A New Highly Efficient Algorithm For Lossless Binary Image Compression

Lele Zhou

B.Sc., The University Of Northern British Columbia, 2004

Thesis Submitted In Partial Fulfillment Of

The Requirements For The Degree Of

Master Of Science

in

Mathematical, Computer, and Physical Science

(Computer Science)

The University Of Northern British Columbia

December 2006

Your file Votre référence
ISBN: 978-0-494-28421-6
Our file Notre référence
ISBN: 978-0-494-28421-6

Abstract

Lossless binary image compression is desirable for the enormous amount of images that are stored and transmitted in a wide range of applications. In this thesis, we present a new efficient algorithm for lossless binary image compression. This algorithm consists of two modules: Direct Redundancy Exploitation and Improved Arithmetic Coding. It is referred to as the Two Module Based Algorithm (TMBA).

The Direct Redundancy Exploitation module exploits the two-dimensional redundancy of an image by removing identical consecutive rows and columns of pixels. Binary reference vectors are generated to indicate the exact locations where removals take place so that the removed rows and columns of pixels can be recovered in the decoding process. In the Improved Arithmetic Coding module, the Markov Model order 2 is applied to the reference vectors. A new Static Binary Tree Model is introduced to efficiently model the Reduced Blocks which are produced by the Direct Redundancy Exploitation module. The modelling processes provide the probability distributions which are then used for arithmetic coding.

The Two Module Based Algorithm has demonstrated excellent compression performance. The simulation results showed that the proposed algorithm well outperformed the G3 and G4 coding schemes. In addition, the proposed algorithm has also yielded an increase of compression performance in comparison to the JBIG1 and JBIG2 standards.

The Two Module Based Algorithm is an efficient method for lossless binary image compression. It offers an alternative approach other than the industrial standards. More importantly, it has the comparable or better compression performance than the current industrial standards.

To my wife Jing and my son Matthew

"Imagination is more important than knowledge..."

— Albert Einstein (1879 - 1955)

Contents

List of Tables

List of Figures

List of Programs

Acknowledgments

I would like to extend my most sincere thanks to my advisor, Professor Saif Zahir. His guidance and inspiration were essential to the completion of this work. I am confident that I will benefit from his rigorous scientific approach and critical thinking throughout my future career.

I would like to thank Professors Liang Chen and Kevin Keen for serving on my committee and for their constructive input. I would also like to thank Lyn Benn, the Director of Learning Skills Centre of University of Northern British Columbia, for her generous help and support.

I am indebted to my parents for their encouragement and support in everything I have done. The emphasis they placed on education is what set my sight on higher education. Finally, I would like to thank my wife Jing for her constant support and encouragement which made it possible for me to pursue and complete this work. This thesis I dedicate to her.

Chapter 1

Introduction

In the last decade we have witnessed an information technology revolution which is still under way. The demands for data storage and communication are increasing tremendously. While many research efforts have been made to increase the storage capacity and communication bandwidth, many research efforts are also needed for new compression algorithms. Lossless binary image compression is desirable for many applications, such as digital libraries, newspaper archives, map archives, fingerprint database, facsimile, etc.

In this thesis, we introduce a new highly efficient algorithm for lossless binary image compression. The proposed algorithm consists of two modules: (1) Direct Redundancy Exploitation (DRE); and (2) Improved Arithmetic Coding (IAC), that we will refer to as the Two Modules Based Algorithm (TMBA). The DRE module is a novel method which exploits the two-dimensional redundancy of an image by removing the identical consecutive rows and columns of pixels. Binary reference vectors are generated to indicate the exact locations where removals take place so that the removed rows and columns of pixels can be recovered accordingly in the decoding process. In the Improved Arithmetic Coding module, the Markov Model order 2 is applied to the reference vectors. A new Static Binary Tree Model is introduced to efficiently model the Reduced Blocks which are produced by the Direct Redundancy Exploitation module. The modelling processes provide the probability distributions which are then used for arithmetic coding.

1.1 Thesis Contributions

There are three major contributions in this thesis. Here, a brief discussion will be made of each.

The first contribution is the development of Direct Redundancy Exploitation module. DRE module is a new method to exploit two-dimensional redundancy of an image. It substantially reduces the size of a image. It also forms the "stair" property on the resulting data which can be modelled and coded efficiently by the aithmetic coder and the new context modelling method.

The second contribution is the development of the Static Binary Tree Model. It is specially designed to perform the modelling process on the data with "stair" property. The statistical information provided by the new modelling process is proven to be reliable in increasing the efficiency of arithmetic coding.

The third contribution is that the Two Module Based Algorithm offers us a new alternative method to compress binary images. More importantly, it allows comparable or better compression performance than the current standards in compression.

1.2 Thesis Outline

This thesis is organized as follows:

Chapter 1 is the introduction. The contributions and the outline of the thesis are given in this chapter.

Chapter 2 discusses the background material of binary images compression. Topics include entropy, Huffman Coding, Run Length Coding, Arithmetic Coding and context modelling. The definitions and notations used throughout the thesis are also listed in this chapter.

In Chapter 3, a new algorithm for binary image compression is proposed. The complete algorithm is discussed in detail. The two modules which make up the algorithm are introduced sequentially. The compression results of the proposed algorithm are compared with G3, G4, JBIG1 and JBIG2 standard compression schemes.

In Chapter 4, we make conclusive remarks on the study.

Chapter 2

Background

A digital image can be represented as a matrix where each entry of the matrix represents a pixel value. A binary image is a digital image that has only two possible types of pixels, black and white, which are represented by 1's and 0's respectively. For example, in Figure 2.1 the image of teapot is a binary image, where each cell of the grid is considered as a pixel. It can be represented as a 16×16 matrix I_{teapot} in Equation (2.1).

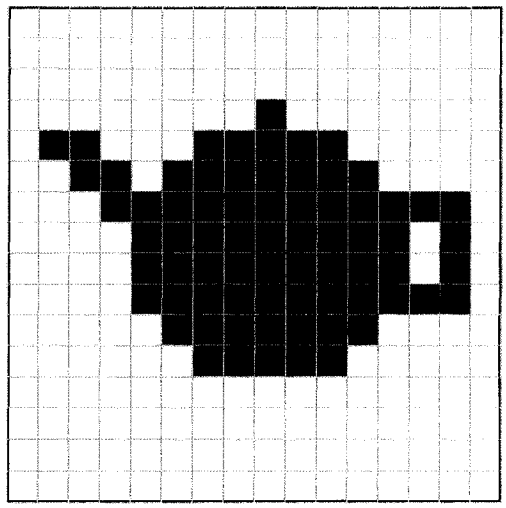


Figure 2.1: Teapot Image

$$I_{teapot} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (2.1)$$

2.1 Definitions

In this section we introduce some definitions and notations which are being used throughout the thesis.

Definition 1: Let I be a matrix, then:

- $I(i, j)$ denotes the element in row i and column j in matrix I.
- $I(i : k, j)$ denotes the elements from row i to k in column j in matrix I.
- $I(i : k, j : l)$ denotes the sub-matrix from row i to k, from column j to l in matrix I.
- $I(i : k, :)$ denotes the sub-matrix from row i to k of all columns in matrix I.
- $I(i : end, j : end)$ denotes the sub-matrix from row i to the last row, from column j to the last column in matrix I.

Definition 2: Let $S = (X_1, X_2, \cdots)$ be a data sequence, each symbol of which belongs to a finite alphabet $A = \{a_1, a_2, \cdots, a_n\}$. For each $a_i, a_j \in A$, then:

- $P(a_i)$ denotes the probability for the occurrences of a_i in source S.
- $P(a_i|a_j)$ denotes the probability for the occurrences of a_i, provided a_j is the immediately preceding symbol.
- $P(a_i|a_j \cdots a_k)$ denotes the probability for the occurrences of a_i, provided $a_j \cdots a_k$ is the immediately preceding symbols sequence.

Definition 3: Compression Ratio is defined as:

$$Compression\ Ratio = \frac{Number\ of\ Bits\ of\ Original\ Data - Number\ of\ Bits\ of\ Compressed\ Data}{Number\ of\ Bits\ of\ Original\ Data}$$

Definition 4: Compression Rate is the average number of bits required to represent a single pixel, which is defined as:

$$Compression\ Rate = \frac{Number\ of\ Bits\ of\ Compressed\ Data}{Number\ of\ Pixels\ of\ Original\ Data}$$

2.2 Image Compression

Image compression, also known as image coding, is a technique that reduces the amount of data to represent a digital image. The ultimate goal of image compression is to minimize the required storage space or transmission time for a given channel capacity. To achieve that, certain properties of an image, especially statistical properties, are used to exploit the redundancy of the image.

Image compression is often referred to its compression algorithm. In fact, two parts of a compression algorithm are compression and reconstruction (decompression), as shown in Figure 2.2. The compression part, also known as the encoding, takes an input image X and generates a representation X_c that requires fewer bits. The reconstruction part, also known as the decoding, operates on the compressed representation X_c to generate the reconstruction image Y. If X and Y are identical, we refer to the algorithm as lossless compression. If Y is different from X, we refer to it as lossy compression. Lossy compression yields higher compression ratios at the price of quality reduction of the reconstructed image. This thesis addresses issues in lossless binary images compression.

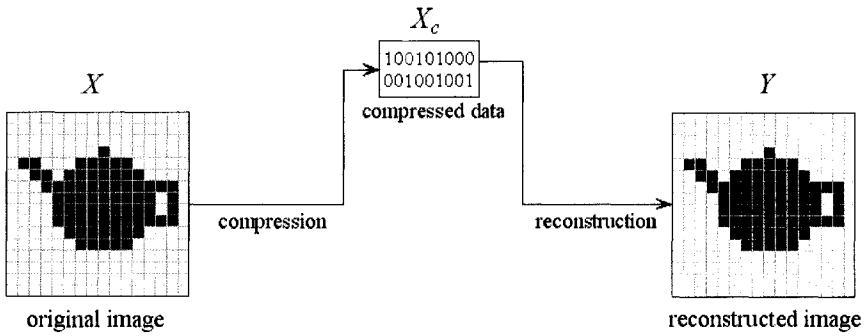


Figure 2.2: Compression and Reconstruction

2.3 Entropy

In 1948, Claude E. Shannon [52] formulated *Information Theory.* He established that there is a fundamental limit for lossless data compression. This limit, called entropy [14], is denoted by H. The value of H can be estimated from the information source.

In this section, we introduce zero order entropy and first order entropy. We will discuss the different coding schemes and their associated entropies in subsequent sections. Higher order entropies are given in Hankersson et al. [16].

2.3.1 Zero Order Entropy

Shannon defined entropy in terms of a discrete random event, known as zero order entropy, which is described as follows:

Let $S = (X_1, X_2, \cdots)$ be a data sequence, where each entry X belongs to a finite alphabet $A = \{a_1, a_2, \cdots, a_n\}$. For each $a_i \in A$, $P(a_i)$ is the probability[1] for the occurrences of a_i in source S. Here,

$$P = \{P(a_i) | i = 1, 2, \cdots, n\} \tag{2.2}$$

defines the probability distribution on A.

Thus, the zero order entropy of S is given by

$$H_0(S) = \sum_{i=1}^{n} P(a_i) \log_2 \frac{1}{P(a_i)} = -\sum_{i=1}^{n} P(a_i) \log_2 P(a_i) \tag{2.3}$$

[1]The probability is estimated by the number of (i=1 to n) appears in source S divided by total number of entries of source S

Consider the matrix representation of the image I_{teapot} in Equation (2.1). I_{teapot} is a binary image in which pixels have two possible values, 0 and 1. Let $N(0)$ and $N(1)$ denote the number of 0's and 1's in I_{teapot} respectively.

$$P(0) = \frac{N(0)}{N(0)+N(1)}, \quad P(1) = \frac{N(1)}{N(0)+N(1)} \tag{2.4}$$

and,

$$N(0) = 184 \quad N(1) = 72$$

hence,

$$P(0) = 0.719, \quad P(1) = 0.281$$

Now applying Equation (2.3) results in the zero order entropy

$$\begin{aligned} H_0(I_{teapot}) &= -\sum_{i=1}^{n} P(a_i)\log_2 P(a_i) \\ &= -\{P(0)\log_2 P(0) + P(1)\log_2 P(1)\} \\ &= -\{0.719 \times \log_2(0.719) + 0.281 \times \log_2(0.281)\} \\ &= 0.857 \end{aligned} \tag{2.5}$$

In this example, nothing more than the probability distribution of each symbol in the alphabet was used. The correlations and dependencies between neigbouring pixels were disregarded. The correlations and dependencies can be exploited by the Markov Model [15], from which first order entropy is derived.

2.3.2 First Order Entropy

In first order entropy, probabilities are dependent on the immediately preceding symbol only. The first order entropy is defined as

$$H_1(S) = -\sum_{i=1}^{n} P(a_j) \sum_{j=1}^{n} P(a_i|a_j)\log_2 P(a_i|a_j) \tag{2.6}$$

where $P(a_i|a_j)$ is the probability of a_i provided that a_j is the immediately preceding symbol.

For example, consider a binary image which has only two types of pixels, black and white. We know there are correlations between neighbouring pixels. Therefore the occurrence of a white pixel as the next observation depends, to some extent, on whether the current pixel is white or black. As a result, we can model the occurrences of pixels as a Markov Chain. In

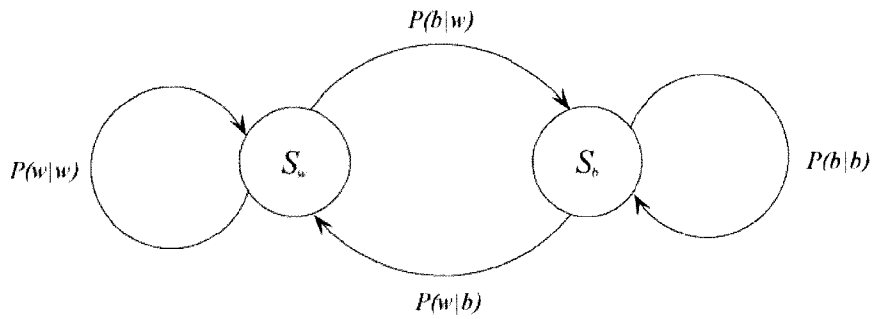


Figure 2.3: Two-state Markov Model for Binary Images

Figure 2.3, two states S_b and S_w are the states of black and white pixel, respectively, and either S_b or S_w is the current state. Let $P(S_w)$ and $P(S_b)$ be the probabilities $P(a_i)$ and $P(b|w)$, $P(w|w)$, $P(w|b)$ and $P(b|b)$ be the probabilities $P(a_i|a_j)$ in Definition 2. Thus, the first order entropy of I_{teapot} can be as follows:

$$P(S_w) = P(0) = 0.719, \quad P(S_b) = P(1) = 0.281$$

and

$$P(w|w) = P(0|0) = 0.929, \quad P(b|w) = P(1|0) = 0.071$$
$$P(w|b) = P(0|1) = 0.181, \quad P(b|b) = P(1|1) = 0.819$$

Applying the definition of the first order entropy in Equation (2.6), we obtain the first order entropy of I_{teapot}:

$$\begin{aligned} H_1(I_{teapot}) &= -\sum_{i=1}^{n} P(a_i) \sum_{j=1}^{n} P(a_j|a_i) \log_2 P(a_j|a_i) \\ &= -\{P(0)\,[P(0|0)log_2(P(0|0)) + P(1|0)log_2(P(1|0))] \\ &\quad +P(1)\,[P(0|1)log_2(P(0|1)) + P(1|1)log_2(P(1|1))]\} \\ &= -\{0.719 \times [0.929 \times log_2(0.929) + 0.071 \times log_2(0.071)] \\ &\quad +0.281 \times [0.181 \times log_2(0.181) + 0.819 \times log_2(0.819)]\} \\ &= 0.458 \end{aligned}$$

Compared with the zero order entropy, the first order entropy of 0.458 bits per pixel is about only a half of 0.857 bits per pixel. This is because the redundancy between the current pixel and its immediately preceding pixel is modelled.

Entropy coding is a data compression approach which exploits the nonuniformity of the probability distribution of the data and encodes them using a variable length code. A typical example of this approach is the well-known Huffman Coding.

2.4 Huffman Coding

Huffman Coding [21] is one of most common entropy encoding algorithms that is widely used for lossless data compression. The main idea behind Huffman Coding is to use the shorter codewords to represent the symbols that occur more frequently and longer codewords to represent the symbols that occur less frequently. In other words, variable-length code is assigned to each symbol based on its probability of occurrence.

If Huffman Coding is directly applied to encode a binary image, no compression will be gained. This is because a binary image only consists of two types of pixels, black and white, which form an alphabet consisting of two symbols: 0 and 1. Regardless of whether one type of pixel occurs more frequently than the other, at least one bit is required to represent each type of pixel. But the entropy calculations for a typical binary image demonstrates that compression should be expected. One way to achieve this is to group neighbouring pixels together so that a larger alphabet may be used, which is explained in this section. Another approach is to consider runs of symbols, i.e. Run Length Coding, and will be explored in the next section.

Suppose we group four neighbouring pixels together in an image, then we should form a larger alphabet A which consists of a maximum number of 2^4 possible symbols. By applying the grouping on image I_{teapot} in Equation (2.1), we obtain a larger alphabet A_{group} which consists of nine symbols as follows:

$$A_{group} = \{0000, 1000, 0110, 0011, 1110, 0111, 0001, 1010, 1111\}$$

For simplicity, a letter a_i is mapped to a symbol in the alphabet A_{group}, and $P(a_i)$ is the probability of occurrence of the symbol a_i. This is done for all symbols in A_{group}. The probabilities are sorted in descending order as shown in Table 2.1.

One way to generate the Huffman Coding is to build a binary Huffman tree in which all external nodes correspond to the symbols. The binary Huffman tree for alphabet A_{group} is shown in Figure 2.4 and it is constructed by the following steps:

Step 1. Select the two parentless nodes with the lowest probabilities. Between those two nodes, assign 1 to the lower probability and 0 to the other.

Table 2.1: the Initial Alphabet of Nine Symbols

Letter	Symbol	Occurrence	$P(a_i)$
a_1	0000	40	0.625
a_2	1111	10	0.156
a_3	1110	4	0.063
a_4	0011	3	0.049
a_5	0111	2	0.031
a_6	1010	2	0.031
a_7	1000	1	0.015
a_8	0110	1	0.015
a_9	0001	1	0.015

Step 2. Create a new node which is the parent of those two lowest probability nodes.

Step 3. Assign the new node a probability equal to the sum of its children's probabilities.

Step 4. Go to Step 1 until there is only one parentless node left in the tree.

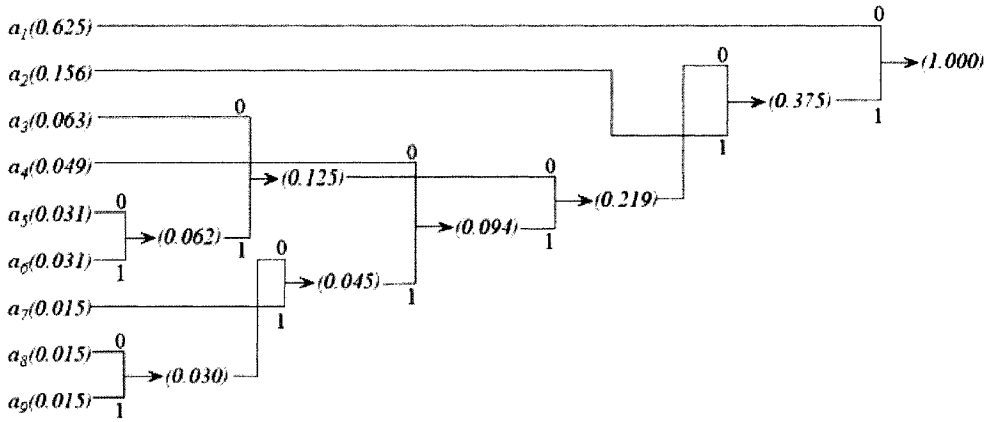


Figure 2.4: Build the Binary Huffman Tree

Once the Huffman tree is constructed, the codeword for each symbol can be obtained by traversing the tree from the root node to the external node corresponding to the symbol. Table 2.2 shows the codeword for each symbol in alphabet A_{group} .

Based on the length of each codeword and the number of occurrences of each symbol,

Table 2.2: the Codeword for Each Symbol

Letter	Symbol	Occurrence	$P(a_i)$	Codewords
a_1	0000	40	0.625	0
a_2	1111	10	0.156	11
a_3	1110	4	0.063	1000
a_4	0011	3	0.049	1010
a_5	0111	2	0.031	10010
a_6	1010	2	0.031	10011
a_7	1000	1	0.015	10111
a_8	0110	1	0.015	101100
a_9	0001	1	0.015	101101

we can calculate the number of bits required to encode I_{teapot} as follows:

$$\begin{aligned} Compressed\ I_{teapot} &= 1\times 40+2\times 10+4\times 4+4\times 3+2\times 5 \\ &\quad +2\times 5+1\times 5+1\times 6+1\times 6 \\ &= 125\ bits. \end{aligned}$$

Therefore the resulting compression rate is about $125/256 = 0.488$ bits/pixel as it should be under the bound of the entropy for Huffman Coding. The entropy is calculated as

$$H_{huffman}(I_{teapot}) = -\sum_{i=1}^{n} P(a_i)\log_2 P(a_i) \tag{2.7}$$

and $P(a_i)$ is given in Table 2.1, thus

$$\begin{aligned} H_{huffman}(I_{teapot}) &= -(0.625)(\log_2 0.625)-(0.156)(\log_2 0.156)-(0.063)(\log_2 0.063) \\ &\quad -(0.049)(\log_2 0.049)-2(0.031)(\log_2 0.031)-3(0.015)(\log_2 0.015) \\ &= 1.890\ \ bits/symbol \\ &= 0.472\ \ bis/pixel. \end{aligned}$$

The above calculation confirms that the compression performance of Huffman Coding is bounded by its entropy.

The decoding procedure of Huffman Coding is straightforward. For the appearance of each Huffman code in the compressed stream, there is a matching symbol. The original data sequence can be recovered by replacing each Huffman code with its matching symbol, which leads to the reconstruction of the image.

Huffman Coding has a number of variations, such as Adaptive Huffman Coding [6] [29] which calculates the frequencies dynamically based on recent actual frequencies of the symbols in the source data, Length-limited Huffman Coding [27] which minimizes the weighted path length to gain better compression, and others [56] [13] [26].

2.5 Run Length Coding

Run Length Coding (RLC) is widely used in binary image compression as well as in other data compression applications. Run Length Coding identifies a sequence with repeated symbols, replacing it with a single symbol followed by the number of times the symbol is repeated. Run Length Coding is a lossless coding scheme. It is capable of achieving high compression performance comparable to Huffman Coding and Arithmetic Coding.

2.5.1 Principle of Run Length Coding

There are various approaches to Run Length Coding, however, the principle behind them is the same. The principle is to replace the consecutive identical symbols (which is called a run) with a single symbol followed by the count number of the consecutive identical symbols. For example, given the following data sequence

$$000000011000000111100$$

it can be encoded by Run Length Coding as

$$08, 12, 06, 14, 02,$$

where the code consists of two part: the first part is the symbol which makes up the run; the second part is the count number to specify the number of symbols within the run. In this example, the first run is 0's and there are 8 of them. Therefore, the codeword 08 is used to encode the first run. Eventually codeword 08 needs to be converted to binary format. Because there are only 0's and 1's in the data sequence, they are directly employed to specify the symbol of the run. 8 is converted to the binary format 111.[2] Table 2.3 shows the code word for each run of the previous data sequence.

In the above example, we use three bits to represent the length of each run. We know that $2^3 = 8$, therefore, it is only able to code a run with size of eight or less. If the size

[2]A run with the length of zero does not exist, as a result binary value 000 is never used. Therefore, we use binary value 000 to represent decimal value 1, binary value 001 to represent decimal value 2 and so on.

Table 2.3: the Codeword for Each Run

		Decimal		Binary	
Orders	Runs	symbols	Codewords	symbols	Codewords
1	00000000	0	8	0	111
2	11	1	2	1	001
3	000000	0	6	0	101
4	1111	1	4	1	011
5	00	0	2	0	001

of a run is over eight, we need to break it into two or more runs. If the size of a run is much less than eight, we still need to encode it with three bits, which in fact should have been encoded in less than three bits. Both of the above cases create a significant amount of overheads. Techniques have been introduced to solve this problem, which include: adaptive coding [20] where the maximum length of a run may vary adaptively; optimum coding [35] which finds the optimum fixed run length.

2.5.2 Probability Based Run Length Coding

Probability Based Run Length Coding is a combination of Run Length Coding and Huffman Coding. It assigns variable length codewords to the runs based on the probability distribution. This technique is used in ITU-T[3] facsimile compression standards: T.4 (Group 3) and T.6 (Group 4) [32], which are also referred to as modified Huffman coding (MH) and modified modified READ (MMR) [49]. MMR has a better compression performance than MH, because MMR exploits the two dimensional correlations while MH only exploits the one dimensional correlations. Besides MMR, coding schemes such as Block-run Run-length Coding [59] and Vector Run-length Coding [57] also make use of the two-dimensional correlations to yield high compression performance.

Consider the image I_{teapot} from Equation (2.1). A raster scan of the image produces such a data sequence $\{000000\cdots\}$. Table 2.4 shows the probability distribution of each run of the data sequence. Huffman Coding is applied so that a unique codeword is assigned to each run.

The codeword alone is not adequate to encode a run. A 0 or 1 must be given before

[3]ITU-T is one of four permanent parts of the International Telecommunications Union. It was also known as the Consultative Committee for International Telephone and Telegraph (CCITT)

Table 2.4: Run Length Probability Distribution for I_{teapot}

Run Length	Occurrences	Probabilities	Codewords
1	6	6/27	10
5	5	5/27	11
7	4	4/27	000
2	2	2/27	0010
9	2	2/27	0011
3	1	1/27	01000
6	1	1/27	01001
8	1	1/27	01010
10	1	1/27	01011
11	1	1/27	01100
12	1	1/27	01101
56	1	1/27	01110
69	1	1/27	01111

the codeword to indicate whether such a run is composed of 0's or 1's. Therefore, the total number of bits to encode I_{teapot} is

$$\begin{aligned} \textit{Compressed } I_{teapot} &= (6+2\times 6)+(5+2\times 5)+(4+3\times 4)+(2+4\times 2)+(2+4\times 2) \\ &\quad +(1+5\times 1)+(1+5\times 1)+(1+5\times 1)+(1+5\times 1) \\ &\quad +(1+5\times 1)+(1+5\times 1)+(1+5\times 1)+(1+5\times 1) \\ &= 117 \textit{ bits.} \end{aligned}$$

Therefore, the resulting compression rate is about $117/256 = 0.457$ bits/pixel as it should be under the bound of the entropy. The entropy is calculated as

$$H_{rlc}(I_{teapot}) = 1 - \sum_{i=1}^{n} P(r_i)\log_2 P(r_i) \tag{2.8}$$

where, 1 is the extra bit to specify the symbol of the run, r_i is a run and n is the total number of r_i. Thus,

$$\begin{aligned} H_{rlc}(I_{teapot}) &= 1-(6/27)(\log_2(6/27))-(5/27)(\log_2(5/27))-(4/27)(\log_2(4/27)) \\ &\quad -2(2/27)(\log_2(2/27))-8(1/27)(\log_2(1/27)) \\ &= 4.306 \textit{ bits/run} \end{aligned}$$

Because there are total number of 27 runs in the image I_{teapot}, we expect a minimum of $4.306 \times 27 = 116.262$ bits to encode the entire image. Therefore, we can also state

$H_{rlc}(I_{teapot})$ as follows:

$$\begin{aligned} H_{rlc}(I_{teapot}) &= 116.262\ bits/256\ pixels \\ &= 0.454\ bits/pixel \end{aligned}$$

The actual compression rate at 0.457 bits/pixel is bounded by the entropy at 0.454 bits/pixel. We observe that the compression rate and the entropy are very close. This is because both the statistical properties and the one-dimensional correlation of the image are well exploited. We say it is a one-dimensional correlation because the data source is obtained from raster scan order. Run Length Coding such as Block-run Run-length Coding [59], or Vector Run-length Coding [57] makes use of the two-dimensional correlation between adjacent lines of pixels, which yields even a higher compression rate.

In the above example, the run length set and its probability distribution is known in advance to build the Huffman codewords. In memoryless data source this is not possible. Alternative Run Length Coding schemes such as Golomb Coding [12], Rice Coding [45] and adaptive truncated run-length coding [55] which use prefabricated codewords to encode the runs, also yield excellent compression performance.

2.6 Arithmetic Coding

Arithmetic coding (AC) is one of the most efficient coding schemes in data compression. It is adopted in a variety of lossless and lossy compression applications, such as H.264/AVC [30] [46], MPEG [11], JPEG2000 [40], JBIG [39], etc. The development of Arithmetic Coding is well discussed in Sayood [50].

In comparison to the Huffman Coding and Run Length Coding, Arithmetic Coding overcomes the constraint that an input symbol is encoded with a specific codeword. Instead, it encodes the entire data sequence into a single decimal number C, $0.0 \leq C < 1.0$. For example, we can choose a decimal number $0.692\cdots$ to represent the image I_{teapot}. This is possible, because there are infinitely many decimal numbers in the interval $[0.0, 1.0)$[4]. In order to do this, we need to associate the data sequence with a designated decimal number. The relationship between the data sequence and the designated decimal number can be described by a function which maps the entire data sequence to a unique decimal number C. This function is given by

$$C = f_{ac}(S) \tag{2.9}$$

[4] $[0.0, 1.0)$ is the range between 0.0 and 1.0, in which, 0.0 is included and 1.0 is excluded

where S is the data sequence. Function f_{ac} is a one-to-one function.

Since the Arithmetic Coding is adopted in the proposed algorithm, the coding scheme will be discussed in detail in Section 3.2.2.

2.6.1 Finite Context Modelling

Finite context modelling assigns a probability to a symbol based on the context where the symbol resides. A good model will admit useful statistical information which can be utilized by Arithemtic Coding to achieve better compression performance.

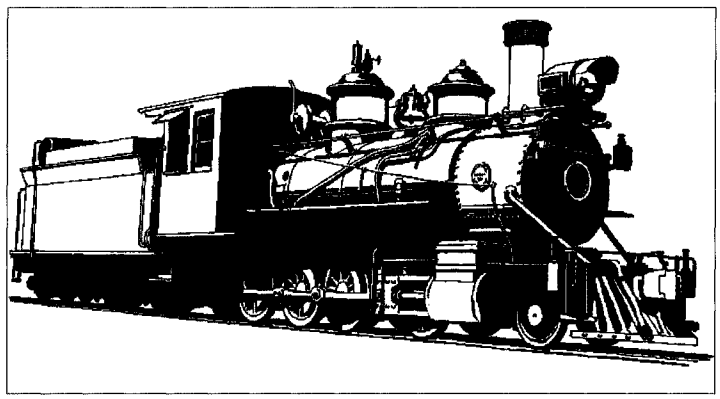


Figure 2.5: Binary Image I_{train}

The simplest finite context model is the order 0 model, in which the probability of each symbol is independent from any previous symbols. It is calculated by the number of occurrences of the symbol divided by the total number of symbols in the data sequence. For example, consider image I_{train} in Figure 2.5. The size of I_{train} is 400×700 and the total number of pixels is $280,000$. The numbers of occurrences of 0 and 1 are $174,995$ and $105,005$ respectively. Therefore, the probability distribution of the order 0 model of image I_{train} is

$$P(0) = \frac{174,995}{280,000} = 0.625 \quad \textit{and} \quad P(1) = \frac{105,005}{280,000} = 0.375.$$

For the order 1 model, the context of a symbol covers its one preceding symbol. The probability of a symbol is dependent on its immediately preceding symbol. Likewise, the context of the order 2 model covers two preceding symbols; the context of the order n model covers n preceding symbols.

In image I_{train}, there are 174,994 contexts where 0 is the immediately preceding symbol, in which, 169,631 contexts have 0 as the current symbol and 5,363 contexts have 1 as the current symbol. There are 105,005 contexts where 1 is the immediately preceding symbol, in which, 5,363 contexts have 0 as the current symbol and 99,642 contexts have 1 as the current symbol. The order 1 probability distributions of image I_{train} is calculated as follows:

$$P(0|0) = \frac{169631}{174994} = 0.969, \quad P(1|0) = \frac{5363}{174994} = 0.031$$
$$P(0|1) = \frac{5363}{105005} = 0.051, \quad P(1|1) = \frac{99642}{105005} = 0.949$$

Table 2.5 shows the compression performance of Arithmetic Coding using order 0 and order 1 models. It is observed that the order 1 model produced much better results than the order 0 model. It is significantly important to have a favorable model to obtain a high compression ratio. As it is stated in [37], "Improvements to the model will yield improved compression effectiveness, that is, a decrease in the size of the encoded data".

Table 2.5: the Compression Results of order 0 and order 1 on I_{train}

$I_{train}(400 \times 700)$	order 0	order 1
compression size(bits)	268220	65390
compression rate(bpp)	0.958	0.234
compression ratio(%)	4.2	76.6

2.6.2 Adaptive Modelling

From the previous example, it seems logical that as the order of the model increases, the compression ratio ought to improve. In fact, this is not true, because the overhead increases as well. The overhead is the probability distributions to be sent from the encoder to the decoder.

Table 2.6 shows the possible number of contexts from order 0 to order 10, as well as the number of bits required to encode the probability distributions, provided that the probability is rounded to three decimal places and encoded in 10 binary bits. It is shown that the number of bits required to encode the probability distributions increases exponentially while the order of the model increases.

Table 2.6: the Number of Bits Required to Code the Probability Distributions

orders	number of contexts	number of bits
order 0	2^0	10
order 1	2^1	20
order 2	2^2	40
order 3	2^3	80
order 4	2^4	160
order 5	2^5	320
order 6	2^6	640
order 7	2^7	1280
order 8	2^8	2560
order 9	2^9	5120
order 10	2^{10}	10240
⋮	⋮	⋮

It is a contradiction that on one hand we want to increase the order of the model to obtain more compression; on the other hand we want to keep a relatively small amount of bits to encode the probability distributions. Adaptive modelling provides an elegant solution. Under adaptive modelling, the encoder does not need to send the probability distributions to the decoder. The decoder estimates the probability distributions. Provided that the encoder and the decoder follow the same procedures, they should estimate the same probabilities. Therefore, all the probabilities are estimated independently during the processes of encoding and decoding.

When the encoder receives a new input symbol, it simply encodes it based on the current existing probabilities. After that, the count of the symbol is updated, which leads to the update of the probabilities. The updated probabilities are used to encode the next symbol. Since the value of the new symbol is known to the encoder before it is encoded, the encoder should be able to update the probabilities and encode the new symbol using the updated probabilities. However, this is restrictive, because the decoder is unable to carry out the exactly same procedure. The decoder cannot update the count of the symbol and its probability without knowing the value of the symbol. Thus, the decoder must first decode the symbol based on the current existing probabilities and update the probabilities afterwards. As a result, the encoder must proceed in the same fashion.

2.6.3 The JBIG Standards

Arithmetic Coding is the coding scheme recommended by the Joint Bi-level Image Processing Group (JBIG)[5] as part of JBIG [23], the international standard for binary image compression.

JBIG uses the arithmetic coder: the QM-Coder [43], patented by three organizations: IBM, Mitsubishi, and Lucent. The QM-coder uses low precision, rapidly adaptable probability prediction combined with a multiply-less arithmetic coder. The probability prediction is intimately tied to the interval calculations necessary for the Arithmetic Coding. The probability prediction is based on the previous 10 pixels on current and previous lines of the pixels. Two different templates used in JBIG are shown in Figure 2.6 and Figure 2.7. In both templates 'C' is the pixel currently being encoded and the previous 10 pixels of 'X' are used to build the context. Therefore, there is a total number of $2^{10} = 1024$ different contexts which leads to 1024 probability distributions. JBIG encodes each pixel using the context for that pixel to determine which of the 1024 probability distributions is applied to encode the pixel. [54]

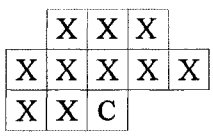


Figure 2.6: JBIG Context Modelling Template 1

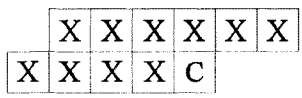


Figure 2.7: JBIG Context Modelling Template 2

JBIG2 [39] is the latest international standard for binary image compression. It was released as the ITU T.88 in 2000, and ISO/IEC 14492 in 2001. With the presence of

[5]JBIG is a joint experts group of the International Standard Organization (ISO), International Electrotechnical Commission (IEC), and the Consultative Committee on International Telephone and Telegraph (CCITT) under the International Telecommunications Union (ITU), which is a part of the United Nations.

JBIG2, the original JBIG compression scheme is often referred as JBIG1. Unlike JBIG1 which encodes an image uniformly, the JBIG2 decomposes an image into several regions and encodes each region separately with different coding schemes. For example, the symbol coding is applied on a text region, the halftone coding is applied on a halftone region. As a result, JBIG2 provides a better compression performance over JBIG1. The coder used in JBIG2 is the MQ-coder [49], which is technically different from the QM-coder of JBIG1, JBIG2 is patented by two organizations: IBM and Mitsubishi.

JBIG2 has another advantage: the capability of lossy, lossless, and lossy-to-lossless compression. JBIG2 is the first international standard that provides for lossy compression; the existing standards are strictly lossless. [18] Although JBIG1 also supports lossy compression, the lossy image produced by JBIG1 has significantly lower quality than the original image. JBIG2 is capable of lossy compression at much higher compression ratios than the lossless ratios of the existing standards, with almost no visible degradation of quality. Since this thesis addresses lossless binary image compression, all data provided in this thesis are based on lossless compression.

Chapter 3

The Proposed Algorithm

In this chapter, a new efficient algorithm for lossless binary image compression is presented. The new algorithm consists of two modules: (1) Direct Redundancy Exploitation (DRE); and (2) Improved Arithmetic Coding (IAC), that we will refer to as the Two Modules Based Algorithm (TMBA). The DRE module is a novel method which exploits the two-dimensional redundancy of an image by removing the identical consecutive rows and columns of pixels. Binary reference vectors are generated to indicate the exact locations where removals take place. In the IAC module, the Markov Model of order 2 is adopted to model the reference vectors. A new Static Binary Tree Model is introduced to efficiently model the Reduced Blocks. Arithmetic coding is applied based on the probability distributions provided by these two models. Figure 3.1 shows the overview of the proposed algorithm.

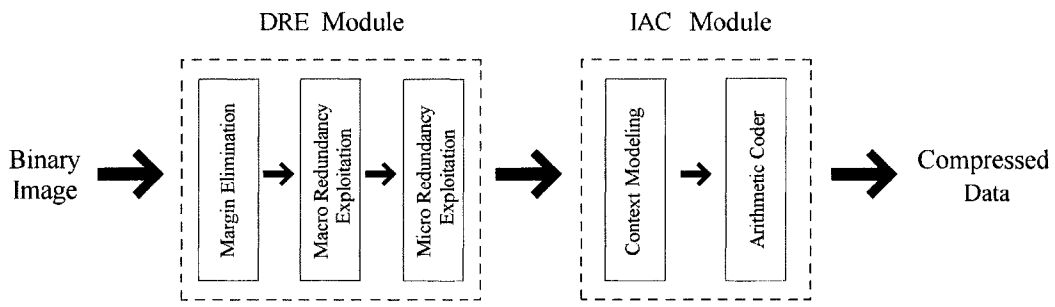


Figure 3.1: The Proposed Two Modules Based Algorithm

3.1 Direct Redundancy Exploitation (DRE)

The Direct Redundancy Exploitation module is a new method which efficiently exploits the two-dimensional redundancy of an image by removing the identical consecutive rows and columns of pixels. Binary vectors are generated to indicate the exact locations of the removed rows and columns of pixels. These binary vectors are referred to as reference vectors. The DRE module comprises three parts: (i) Margin Elimination; (ii) Macro Redundancy Exploitaton; (iii) Micro Redundancy Exploitation.

3.1.1 Margin Elimination

Binary images are most likely to have blank spaces bordering around the objects of the images. The blank spaces can be either black or white and they are referred to as margins in this thesis. Margin Elimination is the process of cropping out the margins from the original image, which results in a smaller image. Therefore, the image is going to be compressed at a reduced size, and it leads to a better compression performance.

Consider an input image I with the size of $N \times M$. Margin Elimination removes the margins on the top, bottom, left and right of the objects. We can find what pixels the margins are composed of by finding the first pixel value of the image, $I(1,1)$. If $I(1,1) = 1$, we consider the margins are black. If $I(1,1) = 0$, we consider the margins are white. Therefore, margins are defined as follows:

$$Top\ Margin = I(1 : t-1, :) \tag{3.1}$$

where, t is the row coordinate of the first row, from top to bottom, containing one or more pixels the values of which are different from the value of the first pixel $I(1,1)$. For example, if $I(1,1) = 0$, we need to locate the first row containing 1 or $1's$, and t is its row coordinate.

$$Bottom\ Margin = I(b+1 : N, :) \tag{3.2}$$

where b is the row coordinate of the first row, from bottom to top, containing one or more pixels the values of which are different from the value of the first pixel $I(1,1)$.

$$Left\ Margin = I(:, 1 : l-1) \tag{3.3}$$

where l is the column coordinate of the first column, from left to right, containing one or more pixels the values of which are different from the value of the first pixel $I(1,1)$.

$$Right\ Margin = I(:, r+1 : M) \tag{3.4}$$

where r is the column coordinate of the first column, from right to left, containing one or more pixels the values of which are different from the value of the first pixel $I(1,1)$.

Upon the identification of each margin, Margin Elimination can be carried out by cropping the image I as follows:

$$I_{margins} = I(t : b, l : r) \tag{3.5}$$

where the size of $I_{margins}$ is $n \times m$ and $n = b - t + 1, m = r - l + 1$. The coordinates: (t, l) and (b, r) to specify where the new image $I_{margins}$ lies must be saved for the reconstruction of the original image. Figure 3.2 shows the process of Margin Elimination applied on image $I_{motorcycle}$.

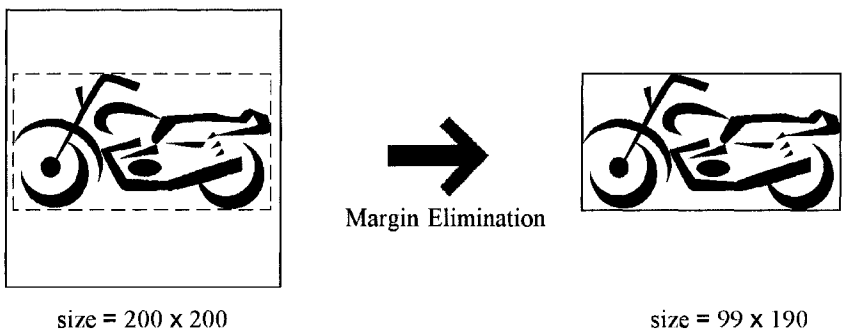


Figure 3.2: Margin Elimination on Image $I_{motorcycle}$

Margin Elimination is the process of removing the margins from an image I, which leads to a new image $I_{margins}$. From this step on, $I_{margins}$ is considered as the image to be compressed.

3.1.2 Macro Redundancy Exploitation

The objective of Macro Redundancy Exploitation is to further reduce the size of the image. It exploits the redundant spaces between two neighbouring rows and columns of pixels of an image. It removes the rows and columns of pixels which are identical to their immediate preceding rows and columns of pixels. The removed rows and columns are referred to as macro redundant rows and columns.

Macro Redundant Rows

Let I be a binary image, $I = (R_1, R_2, \cdots, R_n)$, the size of I is $n \times m$. $R_i \in \{R_1, R_2, \cdots, R_n\}$ is a row of pixels. The macro redundant row (marr) is defined as R_{marr}, and

$$marr = \{i \mid R_i = R_{i-1}\}, \quad 1 < i \leq n. \tag{3.6}$$

In words, if row R_i is identical to its above neighbouring row R_{i-1}, we say R_i is a macro redundant row. Upon its identification, the macro redundant row is removed. This removal process must be reversible in the recontruction process of the original image. In order to achieve that, a reference bit is assigned to each row of the image, which leads to the construction of the reference vector $V_{macro\ row}$:

$$V_{macro\ row}(i) = \begin{cases} 0 & \text{if } i \in marr \\ 1 & \text{otherwise} \end{cases} \qquad 1 \leq i \leq n \tag{3.7}$$

If a row is identified as a macro redundant row, 0 is assigned to the reference bit. In the reconstruction process of the original image, a removed macro redundant row can be recovered by duplicating its immediate preceding row if the value of the reference bit is 0.

Macro Redundant Columns

Likewise, let $I = (C_1, C_2, \cdots, C_m)$, and $C_j \in \{C_1, C_2, \cdots, C_m\}$ is a column of pixels. The macro redundant column (marc) is defined as C_{marc}, and

$$marc = \{j \mid C_j = C_{j-1}\}, \quad 1 < j \leq m. \tag{3.8}$$

In words, if column C_j is identical to its left neighbouring column C_{j-1}, C_j is a macro redundant column. The macro redundant column is removed upon its identification. Similarly, the reference vector $V_{macro\ column}$ is constructed as follows:

$$V_{macro\ column}(j) = \begin{cases} 0 & \text{if } j \in marc \\ 1 & \text{otherwise} \end{cases} \qquad 1 \leq j \leq m \tag{3.9}$$

V_{macro} is the concatenation of $V_{macro\ row}$ and $V_{macro\ column}$, and $V_{macro} = (V_{macro\ row}, V_{macro\ column})$ will be further processed via the IAC module in later section. Figure 3.3 shows that the Macro Redundancy Exploitation is applied to the image $I_{motorcycle}$. Notice that the size of $I_{motorcycle}$ has been further reduced.

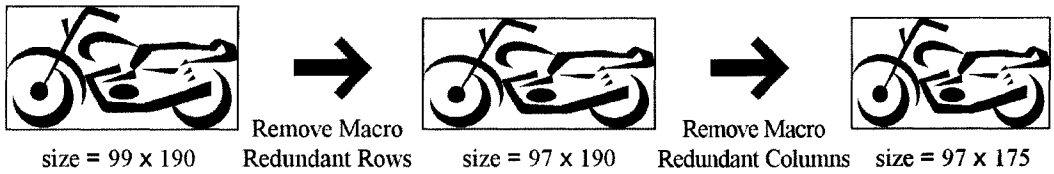


Figure 3.3: Macro Redundancy Elimination on image $I_{motorcycle}$

3.1.3 Micro Redundancy Exploitation

A reduced binary image is partitioned into a number of blocks in this process. Micro Redundancy Exploitation removes those rows and columns which are identical to their preceding rows and columns within a partitiioned block of the image. The removed rows and columns within the block are referred to as micro redundant rows and columns. This part of the algorithm is important because the size of an image is tremendously reduced, yet the algorithm produces a "stair" phenomenon which is exploited by the proposed context tree modelling as explained Section 3.2.1. Micro redundancy consists of two types of redundant spaces: (i) micro redundant rows; (ii) micro redundant columns.

Micro Redundant Rows

Since the micro redundant rows are defined within a block of the image, the image is partitioned into a number of regions which are referred to as blocks.

Consider the image $I_{motorcycle}$ with the size of $n \times m$ in Figure 3.4. It is columnwise partitioned into a number of blocks. The size of each block is $n \times k$ if m is divisible by k, where $0 < k \leq m$. If m is not divisible by k, we have a remainder block which is the rightmost block of the image. Although the size of the remainder block is different from the rest of the blocks, it is treated the same in the subsequent steps. Thus, $I = (B_1, B_2, \cdots, B_u)$ and $B_x \in \{B_1, B_2, \cdots, B_u\}$.

Block B_x in Figure 3.5 is a composition of a number of rows, $B_x = (R_1, R_2, \cdots, R_n)$. $R_i \in \{R_1, R_2, \cdots, R_n\}$, is a row in block B_x. The micro redundant row (mirr) is a row identical to its above neighbouring row in the block, which is defined as R_{mirr}, and

$$mirr = \{i \mid R_i = R_{i-1}\}. \tag{3.10}$$

For example, in Figure 3.5, R_6 and R_7 are micro redundant rows, because $R_6 = R_5$ and

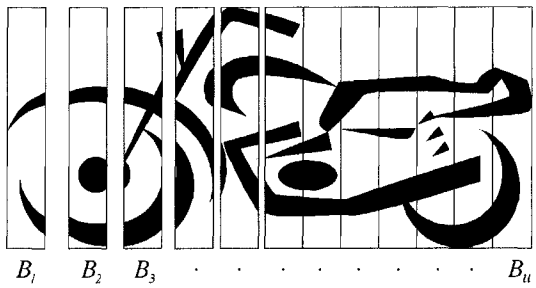


Figure 3.4: Columnwise Partitioning on Image $I_{motorcycle}$

$R_7 = R_6$. The micro redundant rows are removed upon their identification.

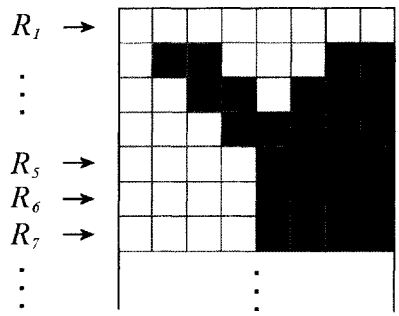


Figure 3.5: Block B_x $size = n \times k$

For decoding purposes, reference vectors are generated to indicate the exact locations of micro redundant rows. Hence, there is a reference vector V_x for every corresponding block B_x and,

$$V_x(i) = \begin{cases} 0 & \text{if } i \in mirr \\ 1 & \text{otherwise} \end{cases} \qquad 1 \leq i \leq n \tag{3.11}$$

$V_{micro\ row} = (V_1, V_2, \cdots, V_u)$ will be further compressed via IAC module in Section 3.2.

Micro Redundant Columns

The resulting bitmap from the previous step is further reduced by eliminating the micro redundant columns. The micro redundant columns are those columns which are exactly

identical or partially identical to their left neighbouring columns within a sub-block.

From the previous step, an image is columnwise partitioned into a number of blocks: $I = (B_1, B_2, \cdots, B_u)$. For every block $B_x \in \{B_1, B_2, \cdots, B_u\}$, B_x is further partitioned rowwisely into a number of sub-blocks, which is $B_x = (b_1, b_2, \cdots, b_v)$ and $b_y \in \{b_1, b_2, \cdots, b_v\}$. For example, consider B_3 in Figure 3.6, it is partitioned into three sub-blocks, which are $B_3 = (b_1, b_2, b_3)$.

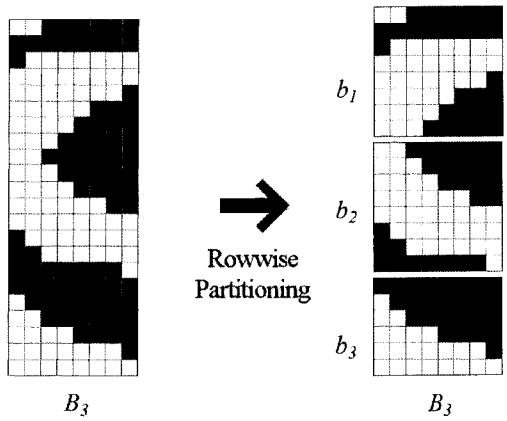


Figure 3.6: Rowwise Partitioning

A sub-block b_y can be considered as a composition of a number of columns, which is $b_y = (C_1, C_2, \cdots, C_k)$ and $C_j \in \{C_1, C_2, \cdots, C_k)\}$, where k is the number of columns within the sub-block b_y. The micro redundant columns consist of two groups: *full* and *partial*.

A full micro redundant column is a column that is exactly identical to its left neighbouring column. It is defined as C_{full},

$$full = \{j \mid C_j = C_{j-1}\}, \quad 1 < j \leq k. \tag{3.12}$$

Consider a sub-block b_2 in Figure 3.7. Column C_6 is identified as a micro redundant column, because $C_6 = C_5$. As C_1 is the first column in sub-block b_2, it cannot be a micro redundant column according to the definition defined in Equation (3.12), $1 < j \leq k$. However C_1 may be identical to its left neighbouring column which is located in the left neighbouring sub-block. Therefore, another definition is needed to manage the first column in every sub-block.

Consider three neighbouring sub-blocks b'_y, b_y, b''_y belong to three different blocks B_{x-1}, B_x and B_{x+1} in Figure 3.8. The length of C_1, denoted as $l(C_1)$, is less than or equal to the

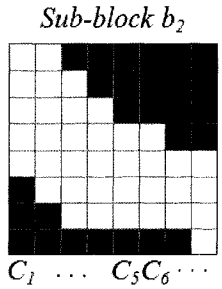


Figure 3.7: Full Micro Redundant Columns

length of C'_k, $l(C_1) \leq l(C'_k)$, and each entry of C_1 is identical to each entry of C'_k in its corresponding position, denoted as $C_1 = C'_k(1 : l(C_1))$. Then C_1 is considered as a partial micro redundant column and will be removed. This is because, C_1 can be fully recovered by duplicating the entries of its neighbouring column C'_k until it reaches its length. Notice that $l(C''_1) \not\leq l(C_k))$, as a result, C''_1 is not considered a partial Micro Redundant Column. This is because C''_1 cannot be fully recovered if it is removed as a partial Micro Redundant Column.

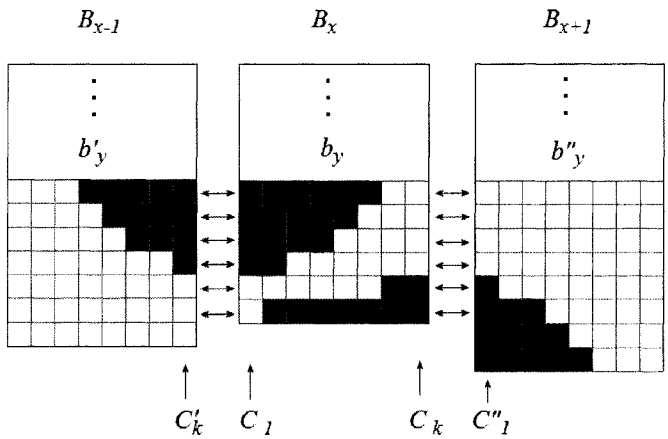


Figure 3.8: Partial Micro Redundant Columns

Therefore, the partial Micro Redundant Column is defined as $C_{partial}$,

$$partial = 1 \;\; if \;\; l(C_1) \leq l(C'_k) \wedge C_1 = C'_k(1 : l(C_1)) \tag{3.13}$$

where, C_1 is the first column of a sub-block b_y in a block B_x, C'_k is the last column of a sub-block b'_y in a block B_{x-1}.

Hence, the micro redundant column (mirc) is defined as C_{mirc}

$$C_{mirc} = C_{full} \cup C_{partial}. \tag{3.14}$$

A reference vector V_y is generated to indicate the locations of micro redundant columns for every corresponding sub-block b_y.

$$V_y(i) = \begin{cases} 0 & \text{if } C_j \in C_{mirc} \\ 1 & \text{otherwise} \end{cases} \qquad 1 \leq j \leq k \tag{3.15}$$

$V_{micro\ column}$ is the concatenation of all reference vectors V_y of all sub-blocks. The process of removing micro redundant columns results in a reduced sub-block, denoted as reduced block (RB).

3.1.4 Summary and Results of the DRE Module

The Direct Redundancy Exploitation (DRE) module is a newly proposed method which efficiently exploits the two-dimensional redundancy of an image. The DRE comprises three parts: (i) Margin Elimination which crops out the blank margins bordering around the binary objects; (ii) Macro Redundancy Exploitaton which removes the identical consecutive rows and columns of pixels of the whole image; (iii) Micro Redundancy Exploitation which removes the identical consecutive rows of pixels within a block as well as the full and partial identical consecutive columns of pixels within a sub-block. Reference vectors are generated in the process of Macro and Micro Redundancy Exploitation to indicate the locations of the removed rows and columns of pixels, which enables the lossless reconstruction of the original image.

A binary image is sequentially processed by the three parts of the DRE module which results in types of data as shown in Figure 3.9. V_{macro} is the concatenation of the reference vectors $V_{macro\ row}$ and $V_{macro\ column}$ which are generated in the process of Macro Redundancy Exploitation. $V_{micro\ row}$ and $V_{micro\ column}$ are the concatenation of the reference vectors which are generated in the process of Micro Redundancy Exploitation. Reduced Blocks (RB) are the remaining sub-blocks.

Table 3.1 shows the results of V_{macro}, $V_{macro\ row}$, $V_{micro\ column}$, RB and total number of bits after DRE module performed on 40 test images. The thumbmails of these test images are given in Appendix A. Figure 3.10 shows the distribution of V_{macro}, $V_{micro\ row}$, $V_{micro\ column}$ and the Reduced Blocks.

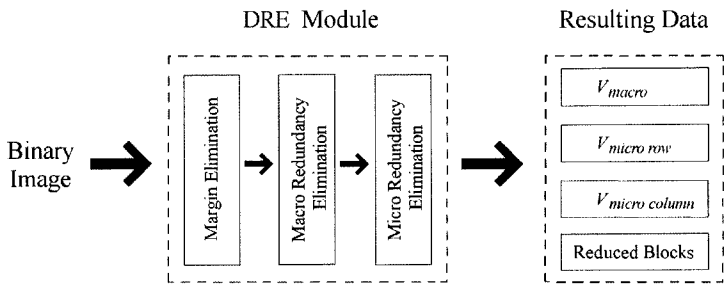


Figure 3.9: the Produced Data After DRE Module

From Table 3.1, we observed that the sizes of the test images have been reduced by 87.62% on average. These Results verify the effectiveness and contribution of the DRE module to the TMBA algorithm.

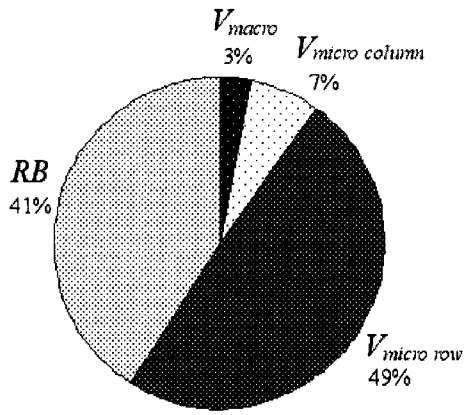


Figure 3.10: The Distribution of V_{macro}, $V_{micro\ row}$, $V_{micro\ column}$ and RB

Table 3.1: DRE Compression Results on 40 Tested Images

Files No.	Original Size (bits)	V_{macro} (bits)	$V_{micro\ row}$ (bits)	$V_{micro\ column}$ (bits)	RB (bits)	DRE Results (bits)
image01	65280	474	7254	2040	13180	22948
image02	35328	338	2840	406	2192	5776
image03	99209	588	3916	613	3047	8164
image04	134784	616	8401	738	3563	13318
image05	250560	384	3625	892	4812	9713
image06	137982	720	5967	727	3931	11355
image07	116160	626	10336	1080	6139	18181
image08	414720	618	10123	3214	18964	32919
image09	141456	727	13377	843	4915	19862
image10	689521	1568	26970	1155	8699	38392
image11	96338	557	9695	2981	18157	31390
image12	202320	811	13104	1280	6572	21767
image13	244400	900	21350	3446	20380	46076
image14	187880	737	10440	1250	6972	19399
image15	26505	300	1170	179	999	2648
image16	81524	422	4176	708	3553	8859
image17	40000	386	4800	1164	6588	12938
image18	114000	606	10200	1498	8841	21145
image19	324864	1063	34505	2034	12079	49681
image20	30880	329	1955	294	1702	4280
image21	264489	1006	16642	1250	6189	25087
image22	40000	361	3982	660	3528	8531
image23	96472	588	9380	1132	6110	17210
image24	159576	706	15351	3796	25683	45536
image25	67104	481	7161	1875	11610	21127
image26	91008	494	4632	501	2843	8470
image27	696320	1309	20055	1042	5691	28097
image28	399224	1002	21156	2724	16338	41220
image29	80775	546	8476	2630	15681	27333
image30	133408	671	13960	4329	28229	47189
image31	98496	645	10950	1851	10118	23564
image32	414720	1123	37526	8891	57303	104843
image33	102408	612	9114	2266	12687	24679
image34	325120	727	13908	979	5197	20811
image35	544640	1177	21525	1057	5542	29301
image36	157248	750	14306	1414	7484	23954
image37	83600	556	6517	977	5655	13705
image38	142464	630	8618	682	3551	13481
image39	250560	469	6369	921	4917	12676
image40	145920	706	12741	906	5623	19976

3.2 Improved Arithmetic Coding (IAC)

The resulting data from the Direct Redundancy Exploitation (DRE) module are V_{macro}, $V_{micro\ row}$, $V_{micro\ column}$ and the Reduced Blocks (RB). In this section, we will discuss how these resulting data are further compressed by the Improved Arithmetic Coding (IAC) module. As illustrated in Figure 3.1, the IAC module consists of two parts: context modelling and the arithmetic coder.

3.2.1 Context Modelling

Context modelling is extremely important for the arithmetic coder. A good model will provide accurate statistical data which favors the arithmetic coder to achieve a high level of compression. Context modelling assigns a probability to a symbol based on the context where the symbol resides. The context modelling presented in this paper is based on the principle: the probability for each encoding symbol is calculated based on the context that the symbol resides in. In fact, the context consists of nothing more than the symbols that have been encountered.

We use two different context models to encode the resulting data from the DRE module. As shown as Figure 3.11, V_{macro}, $V_{micro\ row}$ and $V_{micro\ column}$ are modelled by using the Markov Model [15] and the Reduced Blocks are modelled by using the new proposed Static Binary Tree Model (SBTM).

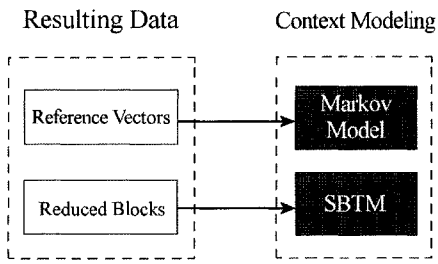


Figure 3.11: The Resulting Data and Two Different Models

Markov Model

The Markov Model assigns the probability to a symbol based on the immediate proceeding symbols which make up the context. The order of a model refers to the number of immediate proceeding symbols within the context. The order 1 model can be characterized by $P(X_i \mid X_{i-1})$, which is the probability distribution of X_i provided that X_{i-1} is the immediate preceding symbol. Thus order 2 can be derived from order 1, which is $P(X_i \mid X_{i-1}X_{i-2})$. The high order Markov Model refers to order 3 or above, which can be described as $P(X_i \mid X_{i-1}X_{i-2}\cdots X_{i-k})$, where k is the number of the order. Figure 3.12 shows the Markov Model from order 1 to order 5 on an example stream.

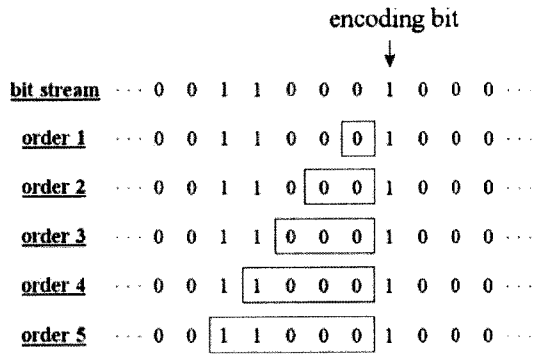


Figure 3.12: Markov Model Order 1 to Order 5

We implemented the Markov Model from order 1 to order 5 followed by the arithmetic coder. Table 3.2 shows the average coding results on the 40 test images. We find that order 2, 3, 4 and 5 models produce very close results. However, while the order of the model increments, the computational cost increases exponentially. Therefore, to avoid the high computational cost, we choose the order 2 model to provide the arithmetic coder with the probability distributions. The final compression results on the 40 test images are based on modelling the V_{macro}, $V_{micro\ row}$ and $V_{micro\ column}$ with the Markov Model order 2. The following are the probability distributions of the Markov Model order 2 on the 40 test images.

$$P(0|0) = 0.923, \quad P(1|0) = 0.077$$
$$P(0|1) = 0.240, \quad P(1|1) = 0.760$$

Table 3.2: Compression Results Using Different Orders of Models

Model	V_{macro} (bits)	$V_{micro\ row}$ (bits)	$V_{micro\ column}$ (bits)
original	683	11664	1661
order 1	315	4363	1266
order 2	285	3883	1255
order 3	271	3750	1258
order 4	273	3728	1327
order 5	275	3705	1336

Static Binary Tree Model

We propose a new context model: the Static Binary Tree Model (SBTM). The SBTM is presented to model the Reduced Blocks (RB). It is based on the unique property: "stair" phenomenon. The "stair" phenonmen is the direct result from Micro Redundancy Exploitation. The formation of the "stair" phenomenon is based on two princicples: (1) the identical lines of pixels are removed in the image, (2) in a relatively small region (Block) two lines of pixels are likely to vary by one pixel if these two lines of pixels are not identical. Consider the example in Figure 3.13. After the process of Micro Redundancy Exploitation is conducted on the block, the Reduced Block forms the "stair" property.

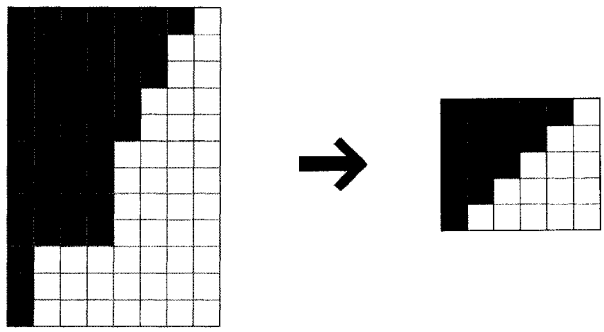


Figure 3.13: Formation of "stair" Property

As shown in Figure 3.14, the "stair" RB may come with the three possible shapes or the combinaion of them. Figure 3.15 shows the four possible orientations of the "stair" RB.

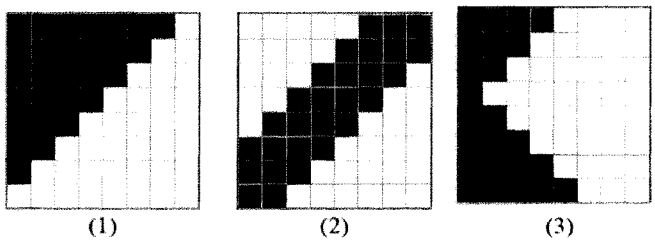


Figure 3.14: Three Possible Shapes

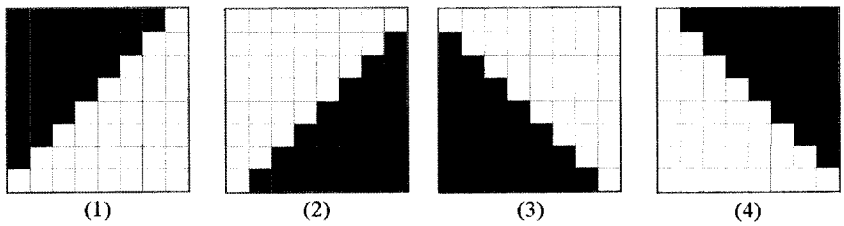


Figure 3.15: Four Possible Orientations

Note that the SBTM model does not require a perfect "stair" RB like these in Figure 3.15 to produce favorable statistics. Consider the examples in Figure 3.16, while the SBTM is able to properly model most of the pixels in the block, only those pixel marked "X" may not be properly modelled, because the model is not designed to handle those special cases.

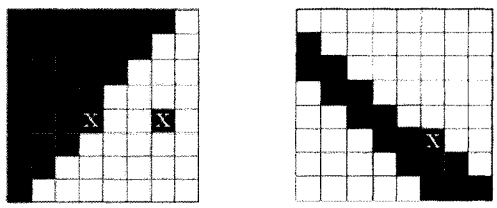


Figure 3.16: Formation of "stair" Property

Based on the "stair" property, the SBTM model calculates the probability of the current encoding bit according to a number of predefined static templates. These templates make up the contexts where the encoding bit resides. They are shown in Figure 3.17. There are

a total number of 20 contexts and they are indexed alphabetically in the Figure.

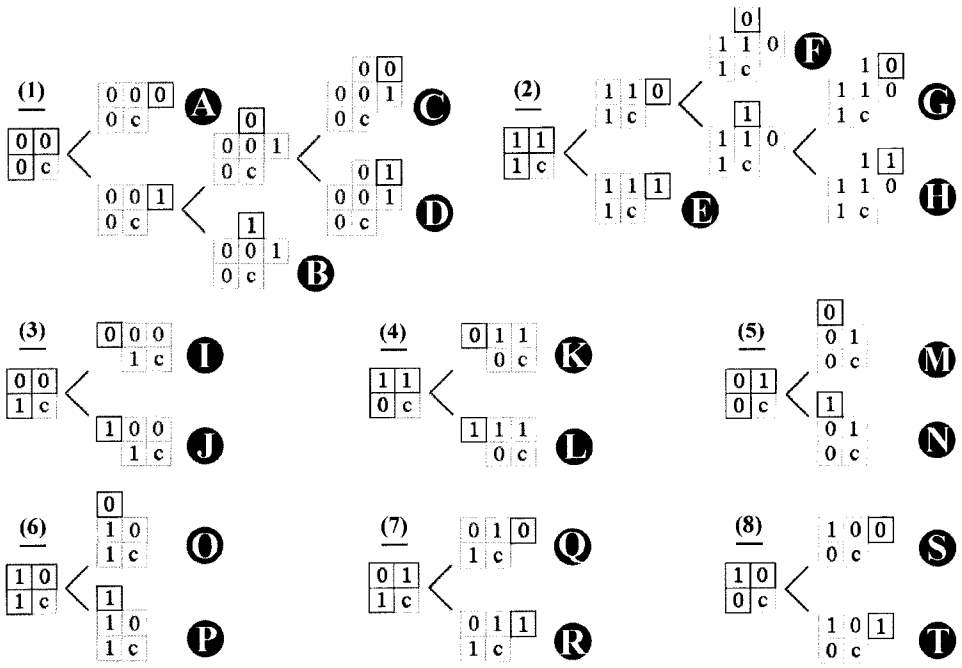


Figure 3.17: Predefined Binary Tree Templates

The contexts are represented by binary trees, in which the context is contructed by three root pixels and other possible branch pixels. The context selection is made by traversing the context tree from the root to the leaf, each time selecting the branch according to the corresponding neighbouring pixel value. Each leaf has a pointer to the statistical model that is to be used. Each node in the tree represents a single context. The two children of a context correspond to the parent context argumented by one pixel. The location of the argumented pixel are fixed in a predefined order.

Let us now work through an example to explain how the SBTM works. Suppose the pixel marked "C" is the current encoding pixel in Figure 3.18. The values of the three root pixels match Template (3). We traverse the context tree according to the value of the corresponding neighbouring pixel which is marked "A" in the figure. The value of the corresponding pixel is 1, therefore the lower leaf of the tree is selected as the modelling context. We stated earlier that each leaf has a pointer to the statistical model. In fact, in the implemental level, we maintain two numbers for each context. One is the total number

of the occurrences of this context, denoted T. The other is the number of the occurrences of 0's in this context, denoted Z. The number of the occurrences of 1's can be found by $T - Z$. Since the current encoding pixel $C = 0$, SBTM calculate the probability as follows:

$$P(C) = \frac{Z}{T}$$

and send the value of $P(C)$ to the arithmetic coder for encoding. After that, the number of the occurrences of the context, T, is incremented by 1. Because the current encoding pixel $C = 0$, the number of the occurrences of 0's is incremented by 1 as well.

The question then arises: since we know the value of the current encoding pixel, why not increment the value of T and Z first, then provide the arithmetic coder with the new probability calculated from the updated values of T and Z? We have to make sure the encoder and decoder have the same probability. The value of the current pixel is known to the encoder. If the frequency of T and Z are incremented, we have a new updated probability and the encoder uses the new updated probability to encode. In order to decode the pixel correctly, the decoder should be provided with the same new updated probability. This is not feasbile because the value of the current pixel is unknown to the decoder before actually decoding it and the new updated probability cannot be obtained until the current pixel is decoded.

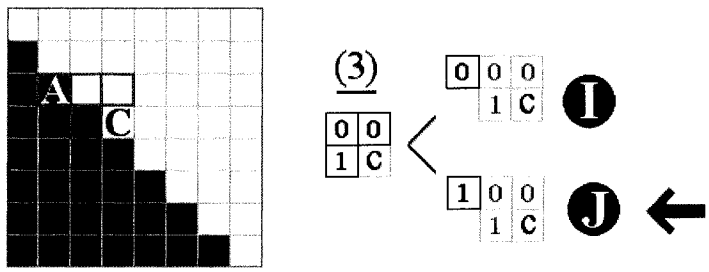


Figure 3.18: An Example of SBTM Modelling Process

The following shows the final probability distributions of the 20 contexts when the SBTM modelling is performed on the 40 test images.

$$
\begin{array}{llll}
P(0|A) = 0.944, & P(1|A) = 0.056, & P(0|B) = 0.906, & P(1|B) = 0.094 \\
P(0|C) = 0.233, & P(1|C) = 0.767, & P(0|D) = 0.727, & P(1|D) = 0.273 \\
P(0|E) = 0.045, & P(1|E) = 0.955, & P(0|F) = 0.099, & P(1|F) = 0.901 \\
P(0|G) = 0.323, & P(1|G) = 0.677, & P(0|H) = 0.833, & P(1|H) = 0.167 \\
P(0|I) = 0.315, & P(1|I) = 0.685, & P(0|J) = 0.800, & P(1|J) = 0.200 \\
P(0|K) = 0.221, & P(1|K) = 0.779, & P(0|L) = 0.721, & P(1|L) = 0.279 \\
P(0|M) = 0.653, & P(1|M) = 0.347, & P(0|N) = 0.899, & P(1|N) = 0.101 \\
P(0|O) = 0.114, & P(1|O) = 0.886, & P(0|P) = 0.417, & P(1|P) = 0.583 \\
P(0|Q) = 0.373, & P(1|Q) = 0.627, & P(0|R) = 0.045, & P(1|R) = 0.955 \\
P(0|S) = 0.652, & P(1|S) = 0.348, & P(0|T) = 0.951, & P(1|T) = 0.049
\end{array}
$$

3.2.2 Arithmetic Coder

The arithmetic coder is applied after each bit is processed by its context model. In this section, we will show how the arithmetic coder works on a specific bit stream. For simplicity, order 0 model is used for modelling process during the illustrated coding process.

Encoding

Arithmetic Coding overcomes the constraint that an input symbol is encoded with a specific codeword. Instead, it encodes the entire data sequence into a single decimal number C, $0.0 \leq C < 1.0$. For example, we can choose a decimal number $0.692\cdots$ to represent an image. This is possible because there are infinitely many decimal numbers in the interval $[0.0, 1.0)$[1]. In order to do this, we need to associate the data sequence with a designated decimal number. The relationship between the data sequence and the designated decimal number can be described by a function which maps the entire data sequence to a unique decimal number C. This function is given by

$$C = f_{ac}(S) \tag{3.16}$$

where S is the data sequence. Function f_{ac} is a one-to-one function, and can be described by the encoding algorithm in Program 3.1, in which, EOF is the end of file, the *upperBound* is

$$upperBound(a_i) = \sum_{i=1}^{k_{a_i}} P_M(a_i) \tag{3.17}$$

[1] $[0.0, 1.0)$ is the range between 0.0 and 1.0, in which, 0.0 is included and 1.0 is excluded

```
low = 0.00;
high = 1.0;
LOOP if not EOF
      a_i = next input symbol;
      range = high - low;
      high = low + range × upperBound(a_i);
      low = low + range × lowerBound(a_i);
end of LOOP
output low;
```

Program 3.1: AC Encoder

and the *lowerBound* is

$$lowerBound(a_i) = \sum_{i=1}^{k_{a_i}-1} P_M(a_i) \tag{3.18}$$

where $P_M(a_i)$ is the probability of a_i under model M, $a_i \in A$ and $A = \{a_1, a_2, \cdots, a_n\}$ is the finite alphabet of the data sequence S. k_{a_i} is the index number of a_i in alphabet A, $1 \leq k_{a_i} \leq n$.

Consider the following data sequence to be encoded by Arithmetic Coding.

$$S = \{0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0\}$$

Thus, the alphabet of S is $A = \{a_1, a_2\} = \{0, 1\}$ and the probability distribution is the following, provided that the order 0 model M is applied.

$$P_M(0) = 0.80 \quad P_M(1) = 0.20$$

and

$$k_0 = 1 \quad k_1 = 2$$

Therefore, the *upperBound* and *lowerBound* of 0 and 1 can be obtained by applying Equations (3.17) and (3.18).

$$upperBound(0) = \sum_{i=1}^{k_0} P_M(a_i) = \sum_{i=1}^{1} P_M(a_i) = 0.80 \tag{3.19}$$

$$lowerBound(0) = \sum_{i=1}^{k_0-1} P_M(a_i) = \sum_{i=1}^{1-1} P_M(a_i) = 0.00 \tag{3.20}$$

and

$$upperBound(1) = \sum_{i=1}^{k_1} P_M(a_i) = \sum_{i=1}^{2} P_M(a_i) = 1.00 \tag{3.21}$$

$$lowerBound(1) = \sum_{i=1}^{k_1-1} P_M(a_i) = \sum_{i=1}^{2-1} P_M(a_i) = 0.80 \tag{3.22}$$

Thus, we can consider symbol 0 owns the interval $[0.0, 0.8)$ and symbol 1 owns the interval $[0.8, 1.0)$. This is graphically presented in Figure 3.19.

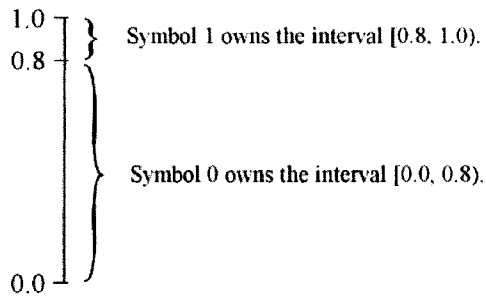


Figure 3.19: Interval of Symbol 0 and 1

Initially, *high* and *low* are assigned to value 1.0 and 0.0, written as $[0.0, 1.0)$. To encode the first symbol 0 of the data sequence S,

$$range = high - low = 1.0 - 0.0 = 1.0$$

$$\begin{aligned} high &= low + range \times upperBound(0) \\ &= 0 + 1.0 \times 0.8 \\ &= 0.8 \end{aligned}$$

$$\begin{aligned} low &= low + range \times lowerBound(0) \\ &= 0 + 1.0 \times 0.0 \\ &= 0.0 \end{aligned}$$

During the process of encoding the first symbol, *high* and *low* are updated, $[0.0, 0.8)$. When the encoder goes on to next iteration, the calculation will be based on the updated values. The next coding symbol is 1,

$$range = high - low = 0.8 - 0.0 = 0.8$$

$$\begin{aligned} high &= low + range \times upperBound(1) \\ &= 0 + 0.8 \times 1.0 \\ &= 0.80 \end{aligned}$$

$$\begin{aligned} low &= low + range \times lowerBound(1) \\ &= 0 + 0.8 \times 0.8 \\ &= 0.64 \end{aligned}$$

The value of *high* and *low* are again updated, $[0.64, 0.80)$. This process of updating the values of *high* and *low* continues until *EOF*. The first four iterations of this process are described graphically in Figure 3.20.

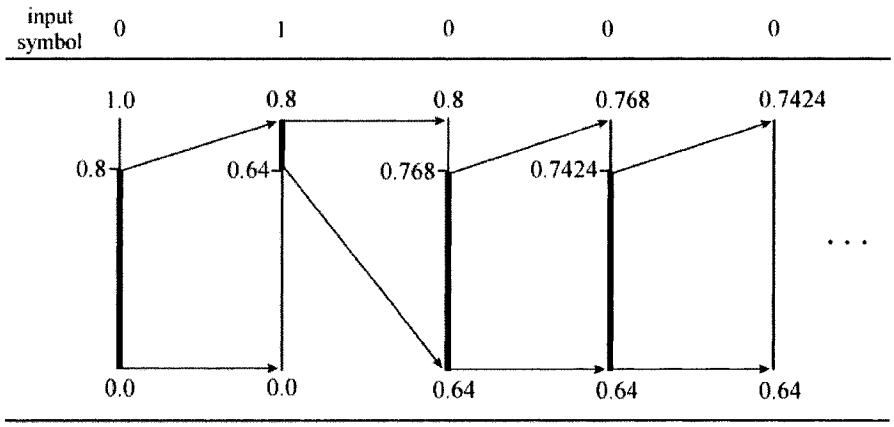


Figure 3.20: the Encoding Process of Arithmetic Coding

Notice that the appearance of each input symbol restricts the updated value *high* and *low* to a subinterval that is disjoint from any other subinterval that may have been generated using this process. For example, by the time the encoder received the third symbol 0, the updated *high* and *low* has been restricted to subinterval $[0.64, 0.768)$. If the third symbol had been 1 instead of 0, the subinterval would be $[0.768, 0.8)$, which is disjoint from the subinterval $[0.64, 0.768)$. Even if the two sequences are identical from this point on, the interval for the two sequences will always be disjoint. This always guarantees that there is

a unique interval for the encoding data sequence. This is why we say function $C = f_{ac}(S)$ in Equation (3.16) is one-to-one.

Table 3.3 shows the values of *high* and *low* after each iteration. The final values of *high* and *low* is $[0.6924288, 0.6991396864)$. In fact, we can choose any number C, $0.6924288 \leq C < 0.6991396864$, to encode the data sequence. For simplicity, the *low* is chosen, therefore 0.6924288 is the final unique number to encode the data sequence 0100001000. It seems that no compression is really gained. This is because the length of the example data sequence is very limited and Arithmetic Coding needs a fair data sequence to demonstrate its elegancy.

Table 3.3: the Updated Values of Each Iteration of the Encoding Process

input symbol	range	low	high
start		0.0	1.0
0	1.0	0.0	0.8
1	0.8	0.64	0.8
0	0.16	0.64	0.768
0	0.128	0.64	0.7424
0	0.1024	0.64	0.72192
0	0.08192	0.64	0.705536
1	0.065536	0.6924288	0.705536
0	0.0131072	0.6924288	0.70291456
0	0.01048576	0.6924288	0.700817408
0	0.008388608	0.6924288	0.6991396864

Decoding

The decoding process is to recover the original data sequence. Given the encoding algorithm, it is relatively easy to see how the decoding process will operate. The encoded number 0.6924288 is given to recover the original data sequence 0100001000. The first symbol of the original data sequence can be found by finding which symbol owns the interval in which the encoded number falls. Since the number 0.6924288 falls in $[0.0, 0.8)$, the first symbol must be 0. Because the first symbol is decoded, its effect needs to be removed from the number 0.6924288 in order to decode the next symbol. Since we know the *high* and *low* values of previously decoded symbol 0, we can remove the effect by reversing this part of the encoding process. First, we subtract the *low* value 0.0 from 0.6924288, resulting in 0.6924288. Then we divide it by its range 0.8. This produces the number 0.865536. Now we need to find

which interval the number 0.865536 falls in. It is obvious the number 0.865536 falls between $[0.8, 1.0)$, therefore the second symbol is 1. This process will be repeated until all symbols are successfully decoded.[2] The decoding algorithm is given in Program 3.2. Table 3.4 shows the decoding process of each symbol of the data sequence 0100001000.

```
C = encoded number;
LOOP until all symbols are decoded
    find the symbol whose interval straddles the encoded number;
    output the symbol;
    range = high - low;
    C = (C-low)/range;
end of LOOP
```

Program 3.2: AC Decoder

Table 3.4: the Decoding Process of Each Symbol

Encoded Number	output symbol	lowerBound	upperBound	range
0.6924288	0	0.0	0.8	0.8
0.865536	1	0.8	1.0	0.2
0.32768	0	0.0	0.8	0.8
0.4096	0	0.0	0.8	0.8
0.512	0	0.0	0.8	0.8
0.64	0	0.0	0.8	0.8
0.8	1	0.8	1.0	0.2
0	0	0.0	0.8	0.8
0	0	0.0	0.8	0.8
0	0	0.0	0.8	0.8

From this example we can see that the length of the encoded decimal number C increases while the encoding is progressing. It will eventually reach the length limit of float numbers supported by the operating system. Therefore, it seems completely impractical to implement the Arithmetic Coding. But in fact, Arithmetic Coding can be implemented using the standard 16 bits or 32 bits integer arithmetics by using the incremental transmission scheme [50].

[2]Provided the decoder has the knowledge of the length of the data sequence

3.2.3 Summary and Results of the IAC Module

The Improved Arithmetic Coding Module further compress the data resulting from the DRE module. The Markov Model order 2 is applied on the reference vectors: V_{macro}, $V_{micro\ row}$ and $V_{micro\ column}$. A new Static Binary Tree Model is proposed in the IAC Module to efficiently model the Reduced Blocks. The arithmetic Coder is applied to each bit based on the probability provided by the modelling process. Table 3.6 shows a before and after comparison of the IAC module. On the average, the IAC module reduced the sizes of reference vectors and Reduced Blocks by 63.3% and 38.4% respectively on the test images.

3.3 Simulation Results

The proposed Two Module Based Algorithm (TMBA) has been tested on 40 images of different size and content. These test images are displayed as thumbnails in Appendix A. The simulation results of the proposed algorithm are compared with four industrial standard coding schemes: G3, G4, JBIG1 and JBIG2. The compression results in bits are shown in Table 3.7 and the compression ratios are shown in Table 3.8.

The TMBA algorithm has improved the compression ratio by about 62% and 30% in comparison to the G3 and G4 standards, on average. It outperforms G3 and G4 on all test images. The TMBA algorithm has also yielded a slight increase of compression performance in comparison to the JBIG1 and JBIG2 standards. On 30 out of 40 test images, TMBA outperforms JBIG1. On 26 out of 40 test images TMBA outperforms JBIG2. The average compression rate (bit/pixel) of each scheme is shown in Table 3.5

Table 3.5: The Average Compression Rate on 40 Tested Images

G3	G4	JBIG1	JBIG2	TMBA
0.1533	0.0830	0.0592	0.0583	0.0581

Table 3.6: IAC Compression Results on 40 Tested Images

	Reference Vectors		Reduced Blocks	
Files Name	Beofre (bits)	After (bits)	Before (bits)	After (bits)
image01	9768	4610	13180	10505
image02	3584	1610	2192	1115
image03	5117	2440	3047	1375
image04	9755	3430	3563	1605
image05	4901	2610	4812	2685
image06	7424	3000	3931	1665
image07	12042	4020	6139	2335
image08	13955	8130	18964	16275
image09	14947	3260	4915	1665
image10	29693	3730	8699	1245
image11	13233	7640	18157	11575
image12	15195	5010	6572	3525
image13	25696	11130	20380	15335
image14	12427	4660	6972	3225
image15	1649	810	999	475
image16	5306	2310	3553	1645
image17	6350	3710	6588	3355
image18	12304	3930	8841	5635
image19	37602	6070	12079	5715
image20	2578	1190	1702	915
image21	18898	5550	6189	2695
image22	5003	2410	3528	1835
image23	11100	4340	6110	2845
image24	19853	9850	25683	17285
image25	9517	4470	11610	8685
image26	5627	2210	2843	1185
image27	22406	4870	5691	2085
image28	24882	9560	16338	7865
image29	11652	7060	15681	11995
image30	18960	10030	28229	21365
image31	13446	5900	10118	5895
image32	47540	22220	57303	43735
image33	11992	7220	12687	7835
image34	15614	3750	5197	2355
image35	23759	4790	5542	2235
image36	16470	5080	7484	3365
image37	8050	3500	5655	2905
image38	9930	3070	3551	1645
image39	7759	3270	4917	2045
image40	14353	3370	5623	1645

Table 3.7: Compression Results in Bits on 40 Tested Images

Files No.	Original	G3	G4	JBIG1	JBIG2	TMBA
image01	65280	26048	19488	15176	15064	15115
image02	35328	10512	4528	4728	4712	2725
image03	99209	19200	7168	5112	5096	3815
image04	134784	21600	8400	6208	6184	5035
image05	250560	19232	10112	6496	6376	5295
image06	137982	21888	8832	5680	5624	4665
image07	116160	22464	8768	6624	6512	6375
image08	414720	50160	34352	21616	23008	24405
image09	141456	22160	7920	5720	5336	4925
image10	689521	45152	20544	6696	6216	4975
image11	96338	31936	22080	17088	16408	19215
image12	202320	29856	12208	8648	8616	8535
image13	244400	60608	34800	26944	25800	26465
image14	187880	26000	11184	8088	8072	7885
image15	26505	7488	3104	3664	3640	1285
image16	81524	14176	6256	5080	5064	3935
image17	40000	15360	8272	7592	7568	7065
image18	114000	23664	13616	9760	9384	9565
image19	324864	34576	17584	10400	10744	11785
image20	30880	8384	3744	4208	4120	2105
image21	264489	29344	12992	7704	7640	8245
image22	40000	11712	5552	5424	5208	4245
image23	96472	21872	9104	7336	7328	7185
image24	159576	46256	34992	25240	27152	27135
image25	67104	22688	16528	13160	14144	13168
image26	91008	15888	5792	5024	4888	3395
image27	696320	56288	19776	7536	6992	6955
image28	399224	54016	25280	15360	15168	17425
image29	80775	32496	21360	18088	19920	19055
image30	133408	52080	39968	31832	29640	31395
image31	98496	21200	13888	10640	10616	11795
image32	414720	102208	81424	62208	58728	65955
image33	102408	26464	17376	15032	13896	15056
image34	325120	26704	11936	6312	6104	6112
image35	544640	45120	16768	7744	7312	7025
image36	157248	23200	11456	7944	7784	8445
image37	83600	20064	8192	7200	6984	6405
image38	142464	21824	8240	5864	5792	4715
image39	250560	20496	9840	6080	5928	5315
image40	145920	24256	8144	5872	5440	5015

Table 3.8: Compression Ratio on 40 Tested Images

Files No.	G3	G4	JBIG1	JBIG2	TMBA
image01	60.10%	70.15%	76.75%	76.92%	76.85%
image02	70.24%	87.18%	86.62%	86.66%	92.29%
image03	80.65%	92.77%	94.85%	94.86%	96.15%
image04	83.97%	93.77%	95.39%	95.41%	96.26%
image05	92.32%	95.96%	97.41%	97.46%	97.89%
image06	84.14%	93.60%	95.88%	95.92%	96.62%
image07	80.66%	92.45%	94.30%	94.39%	94.51%
image08	87.91%	91.72%	94.79%	94.45%	94.12%
image09	84.33%	94.40%	95.96%	96.23%	96.52%
image10	93.45%	97.02%	99.03%	99.10%	99.28%
image11	66.85%	77.08%	82.26%	82.97%	80.05%
image12	85.24%	93.97%	95.73%	95.74%	95.78%
image13	75.20%	85.76%	88.98%	89.44%	89.17%
image14	86.16%	94.05%	95.70%	95.70%	95.80%
image15	71.75%	88.29%	86.18%	86.27%	95.15%
image16	82.61%	92.33%	93.77%	93.79%	95.17%
image17	61.60%	79.32%	81.02%	81.08%	82.34%
image18	79.24%	88.06%	91.44%	91.77%	91.61%
image19	89.36%	94.59%	96.80%	96.69%	96.37%
image20	72.85%	87.88%	86.37%	86.66%	93.18%
image21	88.91%	95.09%	97.09%	97.11%	96.88%
image22	70.72%	86.12%	86.44%	86.98%	89.39%
image23	77.33%	90.56%	92.40%	92.40%	92.55%
image24	71.01%	78.07%	84.18%	82.98%	83.00%
image25	66.19%	75.37%	80.39%	78.92%	80.38%
image26	82.54%	93.64%	94.48%	94.63%	96.27%
image27	91.92%	97.16%	98.92%	99.00%	99.00%
image28	86.47%	93.67%	96.15%	96.20%	95.64%
image29	59.77%	73.56%	77.61%	75.34%	76.41%
image30	60.96%	70.04%	76.14%	77.78%	76.47%
image31	78.48%	85.90%	89.20%	89.22%	88.02%
image32	75.35%	80.37%	85.00%	85.84%	84.10%
image33	74.16%	83.03%	85.32%	86.43%	85.30%
image34	91.79%	96.33%	98.06%	98.12%	98.12%
image35	91.72%	96.92%	98.58%	98.66%	98.71%
image36	85.25%	92.71%	94.95%	95.05%	94.63%
image37	76.00%	90.20%	91.39%	91.65%	92.34%
image38	84.68%	94.22%	95.88%	95.93%	96.69%
image39	91.82%	96.07%	97.57%	97.63%	97.88%
image40	83.38%	94.42%	95.98%	96.27%	96.56%

Chapter 4

Conclusion

In this thesis, we have studied various current existing coding techniques and standards for lossless binary image compression. We have proposed a new highly efficient algorithm: the Two Modules Based Algorithm.

The proposed algorithm consists of two modules: (1) Direct Redundancy Exploitation (DRE); and (2) Improved Arithmetic Coding (IAC). The Direct Redundancy Exploitation module is a reliable and efficient method to exploits the two-dimensional redundancy of an images. It removes the identical consecutive rows and columns of pixels within the image and the partitioned blocks of the image. The empirical results show that this method alone has yielded an average of 87.62% compression ratio on the tested images. The Improved Arithmetic Coding module is the coding scheme to further compress the data resulting from the DRE module. Markov Model order 2 was chosen for the modelling of the reference vectors. A new context modelling method: the Static Binary Tree Model, has been introduced to model the Reduced Blocks. The design of the new model is based on the "stair" property which is the result of Direct Redundancy Exploitation. The Static Binary Tree Model effectively provides the favorable statistic information for arithmetic coding. At the end, the arithmetic coding is adopted for high compression performance. The simulation results show that the Improved Arithmetic Coding module further compressed the reference vectors and the Reduced Blocks by 63% and 38% respectively.

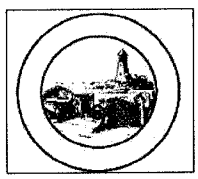
The simulation results show that the proposed algorithm has improved the compression ratio by about 62% and 30% in comparison to the G3 and G4 standards respectively. The proposed algorithm has also yielded an increase of compression performance in comparison to the JBIG1 and JBIG2 standards.

The Two Module Based Algorithm is a reliable and efficient method for lossless binary

image compression. It provides an alternative method for lossless binary image compression. More importantly, the algorithm has demonstrated excellent compression performance which is comparable or better than the current industrial standards.

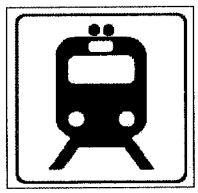
Appendix A

Test Images

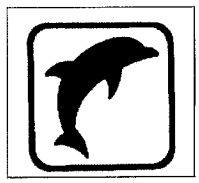


240×272
image01

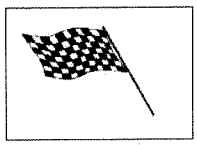
184×192
image02



311×319
image03

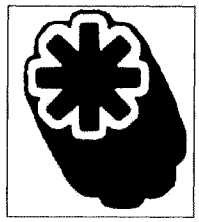


351×384
image04

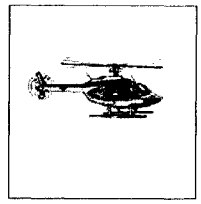


348×720
image05

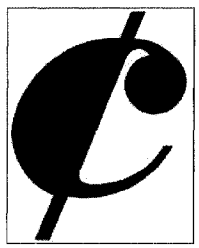
377×366
image06



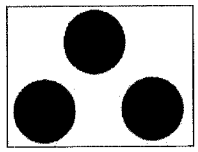
363×320
image07



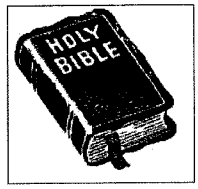
720×576
image08



421×336
image09



719×959
image10



302×319
image11

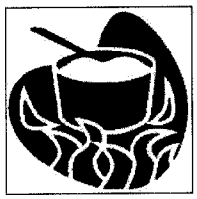
562×360
image12

650×376
image13

474×440
image14

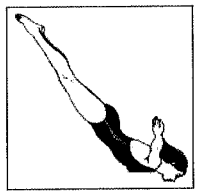
155×171
image15

229×356
image16

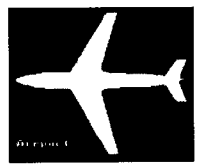


200×200
image17

375×304
image18

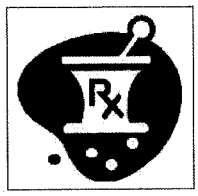


564×576
image19



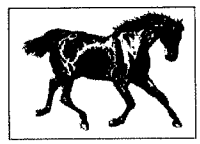
160×193
image20

393×673
image21

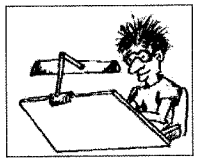


200×200
image22

389×248
image23



327×488
image24



233×288
image25

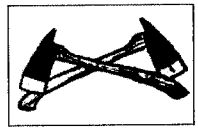
316×288
image26

1088×640
image27

674×576
image28

359×225
image29

379×352
image30



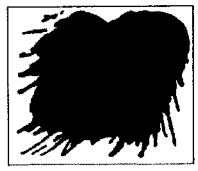
228×432
image31

720×576
image32

251×408
image33

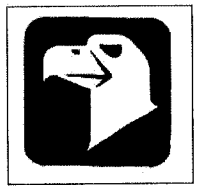
508×640
image34

851×640
image35



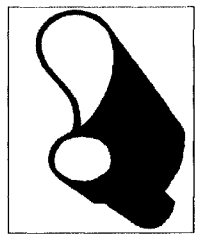
364×432
image36

418×200
image37



371×384
image38

348×720
image39



480×304
image40

Bibliography

[1] E. Ageenko and P. Franti. Lossless Compression of Large Binary Images in Digital Spatial Libraries. *Computers & Graphics*, 24(1):91–98, 2000.

[2] E. Bodden, M. Clasen, and J. Kneis. Arithmetic Coding Revealed. In *Proseminar Datenkompression*, 2002.

[3] F. Bossen and T. Ebrahimi. A Simple and Efficient Binary Shape Coding Technique Based on Bitmap Representation. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP-97)*, volume 4, pages 3129–3132, April 1997.

[4] C. Chamzas and D.L. Duttweiler. Encoding Facsimile Images for Packet-Switched Networks. *IEEE Journal on Selected Areas in Communications*, 7(5):857–864, 1989.

[5] T. Chuang and J. Lin. A New Algorithm for Lossless Still Image Compression. *Journal of Pattern Recognition*, 31(9):1343–1352, 1998.

[6] A. Desoky and M. Gregory. Compression of Text and Binary Files Using Adaptive Huffman Coding Techniques. In *IEEE Conference Proceedings of Southeastcon*, pages 660–663, April 1988.

[7] Robert R. Estes, Jr. and V. Ralph Algazi. Efficient Error Free Chain Coding of Binary Documents. In *Proceedings of IEEE Data Compression Conference*, volume 3, pages 122–131, March 1995.

[8] B. J. Falkowski. Lossless Binary Image Compression Using Logic Functions and Spectra. *Computers & Electrical Engineering*, 30(1):17–43, 2004.

[9] P. Franti and E. Ageenko. On the Use of Context Tree for Binary Image Compression. In *Proceedings of IEEE International Conference on Image Processing(ICIP 99)*, volume 3, pages 752–756, October 1999.

[10] P. Franti and Nevalainen O. Compression of Binary Images by Composite Methods Based on Block Coding. *Journal of Visual Communication and Image Representation*, 6(4):366–377, 1995.

[11] Mohammed Ghanbari. *Standard Codecs: Image Compression to Advanced Video Coding*. The Institution of Electrical Engineers, London, United Kingdom, Herts, United Kingdom, second edition, 2003.

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

[13] P.T. Gonciari, B.M. Al-Hashimi, and N. Nicolici. Variable-Length Input Huffman Coding for System-on-a-Chip Test. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(6):783–796, 2003.

[14] Robert M. Gray. *Entropy and Information Theory.* Spring-Verlag, New York, NY, 1990.

[15] Charles M. Grinstead and J. Laurie Snell. *Introduction to Probability.* American Mathematical Society, second revised edition, 1997.

[16] Darrel Hankersson, Greg A. Harris, and Peter D. Jr. Johnson. *Introduction to Information Theory and Data Compression.* CRC Press, Boca Raton, Florida, 1997.

[17] P. G. Howard. *The Design and Analysis of Efficient Lossless Data Compression Systems.* PhD thesis, Brown University, 1993. Technical Report CS 93-28.

[18] P.G. Howard, F. Kossentini, B. Martins, S. Forchhammer, and W.J. Rucklidge. The Emerging JBIG2 Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 8(7):838–848, 1998.

[19] T. Huang. Coding of Two-Tone Images. *IEEE Transactions on Communications*, 25(11):1406–1424, 1977.

[20] T. S. Huang. Run-Length Coding and its Extensions. *Proc Symp. Picture Bandwidth Compression, Mass. Inst. Tech. Cambridge*, 1969.

[21] D. A. Huffman. A Method for the Construction of Minimum-Redundancy Codes. *Proceeding of IRE*, 40:1098–1101, 1952.

[22] S. Iickho and S. Kassam. A New Noiseless Coding Technique for Binary Images. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 34(4):994–951, 1986.

[23] ISO/IEC JTC1/SC29/WG9. *International Standard 11544, ITU-T Recommendation T.82. Progressive Bi-level Image Compression.* JBIG, 1993.

[24] L. Kamstra. The Design of Nonlinear Binary Wavelet Transforms and Their Application to Binary Image Compression. In *Proceedings of IEEE International Conference on Image Processing(ICIP03)*, volume 3, pages 241–244, September 2003.

[25] B.Y. Kavalerchik. Generalized Block Coding of Black and White Images. *IEEE Transactions on Image Processing*, 1(4):518–520, 1992.

[26] G. Lakhani and V. Ayyagari. Improved Huffman Code Tables for JPEG's Encoder. *IEEE Transactions on Circuits and Systems for Video Technology*, 5(6):562–564, 1995.

[27] L. L. Larmore and D. S. Hirschberg. A fast Algorithm for Optimal Length-Limited Huffman Codes. *Journal of the ACM (JACM)*, 37(3):464–473, 1990.

[28] C. Lin, Y. Chung, and J. Liu. Efficient Data Compression Methods for Multidimensional Sparse Array Operations Based on the EKMR Scheme. *IEEE Transactions on Computers*, 52(12):1640–1646, 2003.

[29] W. Lu and M. P. Gough. A Fast-Adaptive Huffman Coding Algorithm. *IEEE Transactions on Communications*, 41(4):535–538, 1993.

[30] D. Marpe, H. Schwarz, and Wiegand T. Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):620–636, 2003.

[31] B. Martins and S. Forchhammer. Bi-level Image Compression with Tree Coding. In *Proceedings of IEEE Data Compression Conference*, pages 270–279, March 1996.

[32] Kenneth McConnell, Dennis Bodson, and Richard Schaphorst. *Digital Facsimile Technology and Applications.* Artech House, Norwood, MA, third edition, 1992.

[33] B.J. McKenzie and T. Bell. Compression of Sparse Matrices by Blocked Rice Coding. *IEEE Transactions on Information Theory*, 47(3):1223–1230, 2001.

[34] N. Memon and X. Wu. Recent Developments in Context-based Predictive Techiques for Lossless Image Compression. *The Computer Journal*, 40(2/3):127–136, 1997.

[35] H. Meyr, H. Rosdolsky, and T. Huang. Optimum Run Length Codes. *IEEE Transactions on Communications*, 22(6):826–835, 1974.

[36] A. Moffat. Two-level Context Based Compression of Binary Images. In *Proceedings of IEEE Data Compression Conference*, pages 382–391, April 1991.

[37] A. Moffat, R. M. Neal, and I. H. Witten. Arithmetic Coding Revisited. *ACM Transactions on Information Systems (TOIS)*, 16(3):256–294, 1998.

[38] A.A. Moinuddin, E. Khan, and F. Ghani. An Efficient Technique for Storage of Two-Tone Images. *IEEE Transactions on Consumer Electronics*, 43(4):1312–1319, 1997.

[39] ISO/IEC JTC1/SC29/WG1 N1359. *Coding of Still Pictures, Final Committee Draft.* JBIG, July 16, 1999.

[40] ISO/IEC JTCI/SC29/WG1 N1646. *JPEG2000 Final Committee Draft vl.0.* JPEG Committee, March 16, 2000.

[41] T. Nassrin. Entropy and Image Compression. *Journal of Visual Communication and Image Representation*, 4(3):271–278, 1993.

[42] K. Nguyen-Phi and H. Weinrichter. Bi-level Image Compression Using Adaptive Tree Model. In *Proceedings of IEEE Data Compression Conference*, page 458, March 1997.

[43] W.B. Pennebaker, J. L. Mitchell, Langdon, Jr. G. G., and R. Arps. An Overview of the Basic Principles of the Q-coder Adaptive Binary Arithmetic Coder. *IBM Journal of Research and Development*, 32(6):6717–726, 1988.

[44] M.D. Reavy and C.G. Boncelet. An Algorithm for Compression of Bilevel Images. *IEEE Transactions on Image Processing*, 10(5):669–676, 2001.

[45] R. F. Rice. Some Practical Universal Noiseless Coding Techniques. Technical report, Jet Propulsion Laboratory, Canlifornia Institute of Technology, Pasadena, CA, 1979. Technical Report 79-22.

[46] Iain E. G. Richardson. *H.264 and MPEG-4 Video Compression.* John Wiley & Sons Ltd,, West Sussex, England, 2003.

[47] G.R. Robertson, M.F. Aburdene, and R.J. Kozick. Differential Block Coding of Bilevel Images. *IEEE Transactions on Image Processing*, 5(9):1368–1370, 1996.

[48] H. Sakanashi, M. Iwata, and T. Higuchi. Evolvable Hardware for Lossless Compression of Very High Resolution Bi-level Images. *IEE Proceedings-Computers and Digital Techniques*, 151(4):277–286, 2004.

[49] David Salomon. *Data Compression the Complete Reference.* Springer-Verlag, New York, NY, third edition, 2004.

[50] Khalid Sayood. *Introduction to Data Compression.* Morgan Kaufmann Publishers, San Francisco, CA, second edition, 2000.

[51] P. Scheuermann, A. Yaagoub, and M. A. Ouksel. Compression of Binary Images on a Hypercube Machine. *Journal of Parallel and distributed computing*, 23:49–59, 1994.

[52] C. E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27:379–423,623–656, 1948.

[53] H. Shi. Two Image-Ttemplate Operations for Binary Image Processing. *Journal of Mathematical Imaging and Vision*, 3:269–274, 1997.

[54] R. M. Stites. *Two New Methods for Progressive Lossless Binary Image Compression.* PhD thesis, University of Minnesota, 2004. UMI Number: 3124759.

[55] H. Tanaka and A. Leon-Garcia. Efficient Run-Length Encodings. *IEEE Transactions on Information Theory*, 28(6):880–890, 1982.

[56] C. Tsai and J. Wu. On Constructing the Huffman-Code-Based Reversible Variable-Length Codes. *IEEE Transactions on Communications*, 49(9):1506–1509, 2001.

[57] Y. Wang and J. Wu. Vector Run-Length Coding of Bi-Level Image. In *Proceedings of IEEE Data Compression Conference*, pages 289–298, March 1992.

[58] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes.* Morgan Kaufmann Publishers, San Francisco, CA, second edition, 1999.

[59] H. Yuen and L. Hanzo. Block-Run Run-Length Coding of Handwriting and Bilevel Graphics Based on Quadtree Segmentation. *Pattern Recognition Letters*, 18(2):187–191, 1997.