to develop a method for automatically determining the classification document zones without resorting to hand-crafted rules or parameters. The features that are calculated for each zone are described in Section 8.2. These features are used by the machine learning schemes to develop a model that assigns classifications to each zone.

To investigate the feasibility of automated zone classification, each of the test images is segmented and compressed using the manually specified zones in the UW database. Section 8.3 describes this experiment, and shows that the use of the manual classification to choose the compression model can increase the compression ratio. The two machine learning methods C5.0 and Naïve Bayes are used to predict the classification for each zone of the test images.

Both of these schemes are described in Section 1.6. The machine learning predictions are shown to give a significant compression improvement.

Section 8.4 automatically segments the images using the docstrum. Each of the segmented zones is classified using the manual ground-truth classifications and is used to train the machine learning schemes. While the manual zones are rectangular, the docstrum zones have no predefined shape. When the machine learning schemes are used to predict the classification for each zone, the mean file size is reduced.

Instead of using clustering techniques to find zones in an image, and then using machine learning to predict the classification of the zone, Section 8.5 predicts the classification of an individual component. This method is shown to be significantly better than attempting to classify the zones.

8.1 Page segmentation and classification

There are many ways to segment a document into zones, and many ways to assign a classification to them. Of the document segmentation methods, the *X-Y tree decomposition*, *run-length smearing*, and the docstrum methods are used frequently. Unlike the docstrum method, the other methods use pixels instead of components to segment the image. With the exception of X-Y tree decomposition, the methods process the image using a bottom-up strategy.

The X-Y tree decomposition method [WCW82] recursively processes an image, with each stage either dividing the image horizontally or vertically, or deciding that no further split should take place. The X-Y method forms either a horizontal or vertical pixel histogram and decides to split the region based on peaks in the histogram. This method is fast, but limits the segmentation to rectangular zones. Bones *et al.* [BGCS90] introduces a more advanced

type of segmentation which combines bottom-up and top-down blocking with a rank filter to extend zone boundaries.

Run-length smearing [NS84] joins pairs of pixels if they are either vertically or horizontally aligned and are close together. The common implementation of run-length smearing for segmenting creates two images: one smeared in the vertical and one smeared in the horizontal directions. These two images are bitwise OR'ed to form the segmented image. Unlike the X-Y method, the shape of the zones need not be rectangular.

The docstrum was introduced in Section 4.2 and segments the image using components instead of pixels by joining the components together based on the distance to their nearest neighbours.

To assign a classification to a zone, two general approaches are used: *structure based* and *feature based*. Structural approaches classify zones in an image by utilising the previously known structure of the page [FK88]. If the image is known to be a page from a certain journal, the heading size, position of the page number, and number of columns are all known in advance. A more rigorous approach is to calculate features for each of the zones in the image. Statistical methods are then often used to generate classifiers for the zones.

8.1.1 Segmentation

This chapter uses the docstrum method for segmentation because it is fast, there are no limits on the shape of the zones, and the docstrum has already been calculated to determine reading order.

After the images are segmented using the docstrum with the default parameters, the clusters that are formed are used in the classification stage. To illustrate the performance of the segmentation process, Figure 8.1 shows the positions of the manual zones, and Figure 8.2

shows the automatically determined zones.

8.1.2 Classification

Once an image is segmented and the position of the zones is known, the next stage is to classify each zone. One of the earliest methods for automatically determining classifications was pioneered by Wong, Casey and Wahl [WCW82], who present a document analysis system that uses the run length smearing process to find zones, and generates zone classifications based on four features that are calculated for each zone. The features used are the height of each zone, the eccentricity of the zone's bounding box, the number of black pixels as a ratio of the bounding box size, and the mean black pixel horizontal run-length. Wong *et al.* present a linearly separable classification scheme of four rules that assigns four classes. The classes are: *text*, *horizontal solid black lines*, *graphics and halftone images*, and *vertical solid black lines*. They report that text zones that have joined characters, large titles or headings, and vertically oriented text, are sometimes classified as graphic and halftone images.

Wang and Srihari [WS89] describe a method for zone classification that gave 100% accuracy on their images. However, the test images contained just 41 zones—and one of the classification categories occurred only once. Whereas Wong *et al.* used run length smearing, Wang and Srihari used the X-Y cut procedure because it generates larger (and fewer) zones. Wong and Srihari extend the four features of Wong *et al.* by adding features that take textural information into consideration. Two matrices are used to calculate the features: a black-white run-length matrix, and a black-white-black run-length matrix. The black-white matrix is a two dimensional matrix, $p(i, j)$ indexed by the length of the white run length, i, and the proportion of black pixels in the total run length, j. To reduce the size of the matrix, j is quantised into ranges. For example, if the black-white run was

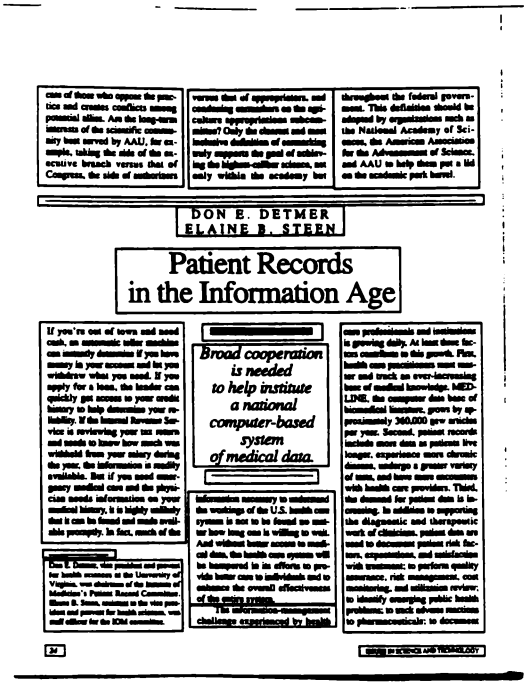


Figure 8.1: Image E00M with manually specified zones

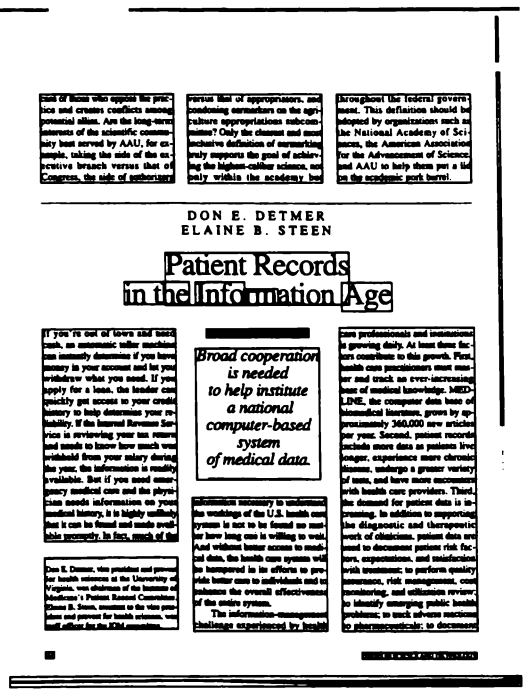


Figure 8.2: Image E00M with zones determined by the docstrum

b b b b b b b b b b b b b w w w w w w w w

then there are 13 black pixels followed by 8 white pixels. The total run length is $i = 21$ and the proportion of black is $j = 13/21 = 62\%$. The value for p(21,62) is updated for each black-white run of this size. The black-white-black matrix is derived in a similar manner.

Wang and Srihari derive three features based on these matrices. The first is

$$F_1 = \sum_{i=1}^{N_c} \sum_{j=1}^{N_r} p(i,j)/j^2 \Bigg/ \sum_{i=1}^{N_c} \sum_{j=1}^{N_r} p(i,j)\,. \tag{8.1}$$

This measure emphasizes short run-lengths. Halftone zones have small values for F_1 because they are composed of small black dots with small white spaces in between them. The second feature is

$$F_2 = \sum_{i=1}^{N_c} \sum_{j=1}^{N_r} j^2 p(i,j) \Bigg/ \sum_{i=1}^{N_c} \sum_{j=1}^{N_r} p(i,j)\,. \tag{8.2}$$

This emphasizes long run-lengths. Large characters have a large value for F_2, while halftone areas have a small value. The third feature is

$$F_3 = \sum_{j=T_1}^{N_r} j^2 \left(\sum_{i=1}^{N_c} p'(i,j) \right) \Bigg/ \sum_{j=T_1}^{N_r} \sum_{i=1}^{N_c} p'(i,j)\,, \tag{8.3}$$

where

$$p'(i,j) = \begin{cases} p(i,j) & \text{if } p(i,j) > T_2 \\ 0 & \text{if } p(i,j) \leq T_2 \end{cases}$$

This applies a threshold to short run-lengths and can be viewed as emphasizing extra long run-lengths. Threshold T_1 is the lower bound on the size of run lengths and is used to eliminate noise. Threshold T_2 is the lower bound for the $p(i,j)$ array. Wang and Srihari use $T_1 = 50$ and $T_2 = 15$.

Fisher *et al.* [FHD90] introduce a system for document image segmentation using decision

rules. They segment images using the run length smearing procedure and classify the zones as either *text* or *non-text* depending on the result of a discrimination system using 14 rules. Five of the 14 rules need to be applied in order, and they use a total of 43 different thresholds. Fisher *et al.* did not describe the process for generating the rule-set, nor the thresholds.

Both of these papers use the same images for both training their system and for evaluation. The authors do not mention the possibility that their systems could be over-fitting the training data, and that their results will not generalise to new images.

8.2 Machine learning features

Instead of forming *ad hoc* rules that are used to classify zones in a page, we present a method that uses established machine learning techniques to determine zone classifications. Because the predictive ability of the machine learning schemes is limited by the accuracy of the manual classifications, we present a set of simple features that attempt to capture the differences between the classification types. The following sections show that these simple features can be combined to give good predictions, and achieve significant improvements when used for choosing the compression model.

The features used are:

- *number*—number of components in the zone;
- *area*—total area of the components in the zone;
- *component area*—mean area of the components;
- *component density*—mean density of black pixels in the components' bounding boxes;
- *aspect ratio*—mean ratio of height to width of the components' bounding boxes;

- *holes*—mean Euler number of the components;

- *edge*—mean number of *edge* pixels in each component.

The first feature counts the number of components in a zone. For text zones this number is quite low, but for halftone zones it is often quite high. The next feature is the total area of the zone. Very small component areas often correspond to salt and pepper noise, and large values often correspond with page borders or visible spine areas. The component area is calculated by dividing the total area by the number of components. The density is calculated by dividing the number of black pixels by the area subtended by the bounding rectangle of the component. Component density is calculated by averaging the densities of each component.

The aspect ratio feature is calculated in a similar way to the component density feature. The aspect ratio for each component is summed and divided by the number of components in the zone. The Euler number of a component is defined as the number of black connected regions minus the number of holes. The number of black regions is easily counted. To count the number of holes, a small transformation is performed: each component is extended by adding a single pixel in both width and height around it, and then the colors of the pixels are inverted. The number of black components in this transformed component is counted and this value, minus one, is the number of enclosed holes in the original component. In the calculation of these features, we define an edge pixel to be a pixel whose colour is different from the one immediately above or to the left of the pixel being processed.

The following sections show that these features are enough to discriminate between the classifications.

8.3 Manually segmented zones

The fifty test images are manually segmented into 723 zones. Of these, 92% are text and the remainder are non-text. Each image contains an average of 14 zones, each of which is described by a non-overlapping rectangular zone. These manual zones do not necessarily cover every black pixel on the image: partial pages, or the spine of the book, are not included in them.

We show in this section that image compression is improved when the manual classifications are used to direct the compression model. When a zone is classified as *text* the components are extracted from the page. If it is non-text, the components are left on the image which is then compressed using the bilevel image model [Mof91]. When this procedure is followed the mean file size is reduced by 5% from 42891 bytes to 40655 bytes. The *Manual* column of Table 8.2 shows the results on the fifty test images, while Table 8.3 shows that when the classification of zones is known, the mean file size is significantly smaller.

This first experiment shows that compression can be increased if the position and class of each zone is known. The next experiment examines how accurately the classification of each of the zones can be determined using machine learning, and how classification accuracy translates into a reduction in compressed file size.

The 929 remaining images from the UW database are used as training data. These images have been manually zoned into 13108 regions, of which the relative proportions of text to non-text are the same. The seven features from Section 8.2 are calculated for each of the zones, and this information is used as the training data. Each of the fifty test images is processed in the same manner to generate the testing data.

The results of compressing each of the test images when using the machine learning predictions for each zone are shown in the last two columns in Table 8.2. Naïve Bayes has a mean

Image	Normal	Manual	Naïve Bayes	C5.0
A006	58450	59109	59109	59109
A034	60595	51539	51553	51553
A03J	46053	44348	44105	44354
A04D	71905	63735	63523	63735
A05E	54449	48063	48063	50206
C03I	46205	42739	43335	44493
C048	33605	31848	31848	31848
D03D	43609	33612	34207	34641
D03M	19583	11703	11793	11703
D048	24075	23252	24038	23279
D04G	54909	49280	47779	49297
D05N	26358	19214	19214	19214
D06C	32403	30839	30802	30802
D06N	47807	44892	46265	45902
E009	24863	22747	24513	22747
E00H	33641	32873	32873	32873
E00M	34777	34245	34281	34245
E01J	65909	61839	61839	61839
E037	74872	69554	69554	69554
E04I	41801	30511	31702	30511
H00C	15656	15667	15562	15667
H019	33022	34268	33285	33285
H041	41672	37566	37566	37566
H04D	25135	26837	26818	26818
H04F	49942	49880	49605	49574
I00J	68221	64203	65120	64203
I00L	68644	62347	62347	62347
I037	38655	30817	30817	30817
J03M	46528	46800	45122	46643
J04A	45215	44243	43955	44332
J04K	64120	52314	52314	52314
K009	36289	34350	34858	35204
N01A	18284	18510	18402	18402
N027	17378	17499	17459	17459
N02A	6866	6824	6824	6824
N02K	53786	51310	51310	51310
N03K	24398	24381	24381	24381
N048	34236	34158	34158	34158
N05I	39746	39668	46426	39668
S021	43699	45867	45933	45933
S03G	51941	51978	51978	51978
S03R	32287	33948	34380	34380
S044	38403	38374	38374	38374
S047	59197	58037	58037	58037
S048	57087	54090	54090	54090
S04H	52252	52425	52425	52425
V004	55166	59673	59363	59363
V00C	34761	44443	44443	44443
V00G	47498	47636	47441	47710
V00N	48576	48678	48599	48599
Mean	42891	40655	40836	40764

Table 8.2: Using manually segmented zones

Normal	Manual	Naïve Bayes	C5.0
42891	40655	40836	40764
	$\surd(t = 3.88)$	$\surd(t = 3.47)$	$\surd(t = 3.78)$

Table 8.3: Comparing manually segmented zones for significance

file size of 40836 bytes and C5.0 has a slightly lower mean file size of 40764 bytes. Table 8.3 shows that both of the machine learning predictions give significant improvements over the normal method that ignores zone information, and that the best of these, C5.0, incurs a compression increase of only 0.3% over the manually classified zones.

The machine learning classification results for Naïve Bayes are shown in Table 8.4. Naïve Bayes classified the zones correctly 93.9% of the time, incorrectly classified 12 text zones as non-text, and incorrectly classified 32 non-text zones as text. Table 8.5 shows the results for the C5.0 decision tree method. C5.0 achieved 96.8% classification accuracy, which is a 3% improvement over Naïve Bayes.

Let us examine a specific image for illustrative purposes. Image A034, shown in Figure 3.1, contains a single drawing towards the bottom of the page. Although this drawing looks like a halftone image, it is actually a stippled drawing and only contains a handful of components. In this particular case, most of the drawing is extracted as a single component and then compressed using a bilevel model. This is why there is very little difference in the file sizes for this image in Table 8.2.

8.4 Segmenting using the docstrum

The previous section shows that it is possible to decrease the compressed file size of the test images significantly by using zones that have been manually specified. Now we attempt to achieve compression gains without using the manually specified image segmentations.

Zone class	Text	Non-text
Text	652	12
Non-text	32	27
Accuracy	93.9%	

Table 8.4: Naïve Bayes classification using the manual zones

Zone class	Text	Non-text
Text	657	7
Non-text	16	43
Accuracy	96.8%	

Table 8.5: C5.0 classification using the manual zones

To determine whether an automated technique increases compression, we use the docstrum for clustering regions of a page into zones. Unlike the manual method, the docstrum groups components together based on their nearest neighbours. This has a tendency to group text and graphic zones separately, but if they are too close to each other they may become joined.

To keep the same experimental framework as the previous section, after each page is segmented by the docstrum into connected zones, we define the zone to be enclosed by the bounding box of the components in it. To allow nesting and to reduce the effects that large components have on the size of the zone, we use the centroids of the components to delimit the zone, not their extents. Although it appears that the zone bisects components at the extremities of the zone, the character extraction process is not limited by the zone boundaries. The zone is simply an indicator that determines where the extraction process begins.

The training data for the docstrum method was generated by calculating the docstrum for each of the 929 images and overlaying the manual classifications. For every zone, each component in it is classified by comparing its position with the manual classification at the same point. When all the components are processed, the classification for the entire zone is determined by a majority vote. If there were no classifications for the zone, as in the case where the operator did not classify all the black pixels, the class of the zone is determined

to be *missing*.

Using this procedure, the fifty test images are segmented into 556 zones. Of these 33% of the zones are text, 28% are non-text and 39% are missing. The number of missing classifications is high because the manual classifiers ignore large regions of noise. For the training data, the docstrum segmented the images in 11171 zones, with a similar classification distribution. When the zones are classified as missing, they are compressed using the non-text model.

When the test images are compressed using the docstrum zones with the manual classifications, the mean file size is slightly increased from 42891 bytes to 42947 bytes. The *Manual* column of Table 8.6 shows the individual results, and Table 8.7 shows that these results are not significantly different. If we assume that the manual classification gives a lower bound on the compression, then when machine learning techniques are used to predict the zones, we expect the mean file sizes to be larger.

The results when these predictions are used to assign the classification of the zone are shown in the last two columns of Table 8.6. The use of Naïve Bayes increased the mean file size slightly from 42891 to 42907 bytes, while C5.0 decreased the mean file size to 42633.

Table 8.7 shows that the two machine learning predictions do not give statistically significant improvements when compared with the normal method. Although the use of the docstrum zones and the C5.0 classifications give an average file size of 42633, the significance test shows the result is not reliable at the 0.01 (99%) confidence level.

The machine learning classification results for Naïve Bayes are shown in Table 8.8. Naïve Bayes classifies the zones correctly only 57.4% of the time. Table 8.9 shows that C5.0 performs much better, with 74.3% classification accuracy. C5.0 classified 43 text zones as non-text and 45 non-text zones as text. Because Naïve Bayes performs so much worse than C5.0, we infer that there is significant interaction between the features.

Image	Normal	Docstrum	Naïve Bayes	C5.0
A006	58450	59432	58555	58684
A034	60595	59543	60605	60257
A03J	46053	46668	46158	47255
A04D	71905	70200	71905	70150
A05E	54449	54130	54369	54299
C03I	46205	44628	45870	46104
C048	33605	33719	33605	33914
D03D	43609	42288	43714	34781
D03M	19583	19660	19660	19657
D048	24075	24280	24075	24075
D04G	54909	56057	54925	54174
D05N	26358	25869	26358	25989
D06C	32403	32403	32403	32403
D06N	47807	48130	47366	48329
E009	24863	23528	24970	23528
E00H	33641	33051	33641	33419
E00M	34777	34813	34777	34854
E01J	65909	65471	66029	65471
E037	74872	72696	74954	72758
E04I	41801	41931	41801	41801
H00C	15656	15656	15656	15571
H019	33022	34327	33147	33135
H041	41672	39908	41769	41412
H04D	25135	26465	25189	25286
H04F	49942	49942	49942	49942
I00J	68221	68206	68221	68221
I00L	68644	68700	68747	68644
I037	38655	39743	38782	38782
J03M	46528	46601	46528	46601
J04A	45215	46302	41613	42348
J04K	64120	53932	64073	63716
K009	36289	36118	36412	36275
N01A	18284	18284	18284	18284
N027	17378	17485	17378	17565
N02A	6866	6866	6866	7080
N02K	53786	51380	52934	51394
N03K	24398	24398	24398	24398
N048	34236	34344	34344	34344
N05I	39746	39746	39746	39746
S021	43699	46412	43872	45742
S03G	51941	52058	51978	51340
S03R	32287	34622	32371	32489
S044	38403	38403	38403	38403
S047	59197	59197	59197	59197
S048	57087	57087	57087	57087
S04H	52252	52283	52379	52283
V004	55166	55287	59341	59488
V00C	34761	49038	34868	34734
V00G	47498	47498	47498	47655
V00N	48576	48576	48576	48576
Mean	42891	42947	42907	42633

Table 8.6: Segmenting using the docstrum

Normal	Docstrum	Naïve Bayes	C5.0
42891	42947	42907	42633
	$\times (t = 0.15)$	$\times (t = 0.14)$	$\times (t = 1.14)$

Table 8.7: Comparing manual docstrum classifications with machine learning

Zone class	Text	Non-text
Text	185	3
Non-text	143	12
Accuracy	57.4%	

Table 8.8: Naïve Bayes classification using the docstrum zones

Zone class	Text	Non-text
Text	145	43
Non-text	45	110
Accuracy	74.3%	

Table 8.9: C5.0 classification using the docstrum zones

8.5 Analyzing individual components

Instead of attempting to automatically predict the classification of zones on a page, this section investigates the prediction of classification *for each component*. To train the machine learning models the training data must include noise, normal characters, page borders, tables and figures. As we do not have a classified training set, a set of 4000 components were extracted and classified as *text* or *non-text*. These components were extracted from images A039 and A00I—two images that were not in the testing corpus. Each image was manually classified, and the classifications verified on a second pass. Components were only classified as text when it was clear they were letters, numbers, or other recognisable characters.

When a component was classified as *non-text* it was not extracted from the image, and this residue was coded using two-level coding (see Section 2.4.9).

The same features that were used to segment the zones were calculated for each of the

Image	Without pruning	Component pruning
A006	58450	57156
A034	60595	51614
A03J	46053	42198
A04D	71905	62097
A05E	54449	46065
C03I	46205	42781
C048	33605	31522
D03D	43609	33002
D03M	19583	11691
D048	24075	23295
D04G	54909	45197
D05N	26358	19339
D06C	32403	30791
D06N	47807	45127
E009	24863	22972
E00H	33641	32765
E00M	34777	34297
E01J	65909	61677
E037	74872	70858
E04I	41801	29896
H00C	15656	14886
H019	33022	29914
H041	41672	33289
H04D	25135	23107
H04F	49942	48788
I00J	68221	59277
I00L	68644	60040
I037	38655	29087
J03M	46528	44547
J04A	45215	41222
J04K	64120	52054
K009	36289	33466
N01A	18284	18424
N027	17378	17093
N02A	6866	6796
N02K	53786	50891
N03K	24398	23805
N048	34236	34073
N05I	39746	39325
S021	43699	43289
S03G	51941	51185
S03R	32287	31227
S044	38403	38137
S047	59197	57478
S048	57087	53619
S04H	52252	50990
V004	55166	59305
V00C	34761	33764
V00G	47498	47042
V00N	48576	47630
Mean	42891	39362

Table 8.10: Pruning individual components

Without pruning	Component pruning
42891	39362 $\surd(t = 6.52)$

Table 8.11: Comparing the two methods for significance

characters. C5.0 processed the features, and calculated a simple rule: A component is text if the area in the component is between 1 mm^2 and 5 mm^2, otherwise it is non-text.

Table 8.10 shows that when components are pruned using this single rule, the mean file size decreases by 8%, with a mean file size of 39362 bytes. This is significantly better than when not pruning based on area, and is better than the previous results that attempted to classify an entire zone in the image. This is also an improvement to the results in Section 4.4.1, as this procedures removes noisy components entirely, and has a less arbitrary definition of noise.

8.6 Summary

When images are manually segmented into zones and the zone information is used to determine whether to compress using the component model or a bilevel image compression model, the mean file size of images is reduced by 5%. When Naïve Bayes or C5.0 is used to predict the classification of these manual zones, the mean file size is similar.

To automatically determine the position of zones in a page without using manually specified regions and classifications, the docstrum can be used to cluster the components into zones. If these docstrum zones are generated, but labelled using the classifications given manually, the difference in mean file size is statistically insignificant. When the same docstrum zones are used, but the classification is determined by machine learning schemes, the results are still not significantly better.

A third segmentation technique is investigated. Instead of attempting to segment and classify large zones, a collection of individual characters is classified, and C5.0 is used to determine a classification for each of them. Each of the individual components is classified into text or non-text, and using a simple rule that is based on the area of component, the mean file size is reduced by 8%.

In the UW database, the number of text zones comprise almost 90% of the total number of zones. If the size of the zones is taken into consideration, this percentage would be even higher. The gains of the methods presented in this chapter increase further when used on images that contain a higher ratio of non-text zones.

Chapter 9

Summary

This thesis makes two claims. First, that we can design a lossless document image compression system that achieves good compression, is tolerant to skew and performs well when compressing multiple pages. Second, that the best performance is obtained when parameters are selected automatically using machine learning techniques.

Each chapter investigates specific components of the document image compression procedure. Areas are examined in isolation, and potential improvements tested against previous results using statistical techniques and added to a base compression system. This thesis introduces machine learning as a novel addition to document image compression, and combines page-layout techniques with skew correction and template matching to improve the compression process.

When the various improvements are combined, our document image compression system achieves compression ratios higher than any other system. The results from Chapter 8 are compared with the DjVu system in Table 9.1 and show that we achieve a mean compressed file size of 39362, which is 2% smaller than DjVu's mean file size of 40159. Figure 9.2 shows that this gain is statistically significant.

9.1 Specific contributions

A novel method of combining adaptive codebook generation with equivalence classes is introduced in Chapter 2 along with a base document image compression system. The base system is extended by each of the following chapters.

The method for extracting components from an image is often overlooked, and Chapter 3 investigates three parameters that influence the extraction process. Components can be extracted using 4-connectivity or 8-connectivity, they can be allowed to nest within each other or not, and their height and width can be bounded. The chapter shows that 8-connected extraction is better than 4-connected extraction, and achieves a 1% compression gain. There is no significant difference between the compression ratios when allowing components to be nested or not, and imposing component width and height bounds causes no significant difference in the mean compressed file size.

A new method for determining the reading order of document images is introduced in Chapter 4. It uses the docstrum, a bottom-up clustering technique that groups components based on proximity to their neighbours. This new method accurately determines the position of line and column boundaries, and the compression results are stable when the test images are perturbed. The docstrum method for reading order determination gives better compression than five other methods on the original image corpus, when the images are skewed by a random amount, and when the images are rotated by 90 degrees. The docstrum method introduced a 1% improvement over the profile method that was used in the base system.

Compressing the symbol indices using a PPMD model incorporating context information about previous components is shown to yield a significant improvement over using gamma encoding. The use of PPMD increases compression by 1%. Because of the small number of components on each page, Chapter 4 shows that there is no need to interpolate the position of spaces in the symbol stream, because there are not enough symbols for PPMD to adapt

to them accurately.

The skew of document images is accurately determined in Chapter 5 using the hierarchical Hough transform. When the manually determined skew angle is combined with the docstrum to determine the reading order, compression is improved. In this chapter we show that it is possible to determine skew angle accurately with a variety of image resolutions and angles, and that when automated skew determination is used, a 2% compression gain is achieved over manually measured angles.

Four methods of template matching are described in Chapter 6 and two new methods are introduced. The first is based on the cross-entropy of two components and achieves a high classification accuracy. The cross-entropy method is combined with the other four methods using machine learning and this new method is found to have even higher classification accuracy. The thresholds for each method are determined automatically and each is evaluated on the test images. While the best template matching method has a high classification rate, and therefore improves lossy compression, the template matching method that achieves the best lossless compression has the lowest classification accuracy. The best compression result is achieved by the simple CSIS method, because the errors that it produces are simple to model using pairwise compression. Using the CSIS template matching method gives a 1% compression increase over the XOR method.

Factors that influence the development of the component codebook are investigated in Chapter 7. When multiple pages from the same document source are compressed sequentially without reinitialising the codebook, compression improves on each subsequent image. Using a testing corpus of twenty-one images, the total compressed file size over all of the images is reduced by 11%. Usually the number of equivalence classes and the number of components in each equivalence class are allowed to grow without bound. This chapter shows that we can limit the maximum number of classes and the maximum number of

examples per class without affecting the compression.

Various contexts for pairwise compression are evaluated in the document image compression system, and a new context is shown to gain over 1% compared with the original. A new ternary context is introduced, which includes a third state that corresponds with whether the position of the pixel is inside the image to be encoded. The ternary context gives a 2% improvement over binary contexts.

Two different alignment techniques for pairwise compression are evaluated in Chapter 7. Alignment on component's centroids gives a 2% compression gain when compared with alignment on component's centres.

The final experiment in Chapter 7 compares matching strategies in the codebook. We investigate whether to match against individual components or against the average from the equivalence class, and whether to stop the search process after the first match is found. The best compression is achieved when a component is matched against the best individual component. This is around 4% better than matching against the best average component in each class.

Pages are composed of many different types of zones, including text, line drawings and images. Chapter 8 investigates methods for classifying different areas in the image. Compression improves if the manually classified page zones are used to determine whether the document image compression model or a pixel model is used. Machine learning techniques are shown to automatically predict the manual classifications accurately. When the docstrum is used to determine the position and shape of the zones, compression is not improved. Instead of attempting to determine the classification of large zones, the machine learning scheme C5.0 is used to derive a rule which determines whether a component represents a letter. When this rule is used in the compression process, an 8% improvement is achieved.

When all the improvements are combined, the compression ratio has increased by 21% when compared with the base system originally presented in Chapter 2. The system described by this thesis achieves the highest known compression ratios and is freely available.

9.2 System parameters

Although we attempted to minimise the number of arbitrary parameters in this system, there remain a number that have been inherited from other systems. These include:

- The use of the Laplace estimator with single byte scaling (Sections 2.4.5 and 2.4.9)
- The choice of arithmetic coder (Section 2.4.4)
- The choice of $k = 5$ for the docstrum (Section 4.2)
- The choice of $C = 5$ and context sizes for Moffat's two-level model (Section 2.4.9)
- The choice of the JBIG1 context (Section 2.4.9)

The Laplace estimator combined with the arithmetic coder was used as they are freely available and well known. The initial values, increments, and scaling of the counts can be tuned for greater performance (eg. the Dirichlet estimator), but this was not formally investigated. O'Gorman shows that the choice of $k = 5$ for the docstrum captures the appropriate nearest neighbours and works well for documents containing mostly text [O'G93]. The JBIG1 context used by Howard's SPM method [How96] is used in this thesis; other variations have not been investigated.

Image	Inglis	DjVu
A006	57156	59176
A034	51614	53770
A03J	42198	44254
A04D	62097	65491
A05E	46065	47390
C03I	42781	45221
C048	31522	31831
D03D	33002	37037
D03M	11691	12139
D048	23295	22746
D04G	45197	48104
D05N	19339	19804
D06C	30791	30121
D06N	45127	46485
E009	22972	24012
E00H	32765	32570
E00M	34297	33835
E01J	61677	64263
E037	70858	74824
E04I	29896	32539
H00C	14886	14920
H019	29914	28346
H041	33289	39436
H04D	23107	22980
H04F	48788	48608
I00J	59277	61992
I00L	60040	62631
I037	29087	30597
J03M	44547	45009
J04A	41222	44217
J04K	52054	55949
K009	33466	34613
N01A	18424	16838
N027	17093	15963
N02A	6796	6224
N02K	50891	53172
N03K	23805	23465
N048	34073	33048
N05I	39325	38310
S021	43289	43492
S03G	51185	51210
S03R	31227	30288
S044	38137	36873
S047	57478	57976
S048	53619	53554
S04H	50990	51895
V004	59305	53186
V00C	33764	33102
V00G	47042	46165
V00N	47630	48279
Mean	39362	40159

Table 9.1: Comparing our system with DjVu

Inglis	DjVu
39362	40159
	$\surd\,(t = 2.82)$

Table 9.2: Comparing the two methods for significance

9.3 Future work

The method for encoding the offsets between components is simple and could be improved by incorporating the improvements of Zhang [Zha97] and Howard [How96]. When Table 9.1 is examined to determine the types of images for which DjVu has a smaller compressed file size, it appears that DjVu gives better results when the page contains mostly text. This could be attributed to the different methods for encoding offsets and the use of a combined probability and entropy coder.

It would be interesting to attempt to model the position of components using machine learning techniques. This could possibly include classification information about individual characters. The count-based model and the arithmetic coder could be replaced with the QM-Coder or ELS-Coder for faster compression times, while incorporating the fast residue encoding of Constantinescu and Arps [CA97].

Although information about the position of spaces in the text does not increase compression when processing a single page, there may be compression gains when compressing multiple pages.

The *best single* matching method is the slowest of the methods, and could possibly be approximated by matching against the average representatives in each equivalence class to find the class, and then matching against the individual components in the class. This would form a hierarchical best match and reduce matching time by a factor proportional to the mean number of components in an equivalence class.

Bibliography

[AH90] Teruo Akiyama and Norihiro Hagita. Automated entry system for printed documents. *Pattern Recognition*, 23(11):1141–1154, 1990.

[AN74] R. N. Ascher and George Nagy. A means for achieving a high degree of compaction on scan-digitized printed text. *IEEE Transactions on Computers*, 23(11):1174–1179, November 1974.

[ATT99] `http://www.djvu.att.com`. The DjVu web site, 1999.

[Bai87] Henry S. Baird. The skew angle of printed documents. In *Proceedings of the SPIE 40th Annual Conference and Symposium on Hybrid Imaging Systems*, pages 21–24, Rochester, NY, USA, 1987.

[Bai93] Henry S. Baird. Document image defect models and their uses. In *Proceedings of the Second International Conference on Document Analysis and Recognition*, Tsukuba Science City, Japan, pages 62–67, IEEE Computer Society Press, October 1993.

[BCW90] Timothy C. Bell, John G. Cleary, and Ian H. Witten. *Text compression*. Prentice Hall, Englewood Cliffs, NJ, USA, 1990.

[BGCS90] Philip J. Bones, Todd C. Griffin, and Chris M. Carey-Smith. Segmentation of document images. In W. Bender and M. Saito, editors, *Image Communications and Workstations, Proceedings of the SPIE*, volume 1258, pages 78–88, 1990.

[CA97] C. Constantinescu and R. Arps. Fast residue coding for lossless textual image compression. In James A. Storer and Martin Cohn, editors, *Proceedings of the IEEE Data Compression Conference*, Snowbird, Utah, USA, pages 397–406, Los Alamitos, CA, USA, 1997. IEEE Computer Society Press.

[Cap59] J. Capon. A probabilistic model for run-length coding of pictures. *IRE Transactions on Information Theory*, IT-5:157–163, December 1959.

[CH94] Su Chen and Robert M. Haralick. An automatic algorithm for text skew estimation in document images using recursive morphological transforms. In *Proceedings of the first International Conference on Image Processing (ICIP'94)*, pages 139–143. IEEE Press, November 1994.

[CHP95] Su Chen, Robert M. Haralick, and Ihsin T. Phillips. Automatic text skew estimation in document images. In *Proceedings of the Third International Conference on Document Analysis and Recognition*, Montreal, Canada, pages 1153–1156. IEEE Computer Society Press, August 1995.

[CW84] John G. Cleary and Ian H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32(4):396–402, April 1984.

[FHD90] James L. Fisher, Stuart C. Hinds, and Donald P. D'Amato. A rule-based system for document image segmentation. In *Proceedings of the 10th International Conference Pattern Recognition*, pages 567–572, Los Alamitos, CA, USA, June 1990. IEEE Computer Society Press.

[FK88] Lloyd Alan Fletcher and Rangachar Kasturi. A robust algorithm for text string separation from mixed text/graphics images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(6):910–918, November 1988.

[FV82] James D. Foley and Andries Van Dam. *Fundamentals of Interactive Computer Graphics*. Addison-Wesley, Reading, Mass., USA, 1982.

[HFD90] Stuart C. Hinds, James L. Fisher, and Donald P. D'Amato. A document skew detection method using run length encoding and the Hough transform. In *Proceedings of the 10th International Conference Pattern Recognition*, pages 464–468, Los Alamitos, CA, USA, June 1990. IEEE Computer Society Press.

[Hol88] Murray J. J. Holt. A fast binary template matching algorithm for document image data compression. *Pattern Recognition*, pages 230–239, 1988.

[Hou62] P. V. C. Hough. Method and means for recognizing complex patterns. US Patent 3,069,654, December 18 1962.

[How93] Paul G. Howard. *The design and analysis of efficient lossless data compression systems*. PhD thesis, Brown University, USA, 1993.

[How96] Paul G. Howard. Lossless and lossy compression of text images by soft pattern matching. In James A. Storer and Martin Cohn, editors, *Proceedings of the IEEE Data Compression Conference*, Snowbird, Utah, USA, pages 210–219, Los Alamitos, CA, USA, March 31–April 3 1996. IEEE Computer Society Press.

[HR80] R. Hunter and A. H. Robinson. International digital facsimile coding standard. *Proceedings of the IEEE*, 68(7):854–867, 1980.

[Huf52] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IEEE*, 40(9):1098–1101, September 1952.

[HX86] Murray J.J. Holt and C. S. Xydeas. Recent developments in image data compression for digital facsimile. *ICL Technical Journal*, pages 123–146, May 1986.

[HYR86] A. Hashizume, P. S. Yeh, and A. Rosenfeld. A method of detecting the orientation of aligned components. *Pattern Recognition Letters*, 4:125–132, 1986.

[Ing95] Stuart J. Inglis. Calculating skew in scanned document images. In S. Reeves, editor, *Proceedings of the Second New Zealand Computer Science Students Conference*, pages 111–118. University of Waikato, New Zealand, April 1995.

[IW94] Stuart J. Inglis and Ian H. Witten. Compression based template matching. In *Proceedings of the IEEE Data Compression Conference*, Snowbird, Utah, pages 106–115, Los Alamitos, CA, USA, April 1994. IEEE Computer Society Press.

[IW95] Stuart J. Inglis and Ian H. Witten. Document zone classification using machine learning. In *Proceedings of Digital Image Computing: Techniques and Applications*, pages 631–636, Brisbane, Australia, December 1995.

[IW96] Stuart J. Inglis and Ian H. Witten. Bi-level document image compression using layout information. Working Paper 96/1, Department of computer science, University of Waikato, New Zealand, January 1996. Also published as a poster in the proceedings of the IEEE Data Compression Conference, Snowbird, Utah, USA, pp. 442, IEEE Computer Society Press, Los Alamitos, April, 1996.

[JBI93] Progressive bi-level image compression. International Standard ISO/IEC 11544 | ITU-T Recommendation T.82, 1993.

[Joh66] Elmer D. Johnson. *Communication: An introduction to the history of writing, priting, books and libraries*. Scarecrow press, New York, third edition, 1966.

[JSC83] Ottar Johnsen, Jakub Segen, and Glenn L. Cash. Coding of two-level pictures by pattern matching and substitution. *Bell System Technical Journal*, 62(8):2513–2545, October 1983.

[Kia97] Omid Ebrahimi Kia. *Document Image Compression and Analysis*. PhD thesis, University of Maryland at College Park, 1997.

[Knu86] Donald E. Knuth. *Computer Modern Typefaces*. Addison Wesley, Reading, MA, USA, 1986.

[LIT92] P. Langley, W. Iba, and K. Thompson. An analusis of Bayesian classifiers. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 223–228, AAAI Press and MIT Press, 1992.

[LR81] Glen G. Langdon and Jorma J. Rissanen. Compression of black-white images with arithmetic coding. *IEEE Transactions on Communications*, 29(6):858–867, June 1981.

[LTW94] Daniel S. Le, George R. Thoma, and Harry Wechsler. Automated page orientation and skew angle detection for binary document images. *Pattern Recognition*, 27(10):1325–1344, 1994.

[Mal77] Dumas Malone. *Jefferson and his time*. Little Brown and Co., Boston, USA, 1977.

[MNW95] Alistair Moffat, Radford Neal, and Ian H. Witten. Arithmetic coding revisted. In J. A. Storer and M. Cohn, editors, *Proceedings of the IEEE Data Compression Conference*, pages 202–211, Los Alamitos, CA, USA, 1995. IEEE Computer Society Press.

[Mof91] Alistair Moffat. Two-level context based compression of binary images. In *Proceedings of the IEEE Data Compression Conference*, Snowbird, Utah, USA, pages 328–391. IEEE Computer Society Press, April 1991.

[Moh82] K. Mohiuddin. *Pattern matching with application to binary image compression.* PhD thesis, Stanford University, Stanford, CA, USA, 1982.

[MRA84] K. Mohiuddin, Jorma J. Rissanen, and Ronald B. Arps. Lossless binary image compression based on pattern matching. In *Proceedings International Conference on Computers, Systems and Signal Processing*, Banagalore, India, pages 447–451, 1984.

[MS94] Peter B. Mark and Stuart M. Shieber. Method and apparatus for compression of images. US Patent 5,303,313, 1994.

[NS84] George Nagy and Sharad C. Seth. Hierarchical representation of optically scanned documents. In *Proceedings of the 7th International Conference for Pattern Recognition (ICPR)*, pages 347–349, Los Alamitos, CA, USA, 1984. IEEE Computer Society Press.

[NSF+90] Yasuaki Nakano, Yoshihiro Shima, Hiromichi Fujisawa, Jun'ichi Higashino, and Masaaki Fujinawa. An algorithm for the skew normalization of document images. In *Proceedings of the 10th Annual Pattern Recognition Conference*, Atlantic City, NJ, USA, pages 8–13. IEEE Computer Society Press, June 1990.

[O'G93] Lawrence O'Gorman. The document spectrum for page layout analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1162–1173, November 1993.

[PCC+80] William K. Pratt, Patrice J. Capitant, Wen-Hsiung Chen, Eric R. Hamilton, and Robert H. Wallis. Combined symbol matching facsimile data compression system. *Proceedings of the IEEE*, 68(7):786–796, July 1980.

[Phi68] A. Phillips. *Computer Peripherals and Typesetting*. Harrow: HMSO Press, 1968.

[Pos86] W. Postl. Detection of linear oblique structures and skew scan in digitized documents. In *Proceedings of the 8th International Conference on Pattern Recognition*, Paris, France, pages 687–689, October 1986.

[PZ92] Theo Pavlidis and Jiangying Zhou. Page segmentation and classification. *Graphical Models and Image Processing*, 54(6):484–496, November 1992.

[RB95] Nadine Rondel and Gilles Burel. Cooperation of multi-layer perceptions for the estimation of skew angle in text document images. In *Proceedings of the Third International Conference on Document Analysis and Recognition*, Montreal, Canada, pages 1141–1144. IEEE Computer Society Press, August 1995.

[RL79] Jorma J. Rissanen and Glen G. Langdon. Arithmetic coding. *IBM Journal of Research and Development*, 23:149–162, 1979.

[RS86] A. Rastogi and Sargur N. Srihari. Recognizing textual block using the Hough transform. Technical Report TR86–01, Department of Computer Science, University of Buffalo (SUNY), USA, 1986.

[RUL99] `http://www.rulequest.com`. The Rulequest web site, 1999.

[Sah93] Abdul Saheed. Processing textual images. Master's thesis, Department of Computer Science, University of Waikato, New Zealand, 1993.

[Sed92] Robert Sedgewick. *Algorithms in C++*. Addison-Wesley, 1992.

[SG89] Sargur N. Srihari and Venugopal Govindaraju. Analysis of textual images using the Hough transform. *Machine Vision and Applications*, 2:141–153, 1989.

[Sha48] Claude E. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 27(379–423):623–656, 1948.

[Smi95] Ray Smith. A simple and efficient skew detection algorithm via text row accumulation. In *Proceedings of the Third International Conference on Document Analysis and Recognition*, Montreal, Canada, pages 1145–1148. IEEE Computer Society Press, August 1995.

[SOF+92] Guy A. Story, Lawrence O'Gorman, David Fox, Louise Levy Schaper, and H. V. Jagadish. The RightPages image-baged electronic library for alerting and browsing. *IEEE Computer*, 25(9):17–26, September 1992.

[Spi92] A. Lawrence Spitz. Skew determination in CCITT group 4 compressed document images. In *Proceedings of the Symposium on Document Analysis and Information Retrieval*, Las Vegas, Nevada, USA, pages 11–25, 1992.

[Tea98] William Teahan. *Modelling English Text.* PhD thesis, University of Waikato, New Zealand, 1998.

[Tri84] J. P. Trincklin. *Conception d'un système d'analyse de documents.* PhD thesis, Université de Franche-Comté Besançon, France, 1984.

[UW93] University of Washington English Document Image Database I. Seattle, WA, USA., 1993.

[WBE+94] Ian H. Witten, Timothy C. Bell, Hugh Emberson, Stuart J. Inglis, and Alistair Moffat. Textual image compression: Two-stage lossy/lossless encoding of textual images. *Proceedings of the IEEE*, 82(6):878–888, June 1994.

[WBH+92] Ian H. Witten, Timothy C. Bell, Mary-Ellen Harrison, Mark L. James, and Alistair Moffat. Textual image compression. In J. A. Storer and M. Cohn, editors, *Proceedings of the IEEE Data Compression Conference*, pages 42–51, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

[WCW82] Kwan Y. Wong, Richard G. Casey, and Friedrich M. Wahl. Document analysis system. *IBM Journal of Research and Development*, 26(6):647–656, November 1982.

[WEK99] `http://www.cs.waikato.ac.nz/~ml/`. The WEKA web site, 1999.

[WF99] Ian H. Witten and Eibe Frank. *Data mining: Practical machine learning tools and techniques with Java implementations*. Morgan Kaufmann, 1999.

[WMB94] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes: Compressing and indexing documents and images*. Van Nostrand Reinhold, NY, USA, 1994.

[WS89] Dacheng Wang and Sargur N. Srihari. Classification of newspaper image blocks using texture analysis. *Computer Vision, Graphics and Image Processing*, 47:327–352, 1989.

[Yan93] Hong Yan. Skew correction of document images using interline cross-correlation. *Graphical Models and Image Processing*, 55(6):538–543, November 1993.

[YJ96] Bin Yu and Anil K. Jain. A robust and fast skew detection algorithm for generic documents. *Pattern Recognition*, 29(10):1599–1630, 1996.

[YTS95] Chiu L. Yu, Yuan Y. Tang, and Ching Y. Suen. Document skew detection based on the fractal and least squares method. In *Proceedings of the Third*

International Conference on Document Analysis and Recognition, Montreal, Canada, pages 1149–1152. IEEE Computer Society Press, August 1995.

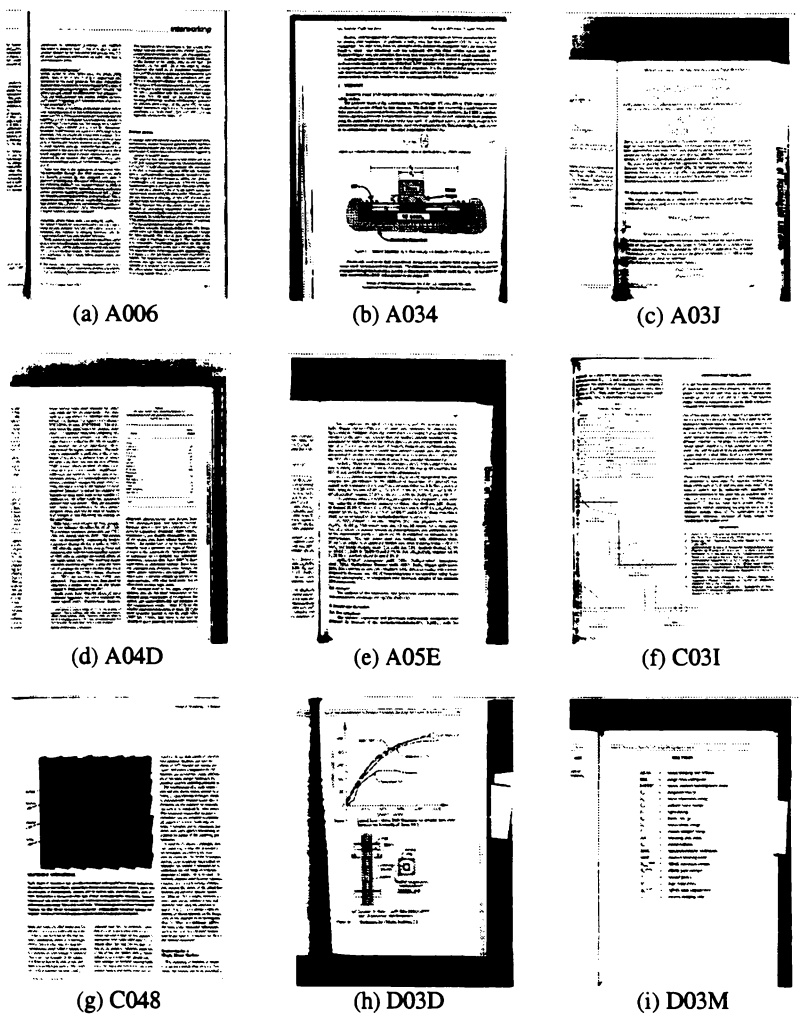
[ZD96] Qin Zhang and John Danskin. Entropy-based pattern matching for document image compression. In *Proceedings of the International Conference on Image Processing*, pages 221–224, 1996.

[Zha97] Qin Zhang. *Document Image Compression via Pattern Matching*. PhD thesis, Dartmouth College, Hanover, New Hampshire, USA, July 1997.

[ZMWSD95] J. Zobel, A. Moffat, R. Wilkinson, and R. Sacks-Davis. Efficient retrieval of partial documents. *Information Processing & Management*, 31(3):361–377, May 1995.

Appendix A

Image corpus

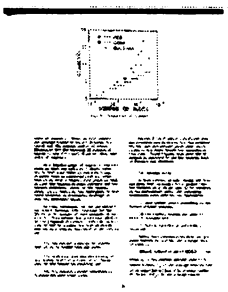


(a) A006 (b) A034 (c) A03J

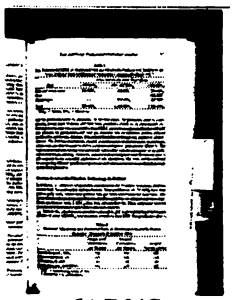
(d) A04D (e) A05E (f) C03I

(g) C048 (h) D03D (i) D03M

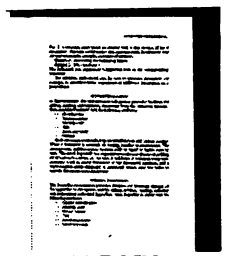
Figure A.1: The test images from the UW corpus



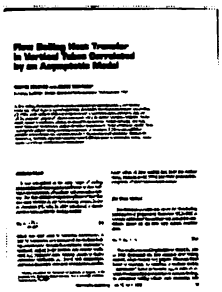
(a) D048



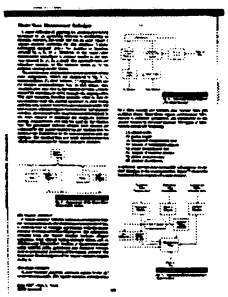
(b) D04G



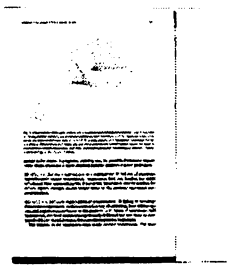
(c) D05N



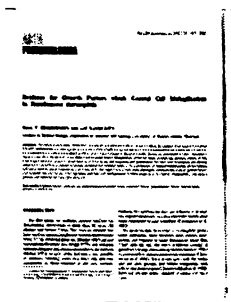
(d) D06C



(e) D06N

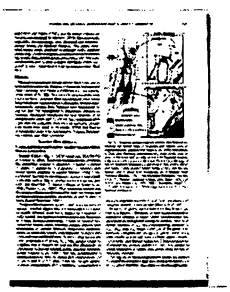


(f) E009

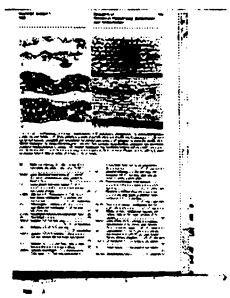


(g) E00H

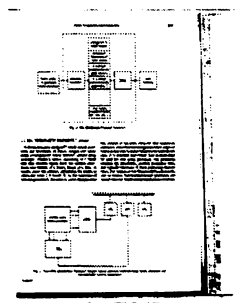
(h) E00M



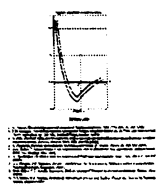
(i) E01J



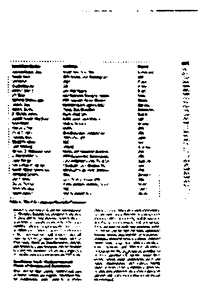
(j) E037



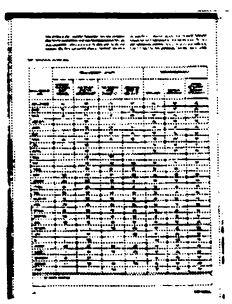
(k) E04I



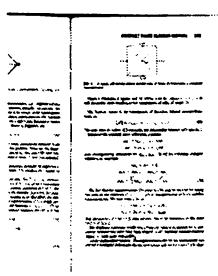
(l) H00C



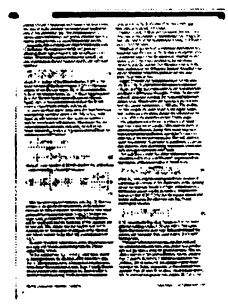
(m) H019



(n) H041

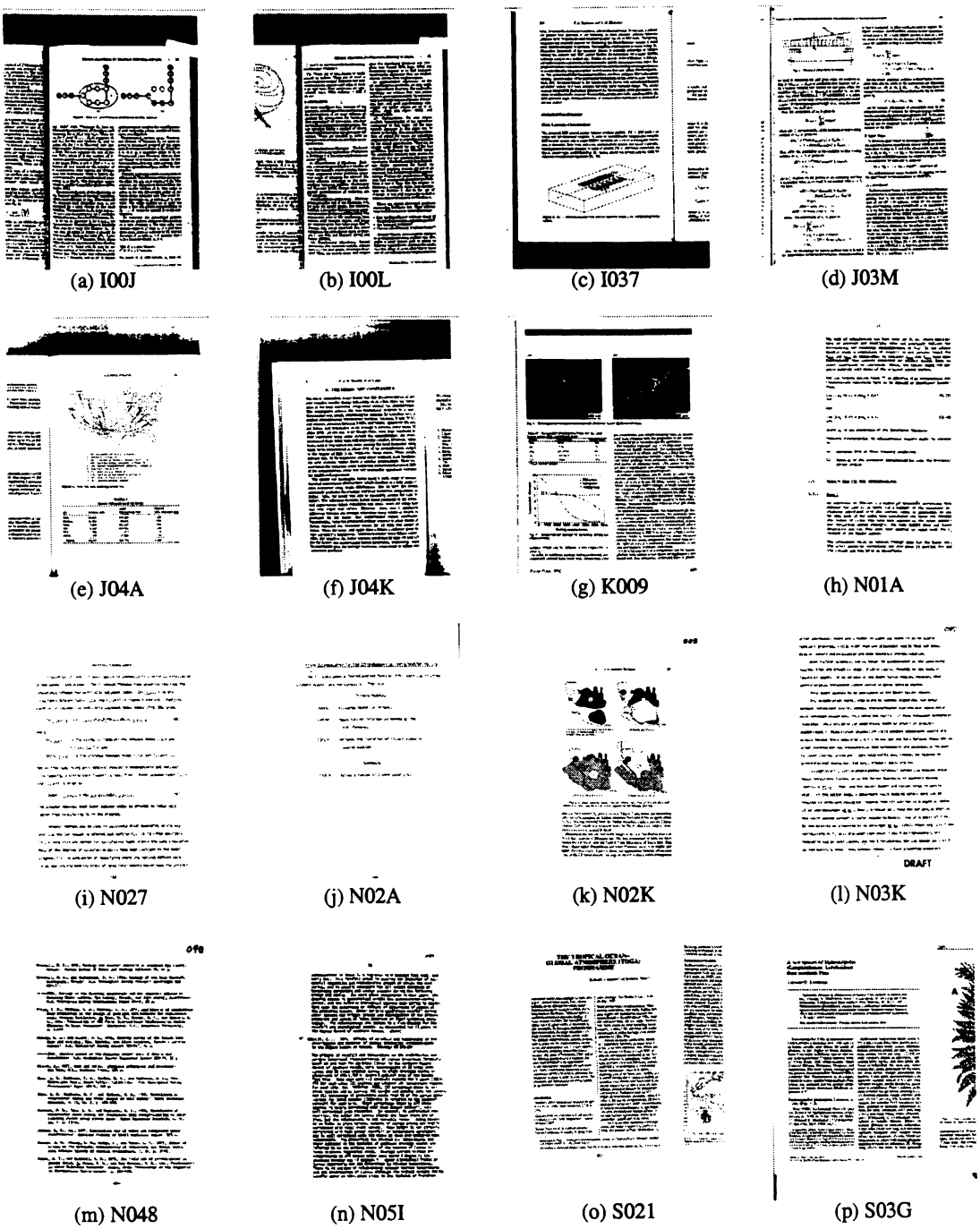


(o) H04D



(p) H04F

Figure A.2: The test images from the UW corpus (continued)



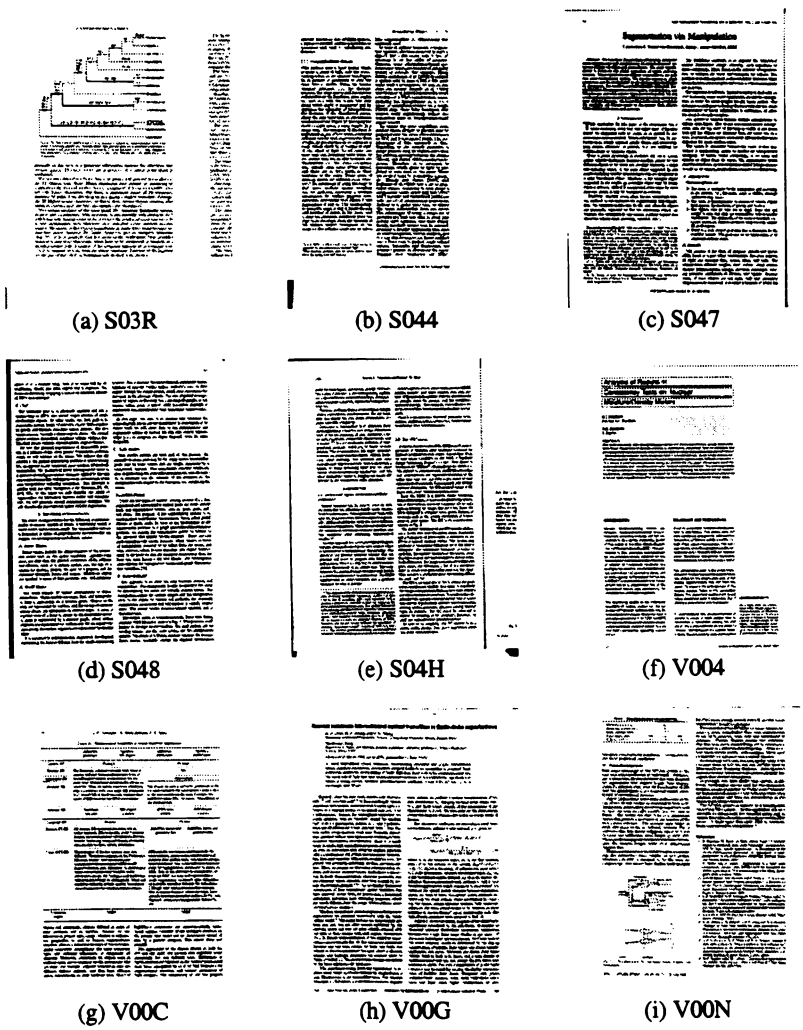
(a) I00J (b) I00L (c) I037 (d) J03M

(e) J04A (f) J04K (g) K009 (h) N01A

(i) N027 (j) N02A (k) N02K (l) N03K

(m) N048 (n) N05I (o) S021 (p) S03G

Figure A.3: The test images from the UW corpus (continued)



(a) S03R (b) S044 (c) S047

(d) S048 (e) S04H (f) V004

(g) V00C (h) V00G (i) V00N

Figure A.4: The test images from the UW corpus (continued)