Analysis and Design of Lossless Bi–level Image Coding Systems

by

Jianghong GUO

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2000

Abstract Lossless image coding deals with the problem of representing an image with a minimum number of binary bits from which the original image can be fully recovered without any loss of information. Most lossless image coding algorithms reach the goal of efficient compression by taking care of the spatial correlations and statistical redundancy lying in images. Context based algorithms are the typical algorithms in lossless image coding.

One key probelm in context based lossless bi–level image coding algorithms is the design of context templates. By using carefully designed context templates, we can effectively employ the information provided by surrounding pixels in an image.

In almost all image processing applications, image data is accessed in a raster scanning manner and is treated as 1–D integer sequence rather than 2–D data. In this thesis, we present a quadrisection scanning method which is better than raster scanning in that more adjacent surrounding pixels are incorporated into context templates. Based on quadrisection scanning, we develop several context templates and propose several image coding schemes for both sequential and progressive lossless bi–level image compression. Our results show that our algorithms perform better than those raster scanning based algorithms, such as JBIG1 used in this thesis as a reference.

Also, the application of 1–D grammar based codes in lossless image coding is discussed. 1–D grammar based codes outperform those LZ77/LZ78 based compression utility software for general data compression. It is also effective in lossless image coding. Several coding schemes for bi–level image compression via 1–D grammar codes are provided in this thesis, especially the parallel switching algorithm which combines the power of 1–D grammar based codes and context based algorithms. Most of our results are comparable to or better than those afforded by JBIG1.

Acknowledgements

Thanks to the supervision of Prof. E. Yang and Prof. J. Mark in Department of Electrical and Computer Engineering. Their serious attitude toward research sets a perfect example for me.

Thanks to Prof. G. Freeman and Prof. A. Khandani in Department of Electrical and Computer Engineering for their review and advice of this thesis. Special thanks to Prof. G. Freeman. I learn a lot from him.

Thanks to all the professors and friends who had helped me in my graduate study at the University of Waterloo. Wish you all good luck in your future work!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Data compression methods play important roles in data storage and transmission technology. In this chapter, we will present a brief review of data compression, including its origination, general methods, application and some international standards adopted in multi–media communication. We will also explain the meanings of some technical terms used in this thesis. The motivation, contributions and composition of this thesis are also included.

1.1 Terminology

Before we step into our subject of this thesis, we would like to explain the meanings for some technical terms used in this thesis.

- Lossless
  A compression algorithm is considered **lossless** if the original file can be fully recovered without any information loss.

- Lossy

  A compression algorithm is considered **lossy** if the original file can not be fully recovered with exactly the same binary data.

- Pixel

  A pixel refers to an element in an image if the image is described by a 2–D array.

- Bi–level image

  Bi–level images refer to those images that have only two colors, either 0 or 1.

- Gray–scale image

  Gray–scale images refer to those images whose colors take value in the range of $\{0, \cdots, 255\}$.

- High color image

  High color images refer to those images whose pixels can take value in the range of $\{0, \cdots, 2^{32}\}$.

- Context

  The context refers to those parts before or behind a message that determine its meaning. In image coding, a context is an integer calculated by some specific templates.

- Context template

  A context template used in image coding refers to some certain spatial pattern by which we determine the context for each pixel.

- Raster scanning

  Raster scanning means we access an image pixel by pixel, from left to right and from top to bottom.

- Sequential

  A coding technique shows sequential behavior if portions of the image near the top are completely described before portions below have been described at all [1].

- Progressive

  A coding scheme shows progressive behavior if an image is first coded as a lowest resolution layer image and then is successively increased in resolution by means of differential layer images [1].

- JBIG1

  JBIG is the abbreviation for Joint Bi–level Image Group. JBIG1 is the current international standard for bi-level image compression [1].

- JPEG

  JPEG is the abbreviation for Joint Photographic Experts Group. It is the current international standard for gray-scale picture compression [2].

1.2 Review of Data Compression

Data compression originates from english text compression. Books made of paper are difficult to keep for a long time. With the introduction of computers, books can be stored easily by simple converting into binary bytes. However, we can not afford the expense if we store all raw data without any processing. How to store text files efficiently using computers interests a lot of people from then on. Considering the fact that english grammar and particular contextual structures between adjacent characters and words exist, it is very natural to use multi–order Markov models to approach the statistical properties of text files. PPM(Prediction by Partial Match)

[3] [4] and CTW(Context Tree Weighting) [5] [6] algorithms give very competitive results in text compression. These algorithms share a common disadvantage of their complexity.

There are several algorithms that play very important roles in the history of the development of data compression and we should pay more attention to those algorithms. Generally, they are called entropy coders. These algorithms served as independent coders performing data compression when they were first introduced. But nowadays they are more likely to be used with the combination of other algorithms, often with model based algorithms which can produce certain probabilities to meet the need of entropy coders as input.

The two most common adopted entropy coders are:

- Huffman coder

  In Huffman coder, each symbol is assigned a pre-determined value according to its probability of appearance in source data. Huffman coder generally produces less compact output than arithmetic coder.

- Arithmetic coder

  The basic idea of adaptive arithmetic coder is to split an interval based on the conditional probability of the current symbol. Generally, the original interval is set to [0,1), then divided into smaller interval using conditional probability for the current symbol. The interval and accumulated frequencies are updated accordingly. For detail, please refer to [7].

Besides Huffman coder and Arithmetic coder, LZ77 and LZ78, the two general lossless compression algorithms proposed by Lempel and Ziv in 1977 and 1978 [8] [9], can also be used as entropy coders. In fact, any lossless compression algorithms can be regarded as entropy coders.

With the astonishing increasing speed of information technology and industry, there are more and more strong demands to perform data compression, not only to text files, but also to image/speech/video data because a typical multi–media computer file may contain text, image, speech and video data at the same time. For text files, a mistaken bit may result in a different character and that is the basic reason that all text compression algorithms are lossless ones. But sources such as image/speech/video rely on human perception systems deeply. For example, two images may seem the same to people but with totally different real binary data. Furthermore, in some applications, there is no need to fully recover original data. Some rebuilt images or speech signals of certain quality are acceptable and thus comes the concept of lossy compression. Most lossy compression algorithms employ the property of human perception systems to discard some information that is not so important to viewers or listeners, such as the high frequency components in images. For different kinds of information sources, different compression algorithms are proposed to capture different statistical properties.

- Lossless data compression

  Lempel and Ziv proposed two lossless data compression algorithm in 1977 [8] and 1978 [9]. The current compression utility software used in UNIX and WINDOWS are based on these two algorithms. Very recently, it is reported that the greedy grammar based codes proposed by Yang and Kieffer [10] get very good results in lossless data compression. For most standard test files, their algorithms perform better than LZ77/78.

- Speech coding

  Speech coding has been paid a lot of attention in the past fifty years. Due to the careful studies on the mechanism of human ears, some precise models are developed for speech coding. VQ(Vector Quantization), LPC(Linear Predictive Coder) and CELP(Codebook Encited Linear Predictive coder) are among the most successful speech compression algorithms. These algorithms can compress speech signal very efficiently. There are also some international standards for speech coding, e.g., G.723/G.729/GSM.

- Still image coding

  According to the color of images, still images can be classified into bi–level, half–tone, gray–scale and high color images. Among them, medical images are of very high resolution. For different type of images and applications, there are many image coding algorithms available for both lossless and lossy modes. Most of those algorithms are either context based or transform based. For bi–level image coding, there is an international standard called T.82 or JBIG1 [1]. JPEG [2] is the international standard for still continuous–tone picture compression. Transform based methods, including DCT(Discrete Cosine Transform) [2] and wavelet transforms [11] [12] [13], have got great achievement in lossy image coding.

- Video coding

  Although video signal can be considered as image sequences, video coding algorithms are a little bit different from still image coding algorithms. For a video data stream, there are not only spatial correlations between pixels in the same image frame, but also temporal correlations between adjacent image frames. For intra–frame coding, still image coding methods are generally adopted. Motion estimation and compensation are more widely used in coding inter–frames. Video coding is the key technique for video conferencing and video phones. For different transmitting channels, there are different international standards, such as H.320/H.321/H.324/, supporting data stream ranging from 32 kbits to 1 mega bytes per second. Consider the success of model based methods in speech coding, model based video coding has been paid a lot of attention. A model based international standard MPEG4 has been already developed for very low bit–rate video coding.

In Table 1.1, we list some current international standards with their applications.

Table 1.1: Some international standards for source coding

	Source	**L**ossless/Lossy	**A**pplication
JBIG1	Bi–level image	Lossless	Fax machine
JPEG	Gray–scale image	Lossless/Lossy	Image applications
MPEG1	Video & Audio	Lossy	Storage
MPEG2	Video & Audio	Lossy	VCD
MPEG4	Video & Audio	Lossy	Very low bit–rate video coding
H.261	Video	Lossy	Video conferencing
H.263	Video	Lossy	Video conferencing
H.263+	Video	Lossy	Video conferencing
G.723	Audio	Lossy	Audio applications
G.729	Audio	Lossy	Audio applications
GSM	Speech	Lossy	Wireless communication

1.3 Image Archives

Since we talk about lossless image coding, we would like to introduce the test images used in this thesis. There are 17 test images in total. 12 of the test images are of size 512×512 and the other 5 are of size 1024×1024. Our test images include human faces, animals, natural scenes and artificial objects. We can check the performance of our algorithms for images with different contents.

All test images in this thesis are retrieved from the Institute of Image and Signal Processing at University of Southern California except flower1 and flower2, which are generated by ourselves. Most of them are deemed as standard test images in bi–level image coding. Some test images are originally gray–scale images. We get bi–level images by taking the most significant bits from each pixel. Those images are shown in Figure 1.1.

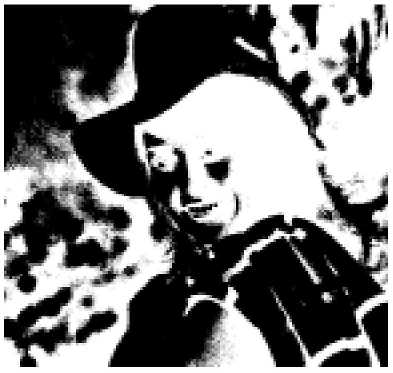
1.4 Motivation and Contributions

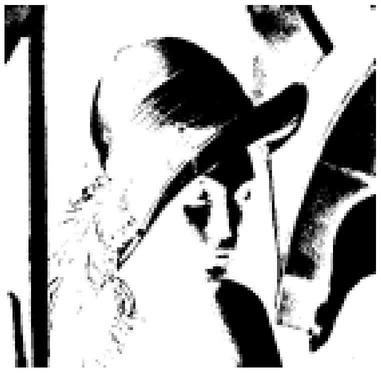
Although context based algorithms are effective in lossless image coding, those reported results are not the best we can reach. Part of the reason can be credited to the fact that images are accessed in a raster scanning manner in almost all image processing applications. The fact is that raster scanning is not the best scanning method for image coding because image data is treated as 1–D integer sequence rather than 2–D data. Our studies show that the adjacent surrounding pixels are more important than other pixels lying several positions away from the current pixel to be encoded though they are generally adopted in raster scanning based algorithms. We will show how to include those adjacent surrounding pixels as many as possible to reach the goal of efficient compression. Also, we will develop a specific

(a) bridge

(b) baboon

(c) barbara


(d) elaine


(e) lenna

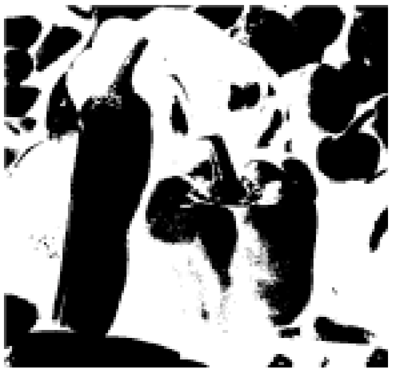
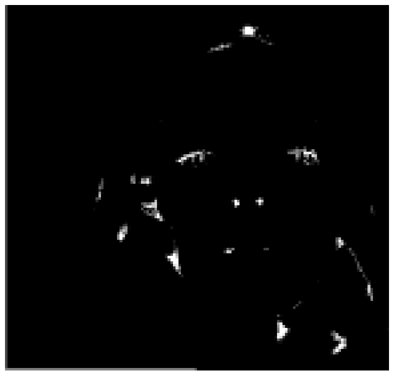
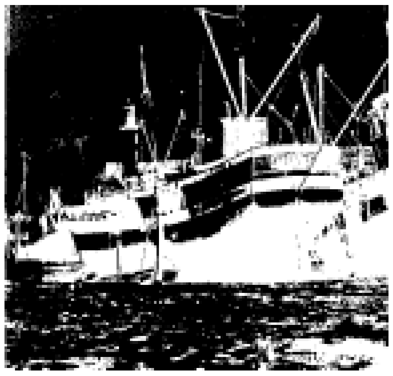

(f) peppers

Figure 1.1: Test images

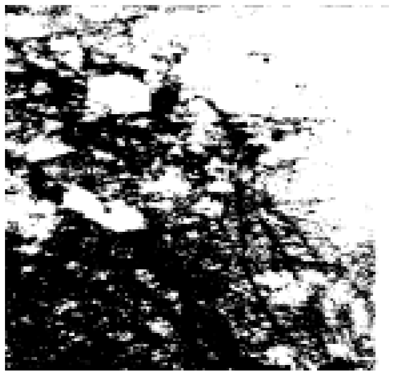
(g) sailboat



(h) tiffany

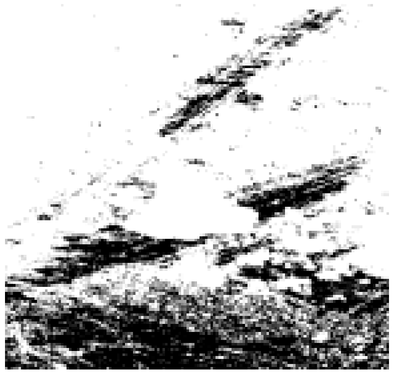


(i) boat



(j) car

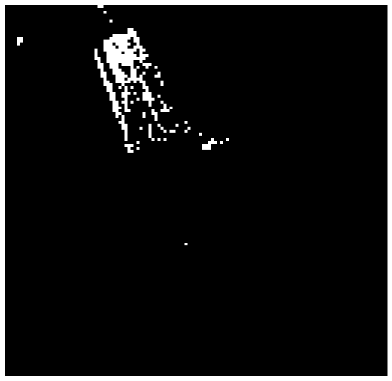
(k) couple



(l) tank

Figure 1.1: Test images cont'd (1)

(m) man


(n) airplane

(o) testpat

(p) flower 1

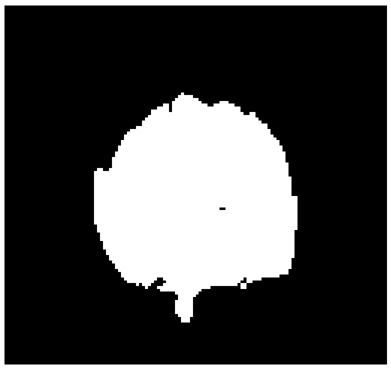

(q) flower2

Figure 1.1: Test images cont'd (2)

scanning method which scans image data in a quadrant by quadrant manner. We call this scanning method quadrisection scanning. By designing corresponding context templates based on quadrisection scanning, we propose several coding schemes for both sequential and progressive lossless bi-level image coding. Our proposed coding schemes provide very good results compared with those afforded by JBIG1.

Very recently, Yang and Kieffer proposed a new kind of universal lossless compression algorithm, which is called greedy grammar based codes [14]. The greedy grammar based codes perform better than those LZ77/78 based codes in general lossless text compression. Furthermore, it is shown that 1–D grammar codes are also effective in lossless image coding since they can exploit long term correlations between pixels in images. We will introduce 1-D greedy grammar based codes to lossless image coding in this thesis. Several coding schemes are presented with compression results. We will pay more attention to the parallel switching algorithm which combines the power of 1–D grammar codes and context based algorithms. Compression results for bi–level images are also provided. Most of our results are comparable to or better than those afforded by JBIG1. However, there are still some open questions regarding the 1–D grammar codes with their application in image coding need to be answered in the future.

1.5 Organization

The organization of this thesis is as follows. In Chapter 1, we briefly review the ideas of data compression and introduce some technical terms and our test images. Motivation and contributions together with the organization of this thesis are also included in Chapter 1. Chapter 2 is an introduction to lossless image coding and several existing coding algorithms are described in brief. In Chapter 3, we will turn to context based image coding algorithms and compare the results provided by existing algorithms with an ideal context model. Quadrisection scanning and context prediction methods are described and several sequential lossless bi–level image coding schemes based on them are provided in detail in Chapter 4, including encoding examples and compression results. Progressive coding of bi-level images is discussed in Chapter 5. 1–D grammar based codes with their application in bi–level image coding are presented in Chapter 6. Several coding schemes based on 1–D grammar codes are provided in detail with compression results, especially the parallel switching algorithm which combines the power of 1–D grammar based codes and arithmetic coding. We end up this thesis with our conclusions in Chapter 7.

Chapter 2

Introduction to Lossless Image Coding

In this chapter, we will briefly introduce the idea of lossless image coding. Several existing algorithms are described, especially JBIG1.

Lossless image coding deals with the problem of representing an image with a minimal number of binary bits from which the original image can be fully recovered without any loss of information. Since no information could be lost in lossless image coding, there is no need to talk about image quality. From Shannon's information theory, we know that theoretically the best result an algorithm can reach is the entropy rate of that specific source. However, for the same image, we may give different entropy rates from diffrent points of view.

Most lossless image coding methods can be classified into two categories: context based algorithms [2] [1] [15] [16] [17] [18] and transform based algorithms [19] [20]. Context based image coding algorithms provide very good results without introducing much complexity. Current international standard for lossless bi–level image coding JBIG1 [1] and gray–scale still picture compression JPEG [2] are the representatives of context based algorithms. In the past several decades, transform coding has been introduced in lossless still picture coding. $S+P$ transform [20] was reported to get very good results compared with JPEG in gray–scale image coding. However, the performance of transform based algorithms are still much worse than context based algorithms in lossless image coding.

2.1 Context Based Lossless Image Coding

It is well known that we can get very good compression results if we can know the statistical property of an information source and then set up an exact model based on that. In [21], Langdon and Rissanen proposed a method to compress bi–level images efficiently. This is actually the beginning of context based image coding algorithms.

In the past twenty years, context based algorithms have got great success in lossless image coding. Currently, the international standard JBIG1 [1] for bi–level image coding and the lossless mode of the international standard JPEG [2] for still image compression are all context based algorithms. Even the new standard JPEG2000 is still a context based algorithm in its lossless mode.

There are mainly two kinds of context based algorithms. One kind is to use a context template to predict the value of current pixel and then to encode the difference between the actual value of current pixel and the predicted value. Most context based lossless gray scale images coding methods [2] [15] work in this way. The other kind is to use a context template to classify current pixel into two groups: LPS (less probable symbol) and MPS (most probable symbol) and then to apply sequential arithmetic coder to finalize output such as JBIG1.

2.1.1 Finite State Machine Model

Since most digital systems can be regarded as finite state machines, in 1983, Langdon and Rissnan applied the idea of FSM (Finite State Machine) to bi–level image coding.

The FSM model for image compression is defined by the following equations:

$$x(t+1) = \mathrm{F}(x(t), s(t+1)) \tag{2.1}$$

$$z(t) = \mathrm{G}(x(t)) \tag{2.2}$$

where

- $s(t)$ denotes the t^{th} symbol in the string $s = s(1)s(2)\cdots$ to be modeled and encoded.
- $x(t)$ denotes the internal state.
- $z(t)$ is an output, called context.

If F maps a string to its last symbol with G the identity mapping, the FSM model actually defines a first–order Markov model. In Markov models, no distinction is made between the internal states and the conditioning class upon which the statistics are based. The FSM model is a generalization of the various versions of Markov models for stochastic processes in which the probability of the next state or the next symbol depends on a fixed number of the past observed values.

In Langdon and Rissanen's algorithm [21], F and G are determined by the neighboring pixels shown in Figure 2.1. The internal states $x(t)$ and $x(t+1)$ in the 7–pixel case are defined by:

$$x(t) = s(t-M-2)\cdots s(t) \tag{2.3}$$

$$x(t+1) = s(t-M-1)\cdots s(t+1) \tag{2.4}$$

where M is the number of pixels per row.

The above equations actually give the definition of function F(x(t),s(t+1)). Similar equations can be developed for 10–pixel case. The choice of 10–pixel case was done by experiments. The pixels two positions above and before pixel **x** are taken out because they are strongly dependent on the others and hence their inclusion adds little to the skewness of conditional counts for pixel **x**.

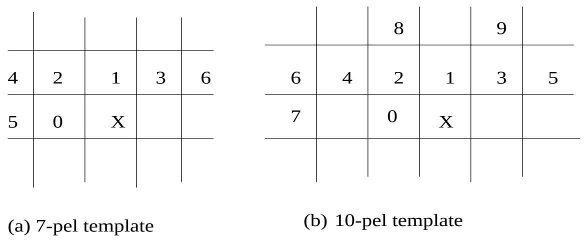


Figure 2.1: Surrounding pixels used by Langdon and Rissanen

Moreover, Langdon and Rissanen think that a compression system includes two distinct steps: modeling and coding. In the modeling step, a model is setup to estimate the statistical property of source data. Then the coding step takes care of this model and finalizes output. For example, the FSM model mentioned above sets up a very good model for image coding and can generate probabilities required by the coding step. In their algorithm, the coding step is done by an arithmetic coder. The meaning of coding here indicates the operation of generating output binary bits rather than compression.

2.1.2 Definition of The Context Template

In fact, the surrounding pixels concerned in determining the internal states of a FSM model define a context template used in image coding. Generally, a context template is composed of surrounding pixels around the pixel which we want to encode, as shown in Figure 2.2. The pixels incorporated in a context template should be those pixels that have been encoded before pixel **x** so that both encoder and decoder know the values of those pixels before processing pixel **x**. Each pixel included in the template contributes one bit to form the context. In this thesis, we refer the pixel to be encoded as pixel **x**. We mark those context pixels with numbers corresponding to their positions in the context.

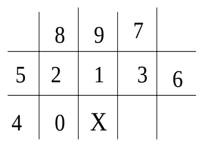


Figure 2.2: Example of the context template

The importance of context templates used in image coding lies in the fact that images are different from text files in having strong 2–D correlations, namely spatial correlations. Context templates can be regarded as certain patterns occurred in images. Due to the strong spatial correlations between adjacent pixels, it is very likely that certain patterns repeat under same contexts. Most context based algorithms are based on this assumption. However, large prediction errors often happen at edges in images, where abrupt intensity changes take place.

2.2 Lossless Bi–level Image Coding: JBIG1

Bi-level images refer to those images that have only two colors: black and white, which means each pixel takes only one of two possible values from $\{0, 1\}$. Generally speaking, facsimile and computer generated text files belong to this category. Bi–level image compression is the key technique in Fax machines because most Fax files can be regarded as bi–level images. Due to the large file sizes and limited bandwidth of the transmitting channel, e.g., the telephone line, we wish to compress Fax files before sending them. We also require that Fax files can be fully recovered to avoid being misunderstood. This asks the coding methods should be lossless and no error at the receiver end.

Bi–level image coding has been studied for a long time. Several efficient compression methods such as adaptive arithmetic coding [21], ITU–T Recommendation T.82 [1] and grammar based codes [22] have been proposed to compress bi–level images over the past several decades. T.82 provides high compression efficiency for bi–level images and was finalized to be the international standard for Fax files compression in 1993. It is also known as JBIG1. This recommendation was proposed by the Joint Bi–level Image Group (JBIG), a collaborative team work for ISO and CCITT. The specification of JBIG1 defines a coding method having single–progression sequential, progressive and progressive–compatible sequential modes and suggests a method to obtain any needed low–resolution renditions. Essentially, JBIG1 can be regarded as a finite context based 2–D arithmetic coder.

JBIG1 is the most successful international standard for Fax machines though there were some other international standards for Fax machines before JBIG1 was accepted in 1993. In JBIG1, an image is scanned and processed in a raster scanning manner, pixel by pixel, from left to right and from top to bottom. During the encoding process, image data is first divided into one or more resolution layers with each layer having one or more strips. An image layer in JBIG1 refers to the original image or a lower resolution image generated from a higher resolution image. A strip in JBIG1 is composed of a certain number of lines in the same image layer. If there is only one resolution layer, it is thus called single–progression sequential mode. If more than one resolution layers are required, e.g., in some applications that need lower resolution images for browsing, progressive and progressive–compatible sequential modes are better choices then. Single–progression sequential mode generally gives much better results than the other two modes.

2.2.1 JBIG1: Single Sequential Mode

JBIG1 is a context based algorithm. It is the extension of the idea in [21] to a carefully designed coder which encompasses strict data format definition. In JBIG1 single sequential mode, two specific context models defined by ten pixels are provided: one is a two line model and the other one is a three line model. In Figure 2.3, the two context templates used in JBIG1 are shown. Although those two context template models are defined in JBIG1, they are not absolutely fixed. Instead, there are some pixels called AT(Adaptive Template) pixels that can move in the neighboring area of pixel **x**. These pixels are marked as **A** in Figure 2.3. AT pixels are strictly confined to those pixels that have been encoded earlier. The function of those AT pixels is supposed to catch the periodic repeated patterns in images. However, AT pixels do not provide much gain in sequential mode though they do provide substantial gain in progressive mode and half–tone image compression.

When pixel **x** is the current pixel to be encoded, the corresponding context is

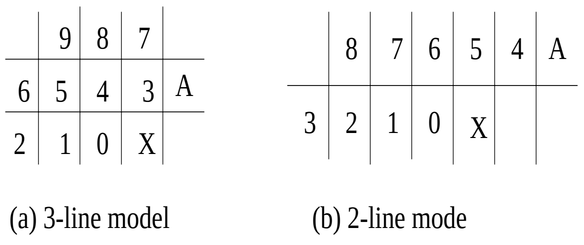


Figure 2.3: Context templates used in JBIG1

calculated first according to the context template. Suppose that the value of pixel **x** is 0 and the corresponding context is **C**. If there are already $n_{0,C}$ zeros and $n_{1,C}$ ones happened under this context, the probability of pixel **x** with value 0 is estimated by using the following equation.

$$\hat{p}(0/C) = \frac{n_{0,C} + \delta}{n_{0,C} + n_{1,C} + \delta}$$

where δ is a constant and set to the value of 0.45 in JBIG1.

2.2.2 JBIG1: Progressive Mode

Progressive and progression-compatible sequential modes are also supported by JBIG1. In JBIG1 progressive mode, a lower resolution image is generated from the original image with, as nearly as possible, half as many rows and half as many columns as the original. A lower resolution image is also called as a differential layer image. We can get lower resolution images as many as we wish by performing the same operation to the newly generated lower resolution image in each step. The following method is provided to get lower resolution images: in Figure 2.4, the value of pixel **x** in the lower resolution image is to be determined. Pixels $\{h_{11}, h_{12}, h_{13}, h_{21}, h_{22}, h_{23}, h_{31}, h_{32}, h_{33}\}$ in the higher resolution image and pixels $\{l_{00}, l_{01}, l_{10}\}$ in the lower resolution image are those pixels used to make the determination. The quantity of

$$4h_{22} + 2(h_{12} + h_{21} + h_{23} + h_{32}) + (h_{11} + h_{13} + h_{31} + h_{33}) - 3(l_{01} + l_{10}) - l_{00}$$

is formed. Pixel $\mathbf{x}$ is assumed to be 1 if the quantity is greater than 4.5; otherwise 0 is assigned.

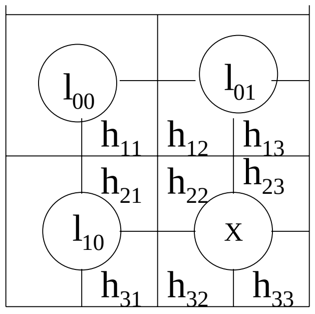


Figure 2.4: Lower resolution images rendition method used in JBIG1

There are four context templates defined in JBIG1 to encode a pixel in a differential layer image, as shown in Figure 2.5. Each template is composed of 6 pixels from the same differential layer and 4 pixels from the lower resolution layer. The phase describes the orientation of the pixel with respect to the lower resolution pixel associated with it. Two bits are used to describe phases. Thus we have 4096 different contexts in coding pixels in a differential layer.

The Q–coder [23] [24] [25] is utilized to compress pixel $\mathbf{x}$ for the final output. The Q–coder is different from a traditional arithmetic coder in that it approximates the probabilities generated from the above equation as powers of 2 and thus lowers implementation complexity for both software and hardware.

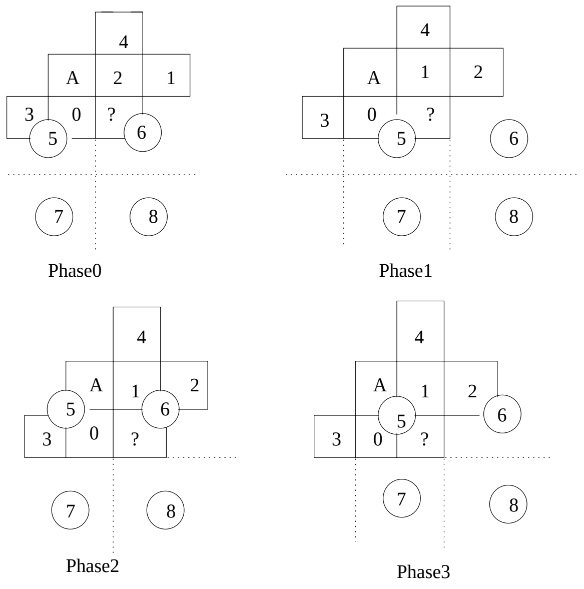


Figure 2.5: Context templates used for differential layer in JBIG1

2.3 Lossless Compression of Gray–scale Images: JPEG

The current international standard for still picture compression JPEG [2] supports four modes of operations, namely sequential DCT based mode, sequential lossless mode, progressive DCT based mode and hierarchical mode. All other modes are lossy ones using DCT transform coding except sequential lossless mode. The sequential lossless mode specifies two coding schemes:

1. Lossless method with Huffman coding.

2. Lossless method with arithmetic coding.

In JPEG lossless mode, a predictive coding technique instead of transform coding is used. The predicted value P_x for pixel $\mathbf{x}$ is calculated by using some linear combination of surrounding three pixels $\{A, B, C\}$ shown in Figure 2.6. JPEG defines eight prediction modes. In mode 0, no actual prediction is used. The definitions of predictors and corresponding index are shown in Table 2.1. R_a, R_b and R_c are the values pixels A, B and C. X is the value of pixel $\mathbf{x}$. A residual value $R = Px - X$ is then output to the entropy coder, either a Huffman coder or an arithmetic coder, to finalize output.

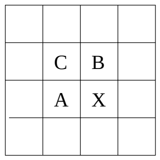


Figure 2.6: Predictor definition in JPEG lossless mode

The idea behind predictive coding is to expect that predictive models can capture the spatial correlations among pixels. If so, prediction residues would be mostly decorrelated. In fact, due to the effectiveness of predictive models, even directly plugging predictive residues into an entropy coder can achieve very good results, as what is done in JPEG lossless mode. But the results of JPEG lossless mode are not the best we can reach because the predictive models are fixed and sometime we are unable to capture the fast changes of edges in images. There are some other methods such as CALIC(Context-based Adaptive Lossless Image Codec) [26] that can adapt to the statistical property of images more quickly than JPEG lossless mode. LOCO–I [27] exploits the idea in CALIC and provides better results. It is adopted by JPEG2000 as the lossless coder for the new standard.

Table 2.1: Selection value and predictor design in JPEG

0	no prediction
1	$P_x = R_a$
2	$P_x = R_b$
3	$P_x = R_c$
4	$P_x = R_a + R_b - R_c$
5	$P_x = R_a + ((R_b - R_c)/2)$
6	$P_x = R_b + ((R_a - R_c)/2)$
7	$P_x = (R_a + R_b)/2$

2.4 Transform Based Lossless Coding of Images

Transform based image coding methods have been used in more and more image processing applications, especially in lossy image coding. The new international standard for JPEG2000 will be based on wavelet transforms. For lossy image coding, wavelet based methods generally provide better results than DCT based ones though some researchers argued that the improvement is due to the careful design of entropy coders instead of wavelet transform itself.

Recent studies show that wavelet transforms can also be used in lossless image coding, but the results reported are much more inferior to their lossy counterparts. Because of the nature of wavelet transforms, one can get multi–resolution representation of an image easily in lossless coding, which is difficult for context based methods to achieve. Since images are often browsed online, there are strong demands for such coding schemes which allow fast inspection, and only when necessary, exact recovery of the original image. Wavelet transform based methods can provide such coding schemes easily.

The key question in applying wavelet transform to lossless image coding is how to map integers to integers because image data can be regarded as 2–D integer arrays. In traditional wavelet transforms, transformed coefficients are real numbers, which makes it hard to keep all information due to the limited precision and accumulated errors of digital systems. However, it is still possible to design some wavelet transforms that can map integers to integers and avoid information loss. For example, the famous $S + P$ transform [20] and binary wavelet decomposition [19].

2.4.1 Binary Wavelet Decomposition

In [19], Swanson and Tewfic construct a theory of binary wavelet decompositions of finite binary images. The new binary wavelet transform uses simple modulo–2 operations. It shares many of the important characteristics of the real wavelet transform. In particular, it yields an output similar to the thresholded output of a real wavelet transform operating on the underlying binary image. They introduced a new binary field transform to use as an alternative to the discrete Fourier transform over GF(2). The corresponding concept of sequence spectra over GF(2) is defined. Using this transform, a theory of binary wavelet is developed in terms of 2–band perfect reconstruction filter banks in GF(2). By generalizing the corresponding real field constraints of bandwidth, vanishing moments and spectral content in the filters, a perfect reconstruction wavelet decomposition is constructed. They also demonstrate the potential use of the binary wavelet decomposition in lossless image coding.

Although this binary wavelet transform is effective in decreasing first order entropy, there is still no lossless image coding algorithm based on this transform reported. In Figure 2.7, image **Lenna** after 3–order binary wavelet transform is shown. We can see that spatial correlations are still existing. We all know that most wavelet transfom based coders such as EZW [12] and SPIHT [13] are all designed carefully to exploit the particular property of transformed coefficients while all the transformed coefficients still take value of either 1 or 0 in binary wavelet transform because it is defined over GF(2). Thus it is hard for us to design image coding scheme based on binary wavelet transform.

2.4.2 $S + P$ Transform

In [20], Said and Pearlman proposed a method combining the power of S transform and Hierarchical Interpolation.

Given an integer sequence $c[n], n = 0, \cdots, N-1$, with N even, we can represent this sequence by two sequences:

$$\begin{aligned} l[n] &= \lfloor (c[2n] + c[2n+1])/2 \rfloor, n = 0, \cdots, N/2 - 1 && (2.5) \\ h[n] &= c[2n] - c[2n+1], n = 0, \cdots, N/2 - 1 && (2.6) \end{aligned}$$

where $\lfloor . \rfloor$ denotes downward truncation operation. The sequences $l[n]$ and $h[n]$ form the S transform of $c[n]$.

The S transform is very efficiently in reducing first order entropy. However, it leaves a residual correlation between the highpass components. To overcome this problem, Said and Pearlman use some values of $l[n]$ and $h[n]$ to estimate a given $h[n]$.

Defining

$$\triangle l[n] = l[n-1] - l[n] \qquad (2.7)$$

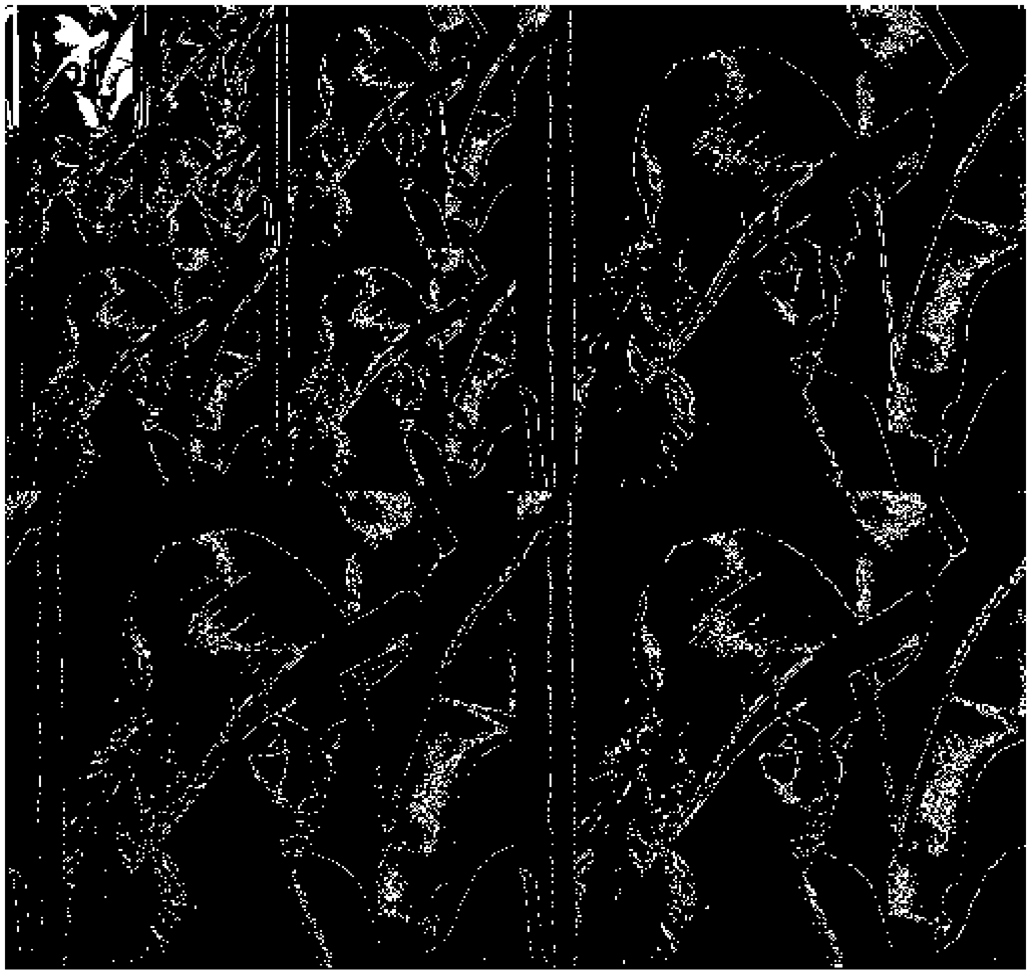


Figure 2.7: Binary wavelet transform of image **lenna**

then the general form of prediction is

$$\hat{h}[n] \;=\; \sum_{i=-L_0}^{L_1} \alpha_i \bigtriangleup l[n+i] - \sum_{j=1}^{H} \beta_j h[n+j] \tag{2.8}$$

where α_i and β_i are predictor coefficients that can be chosen arbitrarily.

Thus

$$h[n] \;=\; \lfloor \hat{h}[n] + 1/2 \rfloor + h_d[n],\; n = 0, 1, \cdots, N/2 - 1 \tag{2.9}$$

where $\hat{h}[n]$ is called an estimate of $h[n]$ and the sequence of $h_d[n]$ forms a lower resolution image.

$S + P$ transform provided very good results in both lossless and lossy gray–scale image coding compared to the results afforded by JPEG. Please refer to [20] for details.

2.5 Other Methods

Besides context based and transform based lossless image coding methods, there are also some other important algorithms, inluding run length coding [2] and pattern matching [28].

Chapter 3

Analysis of Context Based Algorithms

In this chapter, we will show our studies on context based bi–level image coding algorithms. We will also show that the current results provided by existing context based algorithms based on raster scanning are not the best and point out several possible solutions to improve the performance of context based algorithms.

3.1 Definitions and Notations

In early image processing studies, the **adaptive self entropy** of a bi–level image was defined as:

$$H(I) \quad = \quad \sum_{i=1}^{N} log_2 \frac{n_0 + n_1}{n_{p_i}} \qquad (3.1)$$

where

- N is the length of an image file;

- p_i represents the value of the current pixel to be encoded;
- n_0 and n_1 are the counts of 0 and 1 occurred in previous encoding steps before the i^{th} pixel.

In context based image coding algorithms, we can define **conditional self entropy** of a bi–level image with regard to a specific context template model as [17]:

$$H(I) \quad = \quad \sum_{C}(n_{0,C} log_2 \frac{n_{0,C}+n_{1,C}}{n_{0,C}} + n_{1,C} log_2 \frac{n_{0,C}+n_{1,C}}{n_{1,C}}) \tag{3.2}$$

where

- C is the context given by the specified context template for the current pixel;
- summation is over all possible context C;
- $n_{0,C}$ and $n_{1,C}$ are the counts of 0 and 1 occurred under context C in that image.

For i^{th} pixel taken from an image, we use the following equation to calculate its conditional probability:

$$\hat{p}(x/C_i) \quad = \quad \frac{n_{x,C_i}}{n_{0,C_i}+n_{1,C_i}} \tag{3.3}$$

and theoretical code length is calculated as follows:

$$\text{Theoretical Code Length} \quad = \quad \sum_{i=1}^{N} log_2 \frac{n_{0,C_i}+n_{1,C_i}}{n_{x,C_i}} \tag{3.4}$$

where

- N is the length of an image file;
- x is the value of the i^{th} pixel to be coded;

- C_i is the corresponding context of the i^{th} pixel;
- n_{0,C_i} and n_{1,C_i} are the counts of 0 and 1 occurred under context C_i in that image.

To evaluate the compression performance, we define **compression ratio** as:

$$C_{ratio} = \frac{\text{Original Image File Size}}{\text{Encoded File Size}} \tag{3.5}$$

and **compression rate** as:

$$C_{rate} = \frac{\text{Encoded File Size}}{\text{Original Image File Size}} \tag{3.6}$$

Compression ratio actually indicates the compression efficiency and compression rate denotes the number of binary bits required for each pixel. We can easily see that $C_{ratio} \times C_{rate} = 1$. In this thesis, we will use C_{rate} to compare our compression results with those afforded by JBIG1. A direct comparison of compressed file sizes is also adopted in this thesis.

3.2 Analysis of Context Based Algorithms

Raster scanning based algorithms, such as JBIG1, have some apparent advantages, e.g., low memory requirement and low implementation complexity. But their performance are not the best that we can get from context based algorithms. In this section, we will first compare the results of JBIG1 with the results we get by using an ideal context model. After that, we will discuss how to encode an image optimally using context templates.

3.2.1 An Ideal Context Template

In almost all existing context based image coding algorithms [21] [1] [2], the context template is composed of pixels above or before pixel $\mathbf{x}$ because these pixels are definitely available when we process an image in a raster scanning manner. Although algorithms based on those context templates and scanning methods provide very good results, they are not the best we can reach.

Let us look at the following examples. At first, we construct a context template model composed of all 8 adjacent surrounding pixels $\{0, 1, 2, 3, 4, 5, 6, 7\}$, as shown in Figure 3.1. It is an ideal model which we can never put into practice. Suppose we know all these 8 pixels when we encode pixel $\mathbf{x}$, we then calculate the number of binary bytes we need to encode the whole image based on this template for our test images. The equation to calculate the theoretical code length is used. The results are shown in Table 3.1, labeled as template 0. We can see that these numbers of binary bytes required for our test images are much smaller than those achieved by JBIG1, whose results are labeled as JBIG1 in Table 3.1. The differences in compression results may range from 20% to 200%. In [17], Moffat showed that a context template like the one shown in Figure 3.2 which gives optimal conditional entropy for most of his test images. It seems that we can improve the performance of context based image coding algorithms a lot if we can include all those 8 pixels into the actual context template used in coding process. The disappointing fact is that we can't do this because pixels $\{4, 5, 6, 7\}$ are never available when we access an image using raster scanning.

Moreover, even a context template composed of pixels $\{0, 1, 2, 3, 4\}$ gives much better results than JBIG1. These results are also shown in Table 3.1, labeled as template 1. The differences in compression efficiency range from 5% to 50%. With

2	1	3
0	x	6
4	5	7

Figure 3.1: An ideal context template

the introduction of this single pixel 4, we can achieve so much gain over JBIG1. It is really beyond our expectation because we can only get some small gain by setting up some complex models which may include up to 32 pixels lying above or before pixel **x**. All those results lead to the conclusion that pixels $\{4, 5, 6, 7\}$ are more important in the selection of a context template than those pixels lie several positions above or before pixel **x**. This can be explained partly by the fact that pixels $\{4, 5, 6, 7\}$ provide some future information about pixel **x** when we use raster scanning to access image data. However, the information provided by those pixels are discarded when we stick to raster scanning. In this thesis, we will try to develop some ways to employ these information.

			14			
	8	2	1	3	9	
	10	0	x	5	11	
	12	4	6	7	13	
			15			

Figure 3.2: Moffat's optimal context template

Table 3.1: Comparison of theoretical code lengths with JBIG1

	Size	**T**emplate 0	**T**emplate 1	**J**BIG1
lenna	512×512	3391	4228	4870
baboon	512×512	12466	15634	17299
barbara	512×512	5345	8037	8234
peppers	512×512	2699	3489	4024
sailboat	512×512	2989	4314	5094
tiffany	512×512	432	639	928
bridge	512×512	6753	8530	9812
car	512×512	6503	8291	9656
couple	512×512	5140	7389	8173
elaine	512×512	5955	6724	7503
tank	512×512	7245	8948	9988
boat	512×512	4313	6056	7064
airplane	1024×1024	3333	4213	4380
airport	1024×1024	21847	26480	28676
man	1024×1024	14889	19127	21881
flower1	1024×1024	3640	6945	8528
flower2	1024×1024	434	892	1251

3.2.2 Optimality Analysis

From the results shown above, we know that if we can include pixels $\{4, 5, 6, 7\}$ into the context template, we will improve the performance of context based algorithms a lot. However, the following theorem points out that it is impossible for us to have all 8 adjacent surrounding pixels in the ideal context template available for all the pixels in an image for the purpose of encoding no matter what kind of scanning methods we use.

Theorem 1 *No matter what kind of scanning method is used, there are at most* 25% *pixels in an image with all* 8 *directly adjacent pixels available (as shown in Figure 3.1) in the coding process. For these* 25% *pixels, the encoding is considered optimal in the sense that they include more surrounding pixels in their context templates than the remaining pixels. Moreover, there does exist a scanning method that can fulfill this goal.*

It is easy for us to prove Theorem 1. We know that any unencoded pixel can not be included in the ideal context template for pixel $\mathbf{x}$, which means that any two unencoded pixels are not directly adjacent to each other. When there is only one encoded pixel lying between any two unencoded pixels, we get the largest number of the pixels in an image can have all 8 adjacent surrounding pixels available for encoding purpose, as shown in Figure 3.3.

Theorem 1 points out that we can use the ideal context model to encode 25% pixels in an image. However, we still need to find practical ways to encode the remaining 75% pixels optimally even though we can encode these 25% pixels optimally. From our studies in the previous section and Theorem 1, it is very natural for us to reach this goal through the following 3 ways:

- Two pass coding method

  For example, we can construct such a coder which has two context templates: one is a JBIG1 like one and the other one is the ideal template model. All white pixels (as shown in Figure 3.3) are encoded using the JBIG1 like model in the first pass and then black pixels are dealt with by using the ideal model. But this method may not give overall optimal results because these black pixels included in the context templates to encode white pixels are not available in the process of encoding. Any changes in encoding order may influence the statistical property observed in encoding steps though we may reach the same final statistical states. We will show a coding scheme based on this idea in Chapter 5.

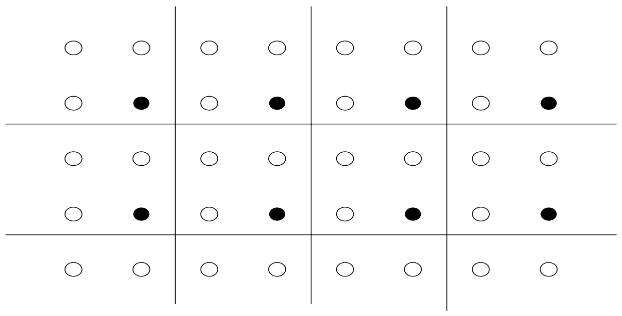


Figure 3.3: Analysis of scanning manner

- Multi–resolution coding methods

  If we follow the idea in Theorem 1, we can construct another coding scheme. Since for 25% pixels in an image, we can encode them optimally in a certain sense, we now only need to find a way to encode the remaining 75% pixels optimally. For example, we can treat those pixels in the same corner of each 2×2 block as a sub–image and then encode each sub–image following the idea in Theorem 1. If we keep on sub–sampling each sub–image into 2×2

blocks, we actually construct a multi–resolution pyramid representation of the original image. We now have more pixels to be encoded based on the ideal context model. However, the above algorithm still may not be overall optimal due to the fact that we can not employ all the local information when we sub–sample and encode pixels an image.

- New scanning methods

  If we can design some new scanning methods other than raster scanning which can give us the opportunity to include all or some of pixels $\{4, 5, 6, 7\}$ in the context template, we can also get some improvement. Since an image can be regarded as composition of many 2×2 blocks, we now can access image data in such a way: to scan an image as the concatenation of many 2×2 blocks. This is the basic idea of quadrisection scanning methods which we will discuss later in detail.

3.3 Remark

In this chapter, we analyze context based algorithms and show that we can improve the compression efficiency if we can introduce more directly adjacent surrounding pixels into the context template for encoding purpose . We also point out several ways to fulfill this goal. In the following Chapter 4 and Chapter 5, we will discuss some lossless bi–level image coding schemes employing those ideas.

Chapter 4

Sequential Lossless Bi–level Image Coding

In this chapter, we will introduce the definition of quadrisection scanning and design some context templates based on quadrisection scanning. Also, we will present several sequential image coding schemes. All coding schemes discussed in this chapter have been implemented and tested for their correctness.

4.1 The Design of Context Template

In this section, we will introduce the definition and properties of quadrisection scanning. In quadrisection scanning, we access pixels in an image in a quadrant by quadrant manner which gives us the opportunity to include more directly adjacent pixels in context templates. An extended context template based on quadrisection scanning is also developed.

4.1.1 Quadrisection Scanning: Definition and Property

Our studies in Chapter 3 show that with the introduction of more directly adjacent pixels in context templates, we can definitely improve the performance of context based algorithms. But we can not incorporate pixels $\{4, 5, 6, 7\}$ if we stick to raster scanning method. To overcome this, we introduce quadrisection scanning. In quadrisection scanning, we access image data in a quadrant by quadrant manner instead of row by row in raster scanning. There are two kinds of quadrisection scanning methods: one by row major and one by column major. Our experimental results show that our algorithms based on the former quadrisection scanning generally provide better compression results than the latter one. We refer to the first kind quadrisection scanning in this thesis unless otherwise specified.

0	1	4	5
2	3	6	7
8	9	12	13
10	11	14	15

(a) Row major

0	2	8	10
1	3	9	11
4	6	12	14
5	7	13	15

(b) Column major

Figure 4.1: Quadrisection scanning

Quadrisection scanning has the following properties:

Property 1 Pixels lie before or right above pixel **x** are always scanned before pixel **x**.

Property 2 Whenever pixel **4** in the ideal context template (Figure 3.1) is available, pixel **3** is available too. On the contrary, when pixel **3** is available, pixel **4** may not be available. This can be clarified when we encode the 9^{th} pixel in Figure 4.1.

Property 3 The frequency of pixel **3** being available is about $\frac{2}{3}$ and that of both pixels $\{3, 4\}$ being available is about $\frac{1}{3}$.

Property 3 can be shown as follows:

- The frequency of pixel 3 in the ideal context template being available.

  Assume a_n is the number of pixels in a $2^n \times 2^n$ block with pixel 3 being available in a quadrisection scanning manner, then

$$a_1 = 3; \tag{4.1}$$

$$a_n = 4 \times a_{n-1} - 1; \tag{4.2}$$

so

$$\begin{aligned} a_n &= 3 \times 4^{n-1} - 4^{n-2} - 4^{n-3} - \cdots - 4 - 1 \\ &= 3 \times 4^{n-1} - \frac{4^{n-1} - 1}{3} \\ &= \frac{2 \times 4^n + 1}{3} \end{aligned} \tag{4.3}$$

and we get frequency(pixel 3 being available) $= \frac{a_n}{2^n \times 2^n} = \frac{2}{3}$.

- The frequency of pixel 4 in the ideal context template being available.

  Assume b_n is the number of pixels in a $2^n \times 2^n$ block with pixel 3 being available in a quadrisection scanning manner, then

$$b_1 = 2; \tag{4.4}$$

$$b_n = 4 \times b_{n-1} - 2; \tag{4.5}$$

so

$$\begin{aligned} b_n &= 2 \times 4^{n-1} - 2 \times 4^{n-2} - 2 \times 4^{n-3} - \cdots - 2 \times 4 - 2 \\ &= 2 \times 4^{n-1} - 2 \times 4^{n-2} + 4^{n-3} - \cdots + 4 + 1 \\ &= 2 \times 4^{n-1} - \frac{2 \times 4^{n-2} - 2}{3} \\ &= \frac{4^n + 2}{3} \end{aligned} \tag{4.6}$$

and we get frequency(pixel 4 being available)$= \frac{b_n}{2^n \times 2^n} = \frac{1}{3}$.

4.1.2 An Extended Context Template

Since in quadrisection scanning we have the opportunity of pixel 4 being available sometimes, we now define a context template composed of pixels $\{0, 1, 2, 3, 4\}$, as shown in Figure 4.2. We call it an extended context template due to the appearance of pixel 4 in the context template. In this context template, pixels $\{0, 1, 2\}$ are always available because they lie right above or before pixel **x**. Although pixel **3** is always available to encode pixel **x** when we use raster scanning, it is sometime not available in quadrisection scanning. However, we can use some special techniques to overcome this bad effect such as some prediction methods.

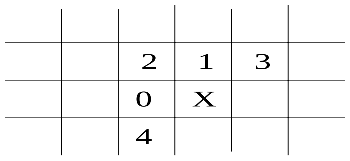


Figure 4.2: An extended context template

4.2 Lossless Image Coding based on Quadrisection Scanning

In most applications, single sequential coding is generally acceptable. In this section, we will present several sequential lossless image coding schemes based on quadrisection scanning with coding steps described in detail. Compression results are provided in next section.

4.2.1 Sequential Compression: Scheme 1

In this coding scheme, the context template is the extended context template descibed earlier, as shown in Figure 4.2. If pixel **3** is available when we encode pixel **x**, the actual value is inserted into the context template; otherwise default value 1 is used since we know this often happens in Fax files. We do the same operation to pixel **4**. If a pixel in the context template is beyond boundary, it is regarded as having a value of 1. A binary arithmetic coder is used to finalize the output.

Because we now have 5 pixels incorporated in the context template, we have 32 possible contexts and 2 possible values under each context. All cumulative frequencies for each symbol under each context are initialized to 1 at the begining. We update the corresponding cumulative frequency after we encode pixel **x** according to its current context and value. If the cumulative frequency of a symbol under a context exceeds certain amount, e.g., 16383, then all cumulative frequencies under that context are then decreased in half. This is actually handled by the binary arithmetic coder.

Figure 4.3 shows a 8×8 block taken from image **lenna**. The number in the upper left corner in each small square denotes the color of the pixel in that position

1 / 1	1 / 2	1 / 5	1 / 6	1 / 17	1 / 18	1 / 21	1 / 22
1 / 3	1 / 4	0 / 7	1 / 8	1 / 19	1 / 20	1 / 23	0 / 24
1 / 9	1 / 10	1 / 13	1 / 14	1 / 25	0 / 26	1 / 29	0 / 30
1 / 11	1 / 12	1 / 15	1 / 16	1 / 27	1 / 28	1 / 31	1 / 32
1 / 33	1 / 34	1 / 37	1 / 38	1 / 49	0 / 50	1 / 53	1 / 54
1 / 35	0 / 36	1 / 39	1 / 40	1 / 51	1 / 52	0 / 55	1 / 56
1 / 41	0 / 42	1 / 45	1 / 46	1 / 57	1 / 58	1 / 61	1 / 62
1 / 43	1 / 44	1 / 47	1 / 48	1 / 59	1 / 60	0 / 63	1 / 64

Figure 4.3: Image block for encoding example

and the number in the lower right corner indicates scanning order, or say encoding order. The encoding steps of this block are shown in Table 4.1 with columns labeled as context 1 and binrate 1.

The following psuedo codes show the encoding process for an image file. P0 in the codes indicates the value of pixel 0 in the context template and so do other pixels. Px is the value for the current pixel **x**.

```
init_arithmetic_coder();
         //initialize the arithmetic coder;
init_scan_order();
         //generate qurdrisection scanning order array
while(!EOF)
{  read(pixel x);
   context = 0;
```

```
        context += p0+2*p1+4*p2;
        if(pixel 3 encoded)
            {context +=8*p3;}
        else
            {context +=8;}
        if(pixel 4 encoded)
            {context +=16*p4;}
        else
            {context +=16;}
        encode_symbol(context, px);
               //output binary bits
        update_model(context, px);
               //update context model
    }
```

The results for scheme 1 are listed in the first column of Table 4.2 for our test images. We can see that for most images, this scheme provides results comparable to or better than those afforded by JBIG1.

This scheme is apparently simple and easy to implement. The disadvantage is that we may not be able to catch the statistical property of an image because we set default values for unknown pixels to 1. However, we may expect to avoid this bad effect if an image is largely composed of 1.

4.2.2 Context Prediction: Scheme 2

Since the frequency of pixels **3** and **4** being available in the extended context template at the same time only equals $\frac{1}{3}$, how to effectively utilize information provided by other adjacent pixels is very important in getting good results. In fact, not only the value of pixel **x** can be predicted by corresponding context, the values of pixel $\{3, 4\}$ can be predicted by their adjacent pixels too. If either of them is not available, we then use some prediction methods to predict its value. In this section, we present a coding scheme which predicts values of unknown pixels incorporated in the extended context template.

In JPEG lossless mode, eight prediction modes and predictors are defined [2]. The prediction strategy used in JPEG can be easily applied to our coding scheme. The context template used in this scheme is the same as the one used in scheme 1 and is composed of pixel $\{0, 1, 2, 3, 4\}$ as shown in Figure 4.2. If a pixel in the context template is not available, the value of the pixel which is directly adjacent to it is used to determine its value. For example, if pixel 3 is not available, the value of pixel 1 is put into context template (which is mode 1 in JPEG prediction modes) instead of a default value 1. So does pixel 0 to pixel 4 (which is mode 3 in JPEG lossless prediction modes). If a pixel in the context template is beyond boundary of the image, it is then regarded as having a value of 1. The same binary arithmetic coder used in scheme 1 is used for the final output. The cumulative frequency updating mechanism is the same as that in scheme 1.

Figure 4.4 shows the basic encoder structure for this coding scheme.

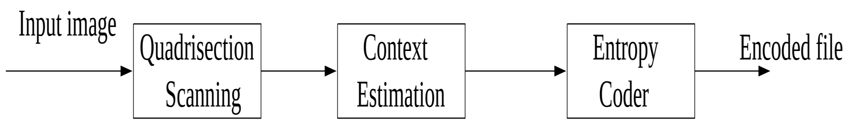


Figure 4.4: Encoder structure using context prediction

We give the encoding steps using this scheme for the block in Figure 4.3 in Table 4.1 with columns labeled as context 2 and binrate 2.

The following psuedo codes show the process for an image file using scheme 2. The symbols have the same meanings that we explained in scheme 1.

```
init_arithmetic_coder();
         //initialize the arithmetic coder;
init_scan_order();
         //generate qurdrisection scanning order array
while(!EOF)
{  read(pixel x);
   context = 0;
   context += p0+2*p1+4*p2;
   if(pixel 3 encoded)
        {context +=8*p3;}
   else
        {context +=8*p0;}
   if(pixel 4 encoded)
        {context +=16*p4;}
   else
        {context +=18*p1;}
   encode_symbol(context, px);
   update_model(context, px);
  }
```

The results for scheme 2 are listed in the second column of Table 4.2 for our test images. We can see that for most images, this scheme provides better results than those afforded by JBIG1 and our coding scheme 1 as we expected. Since a context template is used to predict the value of pixel **x**, it is also feasible for us to predict the values of unknown pixels incorporated in context templates. However, due to the reason similar to scheme 1, we may get lost from the real statistical property of an image if changing edges are frequently encountered.

4.2.3 The 4–line Context Template: Scheme 3

Compression results from previous sections show that our coding schemes based on quadrisection scanning perform better than JBIG1. Furthermore, we can easily improve our algorithmss by refining our context template models.

The extended context template used in scheme 1 and scheme 2 are rather simple and are only composed of 5 pixels. Consider the fact that the 3–line and the 2–line models used in JBIG1 all contain 10 pixels, we construct a 4–line context template incorporating 11 pixels, as shown in Figure 4.5. Among those 11 pixels, 9 of them are marked with numbers and 2 of them are marked as $a1$ and $a2$ separately. In [21], Langdon and Rissanen pointed out that the pixels marked as $a1$ and $a2$ are not recommended to included in context templates. Our experiments confirm their suggestion.

From property 1 of quadrisection scanning, we know that the frequency of pixel **7** in Figure 4.5 being available is no less than that of pixel **3**. Similarly, the frequency of pixel **8** in Figure 4.5 being available is no less than that of pixel **4**. We also know that the future pixels provide more information. Therefore, we introduce pixel $\{7, 8\}$ into the 4–line model. Although pixel $a1$ and $a2$ are not used in the final compression, we use them to predict the values of pixel 7 and pixel 8. For example, we use pixel $a1$ to predict the value of pixel 7.

Our coding scheme based on the 4–line context template provides very good results for most of our test images. For example, for image **barbara**, the result provided by this coding scheme is 7937 bytes while for scheme 2 it is 8773 bytes. For most images of size 512×512 and 1024×1024, scheme 3 provides the best results.

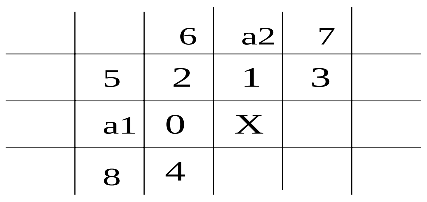


Figure 4.5: The 4–line context template

4.3 Compression Results and Complexity Analysis

In this section, we will present detailed encoding steps of the image block in Figure 4.3 and show compression results for our test bi–level images. Complexity problems are also discussed.

4.3.1 Encoding Examples

In Table 4.1, we show the detailed encoding steps of the image block in Figure 4.3 for scheme 1 and scheme 2. Context 1 are thse contexts used to encode the corresponding pixels for scheme 1 and so do context 2 to scheme 2. Rate 1 and rate 2 indicate the code lengths in each step.

Table 4.1: Encoding example

step	**P**ixel value	**C**ontext 1	**R**ate 1	**C**ontext 2	**R**ate 2
1	1	31	2/1	31	2/1
2	1	31	3/2	31	3/2
3	1	31	4/3	31	4/3
4	1	31	5/4	31	5/4
5	1	31	6/5	31	6/5
6	1	31	7/6	31	7/6
7	0	31	8/1	31	8/1
8	1	30	2/1	14	2/1
9	1	31	9/7	31	9/7
10	1	23	2/1	23	2/1
11	1	31	10/8	31	10/8
12	1	31	11/9	31	11/9
13	1	29	2/1	29	2/1
14	1	27	2/1	27	2/1
15	1	31	12/10	31	12/10
16	1	31	13/11	31	13/11
17	1	31	14/12	31	14/12
18	1	31	15/13	31	15/13
19	1	31	16/14	31	16/14
20	1	31	17/15	31	17/15
21	1	31	18/16	31	18/16
22	1	31	19/17	31	19/17

Table 4.1: Encoding example, cont'd (1)

step	Pixel value	Context 1	Rate 1	Context 2	Rate 2
23	1	31	20/18	31	20/18
24	0	31	21/2	31	21/2
25	1	31	22/19	31	22/19
26	0	31	23/3	31	23/3
27	1	23	3/2	23	3/2
28	1	29	3/2	21	2/1
29	1	22	2/1	22	2/1
30	0	29	4/1	29	3/1
31	1	19	2/1	19	2/1
32	1	29	5/3	29	4/2
33	1	31	24/20	31	24/20
34	1	31	25/21	31	25/21
35	1	31	26/22	31	26/22
36	0	31	27/4	31	27/4
37	1	15	2/1	15	2/1
38	1	31	28/23	31	28/23
39	1	30	3/2	14	3/2
40	1	31	29/24	31	29/24
41	1	23	4/3	23	4/3
42	0	29	6/2	29	5/2
43	1	23	5/4	23	5/4
44	1	29	7/4	21	3/2

Table 4.1: Encoding example, cont'd (2)

step	**P**ixel value	**C**ontext 1	**R**ate 1	**C**ontext 2	**R**ate 2
45	1	26	2/1	26	2/1
46	1	31	30/25	31	30/25
47	1	27	3/2	27	3/2
48	1	31	31/26	31	31/26
49	1	31	32/27	31	32/27
50	0	31	33/5	31	33/5
51	1	23	6/5	23	6/5
52	1	29	8/5	21	4/3
53	1	20	4/3	30	2/1
54	1	31	34/28	31	34/28
55	0	27	4/1	27	4/1
56	1	30	5/4	14	4/3
57	1	31	35/29	31	35/29
58	1	23	7/6	23	7/6
59	1	31	36/30	31	36/30
60	1	31	37/31	31	37/31
61	1	29	9/6	29	6/3
62	1	27	5/3	27	5/3
63	0	31	38/6	31	38/6
64	1	30	6/5	30	3/2

4.3.2 Compression Results

In Table 4.2, our compression results for bi–level images of size 512×512 and 1024×1024 in sequential mode are presented separately. All the compression results are represented by the final file size in bytes after encoding without any additional information. Scheme 1 refers to the coding scheme presented in Section 4.2.1. Scheme 2 is the one described in Section 4.2.2. Scheme 3 is the 4–line model based coding scheme.

The results for JBIG1 are given by the JBIG1 software implemented by M. Kuhn. The source code can be found at [29]. These results contain some additional information about parameters used in compression. Becuase this part contributes only around 20 bytes for each image in the final file size, it would not affect the differences between the results provided by JBIG1 and our coding schemes too much.

From those results listed in Table 4.2, we can see that our coding schemes give better results than those afforded by JBIG1 in sequential mode for most test images. Our improvements range from 5% to nearly 35% for different images. Scheme 3 generally gives the best compression results for sequential compression.

4.3.3 Complexity Analysis

Our proposed coding schemes introduce the quadrisection scanning method and context prediction techniques in lossless bi–level image coding. These ideas are pretty simple and easy to implement. However, because we use two arrays to record the processing order of quadrisection scanning in our coding schemes, if an image is of size 1024×1024, these two array may require a memory space about

Table 4.2: Compression results in sequential mode

	Size	**S**cheme 1	**S**cheme 2	**S**cheme 3	**J**BIG1
lenna	512×512	5142	4900	4738	4870
baboon	512×512	16343	16305	16192	17299
barbara	512×512	8962	8773	7937	8234
peppers	512×512	4334	4165	3995	4024
sailboat	512×512	5018	4880	4793	5094
tiffany	512×512	743	716	725	928
boat	512×512	6883	6730	6592	7064
bridge	512×512	9469	9226	9171	9812
car	512×512	9208	8965	8977	9656
couple	512×512	8073	7912	7691	8173
elaine	512×512	7799	7567	7232	7503
tank	512×512	9550	9319	9299	9988
airplane	1024×1024	4467	4270	3967	4380
airport	1024×1024	29711	28733	27744	28676
man	1024×1024	22223	21333	20537	21881
flower1	1024×1024	9300	8715	8310	8528
flower2	1024×1024	1136	1060	1031	1251

12 megabytes. This may cause a problem when working online. This can be easily overcome by hardware implementation.

Another disadvantage of our coding schemes is that there might be a large time delay since we have to wait to process a whole $2^n \times 2^n$ block in an image. This is different from JBIG1 which processes an image line by line and almost needs no time delay. However, no noticeable time delay is observed when we compare our coding schemes with JBIG1.

4.4 Remarks

From the compression results shown in Table 4.2, we conclude that our coding schemes based on quadrisection scanning and context prediction technique perform pretty well in sequential lossless bi–level image coding. There is still room for us to improve our algorithms if we can design more accurate prediction methods.

Our results also show that scanning methods play very important roles in lossless bi–level image coding because we can set up different context template models to take care of the 2–D correlation information in an image as much as we can. Although quadrisection scanning is used in our algorithms, this scanning method may not be the best. We are expecting to get some improvement by designing more efficient scanning methods.

Chapter 5

Progressive Lossless Bi–level Image Coding

In Chapter 4, we present several sequential lossless bi–level image coding schemes based on quadrisection scanning. As we have pointed out in Chapter 3, in this chapter, we will approach efficient lossless image compression via another way: progressive coding. In many applications such as medical imaging, progressive compression is preferable. In this chapter, we will discuss two progressive coding schemes with compression results.

5.1 Progressive Compression via Resolution Reduction

Follow the idea in Theorem 1, it is easy for us to build a progressive–compatible sequential coding scheme. In this coding scheme, an image is first divided into two different resolution images, namely a lower resolution image and a residue image.

The lower resolution image is first encoded and transmitted. After that, the residue image is encoded based on the lower resolution image. It can also be regarded as having only one sequential layer. In this sense, it is a progressive-compatible sequential coding scheme.

5.1.1 Resolution Reduction

There are a lot of methods to generate lower resolution image in lossless image coding. In our coding scheme, we employ different method to get the lower resolution image from what is used in JBIG1. If we take all those pixels that lies in the upper left corner of each 2×2 square block shown in Figure 5.1, we get an image with $\frac{1}{4}$ size of the original image.

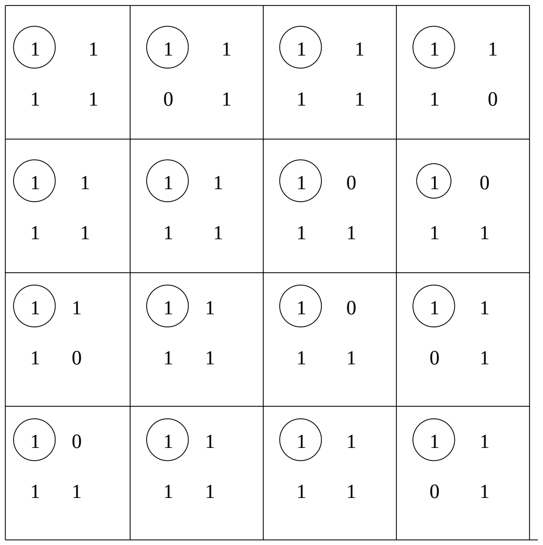


Figure 5.1: Example for lower resolution image rendition

Now we have two parts of image data: the first part is composed of those pixels we get by using the sub–sampling method and the second part includes all remaining pixels. We call the first part the lower resolution image and the second part the residue image.

5.1.2 Coding Strategy

At first, we encode the lower resolution image using the method described in scheme 2 in Chapter 4. The context template used is the ideal context model composed of 8 pixels. The values of pixel $\{5, 6, 7\}$ in the context template are always assumed 1 since they are never available for the pixels in the lower resolution image. If a pixel in the context template is encoded already, its real value is inserted into the context template; otherwise, predicted value is used. If a pixel in the context template is beyond the boundary, it uses default value 1. The reason we use scheme 2 rather than scheme 3 is that we do not observe advantage of scheme 3 over scheme 2 for small size images. Besides that, since quadrisection scanning based algorithms produce better results than raster scanning based ones, we adopt scheme 2 in coding the lower resolution image.

Now we encode the remaining pixels using the ideal context template. We still access those pixels along quadrisection scanning order. For those pixels, any pixel in the ideal context template may have been encoded, including pixel $\{5, 6, 7\}$. If so, its real value is inserted into the context template; otherwise, default value 1 is used. No prediction methods are used in coding the residue image.

The results for this coding scheme are shown in Table 5.1 in the column labeled as scheme 4.

5.2 Progressive Image Coding via Pattern Classification

Scheme 4 is not a complicated coding scheme and we get some good results. In scheme 4, we treat the remaining $\frac{3}{4}$ pixels in the residue image with no distinction. In the following coding scheme, we will divide the original image into four sub–images of the same resolution. We use the same sub-sampling methods described in scheme 4 to form the 4 sub–images.

5.2.1 Pattern Classification

In Figure 5.2, four pixel patterns are shown. The pixels marked with an **x** are the pixels to be encoded. The white pixels are those pixels that have not been encoded and black pixels are those have been already encoded in the original image. In fact, all the pixels marked with **x** in pattern 1 belong to the same sub–image and so do pixels marked with **x** in pattern 2, pattern 3 and pattern 4. The pattern numbers also indicate the coding order for the sub-images.

5.2.2 Coding Strategy

For different sub–images, different coding strategies are employed to encode the pixels in different sub–images. The ideal context templates composed of 8 adjacent surrounding pixels are used in this coding scheme.

Pattern 1 We regard the pixels in pattern 1 as a whole image whose size is $\frac{1}{4}$ of the original image. This sub–image is encoded by using the coding scheme 2 described in Chapter 4.

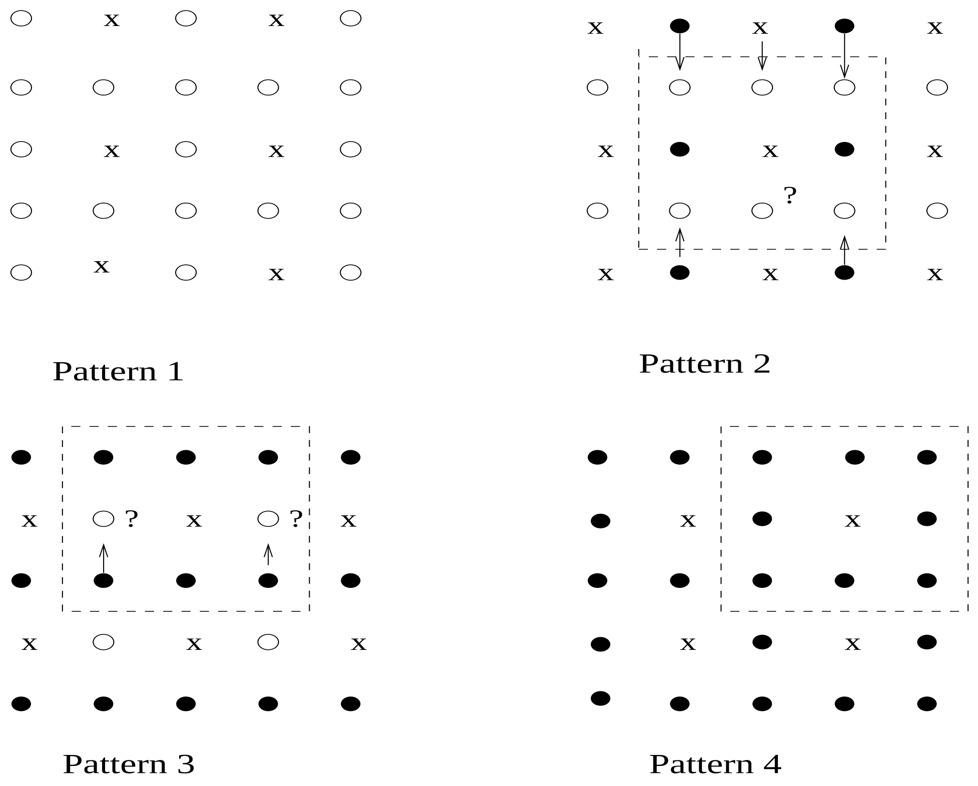


Figure 5.2: Pixel patterns

Pattern 2 After we already finished encoding of all the pixels in pattern 1, we put those pixels back to the original image and use them to encode the pixels in pattern 2. The idea of context prediction is used in this part. The pixels involved in predicting the values of unknown pixels in coding a pixel in pattern 2 are shown in Figure 5.2 with arrows pointing to their target pixels. Generally, we use the pixels directly above or under the unknown pixels to predict their values. The pixel marked with a **?** is predicted by its four adjacent black pixels.

Pattern 3 For each pixel in pattern 3, we find that there are only two adjacent pixels are not available for coding purpose. They are the pixels one position before and the one one position behind the current pixel **x**, marked with a **?** in Figure 5.2. We use a rather simple method to predict the values of these two pixels by using the value of the pixels directly under them.

Pattern 4 The encoding of pixels belonging to pattern 4 is the easiest. All the adjacent pixels included in the ideal context template are available when encoding pixels in pattern 4.

However, from the results listed in Table 5.1, we can find that this algorithm does not give better results than scheme 4 though it does provide several best compression results for our test images. Those results are not beyond our expectation because the coding techniques used in scheme 5 are too simple. Not all the available 2–D correlation information is used. The reason we present this coding scheme is to show our approach to lossless image coding via multi-resolution representation. In fact, we can improve the performance of this algorithm very easily by using more available pixels to predict the unknown pixels.

5.3 Compression Results

In Table 5.1, compression results for bi–level images of size 512×512 and 1024×1024 using our progressive coding schemes are provided. All the compression results are represented by the final file size(in bytes) after encoding without any additional information. Scheme 4 refers to the coding scheme presented in Section 5.1 and scheme 5 is the one discussed in Section 5.2.

The results for JBIG1 are the same as those used in Chapter 4. Those results are for JBIG1 single–sequential mode becuase we know that the results in JBIG1 progressive modes are generally much worse than its sequential mode.

5.4 Remarks

In coding scheme 4 and scheme 5, we show how to construct coders for progressive lossless image coding. If we keep on sub–sampling the lower resolution image, we can get a multi–resolution pyramid representation of the original image. All methods used in coding the two layer case are also applicable to encode multi-resolution images.

Although the results for JBIG1 progressive mode are generally much worse than its single–sequential mode, we do find that some of our results in progressive mode are even better than those provided in Chapter 4.

Table 5.1: Compression results in progressive mode

	Size	**S**cheme 4	**S**cheme 5	**J**BIG1
lenna	512×512	4825	4706	4870
baboon	512×512	16447	16924	17299
barbara	512×512	8807	8448	8234
peppers	512×512	4065	3982	4024
sailboat	512×512	4928	4977	5094
tiffany	512×512	757	895	928
boat	512×512	6790	6886	7064
bridge	512×512	9312	9491	9812
car	512×512	9031	9259	9656
couple	512×512	7967	8179	8173
elaine	512×512	7444	7272	7503
tank	512×512	9464	9693	9988
airplane	1024×1024	4364	4300	4380
airport	1024×1024	28160	27593	28676
man	1024×1024	21213	20874	21881
flower1	1024×1024	8657	8338	8528
flower2	1024×1024	1137	1128	1251

Chapter 6

Lossless Image Coding via 1–D Grammar Codes

Very recently, Yang and Kieffer proposed a new kind of universal lossless compression algorithm [14], which is called grammar transform codes. The grammar based codes outperform those LZ77/78 based codes in general text compression. In fact, they are also effective in lossless image. In this chapter, we will show that grammar based codes can outperform most block based codes used in lossless image coding and present some preliminary compression results.

6.1 Review of Grammar Based Coding

Following [10], [14], a grammar based code has the structure shown in Figure 6.1. The original data sequence x is first transformed into a context–free grammar (or simply a grammar) G from which x can be fully recovered. The grammar G also gives rise to a non–overlapping, variable–length parsing of x. The sequence x is

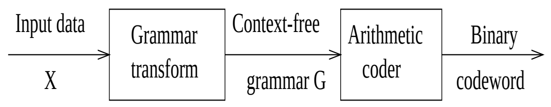


Figure 6.1: Structure of a grammar based code

then compressed by using a zero order arithmetic code to compress either G or the corresponding sequence of parsed phrases. To get an appropriate G, string matching is often used in some manner. One of major problems in grammar based coding theory is to find suitable grammar transforms so that arithmetic coding and string matching capability can be jointly optimized in some sense.

6.1.1 Context Free Grammars

Let $\mathcal{A}$ be our source alphabet with cardinality greater than or equal to 2. In bi–level image coding, $\mathcal{A} = \{0, 1\}$; in gray-scale image coding, $\mathcal{A}$ consists of all integers from 0 to 255. Let $\mathcal{A}^+$ be the set of all finite strings of positive length from $\mathcal{A}$. The notation $|\mathcal{A}|$ stands for the cardinality of $\mathcal{A}$, and for any $x \in \mathcal{A}^+$, $|x|$ denotes the length of x. To avoid possible confusion, a sequence from $\mathcal{A}$ is sometimes called an $\mathcal{A}$–sequence.

Fix a countable set $\mathcal{S} = \{s_0, s_1, s_2, \ldots\}$ of symbols, disjoint from $\mathcal{A}$. Symbols in $\mathcal{S}$ will be called *variables*; symbols in $\mathcal{A}$ will be called *terminal symbols*. For any $j \geq 1$, let $\mathcal{S}(j) = \{s_0, s_1, \cdots, s_{j-1}\}$. For our purpose, a context-free grammar G is a mapping from $\mathcal{S}(j)$ to $(\mathcal{S}(j) \cup \mathcal{A})^+$ for some $j \geq 1$. To describe the mapping explicitly, we shall write, for each $s_i (i < j)$, the relationship $(s_i, G(s_i))$ as $s_i \rightarrow G(s_i)$, and call it a production rule. Thus the grammar G is completely described by the set of production rules $\{s_i \rightarrow G(s_i) : 0 \leq i < j\}$. Start with the variable s_0. Replacing in parallel each variable s in $G(s_0)$ by $G(s)$, we get another sequence from $\mathcal{S}(j) \cup \mathcal{A}$. If we keep doing this parallel replacement procedure, one of the following will hold:

(1) After finitely many parallel replacement steps, we obtain a sequence from $\mathcal{A}$;

(2) the parallel replacement procedure never ends because each string so obtained contains at least one variable.

For the purpose of data compression, we are interested only in grammars G for which the parallel replacement procedure terminates after finitely many steps and every variable $s_i (i < j)$ is replaced at least once by $G(s_i)$ in the whole parallel replacement process. Such grammars G are called *admissible grammars* and the unique sequence from $\mathcal{A}$ resulting from the parallel replacement procedure is called a sequence represented by G or by s_0. Since each variable s_i is replaced at least once by $G(s_i)$, it is easy to see that each variable $s_i (i \neq 0)$ represents a substring of the $\mathcal{A}$-sequence represented by s_0, as shown in the following example.

Example 1: Let $\mathcal{A} = \{0, 1\}$. Below is an example of an admissible grammar G with variable set $\{s_0, s_1, s_2, s_3\}$.

$$\begin{aligned} s_0 &\rightarrow 0 s_3 s_2 s_1 s_1 s_3 10 \\ s_1 &\rightarrow 01 \\ s_2 &\rightarrow s_1 1 \\ s_3 &\rightarrow s_1 s_2 \end{aligned}$$

Perform the following parallel replacements:

$$\begin{aligned} s_0 &\overset{G}{\rightarrow} 0 s_3 s_2 s_1 s_1 s_3 10 \\ 0 s_3 s_2 s_1 s_1 s_3 10 &\overset{G}{\rightarrow} 0 s_1 s_2 s_1 10101 s_1 s_2 10 \\ 0 s_1 s_2 s_1 10101 s_1 s_2 10 &\overset{G}{\rightarrow} 001 s_1 1011010101 s_1 110 \\ 001 s_1 1011010101 s_1 110 &\overset{G}{\rightarrow} 0010110110101010110 \end{aligned}$$

In the above, we start with s_0 and then repeatedly apply the parallel replacement procedure. We see that after four steps—each appearance of the notation $\overset{G}{\rightarrow}$ represents one step of parallel replacement—we get a sequence from $\mathcal{A}$ and the parallel replacement procedure terminates. Also, each variable $s_i (0 \leq i < 4)$ is replaced at least once by $G(s_i)$ in the whole parallel replacement process. Therefore in this example, s_0 (or G) represents the sequence $x = 00101101101010101110$. Each of the other variables represents a substring of x: s_1 represents 01, s_2 represents 011, and s_3 represents 01011.

6.1.2 Grammar Transforms

An admissible grammar G is said to be *irreducible* if the following properties are satisfied:

- Each variable s in G other than s_0 appears at least twice in the range of G.
- There is no non–overlapping repeated pattern of length greater than or equal to 2 in the range of G.
- Each distinct variable in G represents a distinct $\mathcal{A}$–sequence.

A grammar transform is one that converts any sequence $x \in \mathcal{A}^+$ into an admissible grammar G_x that represents x. A grammar transform is said to be *irreducible* if it converts each x into an irreducible grammar G_x.

It was proved in [10], [14] that any grammar based code with an underlying irreducible grammar transform is universal and can outperform asymptotically any finite state code.

6.1.3 The YK Algorithm

In [14], an irreducible grammar transform called the greedy grammar transform was proposed. This grammar transform constructs sequentially a sequence of irreducible grammars from which the original data sequence can be recovered incrementally. Based on this grammar transform, three universal lossless compression algorithms were then developed in [14], among which is the so–called improved sequential algorithm, e.g., the YK algorithm. It has been shown that the YK algorithm combines the power of arithmetic coding with that of string matching and outperforms significantly LZ77, LZ78, and some other standard 1-D compression algorithms. In fact, Theorem 2 in [14] already points out that the YK algorithm can perform better than any context based lossless image coding algorithms if they follow the same scanning methods. Furthermore, a more strict worst case redundancy can be induced in image coding. For the detailed description of the YK algorithm, please refer to [14].

An interesting fact about the YK algorithm is that it partitions a 1–D integer sequence into segments of different sizes based on the patterns already occured in previous parsings. In fact, it tries to find the longest matching string in each coding step. If we apply the same parsing strategy to an image, we then get a block of image each time. If we use raster scanning to access an image, we actually get repeated blocks within the same row or several adjacent rows. In this sense, the 2–D correlation information is not fully employed. As what we did in Chapter 4, we can use some special scanning methods to get more 2–D information provided by surrounding pixels. However, we must point out that we still do not meet the power of grammar based codes since the 2–D information we can employ is limited no matter what scanning method we use.

6.2 Lossless Image Coding Scheme without Prediction

As discussed in the last section, grammar based codes are designed originally for compressing 1–D data. In this section, we extend grammar based codes to 2–D image data. An interesting question arising from the 2–D extension is how to utilize 2–D correlation of images. Following the work[30] of Lempel and Ziv, our approach is to scan images in a proper manner and let both the scanning method and the subsequently used grammar based code jointly take care of 2–D correlation of images. We call the process of scanning images as linearization of images. Then our proposed image coding scheme without prediction has the structure shown in Figure 6.2. The input image is first scanned in a proper manner and encoded by a

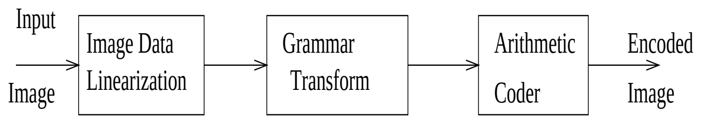


Figure 6.2: Image coding scheme without prediction

grammar based code. As we shall see later, if images are scanned in a quadrant by quadrant manner and if the underlying grammar transform is irreducible, then the above image coding scheme outperforms asymptotically any finite 2–D block codes and any finite context 2–D arithmetic codes, which include JBIG1 as a special case, as the size of images gets larger and larger. It should be pointed out that here we do not call the above image coding scheme as a 2–D grammar based code. Instead we will reserve the name of 2–D grammar based codes for grammar based codes which encode images directly and have three-dimensional structures. 2–D grammar based codes are currently under investigation.

6.2.1 Linearization of 2–D Image Data

The scanning methods we will use are the quadrisection scanning and Peano–Hilbert scanning [30]. The quadrisection scanning is shown in Figure 6.3(a). Peano–Hilbert scanning is shown in Figure 6.3(b). Both scanning methods scan images in a quadrant by quadrant manner. These scanning methods and the subsequently used grammar based code together can utilize effectively 2–D global correlation of images, as shown in Theorem 2 below.

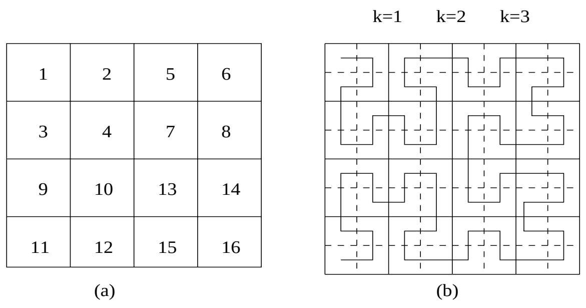


Figure 6.3: (a)Quadrisection scanning (b)Peano–Hilbert scanning

6.2.2 Optimality Analysis

Assume that the linearization process is achieved by either the quadrisection scan or the Peano–Hilbert scan. We now compare the compression performance of our proposed image coding scheme against the best compression performance afforded by all finite 2–D block codes with bounded block size and all finite context 2–D arithmetic codes with bounded number of 2–D contexts. To be specific, assume that the grammar based code used is the YK algorithm. Theorem 2 below, however, is valid for any grammar based code with an underlying irreducible grammar transform.

Let I_n be an $n \times n$ image with pixel values from $\mathcal{A}$. Consider an arbitrary 2–D block code $\mathcal{C}$ with block size $\leq k$ or an arbitrary 2–D arithmetic code $\mathcal{C}$ with the number of 2–D contexts $\leq k$, where for a 2–D block code, its block size is defined as the number of pixels contained in the corresponding rectangular block. Let $r_{\mathcal{C}}(I_n)$ be the compression rate in bits per pixel resulting from using $\mathcal{C}$ to encode I_n. Let

$$r_k^*(I_n) = \min_{\mathcal{C}} \{r_{\mathcal{C}}(I_n)\}.$$

The quantity $r_k^*(I_n)$ represents the smallest compression rate in bits per pixel among all possible 2–D block codes and 2–D arithmetic codes considered above. Let $r(I_n)$ be the compression rate in bits per pixel resulting from using the proposed image coding scheme to encode I_n. Then we have the following theorem.

Theorem 2 *There is a constant d_k, which depends only on $|\mathcal{A}|$, such that*

$$r(I_n) - r_k^*(I_n) \leq \sqrt{\frac{\log\log n^2}{\log n^2}}$$

for any $n \times n$ image with pixel values from $\mathcal{A}$.

The above theorem is similar to Theorem 2 in [14]. In fact, the above theorem is a little more strict than the one in [14] for it is the worst case redundancy comparison when a context based algorithm and the YK algorithm may exploit different scanning methods. The proof of this theorem is not provided in this thesis.

6.2.3 Compression Results

Assume that the grammar based code used is the YK algorithm. Table 6.1 lists some compression results in bits/byte of our proposed image coding scheme for

Table 6.1: Compression rates using YK algorithm w/o prediction

	size	**Q**uadrisection	**P**eano-Hilbert	**J**BIG1
lenna	512×512	0.1923	0.1800	0.1656
baboon	512×512	0.5637	0.5629	0.5213
peppers	512×512	0.1718	0.1584	0.1307
tiffany	512×512	0.0281	0.0279	0.0237
bridge	512×512	0.3455	0.3385	0.3058
car	512×512	0.3410	0.3361	0.3128
couple	512×512	0.3192	0.3094	0.2602
elaine	512×512	0.2667	0.2539	0.2400
tank	512×512	0.3233	0.3272	0.3112
airplane	1024×1024	0.0329	0.0315	0.0334
airport	1024×1024	0.2556	0.2490	0.2188
man	1024×1024	0.2063	0.1963	0.1670
testpat	1024×1024	0.0280	0.0333	0.0352
flower2	1024×1024	0.0134	0.0117	0.0106

different scanning methods. Also shown in Table 6.1 are compression rates afforded by JBIG1.

Compression results shown in Table 6.1 are consistent with our theoretical analysis. Indeed, for some large images, our proposed algorithm outperforms JBIG1. For small images, however, the compression rates achieved by our proposed algorithm are a little worse than those reached by JBIG1. Generally speaking, the YK algorithm and the Peano–Hilbert(or quadrisection) scanning method together can effectively remove global redundancy of images. This advantage, however, comes with a disadvantage of the so–called boundary effect. In its encoding process, the YK algorithm, together with the Peano–Hilbert(or quadrisection) scanning method, partitions an image into non-overlapping irregular areas. Over the boundaries of these irregular areas, our proposed coding scheme currently loses its encoding efficiency, as compared to 2–D arithmetic coding. It is this boundary effect that contributes the redundancy term $\sqrt{\frac{\log\log n^2}{\log n^2}}$. In other words, the image coding scheme proposed in this section can not effectively remove all local redundancy of images. To overcome this problem, in the next section, we will use a prediction mechanism to help removing local redundancy.

6.3 Lossless Image Coding Scheme with Prediction

In this section, we will use an adaptive context template as a prediction mechanism to improve the ability of removing local redundancy. Accordingly, we get a modified image coding scheme which has the structure shown in Figure 6.4.

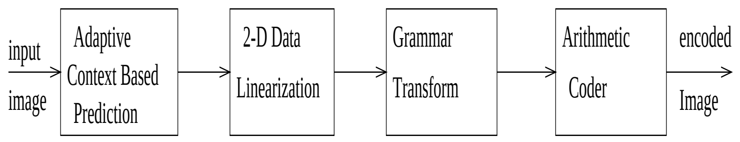


Figure 6.4: Image coding scheme with prediction

6.3.1 Context Based Prediction

A key question in using context based prediction is how to define a context template. Different definitions of context templates result in different coding performance. We have tested several context templates, including two line and three line models used in JBIG1 and our extended context template discussed in the previous chapters. The seven pixel template shown in Figure 6.5 gives the best compression results so far in our compression, though it may not be the best one. There are 128 contexts at all according to this context template definition. For each pixel, there is one specific context corresponding to it. For the pixels in one context template beyond the image boundary, we set the values to 1. Instead of encoding the value of the current pixel directly, we encode the differential value between the actual pixel value and the most probable pixel value under this context based on the pixels encoded already. Because we work on bi–level images, the differential pixel value under one context can take either 1 or 0.

4	1	2	5	6
3	0	X		

Figure 6.5: The 7–pixel context template

6.3.2 Compression Results

By following the steps given above, we re–tested images listed in Table 6.1. As shown in Table 6.2, the simple prediction mechanism indeed improves compression performance. For some images of size 512×512, this lossless image coding scheme can almost match JBIG1 and for images of size 1024×1024, this scheme provides better compression efficiency. We can also find that there is no significant improvement for some images of size 512×512. We think it is because prediction is rather poor when encountering changing edges in those images.

Table 6.2: Compression rates using YK algorithm with prediction

	Size	**Q**uadrisection	**P**eano-Hilbert	**J**BIG1
lenna	512×512	0.1695	0.1661	0.1656
baboon	512×512	0.5643	0.5548	0.5213
peppers	512×512	0.1522	0.1474	0.1307
tiffany	512×512	0.0270	0.0260	0.0237
bridge	512×512	0.3330	0.3242	0.3058
car	512×512	0.3381	0.3279	0.3128
couple	512×512	0.2860	0.2784	0.2602
elaine	512×512	0.2495	0.2422	0.2400
tank	512×512	0.3253	0.3166	0.3112
airport	1024×1024	0.2486	0.2455	0.2188
man	1024×1024	0.1914	0.1880	0.1670
testpat	1024×1024	0.0327	0.0350	0.0352
airplane	1024×1024	0.0315	0.0307	0.0334
flower2	1024×1024	0.0104	0.0102	0.0106

6.4 The Parallel Switching Coding Scheme

From the success of arithmetic coding like coders (i.e., JBIG1) and YK algorithm in bi–level image compression shown in previous sections, we develop the third coding algorithm which combines the power of both YK algorithm and block based arithmetic coders. We call this algorithm PSA(Parallel Switching Algorithm) since it always switches between two parallel encoders.

6.4.1 The Encoder Strcture

The basic structure of PSA encoder is shown in Figure 6.6. At first, we scan the image using the quadrisection scanning method, then we combine four pixels into one byte. We parse the pre–processed image by using YK algorithm, thus we get a block of image pixels each time what we call a longest matching string in YK algorithm. We now have two choices to encode this block (or string) by either YK coder or block based arithmetic coder. We then choose one which gives better compression efficiency and encode an additional symbol to signal the switching operation between the two coders.

The arithmetic coder we use in the parallel switching encoder is similar to the scheme 2 we described in Chapter 4.

6.4.2 The Dynamic Variable Tree

One problem we meet in PSA is how to terminate a block without introducing much redundancy. There is no need to indicate the end of a string when we use YK algorithm. But we have to signal the end of a block if we switch to arithmetic coder otherwise we don't know what we should do. In the original YK algorithm, we create

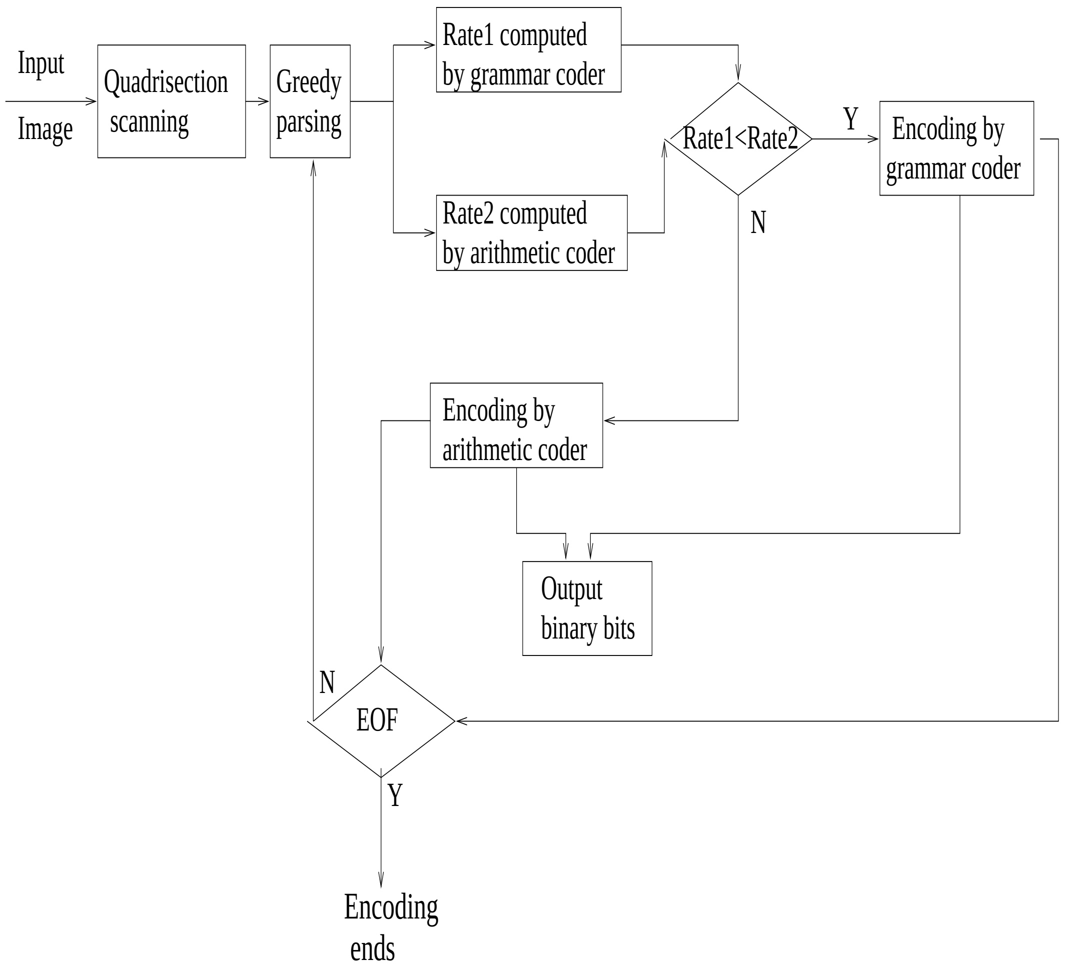

Input
Image
Quadrisection scanning
Greedy parsing
Rate1 computed by grammar coder
Rate2 computed by arithmetic coder
Rate1<Rate2
Y
N
Encoding by grammar coder
Encoding by arithmetic coder
Output binary bits
EOF
N
Y
Encoding ends

Figure 6.6: The encoder structure for PSA

a variable list to record each string occurred in previous parsings. We construct a dynamic variable tree in the parallel switching algorithm instead of a variable list which makes us terminate the block coder easily. We update this variable tree dynamically when there is a new variable introduced after each encoding step.

In Figure 6.7, we show how to construct a binary variable tree. s_1, s_2 and s_3 in Figure 6.7 are variables given by 1–D grammar based codes. Although in our experimental coding scheme, our alphabet size is 16 rather than 2, we use binary variable tree to fulfill our compression.

Each time we get a block after parsing, we encode this block along the dynamic variable tree using the arithmetic coder. If the last pixel in this block is a leaf of this tree, we do not need any further information because we know which variable it is. But if the last pixel is not a leaf of this tree, i.e., an internal node, one strategy to terminate current encoding step is to encode more pixels along the variable tree until the first difference between real pixels and tree node appears.

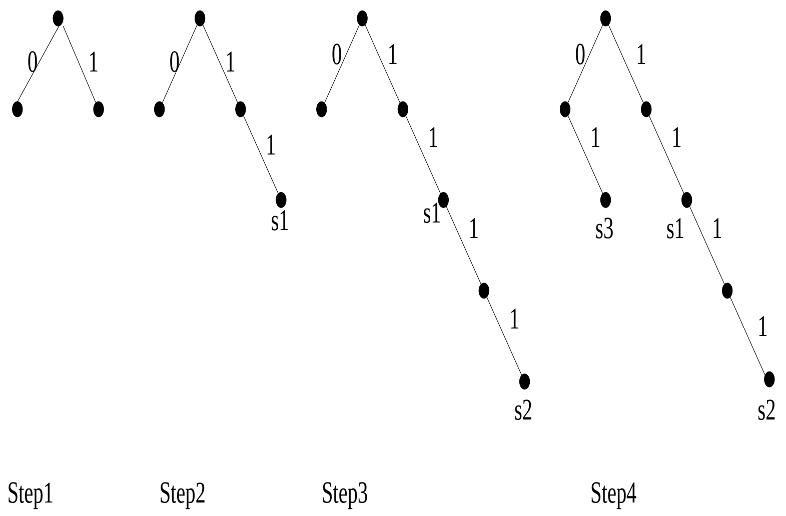


Figure 6.7: The dynamic variable tree

6.4.3 Switching Criteria

Another main factor influencing the compression efficiency in the parallel switching algorithm is the switching criterion, which makes decision when to use the 1–D grammar based codes and when to use the arithmetic coder. In Figure 6.8, we take one coding step in the parallel switching algorithm as an example.

Symbols in Figure 6.8 indicate different blocks or positions in an image.

- PB indicates the parsing block got by using 1–D grammar based codes.
- TB_0 indicates the tail block encoded by the arithmetic coder in the previous coding step if it exists. If 1–D grammar based codes was used in the previous step, then the size of TB_0 equals to zero.
- TB_1 indicates the tail block encoded by the arithmetic coder in the current coding step if the arithmetic coder is used.
- $AB + TB_1$ denotes the block coded by using the arithmetci coder if we switch to it in the current coding step.
- np indicates the beginning position of n^{th} parsing block given by the 1–D grammar codes.
- np_0 indicates the actual beginning position in the current coding step.

We consider four switching criteria in our algorithm as follows:

Criterion 1 Realize that 1–D grammar transfrom based codes parse an image into blocks and apply encoding strategy for different cases. For example, in [10], four cases are addressed and we can make decision about when to

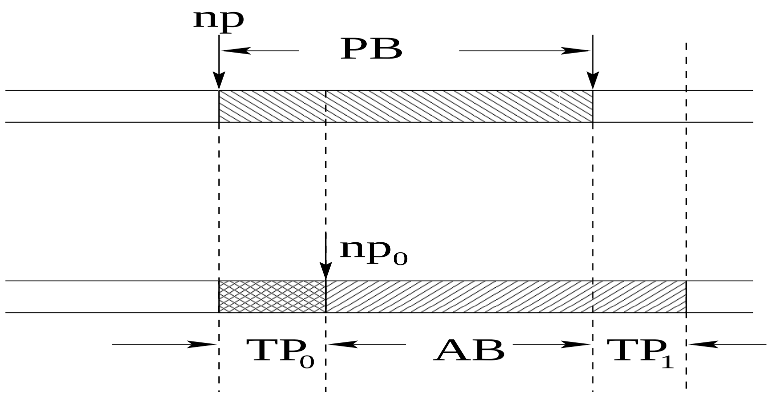


Figure 6.8: Encoding example for PSA

switch according to the current case. For example, for most {0,1} and {1,1} cases, 1–D grammar codes generally provide better compression efficiency than arithmetic coding, but for {0,0} and {1,0} case, the results are different. Thus we can switch to the arithmetic coder when the latter two cases are met.

Criterion 2 The most natural switching criterion is that we compute both rates for the same parsing block PB. We compare the two rates and switch to the coder which gives a smaller rate.

Criterion 3 Consider that once we use the arithmetic coder in the previous coding step, there is no need to encode TB_0. We then get our criterion 2. We compute the coding rate for PB based on the information provided by TB_0. We denote it as rate 0. At the same time, we compute the rate for $AB + TB_1$ since we have to encode the tail block TB_1 if we switch to the arithmetic coder in the current coding step. We denote this rate as rate 1. We compare rate 0 with rate 1 and take the smaller one.

Criterion 4 One advantage of 1–D grammar based codes is that it provides a new partition method in finding repeated patterns in images. For small blocks, the arithmetic coder may give better compression results than 1–D grammar based codes since small blocks tend to lying on the changing edges. For large blocks, 1–D grammar codes may give better results. Thus, criterion 3 makes the decision whether to switch or not depending on the block size. However, we may lost overall efficiency since we have to encode part of the block size information.

Different switching criterion results in compression differences for our test images. Generally, criterion 3 gives good theoretical results for most of our test images. However, the results may vary for different test image. In the implementation of the programs for both encoder and decoder of the 1–D grammar based codes, we adopt the multi–level arithmetic coding algorithm described in [31].

6.4.4 Compression Results

In Table 6.3, we show our theoretical results for criterion 3 of PSA. The final compression rates are the sum of coding rates and switching rates. We can see that our coding rates are mostly less than those afforded by JBIG1 but the final compression rates are a little worse than coding rates because the switching rates are pretty large due to the frequent switching between the two algorithms. However, there is still a lot to improve since we get additional information when we switch to arithmetic coder in the previous coding step.

In Table 6.4, we list our preliminary compression results using criterion 2. The results for criterion 2 do not include the coding rates for the end–of–block symbol. From the results lised in Table 6.4, we can see that we can get substantial gain by switching between two coders if switching symbol and end–of–block symbol for the binary arithmetic coder are not considered. However, how to efficiently encode those two symbols are still under further investigation.

For some images of size 512×512, this lossless image coding scheme outpreforms JBIG1 and for others, it can almost match the rates reached by JBIG1 and the difference is always less than 0.02 bits per pixel. For most images of size 1024×1024, this scheme provides better compression results than the algorithm we discussed in previous two sections. We can also find that there is no significant improvement for some test images.

Table 6.3: Theoretical compression rates of PSA

	Size	**C**oding rate	**S**witching rate	**J**BIG1
lenna	512×512	0.1423	0.0143	0.1656
baboon	512×512	0.4827	0.0442	0.5213
peppers	512×512	0.1213	0.0126	0.1307
tiffany	512×512	0.0231	0.0008	0.0237
bridge	512×512	0.2746	0.0258	0.3058
car	512×512	0.2695	0.0241	0.3128
couple	512×512	0.2345	0.0227	0.2602
elaine	512×512	0.2145	0.0234	0.2400
tank	512×512	0.2776	0.0274	0.3112
airport	1024×1024	0.2164	0.0206	0.2188
man	1024×1024	0.1664	0.0159	0.1670
testpat	1024×1024	0.0274	0.0025	0.0352
flower2	1024×1024	0.0103	0.0003	0.0106

Table 6.4: Compression rates of PSA

	Size	**C**oding rate	**S**witching rate	**J**BIG1
lenna	512×512	0.1263	0.0201	0.1656
baboon	512×512	0.4554	0.0543	0.5213
peppers	512×512	0.1075	0.0171	0.1307
tiffany	512×512	0.0161	0.0034	0.0237
bridge	512×512	0.2540	0.0329	0.3058
car	512×512	0.2463	0.0337	0.3128
couple	512×512	0.2154	0.0292	0.2602
elaine	512×512	0.1971	0.0286	0.2400
tank	512×512	0.2523	0.0358	0.3112
airport	1024×1024	0.1947	0.0241	0.2188
man	1024×1024	0.1427	0.095	0.1670
testpat	1024×1024	0.0193	0.0040	0.0352
flower2	1024×1024	0.0059	0.0015	0.0106

Note: The arithmetic coding rates for end–of–block symbol are not included.

6.5 Remarks

From the compression results shown in this chapter, we can conclude that grammar based codes, particularly the YK algorithm, achieve pretty good performance in bi–level images coding, which is consistent with what is pointed out in Theorem 2. The coding schemes used in our compression are just simple extensions of the YK algorithm to 2–D image coding. One possible reason influencing the YK algorithm in image coding is the mapping of the same sub–string to different image blocks. Besides that, we still have to face the problem of changing edges, which is difficult to overcome for any existing algorithms. We do believe that we can improve the performance once we find more useful ways to take the advantage of 2–D correlation rather than data linearization and context based prediction. Also, how to encode the switching symbol and the end–of–block symbol for the binary arithmetic coder are very critical in determining the overall performance of the PSA algorithm. Due to some reasons, we only show our preliminary compression results in bi–level images coding. In fact, 1–D grammar based codes are also suitable for gray–scale image coding.

Chapter 7

Conclusions

In this thesis, we present our studies on the analysis and design of lossless bi–level image coding systems. In Chapter 3, Chapter 4 and Chapter 5, context based algorithms are studied carefully. A quadrisection scanning method is proposed with several coding schemes based on it. The 4–line model provides competitive results in sequential bi–level image coding. Most of our results are better then those afforded by JBIG1.

Progressive lossless image compression has also been paid a little attention in this thesis. Two algorithms employ quadrisection scanning and context prediction methods are discussed in detail. Multi–resolution image coding is a very interesting topic that I like to do more work on.

The results for these algorithms listed in this thesis are very good, but we still have room to improve compression efficiency by further refining the context template, e.g., increasing the numbers of pixels included in context templates and by designing more accurate context prediction methods. Of course, it is also possible to improve our results by designing new scanning methods which can provide more

2–D information between pixels in an image.

The properties of grammar based codes and their application in lossless image coding still need further verification. The current implementation of the PSA algorithm is not a perfect one. There is still a lot to improve. 1–D grammar based codes provide a new way to have an overall look at an image, or say, a new partition method to find repeated patterns in an image. However, we are not able to employ full advantage provided by this algorithm so far. The PSA algorithm tries to combine the power of 1–D grammar based codes and context based arithmetic coding. From the results listed in Chapter 6, we can see that we can get substantial gain by switching between the two coders. However, we pay too much for something unexpected, such as the switching rates mentioned in this thesis. How to efficiently encode the switching symbol and the end–of–block symbol are still under investigation.

Because it is easy for us to generate the quadrisection scanning order array for images of size $2^n \times 2^n$, we only show our results for this kind of images. It doesn't necessarily mean that we can only compress those images. All of our algorithms can be extended to the coding of large size images and standard Fax files. Our ideas can also be applied to gray–scale image compression.

Bibliography

[1] ISO/IEC International Standard 11544, *Progressive bi-level image compression*, 1992.

[2] ISO/IEC International Standard 10918-1, *Digital compression and coding of continuous-tone still images*, 1991.

[3] J. G. Cleary and I. H. Witten, "Data compression using adaptive coding and partial string matching," *IEEE Transactions on Communications*, vol. 32, pp. 396–402, Apr 1984.

[4] A. Moffat, "Implementing the PPM data compression scheme," *IEEE Transactions on Communications*, vol. 38, pp. 1917–1921, Nov 1990.

[5] F. M. J. Willems, Y. M. Shtarkov, and T. J. Tjalkens, "The context-tree weighting method: basic properties," *IEEE Transactions on Information Theory*, vol. 41, pp. 653–664, May 1995.

[6] J. Rissanen, "Fast universal coding with context models," *IEEE Transactions on Information Theory*, vol. 45, pp. 1065–1071, May 1999.

[7] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data

compression," *Communications of the Association for Computing Machinery*, vol. 30, pp. 520–540, Jun 1987.

[8] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory*, vol. 23, pp. 337–343, May 1977.

[9] J. Ziv and A. Lempel, "Compression of individual sequences via varaible-rate coding," *IEEE Transactions on Information Theory*, vol. 24, pp. 530–536, Sep 1978.

[10] J. Kieffer and E. H. Yang, "Grammar based codes: a new class of universal lossless codes," *IEEE Transactions on Information Theory*, vol. 46, pp. 737–754, May 2000.

[11] R. A. DeVore, B. Jawerth, and B. J. Lucier, "Image compression through wavelet transform coding," *IEEE Transactions on Information Theory*, vol. 38, pp. 719–745, Mar 1992.

[12] J. M. Sharpio, "Embeded image coding using zerotrees of wavelet coefficients," *IEEE Transactions on Signal Processing*, vol. 41, pp. 3445–3462, Dec 1993.

[13] A. Said and W. A. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, pp. 243–250, Jun 1996.

[14] E. H. Yang and J. Kieffer, "Efficient universal lossless data compression algorithm based on a greedy sequential grammar transform-Part 1: Without context models," *IEEE Transactions on Information Theory*, vol. 46, pp. 755–777, May 2000.

[15] P. E. Tischer, R. T. Worley, A. J. Maeder, and M. Goodwin, "Context-based lossless image compression," *The Computer Journal*, vol. 36, pp. 68–77, 1993.

[16] S. Todd, G. G. Langdon, Jr., and J. Rissanen, "Parameter deduction and context selection for compression of gray-scale images," *IBM Journal of Research and Development*, vol. 29, pp. 188–193, 1985.

[17] A. Moffat, "Two-level context based compression of binary images," *Proceedings of Data Compression Conference*, pp. 382–391, Aug 1991.

[18] M. J. Weinberger, J. J. Rissanen, and R. B. Arps, "Applications of universal context modeling to lossless compression of gray-scale images," *IEEE Transactions on Image Processing*, vol. 5, pp. 575–586, Apr 1996.

[19] M. D. Swanson and A. H. Tewfik, "A binary wavelet decomposition of binary images," *IEEE Transactions on Image Processing*, vol. 5, pp. 1637–1650, Dec 1996.

[20] A. Said and W. A. Pearlman, "An image multiresolution representation for lossless and lossy compression," *IEEE Transactions on Image Coding*, vol. 5, pp. 1303–1310, Sep 1996.

[21] G. G. Langdon, Jr., and J. Rissanen, "Compression of black-white binary images with arithmetic coding," *IEEE Transactions on Communications*, vol. 29, pp. 858–867, Jun 1981.

[22] E. H. Yang and J. Guo, "Lossless image coding via one-dimensional grammar based codes," *Proceedings of the 16th IFIP World Computer Congress—2000 International Conference on Communication Technology*, pp. 966–972, Aug 2000.

[23] W. B. Pennebaker, J. L. Mitchell, G. G. Langdon, Jr., and R. B. Arps, "An overview of the basic principles of the Q-coder adaptive binary arithmetic coder," *IBM Journal of Research and Development*, vol. 32, pp. 717–726, Nov 1988.

[24] J. L. Mitchell and W. B. Pennebaker, "Optimal hardware and software arithmetic coding procedures for the Q-coder," *IBM Journal of Research and Development*, vol. 32, pp. 727–736, Nov 1988.

[25] W. B. Pennebaker and J. L. Mitchell, "Probability estimation for the Q-coder," *IBM Journal of Research and Development*, vol. 32, pp. 737–752, Nov 1988.

[26] X. Wu and N. Memon, "Context-based, adaptive, lossless image codec," *IEEE Transactiosn on Communications*, vol. 45, pp. 437–444, Apr 1997.

[27] M. J. Weinberger, G. Seroussi, and G. Sapiro, "The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS," *IEEE Transactions on Image Processing*, vol. 9, pp. 1309–1324, Aug 2000.

[28] J. A. Storer and H. Helfgott, "Lossless image compression by block matching," *The Computer Journal*, vol. 40, pp. 137–145, 1997.

[29] M. Kuhn, "Implementation for JBIG1 codec." Available on line at http://www.cl.cam.ac.uk/ mgk25.

[30] A. Lempel and J. Ziv, "Compression of two-dimensional data," *IEEE Transactions on Information Theory*, vol. 32, pp. 2–8, Jan 1986.

[31] E. H. Yang and Y. Jia, "Universal lossless coding of sources with large and unbounded alphabets," *Numbers, Information and Complexity*, pp. 421–442, Feb 2000.