MULTUM IN PARVO: TOWARD A GENERIC COMPRESSION METHOD FOR BINARY IMAGES

by

Arber Borici

B.S., State University of New York, Empire State College, 2006

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
MATHEMATICAL, COMPUTER, AND PHYSICAL SCIENCES
(COMPUTER SCIENCE)

THE UNIVERSITY OF NORTHERN BRITISH COLUMBIA

November 2010

Your file Votre référence
ISBN: 978-0-494-75135-0
Our file Notre référence
ISBN: 978-0-494-75135-0

Abstract

Data compression is an active field of research as the requirements to efficiently store and retrieve data at minimum time and cost persist to date. Lossless or lossy compression of bi-level data, such as binary images, has an equally crucial factor of importance. In this work, we explore a generic, application-independent method for lossless binary image compression.

The first component of the proposed algorithm is a predetermined fixed-size codebook comprising 8×8-bit blocks of binary images along with the corresponding codes of shorter lengths. The two variations of the codebook—Huffman codes and Arithmetic codes—have yielded considerable compression ratios for various binary images. In order to attain higher compression, we introduce a second component—the row-column reduction coding—which removes additional redundancy.

The proposed method is tested on two major areas involving bi-level data. The first area of application consists of binary images. Empirical results suggest that our algorithm outperforms the standard JBIG2 by at least 5% on average. The second area involves images consisting of a predetermined number of discrete colors, such as digital maps and graphs. By separating such images into binary layers, we employed our algorithm and attained efficient compression down to 0.035 bits per pixel.

Table of Contents

List of Tables

List of Figures

Abbreviations

AC	Arithmetic coding
AD	Anderson-Darling test
BCC	Break-codebook-coding signal
bpp	bits per pixel
CRV	Column reference vector
CR	Compression ratio
EOF	End-of-file signal
GIF	Graphics Interchange Format
ICB	Incompressible-block signal
JBIG	Joint Bi-Level Image Experts Group
JPEG	Joint Photographic Experts Group
KS	Kolmogorov-Smirnov test
MPEG	Moving Picture Experts Group
PNG	Portable Network Graphics
RB	Reduced block
RCRC	Row-Column Reduction Coding
RRV	Row reference vector
TIFF	Tagged Image File Format
VMR	Variance-to-mean ratio

Acknowledgements

I am obliged to my supervisor, Saif Zahir, who exposed a tolerant degree of patience toward the successful completion of my thesis work, despite brief moments of asymmetric coordinations between my pro-attitudes and objectives. I thank my committee members, Charles Brown, Michael Rutherford, and Iliya Bluskov for their insightful comments and questions that lead me to dig up certain important components for my work. I have learned from Charles Brown that analytical thinking attired with proper methodology leads to appreciable answers for many theoretical and practical problems. Michael Rutherford serenely examined my work and rendered attention-drawing questions from the perspective of a non-Computer Scientist. Iliya Bluskov provided necessary suggestions for rigorously improving the mathematical elegance of some theoretical results of my work.

Special thanks go to Jennifer Hyndman for her savvy advice on technological considerations related to the final appearance of this document. Samuel Watson was kind enough to point out several analytical elements regarding asymptotic analyses in my work, despite the incompleteness of my questions. Warm acknowledgments go to Desa and Jernej Polajnar for useful discussions and comments about my work in its final state.

Above all, I am sincerely grateful to my mother and brother for substantiating my decisions and for bearing with my whims and for having stood by me heart and soul Throughout my life, the deep-rooted intellect and will of my parents have encouraged me to preserve the eagerness for pursuing knowledge. My veneration towards such a wonderful family deserves more than two lines, but that is a story that belongs solely to me.

For Viktoria

There is a pleasure in the pathless woods,
There is a rapture on the lonely shore,
There is society, where none intrudes,
By the deep sea, and music in its roar:
I love not man the less, but Nature more,
From these our interviews, in which I steal
From all I may be, or have been before,
To mingle with the Universe, and feel
What I can ne'er express, yet cannot all conceal.

– Byron

Chapter 1

Introduction

I do not pretend to start with precise questions I do not think you can start with anything precise You have to achieve such precision as you can, as you go along

– BERTRAND RUSSELL

Data compression is generally defined as the task of transforming or representing data with a smaller amount of units of information than the original size Compression algorithms are used to transform an initial amount of information to a reduced amount, thus representing information in a compact form Instances of data include, but are not limited to, text, black and white (also referred to as binary, two-color, or bi-level) images, color pictures, scanned documents, sound, videos, and other digital signals [1]

Data compression is ubiquitous despite the paradoxical fact that storage and transmission costs keep decreasing as technological advances increase Web page images, video streams, digital TVs, cellular communications and many other technologies exploit compression, these technologies would otherwise lose clarity or practicality in performing their services [2] Famous examples of standard compression methods are JBIG2 for binary and half-tone images, JPEG, PNG, and GIF for images in general, and MPEG for videos In this work, we illustrate the theoretical development and empirical results of a novel compression method for binary images Binary image compression, too, is an active area of research, as shall be seen in Chapter 4.

The purpose of Section 1.1 is to expose the motivation that incited us to develop a new yet conceptually distinct approach to compressing binary images and, to some extent, other bi-level data, such as circuit test vectors. In Section 1.2, a succinct list of the major contributions is posited. Chapters 2 and 3 attempt to fill in the gaps in that list. Finally, an overview of the thesis is provided in Section 1.4.

1.1 Motivation

Consider devising a generic compression method for some predetermined set of data such as binary images. On one side, having a unified theoretical approach could be advantageous in focusing research in one main stream to ameliorate the generic method. On the other side, empirical results should ascertain an appreciable degree of compression efficiency in order for the method to survive.

Imagine a dynamical system comprising an information source, which assembles uniformly sized chunks of binary images, and a channel that outputs them. Think of these chunks as being analogous to the letters of some huge alphabet and consider the images as being analogous to words or even sentences of some not necessarily meaningful language, in the sense that one would think of meaning and language. Or, imagine these chunks are mosaic tiles, which, when assembled in some way, will reproduce and give meaning to the image conveyed by the mosaic. In light of that metaphor, suppose that the procedure of generating those image chunks obeys a stationary stochastic process. For every time shift, the distribution of such chunks should remain the same. Consequently, the probabilities may be used to determine the compression terminus for each and every possible binary image that the fictitious source can yield. An appropriate theoretical compression method could then be devised

And this approach would lead to a much aspired universal yet simple method for compressing binary images

In truth, such a presupposed information source implies the involvement of an infinitude of binary images and one should be very well aware of this However, the Law of Large Numbers proves to be a strong mathematical aegis which enables one to examine a relatively large sample of binary image chunks and deduce, to some theoretical approximation, a quasi-universal compression method The idea behind the proposed method in this work lies in between these strains

For reasons which will be laid out in Chapter 2, we specified the aforementioned chunks as 8×8-bit blocks We specify theoretical as well as empirical reasons for choosing non-overlapping 8×8 blocks In order to construct a large sample of binary images, we considered collecting and partitioning binary images into 8×8 blocks to examine the overall system entropy The latter provides a useful guide in learning the theoretical upper-bound of compressing binary images using a yet-to-be-devised application-independent method

The existence of entropy coders such as Huffman and Arithmetic coding, adds to the idea of developing a universal dictionary (or codebook) comprising pairs of 8×8 blocks and then Huffman codes or cumulative probabilities for the case of Arithmetic codes In order to achieve such a dictionary, we constructed a system of binary images randomly collected from different sources and were as diverse as possible in their pixel representations Thereafter, we eliminated images that contained salt-and-pepper noise This preprocessing proved to be useful in constructing an unbiased codebook Finally, we studied the relative probabilities of all blocks in the sample and, thus, we calculated the entropy of binary images based on the relatively large sample We used the probability distribution of 8×8 blocks to construct extended Huffman codes for all blocks with absolute frequency greater than 1, as shall be seen in a later chapter

All in all, we are aware that a stochastic discrete dynamic system may assemble an infinitude of binary images for a precise entropy value to be determined A hypothetical machine (or source) such as the one described here may not produce improbable sequences of blocks, any more than an equivalent source may not produce sequences of incomprehensible, say, English words As we shall state subsequently, such sequences are merely unlikely We focus on the theoretical and empirical average measures pertaining to blocks of binary images In light of that, the sample we constructed is representative of the most frequently occurring binary image blocks Furthermore, based on our literature review, this is the first attempt to model a generic codebook for compressing binary images using Huffman or Arithmetic codes

1.2 Contributions

As stated in Section 1 1, a universal codebook could be constructed, in principle, by considering chunks of all possible binary images assembled by some dynamic system In light of that the following list provides the cardinal contributions of this research

- **The compression apparatus.** The proposed method comprises a codebook and the row-column reduction coding, an algorithm which removes additional redundancy in an 8×8 block This method may be viewed as an application-independent compression apparatus, since the codebook component attempts to endorse a generic coding scheme

- **Efficient compression of binary images.** The proposed method achieves, on average, higher compression than the standard JBIG2 on binary images which do or do not favor the latter In addition, the method can efficiently compress textual images, such as scanned documents, books, and so forth. Naturally, in order to attain even higher compression, the codebook must be extended to include empirical distributions and Huffman codes of 8×8 blocks that have not appeared in our data sample.

- **Efficient compression of discrete-color images.** The method has been observed to efficiently compress discrete-color images through color separation. The latter procedure slices a color image into binary layers and compresses each layer individually. Examples of discrete-color images include, but are not limited to, maps, graphs, charts, and the like.

The first item in the list will be explored in detail in Chapter 2, whereas the remaining contributions will be clarified in Chapter 3.

1.3 Mechanism of the Proposed Approach

The general mechanism of the proposed method may be succinctly described as follows The lossless compression algorithm consists of two components· (i) a predetermined codebook; (ii) an additional coding algorithm--the row-column reduction coding (RCRC)—designed to further compress data. Details of these two components are exposed in Chapter 2. As per the proposed scheme, a binary image is partitioned into non-overlapping 8×8 blocks and each block is compressed individually.

In order to construct the codebook, we randomly collected 120 binary data samples, such as binary images, textual and document images, and so forth. The samples are of different dimensions and were gathered from various applications. The dimensions vary between 149×96 and 900×611 bits, whereas their representations vary in complexity and redundancy

The proposed method works as follows. For each 8×8 block, the codebook is searched to check if the block is found. If so, the block of size 64 bits is replaced by the corresponding code in the codebook. The latter code has a shorter length. The minimum and maximum lengths of such corresponding codes are 1 bit and 17 bits, respectively. If the block is not found in the codebook, we resort to an additional coding procedure, the row-column reduction coding (RCRC), to compress the block. If the size of the RCRC-compressed block is smaller than its original size (that is, smaller than 64 bits), we use the compressed bit string. Otherwise, we do not compress and represent the block with its original bits.

In general, blocks may be classified as compressible by the codebook, compressible by the row-column reduction coding, or incompressible if the first two attempts fail. Based on empirical results, the portion of incompressible blocks is, on average, less than 6% of the total number of distinct blocks in a given binary image.

1.4 Thesis Overview

The remainder of the thesis is organized as follows. In Chapter 2, we expose the proposed compression method in detail. We provide a basic theoretical background and lay some definitions pertaining to this work. Then, we exhibit the construction of a codebook per the motivation described above and the row-column reduction coding, an algorithm that removes additional redundancy in 8×8 blocks.

Empirical results are exposed in Chapter 3, categorized according to the related areas of applications. The proposed method achieves efficient compression in various classes of binary images. In addition, we observed that images comprising discrete colors, which can be separated into binary layers, are highly compressible via the proposed method. Finally, with a slight modification to the second component, the proposed algorithm attains good compression for integrated circuit test vectors.

Chapter 4 provides a summary of recent and mainstream compression techniques related to binary images and discrete-color images. Conclusions and Future Work follow in Chapter 5.

Chapter 2

The Proposed Method

You have your way. I have my way. As for the right way, the correct way, and the only way, it does not exist.
– FRIEDRICH NIETZSCHE

The purpose of this chapter is to expose the analytical details and components of the proposed compression method. The foundational armor of the method consists of an admixture of theoretical and empirical arguments. It is the objective of each subsequent section to provide the reader with these arguments as well as with the theoretical context pertaining to the proposed lossless compression algorithm.

2.1 Background

In this section, we provide a basic overview of compression methods and the modeling and coding paradigm. We also cover the definitions of entropy and joint entropy and an information-theoretic result that has been of central importance to the development of efficient entropy coders. We conclude with a general exposition of two famous entropy coders—Huffman Coding and Arithmetic Coding—that will be encountered throughout the remainder of the chapter.

2.1.1 Compression Methods

Compression methods generally operate in two phases. The first phase consists of the compression algorithm, which takes input (or source) data, denoted by $\Im_0$, and transforms them into $\Im_C$, which is expected to contain fewer bits of information. The second phase is the inverse operation of the first phase: the decompression algorithm—also referred to as reconstruction or decoding—takes the compressed data $\Im_C$ and reconstructs the original data $\Im_0$. In what follows, decompression, reconstruction and decoding refer to the same process and may be used interchangeably. Figure 2.1 illustrates a general exhibit of the two compression phases.

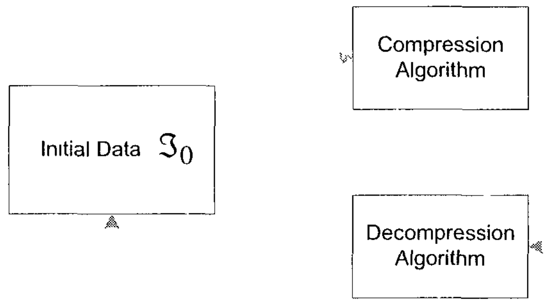


Figure 2.1: The two phases of compression methods

Compression methods are classified into two major categories: *lossless compression schemes*, in which case the compressed data must be recovered exactly, and *lossy compression schemes*, where compressed data is allowed to be different from the original data to some predetermined extent. This research focuses on lossless compression methods.

Lossless compression methods involve the exact reconstruction of the original data from the compressed data. This implies that the compression technique applied on the input data $\Im_0$ to generate the compressed data $\Im_C$ should be such that the decompression method applied on $\Im_C$ reconstructs $\Im_0$ with no loss of information. Figure 2.1 may be viewed as a schematic representation of lossless compression.

Lossless compression methods have a wide realm of applications, particularly when the integrity of data must be preserved. Instances of such applications include text compression, where the exact reconstruction of a particular text message is required [2]. For instance, a bank statement containing important information such as "Credit card balance due April **15**, 2010" and "Credit card balance due April **5**, 2010" convey perceptually almost the same data, but completely diverging information if not reconstructed exactly. Also, binary, grayscale, or color images, such as medical MRI or similar graphics must be reconstructed exactly, otherwise the nearly, yet not completely, reconstructed information may lead to a completely different, and plausibly erroneous, interpretation of the data. Other examples include scientific databases and images arising in remote sensing applications [3]. Last, but not least, lossless compression is crucial for cryptographic data, in which case data are compressed for added security and must be precisely reconstructed in order to preserve cryptographic keys.

2.1.2 The Modeling and Coding Paradigm

Archetypal to lossless compression methods is the Modeling-Coding paradigm [2, 3]. Based on this paradigm, the set of central components of any compression method comprises a mathematical model and a coder. The model is generally a stochastic model describing the distribution of the source data symbols, $\mathfrak{I}_0$, that are to be compressed. For example, if the coder is intended to compress English text, then the stochastic model could be a second-order Markov chain describing the distribution of English trigrams. Given the distribution description for each symbol, the coder attempts to represent the symbol into codewords of shorter length. The coder output will be a concatenated string of codes, $\mathfrak{I}_C$, along with additional information for updating the stochastic model, which may require prior knowledge of symbols. As this paradigm implies, compression may be referred to as coding or encoding, while decompression is synonymous to decoding. A codeword is simply defined as a sequence of binary digits. Huffman and Arithmetic Coding are two famous coding schemes, and will be discussed subsequently.

2.1.3 Entropy: The Coding Terminus

Data compression may be considered as a branch of Information Theory, the purpose of which is the study of efficient coding or quantification of information. From the information theoretic standpoint, the data being compressed are referred to as the *message*. A central question that is addressed to compression methods is how efficient they are. In his seminal paper,[1] Shannon introduced the concept of entropy in Information Theory as an attempt to answer that question.

Entropy, analogous to its counterpart in Thermodynamics, is a quantitative measure of the uncertainty contained in a particular message or, in general, a system [4]. The more random or disordered a system is, the more information is contained in that system and the higher its entropy becomes—that is, the predictability of the next object of the system given a previous object of that system depends on the system entropy. This implies that the predictability of the next object given the previous object increases by reducing the entropy of the system. Note that an object may be a letter of the alphabet if the system under observation is a natural language.

Entropy is also referred to as the Shannon entropy, to distinguish between the concept in Physics, or more accurately as the *first-order entropy*, and is defined as follows.

Definition 2.1. ***First-Order Entropy***

Consider a dynamical system (Ω, F, P, T), where Ω is the sample space, F is a $\sigma-$

[1]See [4].

algebra, P is the probability operator, and T is a time shift operator Let $X \quad \Omega \longrightarrow \Omega$ be a discrete random variable that has a finite alphabet $\mathcal{A} = \{\alpha_1, \alpha_2, \quad , \alpha_{\|\mathcal{A}\|}\}$, where $\|\mathcal{A}\|$ is the size of the alphabet, and let $\{X_n, n \in \mathbb{Z}^+\}$ be a stationary discrete stochastic process defined on the probability space (Ω, F, P) Then, the entropy of X is defined as

$$H(X) \equiv H(p_X) = -\sum_{\alpha \in \mathcal{A}} p_X \ln p_X = -E_p[\ln p_X], \qquad (2\ 1)$$

where $p_X = P(X = \alpha)$ and $E[\,]$ is the expectation operator Note that $0 \ln 0 = 0$ based on the continuity argument that $\lim_{x \to 0^+} x \log x = 0$ Also, $0 \leq H(X) \leq \ln(\|\mathcal{A}\|)$ is a concave function If the outcomes are equiprobable, the entropy is at maximum and equals $\ln(\|\mathcal{A}\|)$

Entropy is expressed in *bits per object*, where an object is any member of a pre-defined system See [5, 6] for a rigorous treatment of the subject

Consider the following examples

Example 2.1. Consider the message $\mathcal{M}$ = "aaaabbaaccccaaaa", containing three symbols (or objects) 'a', b', and 'c' Letter a occurs more frequently than letters b' and c' In other words, it seems normal to expect letter 'a' to appear more frequently should the message be shifted in time The probability distribution for the three letters is $p(a) = 10/16$, $p(b) = 2/16$, and $p(c) = 4/16$ Based on equation (2 1), the entropy of the given message is $H(\mathcal{M}) = 1\ 3$ bits per letter That is, we need on average 1 3 bits to encode each letter in message $\mathcal{M}$ Here, the message may be considered as a system whose objects are English letters

Example 2.2. The message $\mathcal{M}$ = "vbdkfawrptlhksaq' is apparently more *disordered* than the message in example 2 1 In other words, the entropy of this message is higher, given the highly random distribution of its letters Here, $H(\mathcal{M}) = 4\ 13$ bits per letter

Entropy provides a theoretical lower bound for coding and serves as a compression target. If the entropy of a particular message is H, the highest compression ratio that can be achieved for that message is $(S - H)/S$, where S is the size (in bits) of the message [7]. This implies that the smaller the entropy, the higher the compression ratio, and conversely. First-order entropy can be extended to define a vector $\mathbf{X}$ of random variables. The entropy in a dynamical system of two or more discrete random variables is referred to as the *joint entropy* and is defined as follows.

Definition 2.2. ***Joint Entropy***

Let $\mathbf{X}$ be a vector of k random variables $X_1, X_2, \ldots, X_k$. Then, the joint entropy is given by:

$$H(\mathbf{X}) = H(X_1, X_2, \ldots, X_k) = -E_p\left[\ln P(X_1, X_2, \ldots, X_k)\right], \tag{2.2}$$

where $P(X_1, X_2, \ldots, X_k)$ is the joint probability distribution of the k discrete random variables in $\mathbf{X}$. Note that joint entropy is non-negative and satisfies the subadditivity property: $H(X_1, X_2, \ldots, X_k) \leq H(X_1) + H(X_2) + \ldots + H(X_k)$ with equality only if the k random variables are independent in the sense of probability theory. Also, observe that $H(X, X) = H(X)$.

In order to lay out the rational foundation of the proposed method in this work, it is useful to define the entropy function for systems based on binary alphabets.

Definition 2.3. ***Binary Entropy***

Let $\mathcal{A} = \{0, 1\}$, $p(0) = P(X = 0) = p$, and $p(1) = P(X = 1) = 1 - p$. The binary entropy function is defined as:

$$H_b(X) = -p \log_2 p - (1 - p) \log_2 (1 - p). \tag{2.3}$$

Equation (2.3) is easily derived from the general model of first-order entropy given in (2.1). Definition 2.3 posits the following theorem.

Theorem 2.1. Let $\mathcal{A}^{(n)} = \{0,1\}^n$ be the extended alphabet of $\mathcal{A} = \{0,1\}$ That is, the members of $\mathcal{A}^{(n)}$ are all the binary n-tuples, where $\|\mathcal{A}^{(n)}\| = 2^n$ We refer to these groups of symbols as block symbols or, simply, *blocks* Then,

$$H\left[X^{(n)}\right] = nH_b(X) \tag{2 4}$$

The proof follows from equation (2 2) See [2] for the proof of the weak case for extended alphabets

Theorem 2 1 is important for the following two main reasons First, it expresses the entropy of random functions defined over an extended alphabet in terms of the entropy of the same functions defined over the basic alphabet Second, the entropy of longer groupings of symbols guarantees a rate closer to the system entropy—that is, higher compression can be attained by considering blocks of symbols rather than single symbols In general, encoding blocks of symbols defined over an extended alphabet guarantees an average codeword length upper bound closer to the entropy rate This observation is further clarified when the proposed method is posited

2.1.4 Huffman Coding

Huffman coding is a popular entropy encoding algorithm that can generate optimal prefix codes The basic principle behind this method is to optimally assign shorter codes to symbols that appear more frequently in a given message Therefore, source statistics are supposed to be available in advance For instance, in the string in Example 2 1, letter 'a' will be assigned the shortest Huffman code because it has the highest relative probability

If Huffman coding is used to encode binary messages, where symbols are either 0 or 1, then based on equation (2 3), whatever the probability distribution and entropy, the binary symbols will still require one bit to be encoded. Therefore, no compression can be achieved. However, according to Theorem 2.1, if binary symbols are grouped together to form blocks of symbols, then Huffman codes will guarantee compression. When Huffman codes are applied on extended alphabets, they are referred to as extended Huffman codes [2].

2.1.5 Arithmetic Coding

In cases when the probability distribution of symbols is skewed and when symbol probabilities cannot be redefined, Huffman coding may be inefficient to employ [1, 2, 8]. A competitive alternative is Arithmetic Coding, which is a core component of standard compression schemes, such as JBIG [9], JPEG, and MPEG. This method does not encode symbols with specific codes; rather, it encodes an entire sequence of symbols with a real number C, $0 \leq C < 1$. This mapping is accomplished through a simple bounding function.

Arithmetic coding has a higher complexity than Huffman coding, but achieves better results in practice for small alphabet sizes. However, when the alphabet size is very large and the probability distribution of symbols is not too skewed, the efficiency of the two methods is comparable. If used on a very large alphabet, Arithmetic coding may become inefficient in terms of complexity relative to Huffman coding [1]. In addition, Arithmetic coding is affected by inaccurate probabilities more often than Huffman coding [8]. All in all, Huffman codes are fast and efficient and are preferable for most applications.

2.2 Definitions

It is necessary to provide definitions of certain concepts that will prove useful in understanding details of the proposed method, and then construct the denotational aspect of this work.

Definition 2.4. ***Compression Ratio***

Image compression ratio, CR, is measured by an index defined as follows:

$$CR = \frac{hwB}{\|CF\|}\,, \tag{2.5}$$

where w and h represent the width and height of the image, B is the number of bits required to represent each pixel in the image, and $\|CF\|$ is the size, in bits, of the compressed data.

In the case of binary images, $B = 1$ bit/pixel, abbreviated as *bpp*. For images containing k discrete colors, there are $k - 1$ binary layers, where one of the colors represents the background color which is common to all layers. Let CF_i, $i = 1, \ldots, k-1$ denote the layer size in bits Then the compression ratio for discrete-color images is given by:

$$CR = \frac{\sum_{i=1}^{k-1} CF_i}{hw(k-1)}\,. \tag{2.6}$$

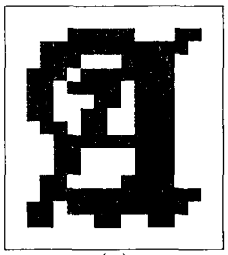
Compression ratio measures the average number of bits required to encode one pixel and may also be expressed as the percentage decrease in input file size. We use these measures interchangeably.

A binary image may be defined topologically such as in [10]. For simplicity, we define a binary image as follows.

Definition 2.5. ***Binary Image***

Also referred to as bi-level image, a binary image is a collection of picture elements *(pixels), each of which conveys either the color black or white. By convention, we use symbol '0' to denote a white pixel, and symbol '1' to denote a black pixel.*

Figure 2.2(a) shows an enlarged 16 $\times$ 16 binary image (letter 'A' in 12 pt Old English font face). The size of this image is 256 bits. Partitioning the image into 8 $\times$ 8 blocks will yield four such blocks as depicted in Figure 2.2(b). For computational simplicity, a binary image is represented as a binary matrix data structure.


(a)

0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1	1 0 0 0 1 1 0 0
0 0 1 1 1 1 1 1	1 1 1 1 1 0 0 0
0 0 1 1 1 0 0 0	1 1 1 1 0 0 0 0
0 1 1 1 0 1 1 1	1 1 1 1 0 0 0 0
0 1 1 0 1 1 1 1	0 1 1 1 0 0 0 0
0 1 1 0 0 0 1 1	0 1 1 1 0 0 0 0
0 1 1 0 0 1 1 0	0 1 1 1 0 0 0 0
0 1 1 0 0 1 1 0	0 1 1 1 0 0 0 0
0 0 0 1 1 1 1 1	1 1 1 1 0 0 0 0
0 0 0 1 1 0 0 0	0 1 1 1 0 0 0 0
0 0 0 1 1 0 0 0	0 1 1 1 0 0 0 0
0 0 1 1 0 0 0 0	1 1 1 1 0 0 0 0
0 0 1 1 1 1 1 1	1 1 1 1 1 1 0 0
0 1 1 0 1 1 1 1	1 1 1 1 1 0 0 0
0 1 1 0 0 0 1 1	0 0 1 1 0 0 0 0

(b)

Figure 2.2: (a) An enlarged 16 $\times$ 16 binary image; and (b) the corresponding bit matrix

The central component of the proposed method is a codebook comprising pairs of fixed-length (8 $\times$ 8) blocks and variable-length Huffman codes. Such a codebook is referred to as a *fixed-to-variable* dictionary. A *block* is any member of alphabet $\mathcal{A}^{(n)} = \{0, 1\}^n$, which is the extended alphabet of $\mathcal{A}^{(1)} = \{0, 1\}$ (see Theorem 2.1). Hence, an 8 $\times$ 8 block is a member of $\mathcal{A}^{(64)}$, with a cardinality of 2^{64} symbols.

An encoding scheme $\mathcal{C}$, defined as a mapping $\mathcal{C}\colon \mathcal{A}^{(64)} \rightarrow \mathcal{A}$, where $\mathcal{A} = \bigcup_{i=1}^{64} \mathcal{A}^{(i)}$, is a function that maps an 8 $\times$ 8 block (64 bits) to a bit string of shorter length. That is, an 8 $\times$ 8 block can be encoded as a sequence of 1 bit ($\mathcal{A}^{(1)}$), a sequence of 2 bits ($\mathcal{A}^{(2)}$), ..., up to a sequence of 64 bits, in which case the compression ratio would equal 0. A Huffman encoding scheme complies with such a definition, since in general no Huffman code can be longer than the alphabet size less one.[2]

There exist schemes that may encode some low-redundancy data into longer bit strings than the original data length. In such cases, the original data are preserved rather than compressed. The row-column reduction coding may encounter such cases as we shall see in Section 2.4. Such an encoding scheme may be defined as $\mathcal{C}_{RCRC} \colon \mathcal{A}^{(64)} \to \mathcal{A} \times \mathcal{A}$.

Define function $L \colon \mathcal{A} \to \mathbb{N}^{+}$, where $\mathcal{A} = \bigcup_{i=1}^{64} \mathcal{A}^{(i)}$. Function L gives the length (in bits) of the encoded data.

The inverse procedure of encoding is referred to as decoding. A lossless compression algorithm should be able to recover the original data exactly. The codebook component of the proposed method may be viewed as a partial injective mapping [12], wherein encoding and decoding are well-defined. The same applies to the second component, the row-column reduction coding.

An input image $\mathfrak{I}$ with dimensions $h \times w$ is defined as a multiset of 8×8 blocks, since a block may appear at least once. Suppose that $8|h \wedge 8|w$, then the cardinality of $\mathfrak{I}$ is $\|\mathfrak{I}\| = \frac{1}{64}wh$

Definition 2.6. *Let $B_{\mathfrak{I}}$ be the set of blocks in image $\mathfrak{I}$ and let $\mathcal{D}$ denote the codebook, defined as a set of pairs $\langle b, \mathcal{C}(b)\rangle$. Then, define the following sets:*

(i) $B_H = \{b | b \in B_{\mathfrak{I}} \wedge b \in \mathcal{D}\}$. Set B_H contains all blocks of image $\mathfrak{I}$ that are in the codebook, i.e. that can be compressed with Huffman codes.

(ii) $B_R = \{b | b \in B_{\mathfrak{I}} \wedge b \notin \mathcal{D} \wedge \mathcal{C}_{RCRC}(b) \neq \emptyset \wedge L(\mathcal{C}_{RCRC}(b)) < 64\}$. Set B_R contains all blocks that are not in the codebook, but that can be compressed by

[2]In [11], it is shown that the maximum length of Huffman codes is:

$$\min\left\{\left\lfloor \log_{\Phi}\left(\frac{\Phi+1}{p_1\Phi+p_2}\right)\right\rfloor, n-1\right\},$$

where n is the number of tree levels, $\Phi = \frac{1+\sqrt{5}}{2}$, and p_1 and p_2 are the two smallest probabilities.

the row-column reduction coding in less than 64 bits

(iii) $B_U = \{b | b \in B_{\mathfrak{I}} \wedge b \notin \mathcal{D} \wedge [\mathcal{C}_{RCRC}(b) = \emptyset \vee L(\mathcal{C}_{RCRC}(b)) \geq 64]\}$ Set B_U contains all incompressible blocks

It should be noted that sets B_H, B_R, and B_U are partitions of set $B_{\mathfrak{I}}$

We will recur to these definitions and notations whenever it is deemed necessary and appropriate throughout the detailed explanation of the proposed method

2.3 Toward a Universal Codebook

The proposed method operates on a fixed-to-variable codebook, wherein the fixed part consists of 8×8 blocks and the variable part comprises Huffman codes corresponding to the blocks In order to devise an efficient and practical codebook, we conducted a frequency analysis on a sample of more than a quarter million 8×8 blocks obtained by partitioning 120 randomly chosen binary data samples By studying the natural occurrence of 8×8 blocks in a relatively large binary data sample, the Law of Large Numbers motivates us to devise a general (empirical) probability distribution of such blocks In principle, this could be used to construct a universal codebook based on extended Huffman codes, which can be employed for compressing efficiently (on average) all sorts of bi-level data In this section, we provide details on the data sample we constructed to generate a codebook and how we constructed the codebook Thereafter, we expose how the codebook is employed to compress binary images

2.3.1 The Sample of Binary Images

In order to perform a frequency analysis on 8×8 blocks, a sample of more than 300 binary images of various dimensions and compositions was compiled. The images varied from complex topological shapes, such as fingerprints and natural sceneries, to bounded curves and filled regions. The candidates were extracted from different sources, mainly randomly browsed web pages and public domain databases. Because these representative images are widely available, it is reasonable to deduce that they are more likely to be considered for compression. Also, the main criterion in constructing an unbiased data sample was that images should convey the clear meaning they were constructed to convey without unintentional salt-and-pepper noise. Such a noisy image is illustrated in Figure 2.3(a) along with the corresponding "noiseless" counterpart in Figure 2.3(b). Perceptually, the images in Figure 2.3 may be regarded as conveying the same meaning. However, we assume that the observer's perception is strictly defined.[3]

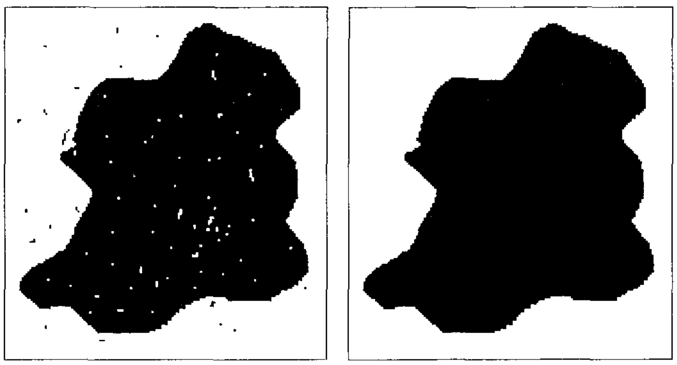

(a) Salt-and-pepper noise (b) No noise

Figure 2.3: An image containing salt-and-pepper noise and its noiseless counterpart

Having removed noisy binary images, the initial data sample reduced to 120 images with dimensions varying from 149×96 to 900×611 bits yielding approximately 250000 8×8 blocks. Before proceeding with the frequency analysis, we preprocessed binary

[3]Consider, for instance, a machine that cannot distinguish between the two images in Figure 2.3.

images in two steps The first step consisted of trimming the margins (or the image frames) in order to avoid biasing distribution of 0-valued or 1-valued 8×8 blocks In the second step, we modified image dimensions to make them divisible by 8 for attaining an integral number of 8×8 blocks

Trimming images in order to remove redundant background frame is important for the first preprocessing step Preserving such frames increases the relative probability of 8×8 blocks consisting of zeros or ones if the background is white or black, respectively Consequently, the probability of such blocks creates a skewed distribution of blocks in the codebook It has been reported that Huffman codes do not perform well with such a distribution of symbols [2, 8] [4]

Consider the binary image shown in Figure 2 4(a) Prior to trimming the margins, which comprise 8×8 blocks filled with zeros, it is necessary to determine the four extreme points depicted with the lines tangent to the closed curve Otherwise, one might clip portions of the image that contribute to the overall meaning the image conveys In addition, it is important that the distance between the tangent point and the actual trimming point is divisible by 8, as depicted in Figure 2 4(b) The reason for this is to avoid biasing the content of an 8×8 block, which would otherwise add to the overall redundancy of the image The latter would positively, but unfairly, influence the compression ratio of the proposed method For instance, using this trimming procedure, the first row of 8×8 blocks will be filled with 0-valued blocks The second such row will comprise blocks that start to represent portions from the fully-trimmed image, as depicted in Figure 2 4(c)

The second preprocessing step consisted of making the image height and width divisible by 8 Let w and h denote the width and height of an image We convert w and h to w^* and h^* such that $8|w^* \wedge 8|h^*$ as follows

[4]It should be stated, however, that the distribution of blocks in the constructed codebook is dominated by 0-valued 8×8 blocks followed by 1 valued 8×8 blocks

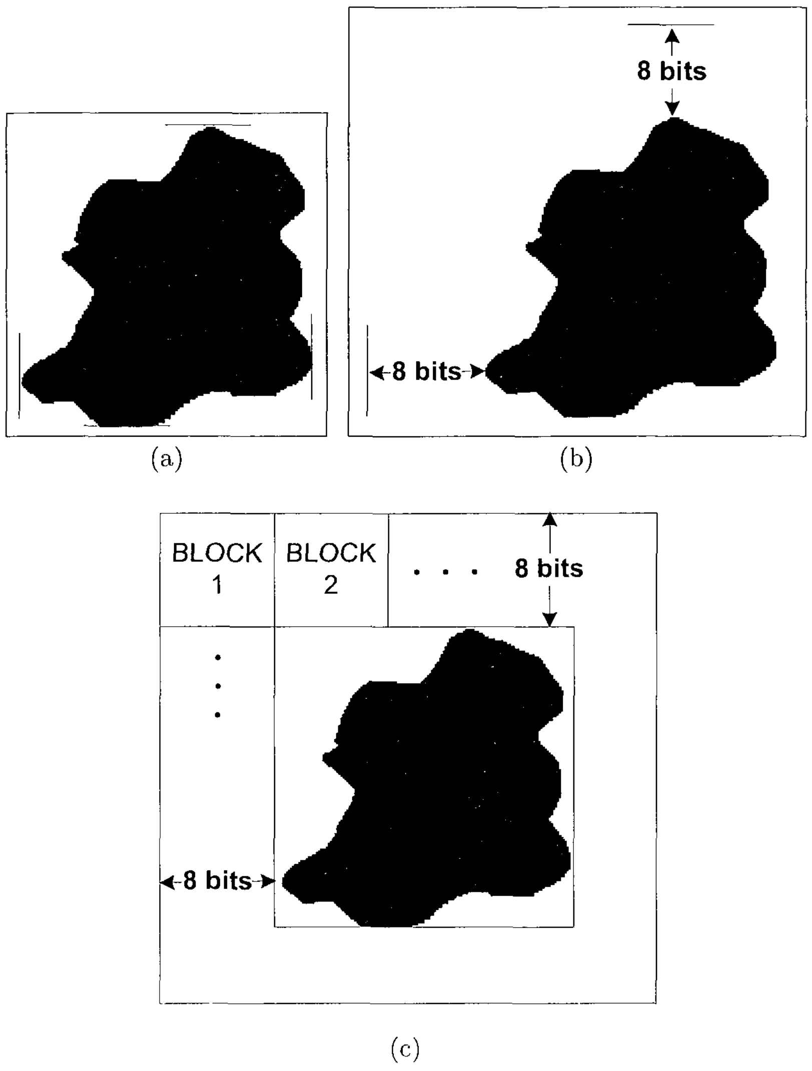


Figure 2.4: Determining the extreme points prior to trimming the binary image

- If $h \mod 8 \neq 0$, then $h^* = h + 8 - (h \mod 8)$;
- If $w \mod 8 \neq 0$, then $w^* = w + 8 - (w \mod 8)$.

Thus, the new image dimensions are $h^* \times w^*$. For instance, a 100×100 image will be padded to become a 104×104 image using the two steps above. The newly padded vector entries are filled with the image background bit. For instance, if the background color in the binary image is white (represented conventionally with 0), the padded entries will be filled with 0 bits.

Having gone through the two preprocessing steps, we conducted a frequency analysis on the 250000 8×8 blocks and we used these relative probabilities to construct the codebook This is the topic of the next subsection

2.3.2 Constructing the Codebook

From an information theoretic standpoint, we consider the images in the data sample described in Section 2 3 1 to have been generated by the hypothetical source described in Section 1 1 As such, the set of 8×8 blocks may be characterized as a discrete stochastic process[5] defined over a very large discrete alphabet of size equal to 2^{64} symbols that represent all possible patterns of zeros and ones in an 8×8 block

Essentially, one can study the distribution of 8×8 blocks for a relatively large data sample, such as the sample described in the preceding subsection It is, however, not possible to estimate empirical probabilities for all 2^{64} 8×8 blocks, and it is certainly not feasible or time efficient to construct a codebook containing all possible blocks and their Huffman codes Therefore one should consider devising a codebook comprising the most frequently occurring blocks In general, the more one increases block dimensions the smaller the waiting probability of observing all possible blocks becomes because the size of the alphabet increases exponentially Thus, it would be reasonable to have an expected value of the number of trial samples required to observe all the possible 8×8 blocks

The latter problem of determining the waiting probability of observing a particular number of blocks and the expected number of samples needed may be viewed as an instance of the more general Coupon Collector's Problem which is elegantly posed in [14] This problem is illustrated in Appendix A 1 In our case, we consider

[5]Information Theory was developed by relying on the assumptions of ergodicity and stationarity [13] Thus, such a random process should be characterized as an ergodic and stationary discrete stochastic process

"coupons" to be the 8×8 blocks for a total of 2^{64} symbols. Hence, we are interested in determining the number of blocks we must collect from the dynamical system in order to have observed all possible blocks. Thereafter, we may deduce an estimate of the expected number of trials required.

Answering these two questions per the Coupon Collector's Problem gives analytical insight on the size of the data sample required to estimate probabilities for observing all 8×8 blocks in the sample. The probability of waiting exactly n trials in order to observe all 2^{64} 8×8 blocks is equal to $P(T = n) = P(T > n-1) - P(T > n)$, where

$$P(T > n) = \sum_{i=1}^{2^{64}-1} (-1)^{i+1} \binom{2^{64}}{i} \left(\frac{2^{64} - i}{2^{64}} \right)^{n} . \tag{2.7}$$

The probability in formula (2.7) is difficult to compute for all possible 8×8 blocks. However, we may resort to an asymptotic approximation of the expected number of trials required to observe all blocks using the following formula:

$$E[T] \approx 2^{64} \ln 2^{64} + \gamma 2^{64} = 8.29 \times 10^{20} , \tag{2.8}$$

where $\gamma \approx 0.5772$ is the Euler-Mascheroni constant. The result in (2.8) implies that we need to compile a practically huge number of samples in order to attain a complete set of 8×8 blocks and to estimate, in turn, all relative probabilities.

Based on the latter remark, the only way to reduce the number of samples needed would obviously be to reduce the block dimensions from 8×8 to, say, 4×4, so that $||\mathcal{A}|| = 2^{16}$. We did experiment with blocks of smaller dimensions in order to decrease the alphabet size. However, the efficiency of extended Huffman codes for smaller block dimensions decreased. This is an expected result in Information Theory, as succinctly stated in Theorem 2.1. For instance, for 2×2 blocks, $||\mathcal{A}|| = 16$, and the expected number of samples needed to observe all possible blocks is approximately equal to 55.

Also, the waiting probability given in formula (2 7) tends to an arbitrarily small value for larger values of the number of samples needed For example, $P(T = 100) = 0\ 0017$, which implies that all 2×2 blocks will certainly be observed For 4×4 blocks, on the other hand, the expected number of samples needed would be approximately equal to 764647 Yet, even this number imposes difficult computations in determining both relative probabilities and extended Huffman codes of 4×4 blocks

The fact that decreasing block dimensions decreases the maximum compression ratio can be explained by the following observation In general, suppose the entropy for $n \times n$ blocks is H Then, the compression ratio upper bound (in bpp) is $\frac{H}{n^2}$, i e the number of bits required to encode an $n \times n$ block is inversely proportional to the square of block dimensions Thus, compression ratio per block increases as longer block dimensions are considered and decreases otherwise For example, the entropy of the system comprising all 65536 4×4 blocks was observed to be equal to 2 12 bits per block, yielding a satisfactory compression bound of 86 74% However, this would be the case if Huffman codes could be derived for *all* such blocks Constructing Huffman codes for such a large cardinality is practically inefficient Therefore, one needs to consider an empirical balancing between block dimensions, entropy, and extended Huffman codes Based on such considerations, we decided to study the empirical distribution of 8×8 blocks

Table 2 1 shows various candidate block dimensions along with the resulting entropy values[6] and the expected sample sizes An increase in entropy is expected as block dimensions increase because the alphabet size becomes larger, thus increasing the number of possible states The theoretical maximum compression ratio in percentage, however, increases proportionally to the square of block dimensions, as

[6]It should be noted that the entropy values given in the table—and, hence the maximum compression ratios, CR_{max}—are extrapolated from the analysis we carried out on 4×4 and 8×8 blocks These approximative values should suffice to give a general idea of how entropy and compression ratio vary with block dimensions

stated above. At this point, one may conclude that using 8×8 blocks consists a better choice than considering 2×2 or 4×4 blocks, or blocks with smaller dimensions than 8×8. After all, even for 4×4 blocks the number of samples required to observe all such blocks and to deduce a more reliable empirical probability distribution is practically unattainable and offers no promising compression ratio.

On the other side, the expected samples needed to observe all blocks increases exponentially, as can be noticed from the last column of Table 2.1. Despite the increase in entropy, the alphabet size imposes an empirical limit in selecting blocks larger than 8×8. Also, the probability that blocks not in the codebook will be compressed by the row-column reduction coding decreases with an increase in vector size, as shall be observed in Section 2.4. In all, our choice of 8×8 blocks is based on these strains of remarks and would probably be no different than selecting 7×7 or 9×9 blocks, except for some decrease or increase in the theoretical compression ratio and the feasibility in handling Huffman codes.

Table 2.1: Effect of block dimensions on entropy and the expected sample size

Block	$\|\mathcal{A}\|$	**Entropy**	CR_{max}	$E[T]$
2×2	2^4	1.36	66.00%	55
4×4	2^{16}	2.12	86.74%	764647
8×8	2^{64}	4.09	93.60%	8.29×10^{20}
12×12	2^{144}	7.89	94.52%	2.24×10^{45}
16×16	2^{256}	9.72	96.20%	2.06×10^{79}

In the data sample of 120 images, we identified a total of 65534 distinct blocks. From this total, we selected the 6952 blocks that occurred 2 times or more and discarded all other blocks with an absolute frequency equal to 1. Thus the cardinality of the fixed-to-variable codebook equals 6952 entries. This is a small number compared to the total number of blocks equaling 2^{61}. Since the codebook contains such a small fraction of 8×8 blocks and since we do not know the theoretical probability distribution of blocks, it is reasonable to provide an estimate for the error between the theoretical and the empirical average code lengths (or entropies).

The observed average code length, L, is given by:

$$L = \sum_{i=1}^{N} q_i \log_2 \frac{1}{q_i} , \tag{2.9}$$

where q_i are the empirical probabilities of blocks and N is the number of blocks. Similarly, we define the theoretical average code length for the theoretical probabilities p_i:

$$H = \sum_{i=1}^{N} p_i \log_2 \frac{1}{p_i} . \tag{2.10}$$

Then, we examine the error model:

$$E = L - H = \sum_{i=1}^{N} \left(q_i \log_2 \frac{1}{q_i} - p_i \log_2 \frac{1}{p_i} \right) . \tag{2.11}$$

Let $\epsilon_i = q_i - p_i$ be the discrepancy between empirical and theoretical probabilities, $\forall i = 1, 2, \ldots, N$. Then, a second-order asymptotic expansion on ϵ_i yields the following error approximation:

$$E = -\sum_{i=1}^{N} \left[\frac{1}{\ln 2} \left(\epsilon_i + \frac{\epsilon_i^2}{2p_i} \right) + \epsilon_i \log_2 p_i \right] + o\left(\max_{i \in \{1, \ldots, N\}} \{\epsilon_i^2\} \right) \quad \text{as} \quad \epsilon_i \to 0 . \tag{2.12}$$

The derivation of formula (2.12) is given in Appendix A.2.1.

The asymptotic approximation in formula (2.12) implies that discrepancies between the theoretical and empirical average code lengths are negligible as $\epsilon_i \to 0$. However, in our case we included only 6952 blocks in the codebook. If we let $N = 6952$ in equation (2.12), we have to add an additional error term for all other possible blocks not included in the codebook. Practically, we consider $q_i = 0, \forall i > 6952$. The additional error term to be added to equation (2.11) would thus be equal to

$-\sum_{i=6953}^{2^{64}} p_i \log_2 p_i$ In our view, these theoretical probabilities are very small and have a minor effect on the code length error, as will be seen in the next paragraph This fact is, however, one of the motivations that incited us to develop the additional coding module—the row-column reduction coding—as will be illustrated in Section 2 4 See Appendix A 2 2 for a detailed discussion on the additional error term

As stated earlier, the constructed codebook is a set of pairs $\mathcal{D} = \{\langle b, \mathcal{C}(b)\rangle\}$, where b is an 8×8 block and $\mathcal{C}(b)$ is the Huffman code of b The Huffman code length $L(\mathcal{C}(b))$ varies from 1 bit to 17 bits, while the observed average code length is 4 094 bits, which is greater than the codebook entropy value of 4 084 bits per block The difference between the observed average code length and the entropy value (defined in formula (2 11)) is equal to 0 01 This difference is referred to as *redundancy* In percentage, the redundancy is found to be 0 252% of the entropy This means that, on average, we need 0 252% more bits than the minimum required in order to code the sequence of blocks in the codebook In compliance with the asymptotic expansion of the error given in (2 12), this value, too, exposes a minor excess in codeword lengths and accounts for a near-optimal encoding of blocks by means of the constructed codebook Table 2 2 summarizes some statistics for the codebook

Table 2.2: Some statistics for the constructed codebook

Entries	Min length	Max length	Mean length	Variance	Entropy	Error
6952	1 bit	17 bits	4 094	1 695	4 084	0 252%

In the context of the modeling and coding paradigm presented in Section 2 1 2, the constructed codebook acts as the static modeling part of the proposed compression method In static modeling, statistics are collected for most or all alphabet symbols in order to construct representative codes While static modeling reduces the complexity of the decoder [15, 16], it is not widely used in practice because sample data may not always be representative of all data [17] Albeit in this section, we tackled a way to construct efficient and representative codes for the most frequent 8×8 blocks of binary images The analysis on the constructed codebook suggests a small lower bound and a negligible asymptotic upper bound on the discrepancy between theoretical and empirical code lengths Moreover, having established a compression model based on a fixed-to-variable codebook, we have selected Huffman and Arithmetic coding to implement the coder The former has been presented in this section, whereas the latter will be exposed subsequently

As a final note to this section, to calculate the frequencies of all distinct 8×8 blocks observed in the data sample, the program we designed executed for approximately 500 hours on an Intel Dual Core machine at 1 6 GHz per processor and 2 4 GB of RAM

2.3.3 Distribution of Blocks and Huffman Codes

A normalized measure to study the dispersion of the blocks in the constructed codebook could be the variance-to-mean ratio (VMR) for the block counts Such a measure can provide insight on the theoretical distribution of 8×8 blocks If $VMR = 1$, the data can be modeled by a Poisson process If $VMR > 1$ the data are over-dispersed, in the sense that they are spatially concentrated, and if $VMR < 1$ the data are said to be under-dispersed In our case, the mean occurrence of blocks is $\bar{x} = 62232\ 88$, the variance is $s^2 = 1723497\ 76$, and $VMR = 27\ 69$ Because $VMR = 27\ 69 > 1$, the blocks are over-dispersed and do not follow a Poisson distribution This result also suggests a relatively high degree of randomness in the distribution of the 6952 8×8 blocks

Figure 2 5 illustrates the distribution of the 20 8×8 blocks with the largest probabilities. Observe that the blocks with the largest probabilities—namely $P(k = 1) = 50.4\%$ and $P(k = 2) = 26.3\%$—are, respectively, filled only with zeros and only with ones. In addition, Figure 2.6 shows the cumulative probability of the 6952 8×8 blocks in the codebook.

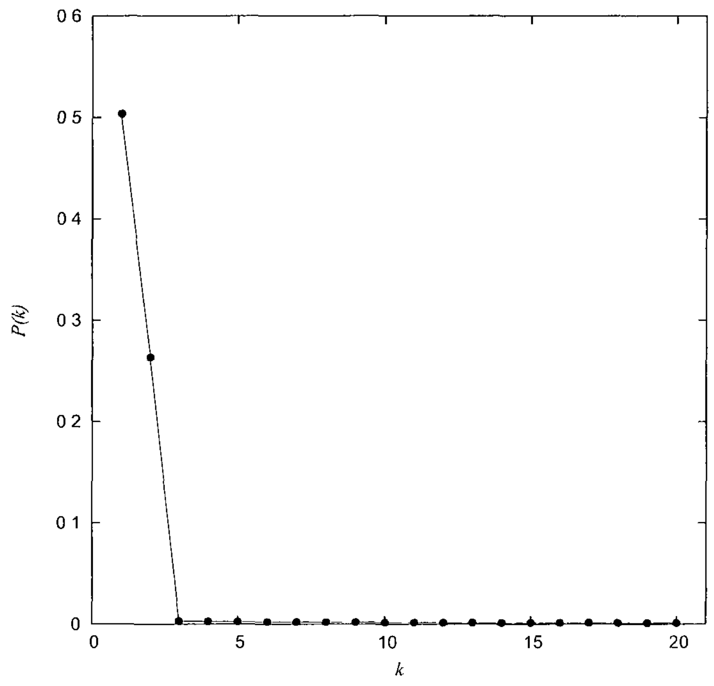


Figure 2.5: Distribution of the first 20 8×8 blocks

At this point, a goodness-of-fit test is useful in ascertaining whether the sampled 8×8 blocks follow any of the known discrete distributions. We test the following hypotheses at the 5% significance level using the Kolmogorov-Smirnov and Anderson-Darling tests:

$\mathcal{H}_0$: The 8×8 blocks follow distribution $\mathcal{P}$.

$\mathcal{H}_A$: The 8×8 blocks do not follow distribution $\mathcal{P}$,

where $\mathcal{P}$ denotes any of the following discrete distributions [14]:

(i) Log-series, with probability mass function (pmf) $P(n; \theta) = \frac{-\theta^n}{n \ln(1-\theta)}$;

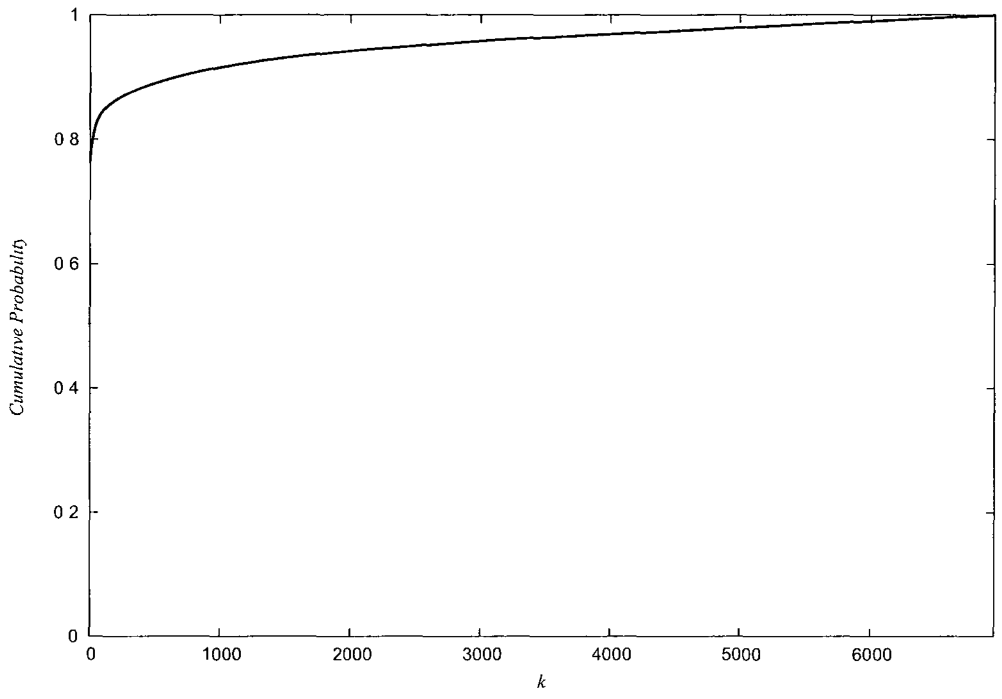


Figure 2.6: Cumulative probability of the 6952 blocks

(ii) Geometric, with pmf $P(n,p) = p(1-p)^n$, $0 < p < 1$,

(iii) Hypergeometric, with pmf $P(k,m,n,N) = \frac{\binom{m}{k}\binom{N-m}{n-k}}{\binom{N}{n}}$, where m denotes the total number of successes and $N-m$ denotes the total number of failures for n draws

(iv) Negative binomial, with pmf $P(k,r,p) = \binom{k+r-1}{r-1}(1-p)^r p^k$, where r denotes the number of failures until the process is stopped,

(v) Poisson with pmf $P(n,\lambda) = \frac{\lambda^n e^{-\lambda}}{n!}$

Test results are given in Table 2.3 The last two columns display the Kolmogorov-Smirnov (KS) and Anderson-Darling (AD) statistics, which are compared with the respective critical values equal to 0.019 and 2.5 at the significance level $\alpha = 0.05$ In all fitted cases, the null hypothesis is rejected in favor of the alternative hypothesis The log-series (discrete logarithmic) distribution is, however, ranked first based on the fact that it has the smallest test statistic.

Table 2.3: Hypotheses testing for the distribution of 8 × 8 blocks

Rank	Distribution $\mathcal{P}$	KS Statistic	AD Statistic	Parameter
1	Log-series	0.324	1089.5	$\theta = 0.995$
2	Geometric	0.619	4390	$p = 0.026$
3	Poisson	0.948	101910	$\lambda = 37.737$
4	Hypergeometric	No fit	No fit	—
5	Negative binomial	No fit	No fit	—

An index of dispersion measure can be given for the distribution of the constructed Huffman code lengths as well. Among other statistics, Table 2.2 shows the mean and the variance of the constructed Huffman code lengths. The index of dispersion for this case is $VMR = 0.41$, which implies that data are under-dispersed. In other words, the constructed Huffman code length values are more regular than the randomness associated with Poisson-distributed data.

2.3.4 Employing the Codebook

Let $\mathbf{V}$ be the binary matrix representing some input image $\mathfrak{I}$ that is to be compressed. First, we pad matrix $\mathbf{V}$ to make its dimensions divisible by 8, as shown in Section 2.3.1. Then, $\mathbf{V}$ is partitioned into 8×8 blocks, $b_{\mathbf{V}}$. For each block $b_{\mathbf{V}}$, the codebook $\mathcal{D}$ is searched for a match $b_{\mathcal{D}}$. If a match is detected, the input block $b_{\mathbf{V}}$ is encoded by the Huffman code of block $b_{\mathcal{D}}$. We denote this operation as $b_{\mathbf{V}} \leftarrow \mathcal{C}(b_{\mathcal{D}})$. This procedure iterates until all blocks in matrix $\mathbf{V}$ have been processed.

Decoding a compressed bit stream is simple. The Huffman code is searched in the codebook and the corresponding 8×8 block is then retrieved. For faster sequential search, the codebook entries are sorted in descending order based on the probabilities of the 8×8 blocks.[7] Figure 2.7 shows the first three entries of the codebook.

[7]Sequential search is expected to run practically fast because the most frequently occurring

8×8 **block**	**Huffman Code**
$\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \\ 0 & 0 & 0 \end{matrix}$	0
$\begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ \\ 1 & 1 & 1 \end{matrix}$	10
$\begin{matrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ \\ 0 & 0 & 0 \end{matrix}$	11101101

Figure 2.7: Sample codebook entries

It can be observed from Figure 2.7 that the 0-valued 8×8 block has the shortest Huffman code length, equal to 1 bit, followed by the 1-valued 8×8 block. In terms of the probability distribution, these two blocks alone comprise approximately 75% of the 6952 blocks in the codebook. This is an expected result as, in general, binary images have a white background and regions filled with black.

On the other hand, if no codebook match for block b_{V} is found, RCRC attempts to compress b_{V}. The RCRC algorithm is explained in the next section.

2.4 The Row-Column Reduction Coding

The codebook component of the proposed method is efficacious in compressing the 6952 blocks it contains. These blocks, as seen in the previous section, are the most frequently occurring symbols as per the empirical distribution. Compared to the alphabet size of 2^{64}, the cardinality of the codebook is very small. Hence, there will be blocks from input images that cannot be compressed via the codebook. For

blocks appear at the beginning of the codebook. In theory, however, the running time is constant since the block dimensions as well as the size of the codebook are fixed.

that purpose, we designed the row-column reduction coding (RCRC) to compress 8×8 blocks of a binary matrix, $\mathbf{V}$, that are not in the codebook, $\mathcal{D}$ In this section, we illustrate how the algorithm works

2.4.1 The RCRC Algorithm

RCRC is an iterative algorithm that removes redundancy between row vectors and column vectors of a block and functions as follows For each 8×8 block $\mathbf{b}$, $\mathbf{b} \in \mathbf{V}, \mathbf{b} \notin \mathcal{D}$, RCRC generates a row reference vector (RRV), denoted as $\mathbf{r}$ and a column reference vector (CRV), denoted as $\mathbf{c}$ Vectors $\mathbf{r}$ and $\mathbf{c}$ may be viewed as 8-tuples which can acquire values $\mathbf{r}_i = \{0, 1\}$, $\mathbf{c}_i = \{0, 1\}$, for $i = 1, 2, \quad , 8$ These vectors are iteratively constructed by comparing pairs of row or column vectors from the block $\mathbf{b}$ If rows or columns are identical in a given pair, then the first vector in the pair eliminates the second vector, thus reducing the block If the two vectors are not identical, then they are both preserved The eliminations or preservations of rows and columns are stored in RRV and CRV, respectively, which are constructed in a similar way The iterative construction procedure is exposed in what follows for the case of RRV, while noting that the same procedure applies to constructing the CRV

Let $\mathbf{b}_{ij}$ denote the i^{th} row of block $\mathbf{b}$, for $j = 1, 2, \quad , 8$ RCRC compares rows in pairs starting with the first two row vectors in the block, $\langle \mathbf{b}_{1j}, \mathbf{b}_{2j} \rangle$ If $\mathbf{b}_{1j} = \mathbf{b}_{2j}$, $\mathbf{r}_1 = 1$ $\mathbf{r}_2 = 0$, and row $\mathbf{b}_{2j}$ is eliminated from block $\mathbf{b}$ Next, $\mathbf{b}_{1j}$ is compared with $\mathbf{b}_{3j}$ and, if they are equal, a value of 0 is stored in $\mathbf{r}_3$ If, however, $\mathbf{b}_{1j} \neq \mathbf{b}_{3j}$, then a value of 1 is assigned to $\mathbf{r}_3$, implying that the third row has been preserved, and the RCRC will create the new pair $\langle \mathbf{b}_{3j}, \mathbf{b}_{4j} \rangle$ to compare as above This procedure iterates until RCRC compares rows in the pair that contains the last row of block $\mathbf{b}$ By convention, $\mathbf{r}_i = 1$ means that the i^{th} row of the block has been preserved while

$\mathbf{r}_i = 0$ marks an eliminated row. Clearly, $\mathbf{r}_1$ and $\mathbf{c}_1$ will always take on a value of 1.

The result of these RCRC operations will be a row-reduced block. Next, RCRC constructs the column reference vector based on the row-reduced block. There will be 7 pairs of column vectors to compare and at most 7 pairs of entries to compare depending on the number of eliminated rows. CRV is constructed in a similar way as RRV through the procedure illustrated above. The end result will be a row-column-reduced block (RB). The block is encoded as a sequence of bits, where the first 8 bits represent RRV, the second 8 bits represent CRV, and the remaining bits represent RB. The minimum size RB can assume is 1 bit. Thus, the maximum compression ratio attainable by RCRC is $(64 - 17)/64 = 73.44\%$. Figure 2.8 illustrates the RCRC algorithm for some input vector $\mathbf{v}$.

The RCRC decoding process is straightforward. The number of 1's in RRV and CRV indicates the number of rows and columns in the reduced block, respectively. If $\mathbf{r}_i = 1$ and $\mathbf{r}_{i+1} = 0$, then the row in block $\mathbf{b}$ having index $i + 1$ will be reproduced exactly by the row with index i. Also, if $\mathbf{r}_i = 1$ and the k consecutive entries are all equal to 0, then the decoding procedure will reproduce k copies of the i^{th} row of block $\mathbf{b}$ to construct that particular portion of the block. Having reconstructed rows, the decoding of columns proceeds in similar ways.

In Table 2.1 of Section 2.3.2, we illustrated how entropy and expected sample size vary with different block dimensions. Having exposed the details of RCRC, Table 2.4 illustrates how RCRC compression changes with varying block dimensions. The last column shows the probability that any two vectors match. Here, we consider pixels to take on values independently. This fact is, however, not realistic because for binary images a pixel is dependent on its neighboring pixels. For simplicity, let $P(\mathbf{v}_i = 0) \equiv p$ and $P(\mathbf{v}_i = 1) \equiv 1 - p$ denote, respectively, the probability that the i^{th} vector entry has a value of 0 and 1. Then, the probability that two vectors $\mathbf{v}$ and

```
Row-column reduction coding
  {Input: Vector $\mathbf{v}$, $\|\mathbf{v}\| = 8$}
  {Output: [RRV, CRV, RB]}
  $i = 1$
  while $i \leq 7$ do
    $\mathbf{r}_i = 1$
    $j = i + 1$
    while $\mathbf{b}_{ik} = \mathbf{b}_{jk}$ and $j \leq 8$, $k = 1, 2, \ldots, 8$ do
      $\mathbf{r}_j = 0$
      $\mathbf{b} = \mathbf{b} \setminus \mathbf{b}_{jk}$
      $j = j + 1$
    end while
    $i = j$
  end while
```

Figure 2.8: The RCRC algorithm

$\mathbf{u}$ of same size n match is $P(\mathbf{v} = \mathbf{u}) = (2p^2 - 2p + 1)^n$.[8]

It can be observed from Table 2.4 that, for 2×2 blocks, the maximum compression ratio RCRC can achieve is -25%. That is, RCRC fails to compress 2×2 blocks; instead, it adds 25% more bits to the compressed data stream. The compression increases as a function of block dimensions, whereas the probability that any two vectors match decreases exponentially.

[8]Assume the random events $\{\mathbf{v}_i = \mathbf{u}_i\}$ are independent. Then, for $i = 1, \ldots, n$, we have.

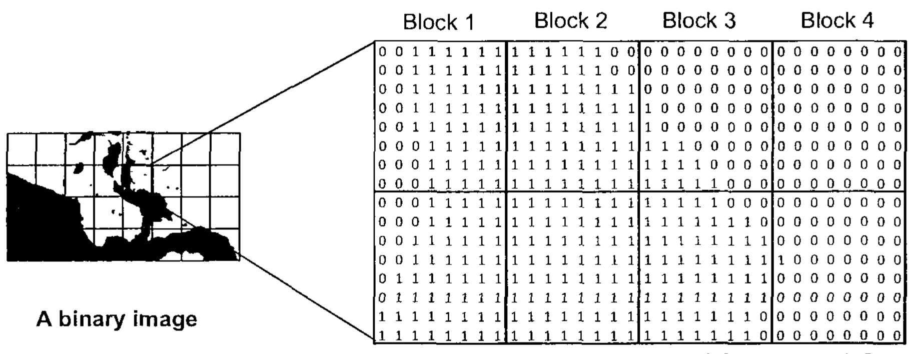
$$\begin{aligned} P(\mathbf{v} = \mathbf{u}) &= P\left\{\bigwedge_{i=1}^{n} [(\mathbf{v}_i = 0 \wedge \mathbf{u}_i = 0) \vee (\mathbf{v}_i = 1 \wedge \mathbf{u}_i = 1)]\right\} \\ &= \prod_{i=1}^{n} [P(\mathbf{v}_i = 0)P(\mathbf{u}_i = 0) + P(\mathbf{v}_i = 1)P(\mathbf{u}_i = 1)] \\ &= \left[p^2 + (1-p)^2\right]^n = (2p^2 - 2p + 1)^n . \end{aligned}$$

Table 2.4: Effect of block dimensions on RCRC

Block	RRV	CRV	RB$_{min}$	CR_{max}	$P(\mathbf{v} = \mathbf{u})$
2×2	2	2	1	-25.00%	$(2p^2 - 2p + 1)^2$
3×3	3	3	1	22.22%	$(2p^2 - 2p + 1)^3$
4×4	4	4	1	43.75%	$(2p^2 - 2p + 1)^4$
5×5	5	5	1	56.00%	$(2p^2 - 2p + 1)^5$
6×6	6	6	1	63.89%	$(2p^2 - 2p + 1)^6$
7×7	7	7	1	69.39%	$(2p^2 - 2p + 1)^7$
8×8	8	8	1	73.44%	$(2p^2 - 2p + 1)^8$
12×12	12	12	1	82.64%	$(2p^2 - 2p + 1)^{12}$
16×16	16	16	1	87.11%	$(2p^2 - 2p + 1)^{16}$

2.4.2 An Example

Figure 2.9 shows a binary image partitioned into regions along with the binary matrix representing a portion of the partition depicted by the extended lines. This binary matrix contains eight 8×8 blocks. We use some of these blocks to illustrate how the RCRC algorithm works.


A binary image
Matrix corresponding to the upper-left part of the depicted section in the image: eight 8x8 blocks

Block 1	Block 2	Block 3	Block 4
0 0 1 1 1 1 1 1	1 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 1 1 1 1 1 1	1 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 1 1 1 1 1 1	1 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 0 0 0	0 0 0 0 0 0 0 0

Block 5	Block 6	Block 7	Block 8
0 0 0 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 0 0 0	0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 0	0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 0	0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 0	0 0 0 0 0 0 0 0

Figure 2.9: Portion of a binary image and its corresponding 8×8 blocks

In Figure 2.10, the row reduction operation is applied on Block 2 of the binary matrix in Figure 2.9. The row reference vector (RRV) is shown on the left of the block. In this case, the first row is identical to the second row, which is removed from the block. Therefore, a value of 1 is placed in the first location of RRV (for the first row), and a value of 0 is stored in the second location of RRV for the second (eliminated) row. Next, row 1 is compared with row 3, but the two rows are not identical. Hence, a value of 1 is placed for row 3 and the pair comparison proceeds between row 3 and rows $4, 5, \ldots, 8$. Finally, a value of 0 is placed for the corresponding RRV locations of rows 4 to 8, which are eliminated since they are identical to row 3.

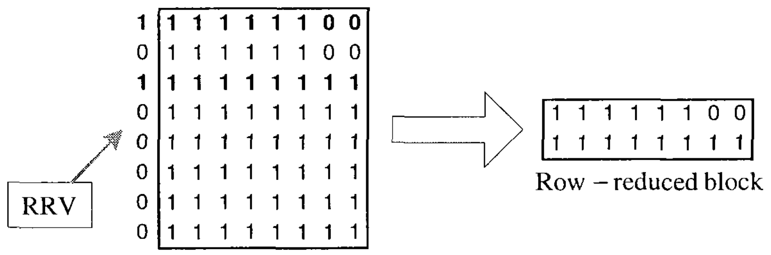


Figure 2.10: The row-reduction operation applied on a block

The column-reduction operation is applied on the row-reduced block, as depicted in Figure 2.11. The column-reference vector (CRV) is shown on top of the block. In this case, the first column is identical to and eliminates columns 2 to 6. Also, column 7 eliminates column 8. This yields the reduced block, RB, shown on the right of the column-reduced block. For this example, the output of RCRC is a concatenated string composed of the RRV (the first group of 8 bits), CRV (the second group of 8 bits), and RB (the last 4 bits), all displayed as one vector: $\underbrace{10100000}_{\text{RRV}}\underbrace{10000010}_{\text{CRV}}\underbrace{1011}_{\text{RB}}$, for a total of 20 bits. The compression ratio achieved for this block is $(64 - 20)/64 = 68.75\%$.

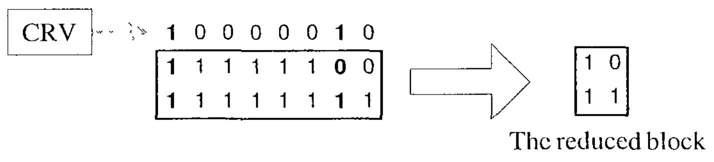


Figure 2.11: The column-reduction operation applied on the row-reduced block in Figure 2.10

To clarify the decoding process, we consider the row-column reduced block of the preceding example. The number of 1's in RRV and CRV shows the number of rows and columns of the reduced block, respectively. The output $\underbrace{10100000}_{\text{RRV}}\underbrace{10000010}_{\text{CRV}}\underbrace{1011}_{\text{RB}}$ contains two ones in the first group of 8 bits (the RRV), and two ones in the second group of 8 bits (the CRV). This means that there are 2 rows and 2 columns in the reduced block. That is, the first two bits of the reduced block, '10', represent the first reduced row, and the second two bits, '11', represent the second reduced row. Then, given the 1's and 0's in the reference vectors, we construct the rows and columns of the original block. Figure 2.12 shows the column reconstruction based on the column-reference vector.

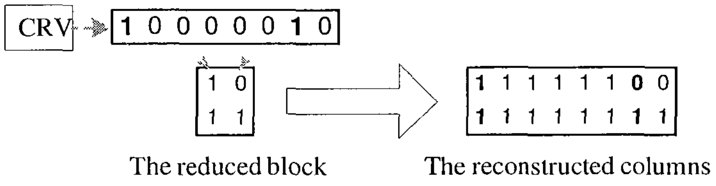


Figure 2.12: Column reconstruction based on the column-reference vector (CRV)

In Figure 2.12, CRV informs the decoder that columns 2 to 6 are exact copies of column 1, and column 8 is an exact copy of column 7. The block on the right depicts this operation. Figure 2.13 shows the row reconstruction process, which terminates RCRC decoding and we obtain the original block of Figure 2.10.

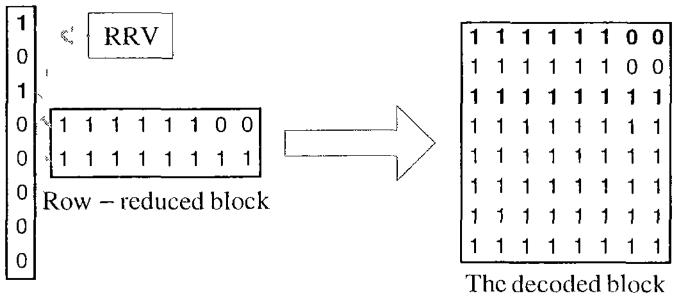


Figure 2.13: Row reconstruction based on the row-reference vector (RRV)

2.4.3 A Word on RCRC Compression Probability

One should likely ponder about the probability of an 8×8 block being compressed by RCRC In Section 2 4 1, we established that the probability of two vectors of length 8 being identical is given by $(2p^2 - 2p + 1)^8$, where p denotes the probability that the pixel has a value of 0 This expression holds for zero-order Markov chains That is to say, the probability of the current pixel being 0 or 1 does not depend on the values of neighboring pixels This assumption is strong, since pixel values in binary images *do* depend on neighboring pixel values For simplicity, however, this assumption should suffice to provide a general idea of the RCRC compression probability

The output of RCRC is a bit stream comprising RRV, CRV, and RB The sizes of RRV and CRV are fixed to 8 bits each The size of RB may vary from 1 bit to 64 bits A block is considered compressible by RCRC if the total length of the RCRC output is less than 64 bits Thus, let R denote the random event that an 8×8 block is compressible by RCRC The objective of this section is find an expression for the probability $P(R)$ Let R' denote the complement of event R Then, $P(R) = 1 - P(R')$ We focus on determining $P(R')$, as it is simpler to consider the cases when RCRC fails to compress a 8×8 block

The length of the RCRC output is $8 + 8 + L(RB)$ and it should be less than 64 bits Therefore, $L(RB) < 48$ The random event R' thus denotes R' $L(RB) \geq 48$ The size of the reduced block, RB, is greater than 48 bits in the following four cases

(1) Only one row and only one column has been eliminated That is, $L(RB) = 49$ bits

(2) Only one row and no column has been eliminated That is, $L(RB) = 56$ bits

(3) No row and only one column has been eliminated That is, $L(RB) = 56$ bits

(4) No row and no column has been eliminated That is, $L(RB) = 64$ bits

Each random event of each case may be viewed as a success/failure event. Therefore, a binomial distribution is suitable to study their probabilities. In the end, the sum of probabilities of these four cases will give the value $P(R')$. Let us now consider these cases. In what follows, we let $(2p^2 - 2p + 1)^8$ denote the probability that two vectors match and $q = (2p^2 - 2p + 1)$.

(1) Let C_1 denote the random event "Only one row and only one column has been eliminated", E_1 denote the random event "One row has been eliminated", and E_2 denote the random event "One column has been eliminated". Events E_1 and E_2 are independent, in the sense of Probability Theory. Thus, $P(C_1) = P(E_1)P(E_2)$. In total, there are only 7 pairs of consecutive rows (or columns) to compare and we require only one pair out of 7 to match. However, when a row is eliminated, there are only 7 entries per column pair to compare. Then,

$$P(E_1) = \binom{7}{1} q^8 (1 - q^8)^6 \tag{2.13}$$

and

$$P(E_2) = \binom{7}{1} q^7 (1 - q^7)^6 \tag{2.14}$$

Finally, from (2.13) and (2.14) we have:

$$P(C_1) = 49q^{15}(1 - q^8)^6(1 - q^7)^6 \ . \tag{2.15}$$

(2) Let C_2 denote the random event "Only one row and no column has been eliminated", E_1 denote the random event "One row has been eliminated", and E_2 denote the random event "No column has been eliminated". Events E_1 and E_2 are independent; thus, $P(C_2) = P(E_1)P(E_2)$. Similar to the previous case, there are only 7 pairs of consecutive rows to compare and we require only one pair out

of 7 to match, and no pairs of columns. Once again, a row is eliminated and there are only 7 entries per column pair to compare. Then,

$$P(E_1) = \binom{7}{1} q^8 (1-q^8)^6 \tag{2.16}$$

and

$$P(E_2) = \binom{7}{0} (q^7)^0 (1-q^7)^7 = (1-q^7)^7 \tag{2.17}$$

Finally, from (2.16) and (2.17) we have:

$$P(C_2) = \binom{7}{1} q^8 (1-q^8)^6 (1-q^7)^7 \,. \tag{2.18}$$

(3) This case is similar to Case (2). Let C_3 denote the random event "No row and only one column has been eliminated". Then,

$$P(C_3) = \binom{7}{1} q^8 (1-q^8)^6 (1-q^7)^7 \,. \tag{2.19}$$

(4) For this case, no row or column is eliminated. Let C_4 denote the random event "No row and no column has been eliminated". Then, the probability of this event is:

$$P(C_4) = \left[\binom{7}{0} (q^8)^0 (1-q^8)^7 \right]^2 = (1-q^8)^{14} \,. \tag{2.20}$$

The random events $C_1, C_2, C_3,$ and C_4 are mutually exclusive and, therefore, $P(R')$ is the sum of the probabilities of these events. From algebraic manipulations of the expressions above, $P(R')$ may be succinctly expressed as:

$$P(R') = (1-q^8)^6 \left[49q^{15}(1-q^7)^6 + 14q^8(1-q^7)^7 + (1-q^8)^8 \right] \,, \tag{2.21}$$

and the required probability $P(R)$ of an 8×8 block being compressed by RCRC is, therefore

$$P(R) = 1 - (1-q^8)^6 \left[49q^{15}(1-q^7)^6 + 14q^8(1-q^7)^7 + (1-q^8)^8\right] , \qquad (2.22)$$

where $q = (2p^2 - 2p + 1)$

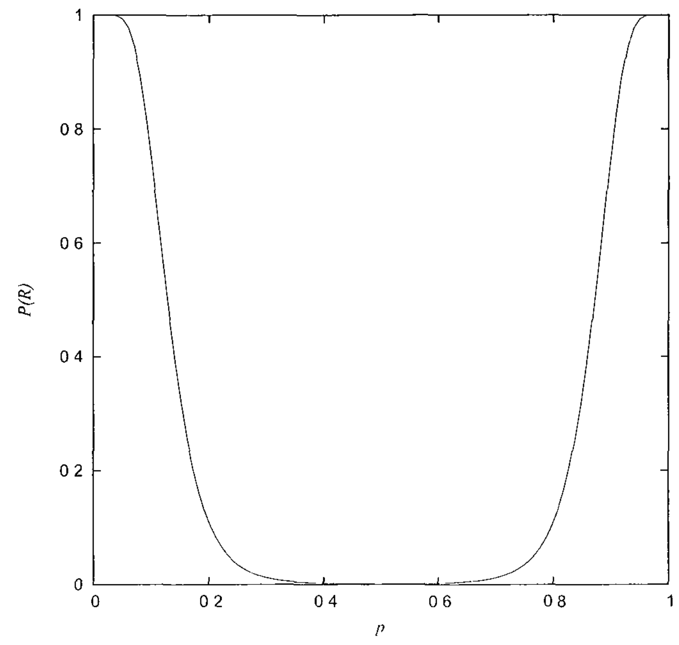


Figure 2.14: A plot of $P(R)$ as a function of p for 8×8 blocks

The plot in Figure 2.14 illustrates the probability in formula (2.22) as a function of p. Recall that p denotes the probability of a pixel assuming a value equal to 0. From the graph, we can observe that RCRC performs well if the probability value p is relatively small or relatively large, and does not do well if 0's and 1's are uniformly distributed. A first-derivative analysis shows that the minimum of the function is reached at $p = 0.5$ when pixel values are uniformly distributed. That is, if $p = 0.5$, $P(R)$ is practically small. A uniform distribution of pixel values among binary images is not realistically the case because of the high degree of inherent correlation and

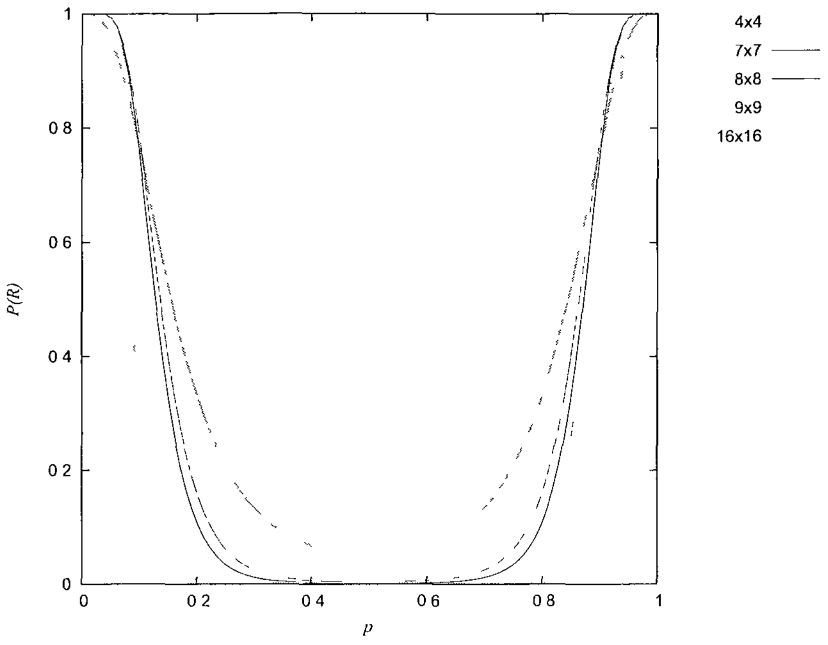


Figure 2.15: A plot of $P(R)$ as a function of p foı various block dimensions

redundancy among pixels [18]. For example, if one is sketching a human eye, then the black and white pixels cannot be uniformly distributed, since the eye has a particular topological shape and the prior sequence of pixel values will determine the cuırent pixel value.

Figuıe 2.15 plots $P(R)$ as a function of p foı various block dimensions. Obseıve that the probability $P(R)$ for 7×7 and 9×9 blocks is close to the probability for 8×8 blocks. As an example, if $p = 0.1$, then $P(R) = 0.28$ for 16×16 blocks, but $P(R) = 0.75$ foı 8×8 blocks.

2.5 Computational Complexity

Here, we give an analytical time complexity analysis for the proposed method. Let h and w be the dimensions of an input binaıy image matrix. Assume, without loss of generality, that the image dimensions are divisible by 8 For each 8×8 block, the algorithm searches the codebook for a matching block If a match is detected, the block is compressed and the next 8×8 block is processed The codebook has a fixed size of 6952 entries, therefore, it has $\Theta(1)$ running time If a match is not found in the codebook, RCRC attempts to compress the block In the context of the proposed method, the RCRC input is of fixed size and has $\Theta(1)$ running time, too The codebook search and RCRC are executed for at most $\frac{1}{64}wh$ 8×8 blocks Thus, the total complexity of the proposed method is $\Theta(hw)$

In Section 2 3 2, we noted that codebook entries are sorted in descending order based on their empirical probabilities While this fact does not contribute to the analytical time complexity, we noticed it had some positive impact on the empirical complexity metric of the proposed method

2.6 The Coding Scheme

The encoding process of the proposed method is simple and straightforward In order to distinguish between blocks compressed by the codebook, blocks compressed by RCRC, or uncompressed blocks, we consider three cases which are summarized in Table 2 5 Based on these cases, we construct a general model for the expected compression ratio attainable by the encoding scheme of the proposed method

Case 1 If a block is found in the codebook, we use the corresponding Huffman code This provides for two sub-cases

Case 1a If the corresponding Huffman code is the shortest in the codebook, i e 1 bit in the case of the constructed codebook, then assign bits 1 1 to encode that block The reason we use two overhead bits for the block that has the shortest code is based on the empirical fact that this entry comprises about 57%–70% of the total blocks found in the codebook, depending on the type of bi-level data.

Case 1b If the corresponding Huffman code has a length $L(\mathcal{C}(b)) > 1$ bit, then assign bits `0 0` to encode the block. As stated in Section 2.3.2, the length of Huffman codes in the codebook varies from 1 bit to 17 bits. Thus, after `0 0`, use 5 additional bits to encode the length of the codeword that follows. Lastly, add the Huffman code to the bit stream. For instance, if a block b that is found in the codebook has a code of length 7, then the block will be encoded as `0 0` + `00111` + $\mathcal{C}(b)$. In this case, the second group of 5 bits (`00111`) tells the decoder that $\mathcal{C}(b)$ has a length equal to 7 bits; thus, the decoder will read the subsequent 7 bits.

Case 2 If the block is compressed by RCRC, then use overhead bits `0 1`. Following these two bits is the bit stream RCRC produces. The decoding process for RCRC is explained in Section 2.4.1.

Case 3 If the block is neither found in the codebook, nor compressed by RCRC, we use the two overhead bits `1 0`, after which the 64 bits of the incompressible block are appended.

Table 2.5: The coding scheme

Case	Coding Bits	Description
1a	`1 1`	For the block with the shortest code in the codebook
1b	`0 0` + 5 bits + $\mathcal{C}(b)$	For other blocks found in the codebook
2	`0 1` + $\mathcal{C}_{RCRC}(b)$	For blocks compressed by RCRC
3	`1 0` + 64 bits	For uncompressed blocks

The decoding process is straightforward. If the decoder encounters `1 1`, then it recognizes the symbol as the codebook entry with the shortest code. If the decoder reads 0 0, it identifies the codebook entry whose code length, $L(\mathcal{C}(b))$, is given by the next 5 bits. Then, the decoder reads the next L bits to determine the codewords which leads to the corresponding block in the codebook. If the decoder encounters 0 1, then the RCRC decoding process follows. Finally, if the decoder encounters 1 0, then it reads the subsequent 64 bits.

The details of the proposed method exposed in the previous sections and the cases illustrated in Table 2.5 motivate the following general encoding model. Let B_H, B_R, and B_U be the partitions of set $B_{\mathfrak{I}}$ (*see* Definition 2.6 in Section 2.2). Recall that $L(\mathcal{C}(b))$ and $L(\mathcal{C}_{RCRC}(b))$ denote, respectively, the length in bits of the Huffman code of block b and the length in bits of the RCRC output. Let ξ be a random variable denoting the random event $\xi = b_{\mathfrak{I}}$, $b_{\mathfrak{I}} \in B_{\mathfrak{I}}$ and let $P(\xi = b_{\mathfrak{I}}) = p(b_{\mathfrak{I}})$ denote the probability of ξ. Based on an empirical approach, one can evaluate the probabilities $P(\xi = b_H) = p(b_H)$, $b_H \in B_H$; $P(\xi = b_R) = p(b_R)$, $b_R \in B_R$; and $P(\xi = b_U) = p(b_U)$, $b_U \in B_U$. Then, the expected compression size in bits is given by the following model:

$$\begin{aligned} E_p[\xi] = \;& 2P(\{b_H \mid L(\mathcal{C}(b_H)) = 1\}) && \text{Case 1a} \\ & + \textstyle\sum_{b_H \in B_H,\, L(\mathcal{C}(b_H))>1} p(b_H)\,[7 + L(\mathcal{C}(b_H))] && \text{Case 1b} \\ & + \textstyle\sum_{b_R \in B_R} p(b_R)\,[L(\mathcal{C}_{RCRC}(b_R))] && \text{Case 2} \\ & + 66 \textstyle\sum_{b_U \in B_U} p(b_U)\,. && \text{Case 3} \end{aligned} \tag{2.23}$$

Here, $E_p[\cdot]$ denotes the expectation operator. Formula (2.23) provides a general model for the compression size in bits. The expected compression ratio, by formula (2.6) for $k = 2$, is equal to:

$$CR = \frac{E_p[\xi] \cdot hw/64}{hw} = \frac{E_p[\xi]}{64}\,, \tag{2.24}$$

where h and w are the image dimensions, and $hw/64$ is the number of 8×8 blocks in the image. In general, the expected compression ratio depends on the distribution of 8×8 blocks of an input binary image.

One may speculate on the overhead amount of bits this scheme uses for encoding blocks. Specifically, if the 8×8 block with a Huffman code length of 1 bit occurs, say, 50% of the time and it will be encoded with two bits (Case 1a), then the expected compression size for that block will double. Also, 7 overhead bits are used for the remaining Huffman codes in the codebook (Case 1b), whereas ideally Huffman codes should solely be employed as per their purpose. While this encoding *per se* is correct, it seems reasonable to look for a more efficacious coding scheme for the proposed method. This is the objective of the next section.

2.7 Alternative Coding Scheme

The reason why the coding scheme introduced in Section 2.6 incurs a considerable amount of overhead encoding information lays on the fact that RCRC interferes with codebook coding. Consequently, the compressed bit stream contains an admixture of strings representing Huffman codes and strings representing the RCRC output per block. Thus, a distinction between such encoded bits needs to be made explicitly for the decoder to function correctly. The cases covered in Table 2.5 are sufficient and necessary to reconstruct the original binary image exactly. This issue being stated, in this section we look at an alternative coding scheme and we conduct a sensitivity analysis between the two schemes to study under what conditions one outperforms the other.

In order to understand the mechanism of the alternative coding scheme, it is important to illustrate with a simple example how Huffman decoding works. Consider the string `LILIANA` of length 7 and relative probabilities of letters: $P(L) = P(I) = P(A) = 2/7$ and $P(N) = 1/7$. The Huffman algorithm will yield the following codes for the four letters: $\mathcal{C}(L) = 00$, $\mathcal{C}(I) = 01$, $\mathcal{C}(A) = 10$, and $\mathcal{C}(N) = 11$. Figure 2.16

is an exhibit of this particular Huffman tree, where the labels on the edges denote the codes employed for encoding and the nodes represent the letters and their parent nodes. The same tree has to be supplied to the decoder, which decodes a given bit string if a leaf node is encountered in the tree.

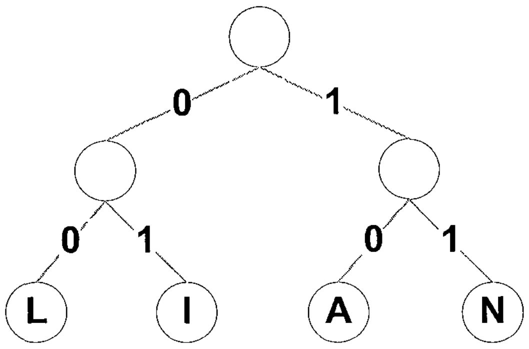


Figure 2.16: Huffman tree for string `LILIANA`

Suppose the decoder receives the string **S** = `00010000011011`. It reads the first bit, **S**[1] = `0`, and starts to traverse the tree in Figure 2.16 from the root to the node on the left, since the label on the left edge is 0 and equals **S**[1]. However, the node is not a leaf node and the decoder reads the next bit in the sequence, which is **S**[2] = `0`. Finally, leaf node `L` is reached and the decoder outputs letter 'L'. This procedure continues until the end of the received string is encountered. It is easy to check that the decoded string will be `LILLIAN`.

Technically, the decoding process terminates when the decoder encounters a special signal called the end-of-file (EOF) signal. In practice, given an alphabet $\mathcal{A}$ of cardinality $||\mathcal{A}||$, an additional EOF symbol is added to the alphabet with a very small probability value. This symbol is treated the same way as the other members of $\mathcal{A}$, and will thus be included in the Huffman tree. The EOF signal will have its own binary code. Since it is assigned a very small probability value (because it occurs only once at the end of the string), then its binary code is usually the longest. The effect of such a code is practically negligible [8].

In light of the aforementioned, we introduce two 'flag' signals for the alternative coding scheme of the proposed method. The first signal is the *break-codebook-coding* (BCC) signal, and the second is the *incompressible-block* (ICB) signal. These two flags are considered as members of the alphabet of 8×8 blocks and will be added to the constructed codebook with probabilities p_{BCC} and p_{ICB}, and the Huffman algorithm will assign binary codes to both flags. The purpose of the BCC block is to mark the interruption of codebook encoding for an input 8×8 block and the commencement of the RCRC encoding for that block. If RCRC encodes the block in less than 64 bits, then the compressed bit stream for that block will consist of the Huffman code of BCC, $\mathcal{C}(BCC)$, and the RCRC output, $\mathcal{C}_{RCRC}(b)$. If the block is incompressible, then the 64 bits are preceded by the Huffman code of ICB, $\mathcal{C}(ICB)$.

Decoding is straightforward. If the decoder encounters bits $\mathcal{C}(BCC)$, it will traverse the tree to decode flag block BCC. That block calls for an interruption of Huffman tree decoding and the decoder turns to RCRC decoding. If bits $\mathcal{C}(ICB)$ are encountered, then the flag block ICB informs the decoder to read the subsequent 64 bits in the compressed bit stream.

The probabilities p_{BCC} and p_{ICB} determine the Huffman codes for the two flag blocks. These values are assigned empirically based on the average rate of RCRC compression and the average percentage of incompressible blocks for large data sets. For a large variety of binary images, we observed that, on average, 93% of the blocks are compressed by the codebook. Out of the remaining 7% of blocks, 5% are compressed by RCRC and 2% remain uncompressed. The constructed codebook for this scheme has 6954 entries. The Huffman codes for flag signals BCC and ICB are `1110` and `110001`, respectively.

The alternative coding scheme has some apparent advantages over the coding scheme described in Section 2.6. First, it eliminates Case 1a in Table 2.5 as it does not require two overhead bits to encode the codebook block with the shortest Huffman code. Second, it eliminates the 7 overhead bits of Case 1b that are used for the remaining Huffman codes. We observed that the empirical occurrence of the blocks covered by Cases 1a and 1b was larger than the blocks compressed by RCRC and the incompressible blocks, on average. Therefore, the alternative coding scheme is expected to yield better compression ratios for binary images. Table 2.6 summarizes the three cases of the alternative coding scheme.

Table 2.6: The alternative coding scheme

Case	Coding Bits	Description
1	$\mathcal{C}(b)$	For blocks in the codebook
2	$\mathcal{C}(BCC) + \mathcal{C}_{RCRC}(b)$	For blocks compressed by RCRC
3	$\mathcal{C}(ICB)$ + 64 bits	For uncompressed blocks

As an example, consider the 8×8 blocks in Figure 2.17. Table 2.7 shows the blocks compressed by the codebook, the blocks compressed by RCRC, and one incompressible block. The resulting encoded bit stream is illustrated in Figure 2.18. In this example, operator $||$ denotes string concatenation. Flag block BCC informs the decoder that the subsequent $8+8+L(RB)$ bits will be decoded using the RCRC algorithm, whereas flag block ICB tells the decoder to merely scan the subsequent 64 bits.

Table 2.7: Encoding of blocks in Figure 2.17

Block b	$b \in \mathcal{D}$	$L(\mathcal{C}_{\mathrm{RCRC}}(b)) < 64$	Incompressible	Final Coding
1	YES			$\mathcal{C}(b_1)$
2	YES			$\mathcal{C}(b_2)$
3		YES		$\mathcal{C}(BCC)\|\|\mathcal{C}_{RCRC}(b_3)$
4	YES			$\mathcal{C}(b_4)$
5	YES			$\mathcal{C}(b_5)$
6	YES			$\mathcal{C}(b_6)$
7		YES		$\mathcal{C}(BCC)\|\|\mathcal{C}_{RCRC}(b_7)$
8			YES	$\mathcal{C}(ICB)\|\|b_8$

Block 1	Block 2	Block 3	Block 4
0 0 1 1 1 1 1 1	1 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 1 1 1 1 1 1	1 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 1 1 1 1 1 1	1 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1	1 1 1 1 1 1 1 1	1 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 0 0 0	0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 0 0 0	1 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 0	0 1 0 0 0 1 0 0
0 0 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	0 0 1 0 0 0 0 1
0 0 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 0 0 1 0 0 0 0
0 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	0 0 0 0 1 0 0 0
0 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	0 0 1 0 0 1 0 0
1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 0	0 0 0 0 0 0 1 0
1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 0	0 0 0 1 0 0 0 1
Block 5	Block 6	Block 7	Block 8

Figure 2.17: Example of eight input 8×8 blocks

$$\boxed{\mathcal{C}(b_1)||\mathcal{C}(b_2)||\mathcal{C}(BCC)\mathcal{C}_{RCRC}(b_3)||\mathcal{C}(b_4)||\mathcal{C}(b_5)||\mathcal{C}(b_6)||\mathcal{C}(BCC)\mathcal{C}_{RCRC}(b_7)||\mathcal{C}(ICB)b_8}$$

Figure 2.18: Compressed bit stream of blocks in Figure 2.17

As for the previous encoding scheme, we provide the following general model for the alternative scheme. Let B_H, B_R, and B_U be the partitions of set $B_\mathfrak{I}$ (*see* Definition 2.6 in Section 2.2). Let ξ be a random variable denoting the random event $\xi = b_\mathfrak{I}$, $b_\mathfrak{I} \in B_\mathfrak{I}$ and let $P(\xi = b_\mathfrak{I}) = p(b_\mathfrak{I})$ denote the probability of ξ. Based on an empirical approach, one can evaluate the probabilities $P(\xi = b_H) = p(b_H)$, $b_H \in B_H$; $P(\xi = b_R) = p(b_R)$, $b_R \in B_R$; and $P(\xi = b_U) = p(b_U)$, $b_U \in B_U$. Then, the expected compression size in bits for the alternative coding scheme is given by the following model:

$$\begin{aligned} E_p^*[\xi] = \; & \textstyle\sum_{b_H \in B_H, b_H \notin \{BCC, ICB\}} p(b_H) L(\mathcal{C}(b_H)) && \text{Case 1} \\ & + \textstyle\sum_{b_R \in B_R} p(b_R) \left[L(\mathcal{C}(BCC)) + L(\mathcal{C}_{RCRC}(b_R)) \right] && \text{Case 2} \\ & + \left[L(\mathcal{C}(ICB)) + 64 \right] \textstyle\sum_{b_U \in B_U} p(b_U) \, , && \text{Case 3} \end{aligned} \tag{2.25}$$

where, $E_p^*[\cdot]$ denotes the expectation operator. Formula (2.25) provides a general model for the compression size in bits. The expected compression ratio, by formula (2.6) for $k = 2$, is equal to:

$$CR^* = \frac{E_p^*[\xi] \cdot hw/64}{hw} = \frac{E_p^*[\xi]}{64}\,, \tag{2.26}$$

where h and w are the image dimensions, and $hw/64$ is the number of 8×8 blocks in the image. We denote the expected compression size of the alternative coding scheme as E_p^* to distinguish it from its counterpart E_p given in formula (2.23).

One may notice that the overhead bits of the alternative coding scheme incurred for blocks compressed by RCRC and for incompressible blocks are larger than the amount given for Cases 2 and 3 in Table 2.5. Specifically, $L(\mathcal{C}(BCC)) = 4$ and $L(\mathcal{C}(ICB)) = 6$, which are greater than the 2 overhead bits used for the same cases in Table 2.5. For that reason, we aim at determining which scheme outperforms the other on average and under which conditions. Thus, we perform a sensitivity analysis using a Monte Carlo simulation between the models in formulas (2.23) and (2.25) to observe how the two proposed coding schemes perform for various input 8×8 blocks and their distributions. From this point onward, we denote as $\mathcal{C}_0$ the coding scheme introduced in Section 2.6 and as $\mathcal{C}_A$ the alternative coding scheme introduced in this section.

2.8 Sensitivity Analysis on the Coding Schemes

Based on the results in Table 2.3, we generate 8×8 block samples from a log-series distribution with parameter $\theta = 0.995$ and probability mass function $P(n) = \frac{-0.995^n}{n \ln(0.005)}$ with support $n = \{1, 2, \ldots\}$. By convention, $n = 1$ denotes the 0-valued block, $n = 2$ denotes the 1-valued block, and the other ordinal values denote the remaining 8×8 blocks in the codebook, which have been initially sorted in descending order per the corresponding empirical probabilities (see Section 2 3 2)

The procedure we constructed for the Monte Carlo simulation is as follows

1 Generate 100 1024×1024 binary images, for a total sample size equal to 25600 8×8 blocks In order to obtain results with an error less than 2%, approximately 24000 blocks per sample are required[9]

   The binary images are generated by retrieving blocks from the codebook with probability D and constructing the remaining 8×8 blocks with probability $(1 - D)$ The probability of generating a white pixel for each of the remaining blocks is denoted as P For instance, $D = 0\ 8$ and $P = 0\ 5$ imply that blocks are generated from the codebook 80% of the time and are constructed uniformly (i e the probability of a white or black pixel is 0 5) 20% of the time Parameter D varies from 0 to 1, whereas parameter P varies from 0 5 to 1 Sensitivity analysis is carried out on both parameters Note that $P < 0\ 5$ implies that the probability of constructing a *black* pixel is $1 - P > 0\ 5$, which is technically covered by the range 0 5 to 1 by considering the inverted pixel This observation reduces the total number of samples required to conduct the simulation and sensitivity analysis Also, $P = 1$ implies the generation of all-white blocks and these blocks are generated from the codebook Therefore, we consider a value equal to 0 98 as the maximum ranging value for parameter P

   In this experiment we let $D = \{0, 10, 20, \quad , 100\}$ and $P = \{0\ 5, 0\ 6, \quad , 0\ 98\}$ For each parameter value, we generated 100 binary images as described above to evaluate, on average, the compression ratios yielded by the two coding schemes

[9]The error ϵ in Monte Carlo analysis is given by the expression $\epsilon = \frac{3\sigma}{\sqrt{N}}$, where σ is the standard deviation and N is the sample size Here, σ is estimated by the population's standard deviation between the minimum and maximum compression ratios per block, which are $-10\ 9375\%$ (with occurrence only when all blocks are incompressible) and 0 9362 (imposed by the entropy), respectively The error is given by the average of the minimum and maximum compression ratio per block multiplied by 2% Substituting these values in the error expression above and solving for N, yields $N \approx 24000$

2. For each value of parameter P and for every D, evaluate E_p and E_p^* averaged over 100 samples.

The results for each value of P are illustrated in Tables 2.8 to 2.13. From the results, we can observe how the two coding schemes perform under various values of parameters D (% of blocks found in the codebook) and P (the probability of a white pixel). The farther away D gets from the break-even point, the more does the discrepancy between $\mathcal{C}_0$ and $\mathcal{C}_A$ increase, wherein a smaller D favors $\mathcal{C}_0$ and a larger D favors $\mathcal{C}_A$. For small values of D (typically $D < 10\%$) and for $0.5 \leq P \leq 0.8$, the coding schemes do not compress: the negative ratios imply an overhead coding size larger than the original image size.

It can be noticed that parameter P does not have any major effect on the relative average performance of the two coding schemes. In all graphs, the break-even point is between 45% and 60% and the two schemes perform almost similarly in this range. Therefore, it may be conjectured that the alternative coding scheme, $\mathcal{C}_A$, is preferable over scheme $\mathcal{C}_0$ if the percentage of blocks found in the codebook is greater than 60%. Moreover, the results in Table 2.13 show that for large values of P, $\mathcal{C}_A$ attains better compression ratios than $\mathcal{C}_0$ for all values of D.

In Section 2.4.3, we illustrated theoretically the probability, $P(R)$, of RCRC compressing a block. We established that $P(R)$ depends on the probability P of white and black pixels in the block. We concluded that for relatively small or relatively large values of P, the chances RCRC compresses are high. In light of that, the parameter value P affects the probability $P(R)$. Observe the results in Table 2.13 for $P = 0.98$. Based on equation (2.22), we have $P(R) = 0.9999$. Here, we may speculate that no block is incompressible; thus, $\mathcal{C}_A$ should be the preferred coding scheme, as also verified graphically.

Table 2.8: Simulation results for $P = 0.5$

D (%)	E_p	E_p^*
0	-0.0311	-0.0935
10	0.0434	-0.0082
20	0.1502	0.1115
30	0.2149	0.1859
40	0.2888	0.2696
50	0.3821	0.3748
60	0.4812	0.4886
70	0.5528	0.5725
80	0.6593	0.6934
90	0.7585	0.806
100	0.8754	0.9366

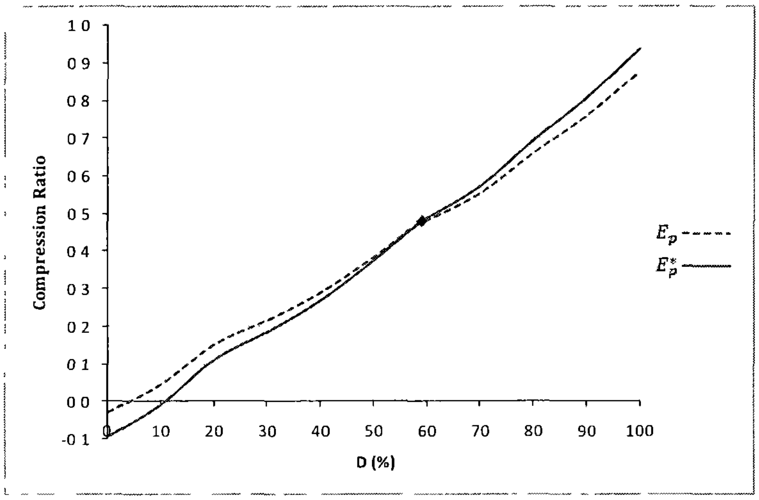


Table 2.9: Simulation results for $P = 0.6$

D (%)	E_p	E_p^*
0	-0.0312	-0.0936
10	0.0529	0.0019
20	0.1273	0.086
30	0.2177	0.1899
40	0.2952	0.2788
50	0.3732	0.3692
60	0.4756	0.481
70	0.5404	0.5575
80	0.6593	0.6892
90	0.7779	0.8216
100	0.8799	0.9404

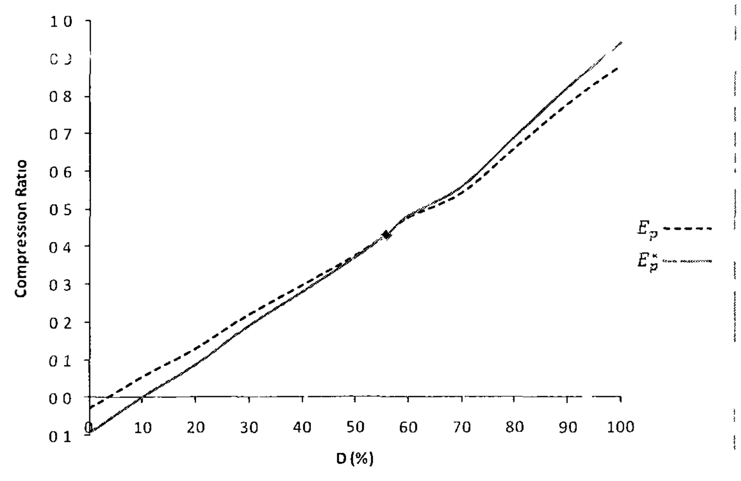


Table 2.10: Simulation results for $P = 0.7$

D (%)	E_p	E_p^*
0	-0.0309	-0.093
10	0.0515	0.0011
20	0.134	0.0942
30	0.2158	0.1869
40	0.308	0.2912
50	0.3867	0.3815
60	0.4766	0.4832
70	0.5427	0.5587
80	0.6562	0.6908
90	0.7665	0.8133
100	0.8748	0.9381

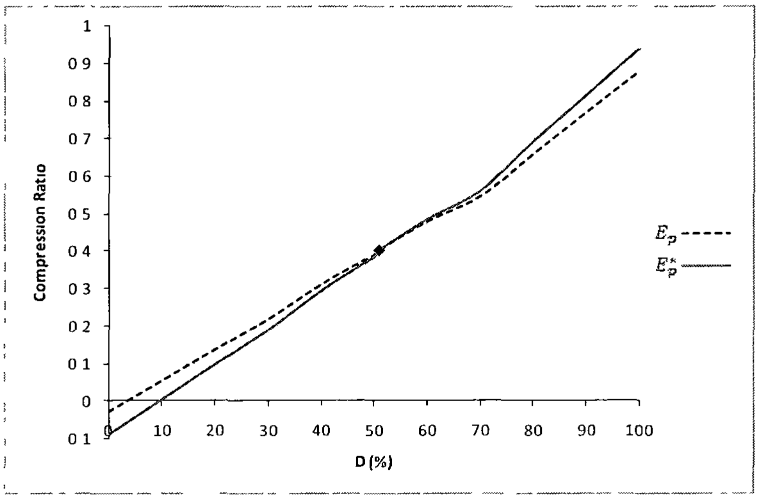


Table 2.11: Simulation results for $P = 0.8$

D (%)	E_p	E_p^*
0	-0.019	-0.0777
10	0.06	0.0116
20	0.1437	0.1074
30	0.2387	0.214
40	0.319	0.3066
50	0.4008	0.397
60	0.4552	0.4616
70	0.5618	0.5814
80	0.6627	0.6954
90	0.762	0.8098
100	0.8762	0.9381

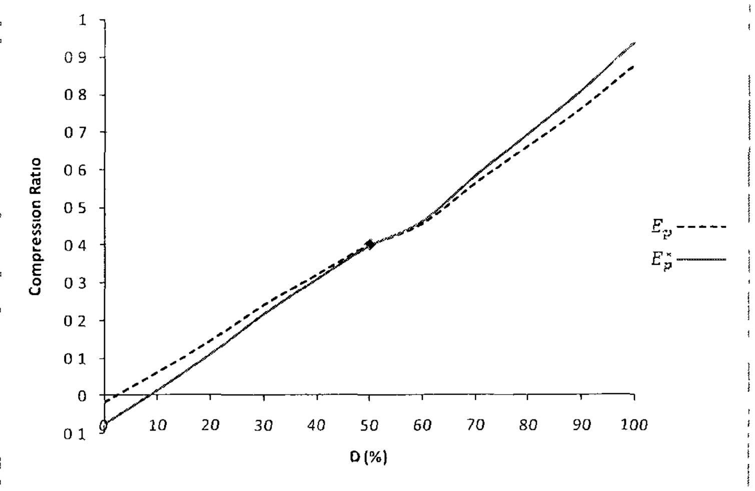


Table 2.12: Simulation results for $P = 0.9$

D (%)	$\mathbf{E_p}$	$\mathbf{E_p^*}$
0	0.116	0.0751
10	0.1792	0.1471
20	0.2402	0.216
30	0.3298	0.3187
40	0.3999	0.3965
50	0.4713	0.4782
60	0.5475	0.5656
70	0.6187	0.6473
80	0.6814	0.7171
90	0.7892	0.8387
100	0.8745	0.9366

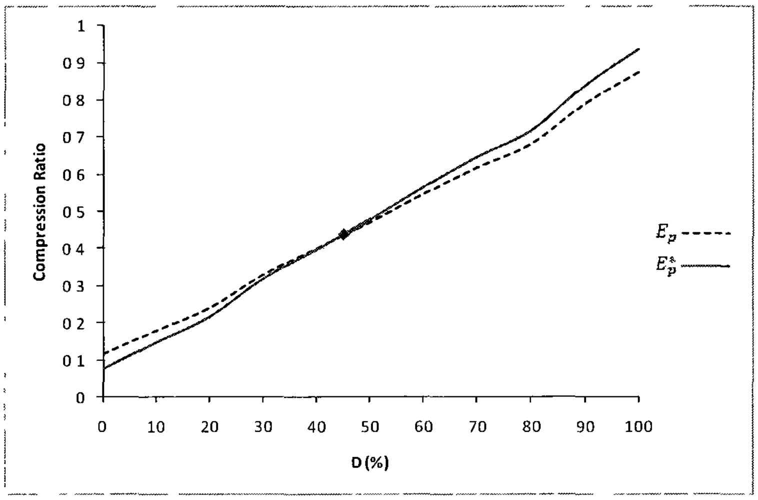
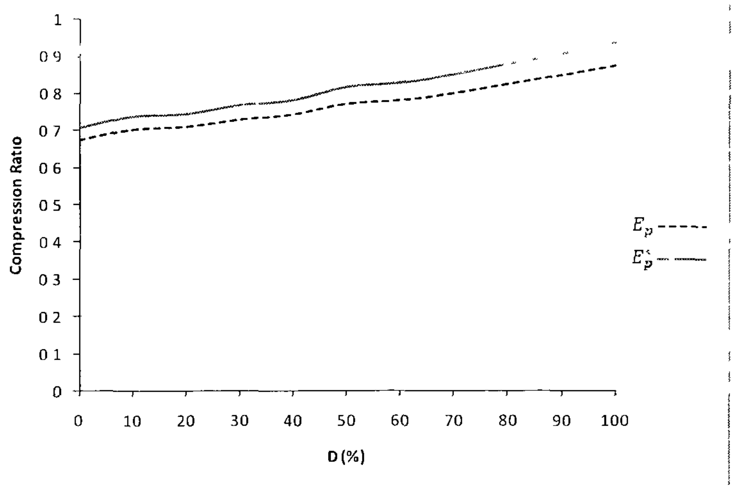


Table 2.13: Simulation results for $P = 0.98$

D (%)	$\mathbf{E_p}$	$\mathbf{E_p^*}$
0	0.6745	0.7057
10	0.7032	0.7369
20	0.7105	0.7434
30	0.7298	0.7678
40	0.7436	0.7816
50	0.7723	0.8175
60	0.7818	0.8285
70	0.801	0.8517
80	0.8254	0.8797
90	0.8496	0.9076
100	0.8749	0.9379



In addition to parameters D and P, we conduct sensitivity analysis on the probabilities p_w and p_b of white and black 8×8 blocks, respectively. There are two reasons we consider these parameters. First, p_w affects the performance of $\mathcal{C}_0$, as discussed in Section 2.6. Also, $\mathcal{C}_A$ was designed precisely to reduce the overhead that white blocks impose on $\mathcal{C}_0$. Hence, it is important to observe how $\mathcal{C}_0$ and $\mathcal{C}_A$ behave under different values of p_w. Second, white and black blocks tend to have the highest frequencies of occurrence in relatively large samples of binary images. The empirical probabilities illustrated in Section 2.3.3 suggest that black and white blocks comprise approximately 73% of blocks. Then, reducing the probability of white and black blocks for the sensitivity analysis brings about an increase in the frequency of occurrence of other blocks with longer code lengths. Under such circumstances, we want to observe whether the two flags of scheme $\mathcal{C}_A$ incur more coding overhead than the straightforward coding scheme $\mathcal{C}_0$.

The ranges we selected for the probability of white and black blocks are, respectively, $p_w \in \{0.1, 0.15, 0.2, \ldots, 0.5, 0.55\}$ and $p_b \in \{0, 0.05, 0.1, 0.15, 0.16, \ldots, 0.25\}$. The lower bound for p_w is based on the empirical judgment that binary images are expected to have a certain amount of white background, whereas the lower bound for p_b is based on the observation that binary images need not necessarily comprise black 8×8 blocks. For instance, binary textual images containing text with thin font faces and small font sizes (such as Arial, 9pt) do not generally yield black 8×8 blocks.

Similar to the simulation for parameters D and P, we generate 25600 blocks for each value of p_w and p_b. White blocks are generated $100p_w\%$ of the time, black blocks $100p_b\%$ of the time, the remaining $1 - (p_w + p_b)$ of the blocks are generated from the codebook following a log-series distribution. This procedure is repeated for various values of D and P, as illustrated in the preceding sensitivity analysis.

Figures 2 19 to 2 24 exhibit results for

$$P = \{50, 60, 70, 80, 90, 98\%\},$$

$$D = \{0, 30, 60, 90\%\},$$

$$p_w = \{0\ 1, 0\ 15, 0\ 2, 0\ 3, 0\ 4, 0\ 5, 0\ 55\},$$

$$p_b = \{0, 0\ 1, 0\ 15, 0\ 17, 0\ 2, 0\ 22, 0\ 23, 0\ 25\}$$

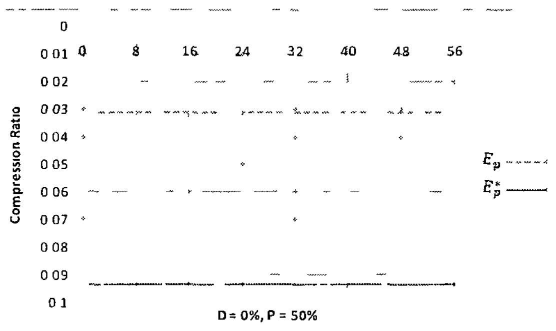
For each of the 56 pairs (p_w, p_b), we graph the average compression ratios yielded by $\mathcal{C}_0$ and $\mathcal{C}_A$ for 100 1024×1024 binary images used per pair

Consider the case when $D = 0\%$, i e no blocks are generated from the codebook The only component left to compress blocks is RCRC with probability $P(R)$ As noted in Section 2 4 3, $P(R)$ depends on the probability P the higher the value of P is, the higher the chances RCRC compresses a block become It can be observed from Figures 2 19 to 2 22 that for $D = 0\%$ and $P = \{50, 60, 70, 80\}$ RCRC fails to compress, and most blocks remain incompressible Notice that for small values of P the resulting compression trend is almost flat For $D = 0\%$ and $P = 90\%$ (Figure 2 23), there is compression but at insignificant rates, whereas for $D = 0\%$ and $P = 98\%$ RCRC compresses significantly most blocks Moreover, for all P, $\mathcal{C}_0$ performs better than $\mathcal{C}_A$ because $\mathcal{C}_A$ incurs more overhead bits with flags BCC and ICB (see Table 2 6) than the 4 overhead bits incurred by $\mathcal{C}_0$ (see Table 2 5)

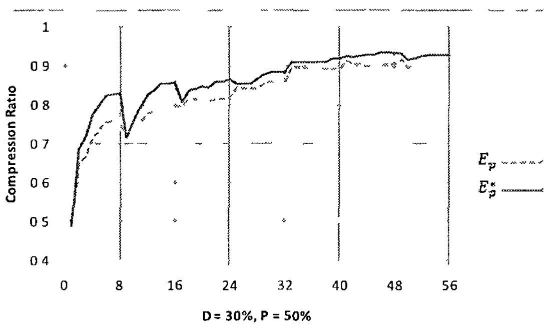
For $D = 30\%$ and higher, $\mathcal{C}_A$ outperforms $\mathcal{C}_0$ and the compression discrepancies tend to increase as D increases Both coding schemes expose increasing compression trends as the probability p_w changes from 0 1 to 0 55 It can also be noticed that the compression rates yielded by $\mathcal{C}_0$ fluctuate more than those yielded by $\mathcal{C}_A$ For example, consider cases (b), (c) and (d) in Figure 2 23 In these cases, for every value of p_w, E_p decreases as p_b increases from 0 to 0 25 whereas E_p^* is always increasing

The reason for this fluctuation lays on the coding of black blocks: $\mathcal{C}_0$ incurs 7 overhead bits, whereas $\mathcal{C}_A$ employs solely the Huffman code of black blocks. Hence, for small values of p_b, $\mathcal{C}_0$ will yield higher average compression ratios, but $\mathcal{C}_A$ is still superior. This fluctuation lessens when both p_w and p_b are large, as observed in the figures.

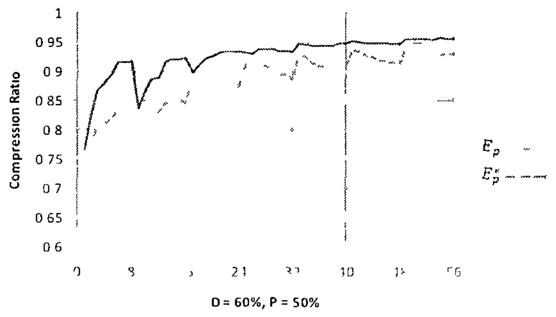
In the preceding simulation results, we stated that scheme $\mathcal{C}_A$ should be chosen over $\mathcal{C}_0$ for $D \geq 60\%$. Based on the sensitivity analysis on p_w and p_b, it can be conjectured that for some value D^* between 0% and 30%, $\mathcal{C}_0$ outperforms $\mathcal{C}_A$ for all $D < D^*$. Hence, we may conclude here that for all $D \geq D^*$ (or, specifically, $D \geq 30\%$), the preferred coding scheme should be $\mathcal{C}_A$. For a more solid conclusion, one needs to conduct sensitivity analyses on all the probability parameters of the two coding schemes. In practice, however, such simulations incur expensive computational costs. In all, the results presented here suffice to conclude that the alternative coding scheme illustrated in Section 2.7 should be the preferred scheme for compressing the average binary image.



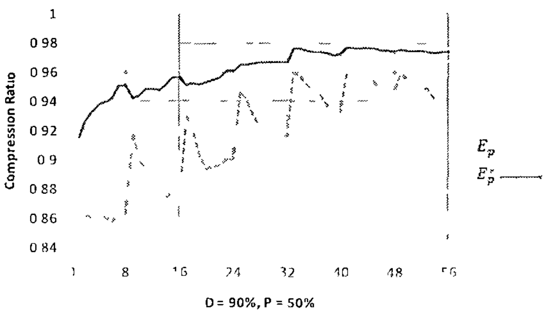
(a) $D = 0\%$



(b) $D = 30\%$

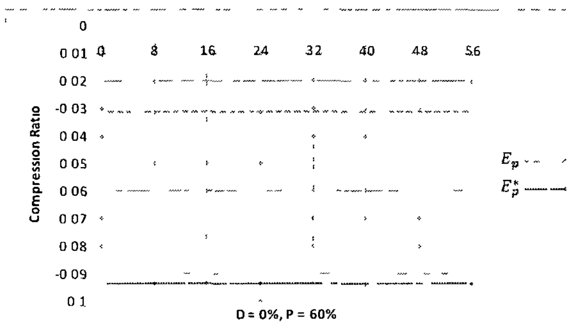


(c) $D = 60\%$

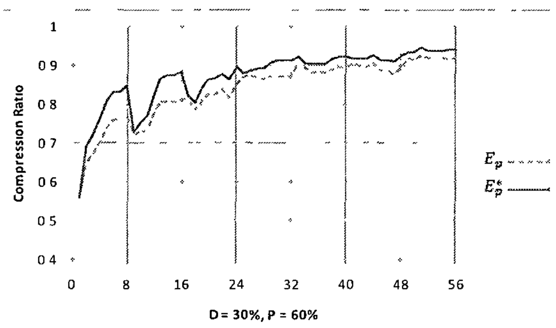


(d) $D = 90\%$

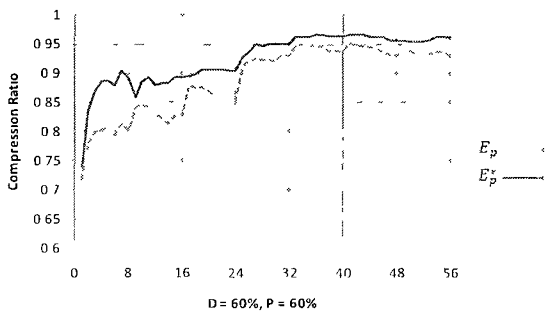
Figure 2.19: Simulation results for $P = 50\%$



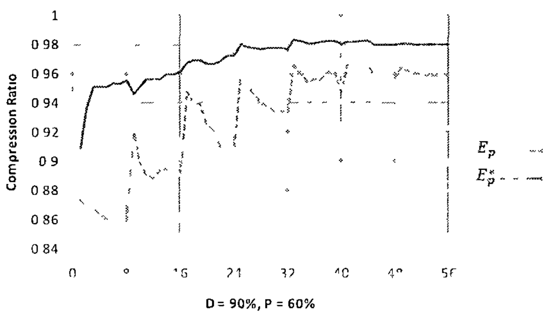
(a) $D = 0\%$



(b) $D = 30\%$

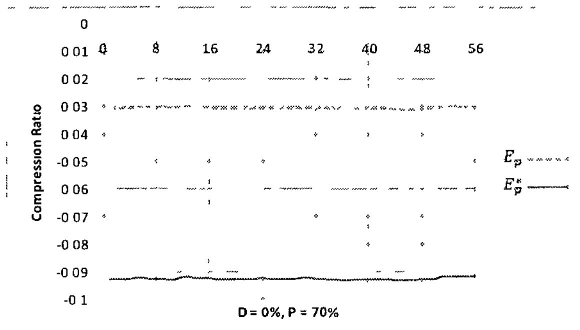


(c) $D = 60\%$

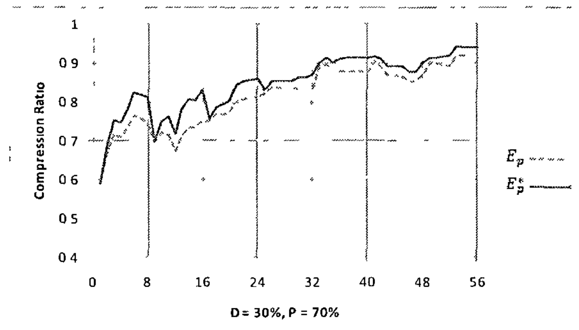


(d) $D = 90\%$

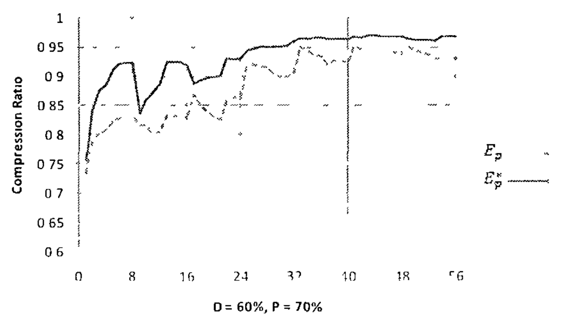
Figure 2.20: Simulation results for $P = 60\%$



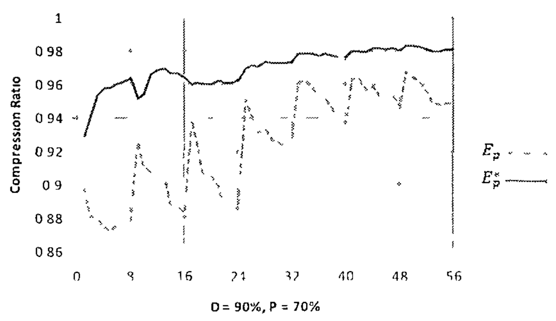
(a) $D = 0\%$



(b) $D = 30\%$

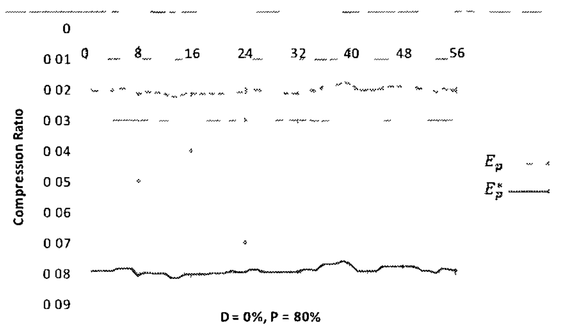


(c) $D = 60\%$

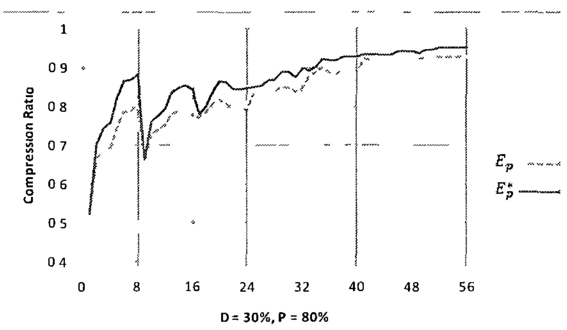


(d) $D = 90\%$

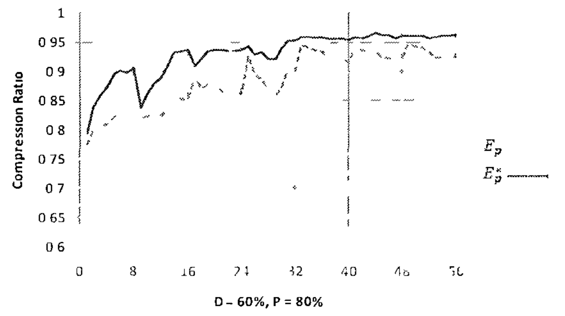
Figure 2.21: Simulation results for $P = 70\%$



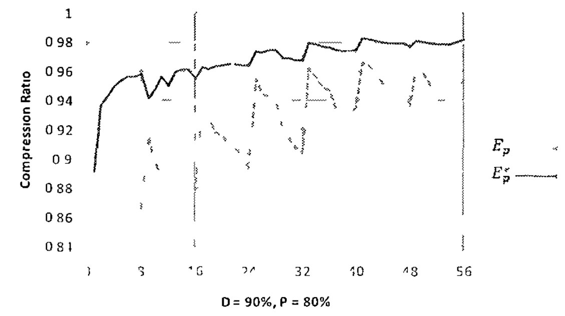
(a) $D = 0\%$



(b) $D = 30\%$

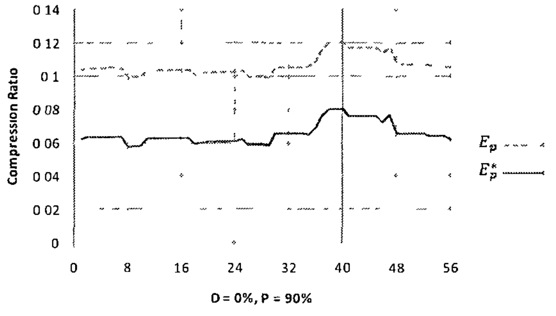


(c) $D = 60\%$

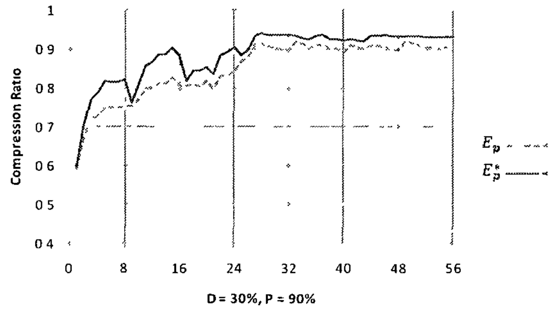


(d) $D = 90\%$

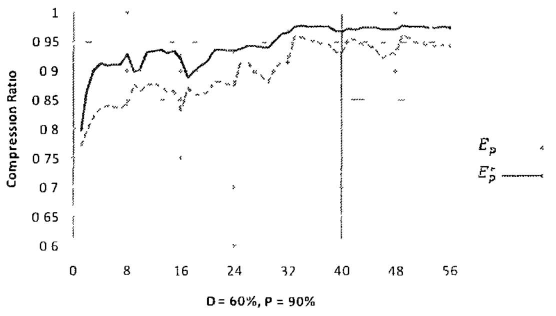
Figure 2.22: Simulation results for $P = 80\%$



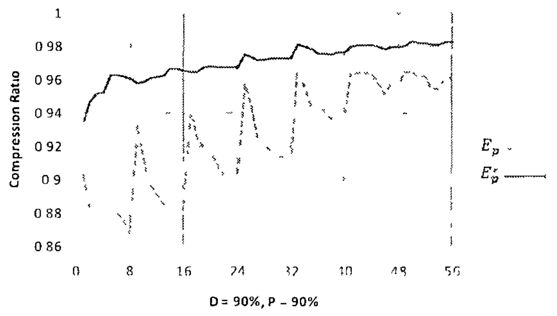
(a) $D = 0\%$



(b) $D = 30\%$

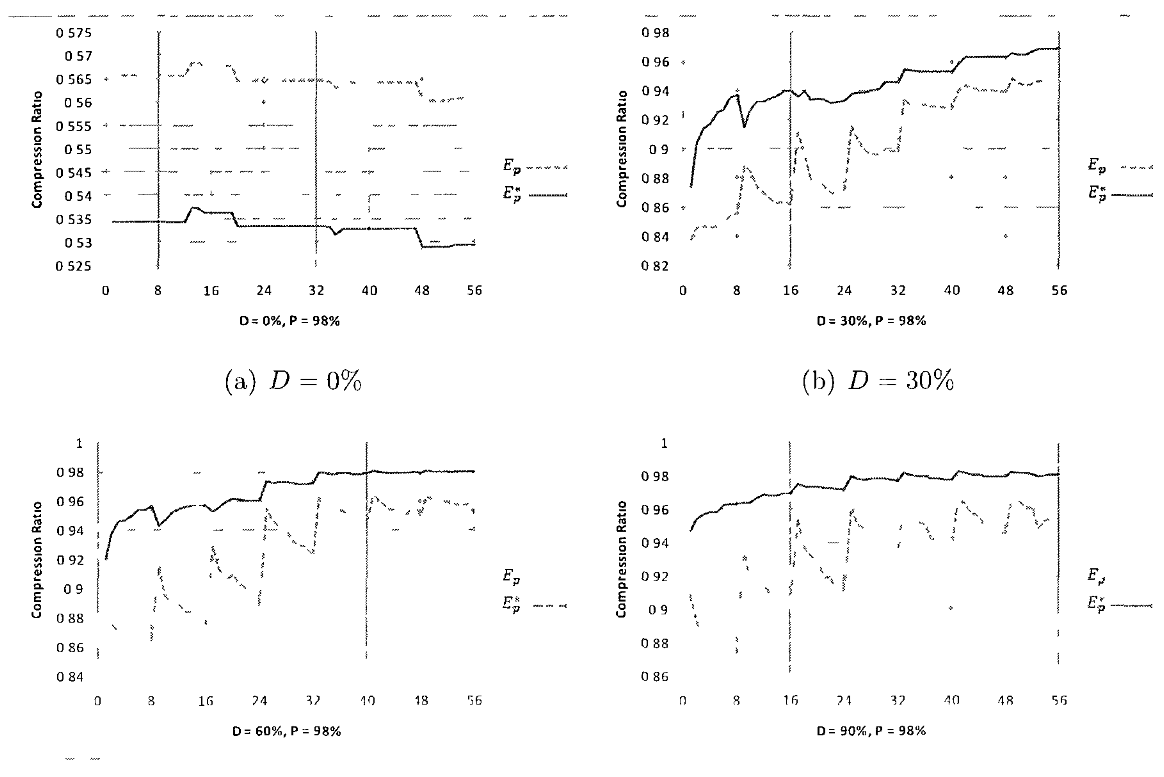


(c) $D = 60\%$



(d) $D = 90\%$

Figure 2.23: Simulation results for $P = 90\%$



(a) $D = 0\%$

(b) $D = 30\%$

(c) $D = 60\%$

(d) $D = 90\%$

Figure 2.24: Simulation results for $P = 98\%$

2.9 The Codebook Model for Arithmetic Coding

To this point, the illustrated codebook model has been used in conjunction with Huffman codes. Two coding schemes were developed based on that model. The alternative coding scheme exposed in Section 2.7 motivated us to use the codebook model along with the empirical probabilities of 8×8 blocks to compress via Arithmetic coding. In this case, the codebook comprises 6954 blocks (including the two flag symbols discussed in Section 2.7) along with the lower and upper probability values of each block. Technically, we implemented an integer-based arithmetic coder [19]. Encoding and decoding for arithmetic coding work the same way as the alternative coding scheme, $\mathcal{C}_A$, illustrated in Section 2.7.

2.10 Protagonists and Antagonists

The 100 most frequently occurring blocks are shown in Figure 2.25.[10] The blocks in cells a1 and a2 depict the 0-valued and the 1-valued blocks, respectively. Observe that the most frequently occurring blocks represent geometric primitives, such as points (cells a4–a7), lines (cells a3, a9, e20, etc.), triangles (cells c4, c10, e15, etc.), rectangles (cells b1–b6, c9, etc.), or a combination thereof. Moreover, there exist blocks that are inverted versions of each-other. For instance, the block in cell d1 is the inverted counterpart of the triangle in cell c7. This is because the data sample from which the codebook was constructed contained a combination of binary images with white and black backgrounds. In general, the codebook comprises regular geometric constructs.

As stated in Section 2.2, the set of all binary images can be partitioned into images compressible by the codebook, images compressible by RCRC but not by

[10]The digits in boldface represent numbers 10–20

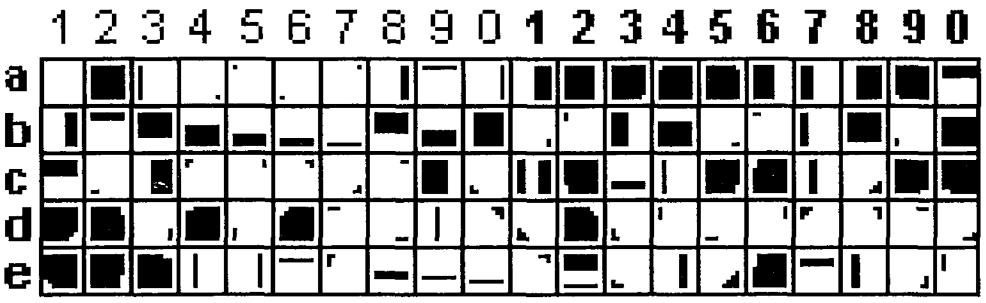


Figure 2.25: Visualization of the first 100 8×8 blocks

the codebook, and incompressible images. Images belonging to the first class expose primitive-geometric construct, given the nature of 8×8 blocks in the codebook. We randomly generated three 64×64 such images using the 6952 8×8 blocks per their probabilities. These images are shown in Figure 2.26. Notice the dominance of basic geometric constructs, which resemble some of the blocks in Figure 2.25. Such binary images are efficiently compressed by the codebook component of the proposed method, but such images can also be compressed using RCRC alone. However, as discussed in Section 2.3.2, the maximum Huffman code length of codebook blocks is 17 bits while blocks compressed with RCRC take on at least 17 bits. In practice, binary images exposing the regularity depicted in Figure 2.26 have a low (empirical) probability of occurrence.

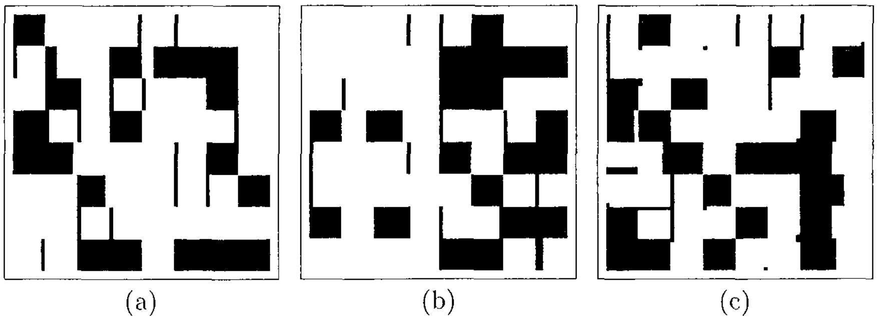


Figure 2.26: Binary images randomly generated using codebook blocks only

Blocks not in the codebook, but compressible by RCRC, may also evince regular geometric constructs For instance, the codebook does not contain all 8×8 blocks consisting of 63 0's and a 1 or 63 1's and a 0 Such blocks are efficiently compressed by RCRC However, RCRC compresses triangles less efficiently the larger the triangle in an 8×8 block, the less efficient RCRC becomes Figure 2 27 illustrates an 8×8 block evincing a triangle This block cannot be compressed by RCRC

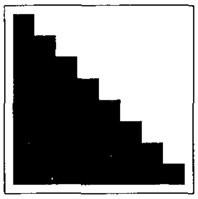


Figure 2.27: An incompressible geometric primitive

In addition, RCRC does not perform efficiently for 8×8 sparse matrices and matrices where the 1's are aligned in a non-linear fashion, such as diagonally, as depicted in Figure 2 28 As an instance, RCRC fails to compress 8×8 permutation matrices, i e matrices that have exactly one entry equal to 1 in each row and each column and 0 elsewhere

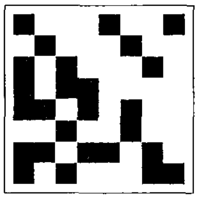


Figure 2.28: An incompressible 8×8 block

Furthermore, there exist blocks that cannot be compressed by the proposed method One way to ensure plausibly higher compression rates could be to resort to additional conventional coding techniques, such as Run-Length Encoding However, such approaches are not efficient for two reasons First, the empirical complexity of the proposed scheme would increase Second, the coding schemes would have to be extended to accommodate the new add-ons, thus adding more overhead bits to compression In general, it is inconclusive whether adding more schemes could increase compression rates, but it is almost certain that such add-ons would increase the complexity

Chapter 3

Applications

There is nothing so agonizing to the fine skin of vanity as the application of a rough truth.

– EDWARD BULWER-LYTTON

In this chapter, we report empirical results of the proposed compression method on binary and discrete-color images in comparison with JBIG2. The main reason why we compare results for binary image compression only with JBIG2—despite the fact that both methods are lossless—lays on that the standard JBIG2 is viewed as a generic compression scheme in much the same way as we claim the proposed method to be. Nevertheless, we note that for specific classes of binary and discrete-color images, ad-hoc compression methods have been proposed and successfully implemented, as exposed in Chapter 4.

The schematic diagram shown in Figure 3.1 illustrates the generic operation of the proposed compression scheme. The input image is appended in both dimensions to become divisible by 8. Then, layers are extracted through color separation yielding a set of bi-level matrices. Note that if the input image is bi-level, such as binary images, then it represents one layer by default. Next, each layer is partitioned into 8×8 blocks. Each 8×8 block of the original data is searched in the codebook. If it is found, the corresponding Huffman or Arithmetic code is selected and added to the compressed data stream. If it is not found, the row-column reduction coding attempts to compress the block. If the output of RCRC is smaller than 64 bits, the reduced block is appended to the compressed data stream. Otherwise, the original block is preserved. An example of color separation is illustrated in Section 3.2.

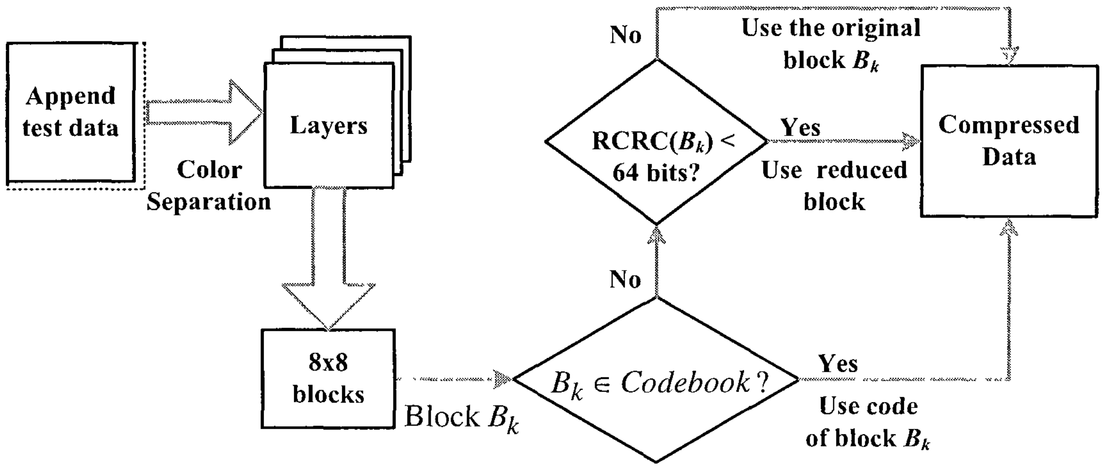


Figure 3.1: Generic diagram of the proposed compression scheme.

3.1 Binary Images

We tested the proposed method on a variety of more than 200 binary images collected from different sources. The sample we compiled comprises varying topological shapes ranging from solid objects to less regular, complex geometries. The empirical results presented here are classified in three categories: solid binary images, irregular geometries, and images JBIG2 compresses more efficiently than the proposed method. In all three cases, a selected set of binary images is given along with compression ratios of the proposed method using Huffman and Arithmetic codes, and JBIG2. A set of 112 labeled images and their compression ratios is exhibited in Appendix B.1.

Table 3.1 displays the compression ratios for 15 solid binary images. In the case of Huffman codes, the alternative encoding scheme, C_A, exposed in Section 2.7 performs better than the other coding scheme, $\mathcal{C}_0$, given in Section 2.6. On average, the proposed method outperforms the standard JBIG2 by approximately 3.06% in the case of Huffman codes when $\mathcal{C}_A$ is employed, and 3.07% when Arithmetic coding is employed. As stated in Section 2.7, the coding scheme $\mathcal{C}_A$ is more efficacious than scheme $\mathcal{C}_0$. The sensitivity analysis on the stochastic parameters of the coding models exposed in Section 2.8 provides a useful reference to apprehend the performance of the two coding schemes. In all cases, the alternative coding scheme, $\mathcal{C}_A$, outperforms the other coding scheme, $\mathcal{C}_0$. Finally, Table 3.2 shows the percentage of blocks compressed by the dictionary, the percentage of blocks compressed by RCRC, and the portion of incompressible blocks. Notice that the percentage of the latter is relatively small. This observation complies with the theoretical analyses on the codebook error as well as the sensitivity analysis for D close to 98% (see, for instance, Table 2.13 in Section 2.8).

Table 3.1: Empirical results for 15 selected binary images: solid shapes.

		Proposed Method			
Image	**Dimensions**	E_p	E_p^*	AC	**JBIG2**
059	200×329	88.99	93.72	94.58	88.58
071	545×393	90.72	93.75	94.22	89.82
074	203×247	86.9	92.66	93.16	87.68
075	790×480	92.78	96.5	96.25	94.8
076	245×226	86.24	92.75	93.41	86.82
077	450×295	88.47	95.65	95.38	94.83
079	245×158	85.35	91.42	91.96	84.91
080	491×449	91.86	95.71	95.86	92.17
081	245×248	89.2	92.84	93.3	86.69
082	491×526	92.21	96.33	96.08	94.31
083	354×260	88.93	95.29	95.48	92.24
085	167×405	86.9	92.55	93.62	87.7
086	335×500	91.12	95.88	95.62	94.97
087	447×459	89.89	96.2	95.73	93.86
090	350×357	86.68	95.02	94.8	92.18
	Average	90.36	95.26	95.27	92.43

Table 3.3 shows empirical results for 15 less regular binary images. These images

Table 3.2: Percentage of blocks compressed by the codebook, RCRC, and incompressible blocks

Image	Codebook	RCRC	Incompressible
059	96.7	3.3	0
071	95.48	4.32	0.2
074	95.04	4.47	0.5
075	98.53	1.46	0.02
076	95.44	4.56	0
077	98.62	1.23	0.14
079	94.19	5.65	0.16
080	98.73	1.25	0.03
081	94.96	4.84	0.2
082	98.9	1.03	0.07
083	98.52	1.28	0.2
085	95.42	4.39	0.19
086	98.49	1.4	0.11
087	99.38	0.52	0.09
090	98.08	1.87	0.05

are not as solid geometrics as the images given in Table 3.1. On average, the proposed method performed better than JBIG2 by 4.32% when Huffman coding with scheme C_A is used and 5.62% when Arithmetic coding is employed. Moreover, Table 3.4 shows the percentage of blocks compressed by the dictionary, the percentage of blocks compressed by RCRC, and the portion of incompressible blocks.

Table 3.5 provides the 8 binary images wherein JBIG2 outperforms either the Huffman coding, or the Arithmetic coding, or both components of the proposed method. On average, JBIG2 scores 0.51% higher compared to the Huffman coding and 0.92% higher than the Arithmetic coding aspect of the proposed method. Moreover, Table 3.6 shows the percentage of blocks compressed by the dictionary, the percentage of blocks compressed by RCRC, and the portion of incompressible blocks.

In addition, Table 3.7 exposes empirical results for 6 binary images comprising contour lines rather than filled regions. A sample image is shown in Figure 3.2, while the six images are given in Appendix B.1. Per the discussion in Section 2.10, 8×8

Table 3.3: Empirical results for 15 selected binary images irregular shapes

		Proposed Method			
Image	**Dimensions**	E_p	E_p^*	AC	**JBIG2**
004	512×800	93 25	95 99	95 73	94 69
005	1024×768	88 35	92 89	93 92	89 06
009	1061×1049	87 9	92 31	94 05	88 44
012	575×426	90 85	94 45	95 29	92 83
014	498×395	88 4	92 74	93 69	87 58
016	400×400	77 45	81 9	86 17	74
018	483×464	89 67	93 73	94 78	89 8
019	791×663	91 68	94 84	95 3	91 81
021	360×441	88 49	90 39	90 98	85 94
029	196×390	84 86	89 17	91 62	81 59
034	372×217	81 71	85 56	87 16	80 1
042	490×481	86 04	89 92	90 96	84 99
056	450×360	85 44	91 16	92 69	86 86
057	180×210	83 56	89 31	91 35	80 17
084	240×394	87 85	92 52	92 63	89 32
	Average	88 44	92 45	93 6	88 62

blocks extracted from such images may be classified as antagonists to the proposed method because of their topological irregularity In addition, JIBG2 has been reported to compress efficiently topological objects enclosed by contour lines On average, the proposed method performs better than JBIG2 by 2 42% for Huffman codes and 2 48% for Arithmetic coding

For empirical purposes, we conducted the following experiment We inverted the bit values in the six binary images discussed above, as illustrated in Figure 3 3 Bit inversion causes the white image background to become black When partitioned into 8×8 blocks, 1-valued blocks will dominate the set of blocks Based on the constructed codebook, 1-valued 8×8 blocks have a Huffman codeword length of 2 bits Hence, all else equal, it is expected that, on average, the proposed method will perform worse on the inverted images, but no change should be expected from JBIG2 since it is a context-based modeling scheme Next, we employed the proposed method and JBIG2, and the results are exposed in Table 3 8 Observe that, on average, JBIG2

Table 3.4: Percentage of blocks compressed by the codebook, RCRC, and incompressible blocks.

Image	Codebook	RCRC	Incompressible
004	97.38	2.54	0.08
005	95.92	3.55	0.54
009	95.82	3.89	0.29
012	96.35	3.65	0
014	95.65	4.03	0.32
016	83.89	15.3	0.81
018	97.03	2.92	0.06
019	97.42	2.52	0.06
021	91.5	7.26	1.24
029	92.98	6.37	0.65
034	85.03	14.36	0.61
042	90.67	8.99	0.34
056	93.59	6.14	0.27
057	92.59	7.09	0.32
084	93.74	6.19	0.06

has gained a relative additional compression of 0.52% from bit inversion, while the individual coding schemes of the proposed method have decreased by 9.97% for $\mathcal{C}_0$, 1.72% for $\mathcal{C}_A$, and 2.2% for the Arithmetic coding. In the case of bit inversion, JBIG2 outperforms Arithmetic coding by a relative difference of 0.3%, but scheme $\mathcal{C}_A$ still outperforms JBIG2 by a relative difference of 0.15%. All in all, in these cases, the proposed method and JBIG2 score relatively close compression ratios with no major cost difference.

Tables 3.9 and 3.10 display the percentage of blocks compressed by the dictionary, the percentage of blocks compressed by RCRC, and the portion of incompressible blocks for the 6 original and inverted binary images, respectively.

Table 3.5: Empirical results for 8 selected binary images: JBIG2 more efficient.

		Proposed Method			
Image	**Dimensions**	E_p	E_p^*	AC	**JBIG2**
008	2400×3000	92.56	97.59	96.98	98.34
022	315×394	84.38	89.66	93.32	91.94
028	2400×1920	94.37	97.73	97.47	98
064	640×439	94.59	96.77	96.93	96.84
088	1203×1200	91.12	95.88	95.3	95.94
094	1018×486	92.43	96.42	96.32	96.57
096	516×687	93.6	97.08	97.06	97.9
100	765×486	95.54	97.54	97.59	97.71
	Average	93.05	97.33	96.93	97.83

Table 3.6: Percentage of blocks compressed by the codebook, RCRC, and incompressible blocks.

Image	**Codebook**	**RCRC**	**Incompressible**
008	99.81	0.18	0.01
022	93.05	6.85	0.1
028	99.65	0.34	0.01
064	98.14	1.77	0.09
088	97.49	2.34	0.17
094	98.49	1.45	0.06
096	99.21	0.79	0
100	99.18	0.8	0.02

Table 3.7: Empirical results for 6 selected binary images: line boundaries.

		Proposed Method			
Image	**Dimensions**	E_p	E_p^*	AC	**JBIG2**
101	512×512	87.47	89.09	89.04	87.68
102	514×514	93.01	94.91	94.82	94.7
103	512×512	85.56	87.4	87.99	83.92
104	512×512	85.85	87.51	87.87	84.82
105	512×512	89.66	91.41	91.11	88.63
106	512×512	88.89	90.55	90.29	88.3
	Average	88.41	90.14	90.19	88.01

Table 3.8: Empirical results for the 6 inverted binary images: line boundaries.

	Proposed Method			
Image	E_p	E_p^*	AC	**JBIG2**
101	78.89	87.64	87.11	88.08
102	83.09	93.36	92.71	95
103	77.51	85.66	85.97	84.45
104	77.42	85.93	85.64	85.26
105	80.66	89.91	89.21	88.83
106	80.02	89.1	88.44	89.18
Average	79.6	88.6	88.18	88.47

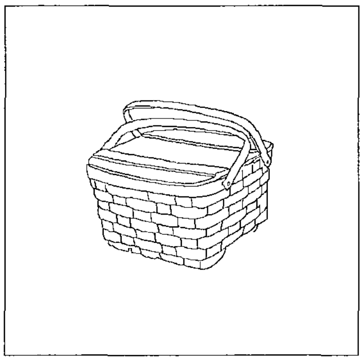


Figure 3.2: Binary image with line boundaries

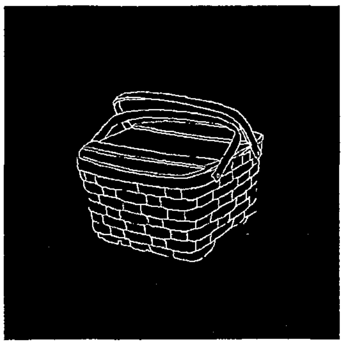


Figure 3.3: Binary image with line boundaries: inverted counterpart

Table 3.9: Percentage of blocks compressed by the codebook, RCRC, and incompressible blocks

Image	Codebook	RCRC	Incompressible
101	87.86	10.51	1.63
102	95.91	3.88	0.21
103	84.9	13.78	1.33
104	87.01	10.6	2.39
105	91.22	7.53	1.25
106	90.04	8.62	1.35
Average	89.49	9.15	1.35

Table 3.10: Percentage of blocks compressed by the codebook, RCRC, and incompressible blocks inverted counterparts

Image	Codebook	RCRC	Incompressible
101	87.01	11.36	1.63
102	95.41	4.38	0.21
103	82.98	15.55	1.47
104	85.59	12	2.41
105	90.56	8.14	1.3
106	89.28	9.37	1.35
Average	88.47	10.13	1.35

3.2 Discrete-Color Images

In addition to binary images, we tested the proposed scheme on two sets of discrete-color images. The first set consists of a sample of topographic map images comprising four semantic layers that were obtained from the GIS lab at the University of Northern British Columbia [20]. Semantic layers include contour lines, lakes, rivers, and roads, all colored differently. Figure 3.4 illustrates layer separation of a topographic map from our test sample. In this case, the map image has four discrete colors (light blue, dark blue, red, and olive), and thus four layers are extracted. Each discrete color is coupled with the background color (in this case being white) to form the bi-level layers.

The second set of discrete-color images consists of graphs and charts, most of which were generated with spreadsheet software. Graphs and charts consist of discrete colors and are practically limited to no more than 60 colors. Such data are extensively used in business reports, which are in turn published over the web or stored in internal organizational databases. As the size of such reports increases, lossless compression becomes imperative in the sense that it is cost-effective for both storage and transmission.

Table 3.11 provides a summarized description of three map images used in this process. Table 3.12 gives the compression results in bits per pixel (bpp) of the proposed method for the three map images shown in Table 3.11. Formula (2.5) has been used to calculate compression ratios for both the proposed method and JBIG2. From the results we may conclude that the proposed method achieves high compression on the selected set of map images. On average, the proposed method achieves a compression of 0.035 bpp (96.5%) on the selected data sample, whereas JBIG2 yields a compression rate of 0.053 bpp (94.7%). These results are higher than or comparable to those results reported in [21]. We note that while compression rates yielded by the proposed method outperform JBIG2 by an average 2%, JBIG2 has been reported to generally compress at 0.22 to 0.18 bits per pixel [22]. Finally, results for charts and graphs are given in Table 3.13, wherein the proposed method and JBIG2 compress, respectively, at 0.03 bpp and 0.087 bpp. In this case, the proposed method outperforms JBIG2 by 6.24%, on average. The maps, graphs and charts used in these experiments are exhibited in Appendix B.2.

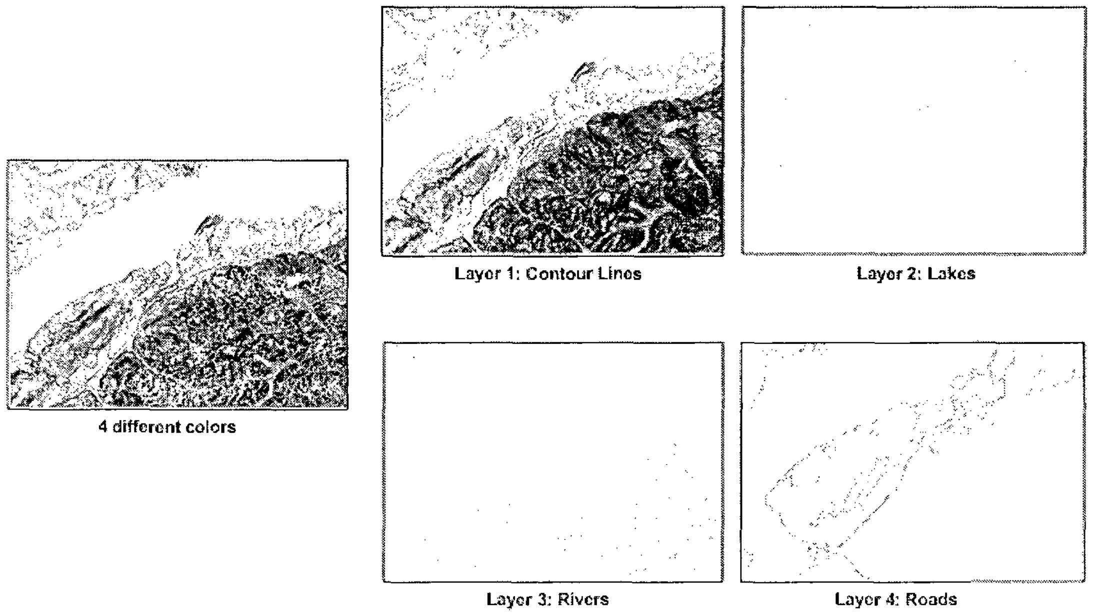


Figure 3.4: Example of color separation: a topographic map of a part of British Columbia containing 4 different colors excluding the white background.

Table 3.11: Description of selected topographic map images, scale 1: 20000 [20]

Map	Dimensions	Size (KB)
1	2200×1700	10960
2	5776×13056	220979
3	5112×11600	173769
	Total Size	406708

Table 3.12: Compression results for map images using the proposed method vs. JBIG2

Map	Compressed Size (KB)	Compression Ratio (bpp)	JBIG2
1	210.77	0.019	0.029
2	7489.27	0.034	0.052
3	6626. 27	0.038	0.055
Total	14326.42	0.035	0.053

Table 3.13: Compression results for charts and graphs using the proposed method vs. JBIG2

Map	Compressed Size (KB)	Compression Ratio (bpp)	JBIG2
1	1281.33	0.014	0.082
2	899.28	0.065	0.063
3	865.97	0.065	0.135
4	863.74	0.045	0.077
5	378.33	0.057	0.143
6	607.67	0.021	0.088
7	590.07	0.021	0.077
8	1039.81	0.018	0.053
9	590.07	0.033	0.099
10	590.07	0.031	0.089
11	773.49	0.033	0.089
12	839.71	0.023	0.079
13	590.07	0.022	0.08
14	590.07	0.032	0.088
15	590.07	0.032	0.087
16	590.07	0.018	0.055
17	590.07	0.028	0.059
18	367.01	0.017	0.126
19	590.07	0.034	0.167
20	590.07	0.07	0.103
21	1050.83	0.017	0.072
22	588.87	0.017	0.065
23	309.45	0.02	0.074
24	607.44	0.022	0.12
Total	16373.62	0.03	0.087

3.3 Discussion

The compression results shown in Table 3.1 reveal that the proposed method outperforms JBIG2 in 92% of the cases. We note that the 100 binary images are mostly solid topological shapes, which intentionally favor the JBIG2 compression algorithm. Moreover, the alternative coding scheme, $\mathcal{C}_A$, modeled in equation (2.25) outperforms the other coding scheme, $\mathcal{C}_0$, modeled in equation (2.23) for all the binary images exhibited in Table B.1 of Appendix B.1. The reason for this is that 98.42% of the blocks are found in the codebook and can be compressed with less overhead bits if the alternative coding scheme is used. This observation is also confirmed by examining the simulation results in Section 2.8 for large values of D, p_w and p_b. It may also be concluded that the probability P should be relatively large (more than 90%) because the portion of incompressible blocks is 0.08%, on average, and the portion RCRC compresses is 99.92% of the blocks not in the codebook.

In the case of the six binary images shown in Table B.2 of Appendix B.1, 89.29% of the blocks were found in the codebook, 9.16% were compressed with RCRC and 1.35% were incompressible. For the inverted counterparts, 88.47% of the blocks were compressed with the codebook, 10.18% with RCRC and 1.35% remained incompressible. We may conclude that RCRC has proved to be an efficient auxiliary coding module. Also, we may surmise as above that the probability parameter P is quite large, thus complying with the simulation results of Section 2.8.

Similar conclusions can be drawn for the coding of discrete-color images.

3.4 Huffman Coding Is Not Dead

The advent of Arithmetic Coding along with the gravity of many results reported in literature over the last two decades has brought about an overshadowing of the power and simplicity of Huffman Coding, but not without principal reasons [1, 8]. Arithmetic codes

- attain compression rates closer to the source entropy than Huffman codes;
- outperform Huffman codes by $p_1 + 0.086$, where p_1 is the probability of the most frequently occurring symbol;[1]
- are adjustable to adaptive models.

These advantages, however, come at a cost. Arithmetic codes do not generally perform better than Huffman codes if incorrect probabilities are fed to the coder. For instance, if the coder encodes according to a probability model $\mathcal{M}$ while the true probabilities are described by model $\mathcal{M}^*$, then Arithmetic coding is expected to perform worse than Huffman coding. A hypothetical example illustrating this observation for a small number of symbols is given in [8]. Consider the following extracts from [13]:

> [G]ottlob Burmann, a German poet who lived from 1737 to 1805, wrote 130 poems, including a total of 20000 words without once using the letter `R`. [...] In 1939, Ernest Vincent Wright published a 267-page novel, *Gadsby*, in which no use is made of the letter `E` (Source: [13], p. 48)

If a probability model for compressing German text is constructed using Burmann's poems as the body of data, then chances are that the incorrect probability of `R` will reduce the efficiency of Arithmetic coding because the occurrence of `R` in other German texts will reveal a conspicuous discrepancy between the theoretical and empirical probabilities. To better comprehend the inefficiency of arithmetic codes resulting from erroneous probabilities, we illustrate the following analysis taken from [8] for the first 10028 words of Wright's novel, *Gadsby*:

> If one uses this novel to estimate the character frequencies in English, the Huffman codeword assigned to `E` would be 14 bits long [...], instead of just 3 bits on regular English text. For arithmetic codes, each `E` would add 15.4 bits.

[1]In [23], it is shown that the redundancy of Huffman codes is bounded from above by $p_1 + 0.086$

In reality, the true model $\mathcal{M}^*$ describing English language requires an average of 4.19 bits per letter for Huffman codes and 4.16 for arithmetic codes. In the case of the erroneous probability model $\mathcal{M}$ described above, the average length of Huffman codewords would increase from 4.19 to 5.46, whereas for arithmetic codes from 4.19 to 5.60. Therefore, one has to carefully choose correct probability models in order to strictly avoid such errors propagating throughout the entire coding procedure.

Empirical results for the solid images exposed in Appendix B.1 show that Huffman coding (via scheme $\mathcal{C}_A$) slightly outperforms Arithmetic coding on average. The rationale for this is that the probability model we constructed for the codebook may have predicted slightly lower or slightly higher relative frequencies for certain 8×8 blocks which appeared with slightly higher or slightly lower probabilities in specific test images. To clarify this point, consider the six binary images with white background (Table 3.7) and the six inverted counterparts (Table 3.8). In the case of white background, Arithmetic coding slightly outperforms Huffman coding per scheme $\mathcal{C}_A$ by 0.044%. This implies that the empirical probability model describing these six images is almost consistent with the theoretical model, probably because of the dominance of white blocks both in the codebook and in the set of the six images. However, when the images are inverted, white blocks are replaced by black blocks, which have a codebook probability equal to 0.263. In this case, Huffman coding outperforms Arithmetic coding by 0.15% (Table 3.8). It may be surmised that the empirical probability model which describes the inverted images is not as particularly consistent with the theoretical probability model that describes the constructed codebook as the probability model which describes the images with white background. Therefore, Arithmetic coding performs less efficiently than Huffman coding, as expected theoretically. The case presented here posits a complex situation involving an alphabet of 2^{64} symbols and a less contiguous body of data, such as is the case of binary images We may, at least in principle, conclude that a more rigorous codebook model needs to be constructed in order to accommodate the strict modeling requirements of Arithmetic coding One way to approach a more correct model would be to enlarge the data sample used to construct the codebook This is, nevertheless, a daunting computational task, as illustrated by the time (500 hours) it took to generate the codebook based on only 120 binary images

In addition to being susceptible to incorrect probabilities, Arithmetic coding is generally slower than Huffman coding in terms of execution time for encoding and decoding [24, 25] Improvements for increasing the speed of Arithmetic coding have brought about sacrifices for coding optimality [8] In this work, we implemented an integer-based arithmetic coder based on the guidelines in [19] For binary images we used for testing, for instance, the overall execution times for scheme C_A and the arithmetic coder were, respectively, 261 and 287 seconds

In terms of optimality, the Huffman codes we constructed for the codebook blocks have an absolute redundancy equal to 0 01, which implies that 0 252% more bits than the entropy circumscribes are required to code blocks in the codebook (see Section 2 3 2) The upper bound for the redundancy is $p_1 + 0\ 086$ in our case $p_1 = 0\ 503$ Thus, we may conclude that the constructed Huffman codes are near-optimal And given the prone-to-incorrect-probabilities facet of Arithmetic coding as well as the high compression results of the proposed method, we conclude that for practical applications the proposed Huffman coding scheme, C_A, should be the preferred compression choice

All in all, the whole purport of the theoretical and empirical observations posited in [1, 8] is that Huffman coding is generally more robust than Arithmetic coding, which can expose emphatic advantage in rare cases The empirical results shown in this work suggest that the proposed codebook model works efficiently with Huffman codes, but models slightly discrepant probabilities for the arithmetic coder to function effectively.

Chapter 4

Related Work

Mankind is not a circle with a single center but an ellipse with two focal points of which facts are one and ideas the other

– VICTOR HUGO

In this chapter, we present the mainstream research pertaining to lossless compression of binary and discrete-color images in the context of block coding

4.1 Binary Image Compression Techniques

Central to the proposed method is the idea of partitioning a binary image or the bi-level layers of discrete-color images into non-overlapping 8×8 blocks Partitioning and encoding binary images into blocks, referred to as *block coding*, has been summarized in [26], wherein images are divided into blocks of totally white (0-valued) pixels and non-white pixels The former are coded by one single bit equal to 0, whereas the latter are coded with a bit value equal to 1 followed by the content of the block in a row-wise order similar to RCRC coding Moreover, the hierarchical variant of the block coding lays on dividing a binary image into $b \times b$ blocks (typically, 16×16), which are then represented in a quad-tree structure In this case, a 0-valued $b \times b$ block is encoded using bit 0, whereas other blocks are coded with bit 1 followed by recursively encoded blocks of pixels with the base case being one single pixel. In [18, 27], it is suggested that block coding can be improved by resorting to Huffman coding or by employing context-based models within larger blocks.

A generalized approach to block coding is illustrated in [28], wherein it is argued that such a method achieves near-optimal encoding of sparse binary images, especially when source statistics are not available.

In [29], a hybrid compression method based on hierarchical blocks coding is proposed. Here, predictive modeling has been employed to construct an error image as a result of the difference between the predicted and original pixel values. Then, the error image is compressed using Huffman coding of bit patterns at the lowest hierarchical level. This work builds upon the ideas presented in [30, 31], wherein block coding with Arithmetic coding has been employed.

Closely related to the idea of (non-)overlapping blocks is rectangular partitioning, wherein 1-valued (black) regions in the input binary image are partitioned into rectangles [32]. In this case, the top-left and bottom-right coordinates of a given rectangle are encoded, whereas a different code is used for isolated pixels.

4.1.1 JBIG2

JBIG2, the successor of JBIG1, is a platform-independent lossless and lossy coding standard primarily designed for compressing bi-level images, but is also capable of encoding layers of multiple-bit pixels, such as halftone images [9, 33]. The underlying method is based on adaptive coding, in which case current information about an image pixel is adapted contextually to preceding pixels. In light of that, JBIG2 uses adaptive arithmetic coding to predict future pixel codes based on previously encountered pixel data.

JBIG2 operates by segmenting an input image into regions, such as text and images, and encodes each region using different methods embodied in the standard If X is the current pixel to be predicted, than JBIG2 resorts to a set of adjacent pixels, referred to as the context, to code X The context includes adaptive pixels as well All in all, it has been observed that JBIG2 compresses at rates higher than other known standards or generic methods

4.2 Discrete-Color Image Compression Techniques

In the context of discrete-color images, lossless compression methods are generally classified into two categories (i) methods applied directly on the image, such as the graphics interchange format (GIF), the portable network graphics (PNG), or lossless JPEG (JPEG-LS), (ii) and methods applied on every layer extracted (or separated) from the image, such as TIFF-G4 and JBIG In this work, we focused on the second category Previous work in literature amounts to several lossless compression methods for map images based on layer separation The standard JBIG2, which is specifically designed to compress bi-level data, employs context-based modeling along with Arithmetic coding to compress binary layers In [21], a lossless compression technique based on semantic binary layers is proposed Each binary layer is compressed using context-based statistical modeling and arithmetic coding, which is slightly different from the standard JBIG2 In [34], a method that utilizes interlayer correlation between color separated layers is proposed Context-based modeling and arithmetic coding are used to compress each layer An extension of this method applied on layers separated into bit planes is given in [35]

Chapter 5

Conclusions and Future Work

I was born not knowing and have had only a little time to change that here and there.

– RICHARD FEYNMAN

5.1 Concluding Remarks

This thesis exposed the details of a novel lossless compression method for binary and discrete-color images. The core of the method lies in generic block coding and operates per the empirical distribution of the most, but not all, frequently occurring 8×8 blocks. Asymptotic analyses suggest that the error incurred from trimming the codebook to a particular number of blocks is small to negligible. The distribution of blocks was employed to construct Huffman and Arithmetic codes. The latter coding algorithms led to the development of two coding schemes for the proposed method. To attain higher compression, an additional coding helper module–the row-column reduction coding–was introduced. The proposed method was tested on various binary and discrete-color images. Results were compared to JBIG2, the standard coder for bi-level layers. Empirical results suggest that the proposed method outperforms JBIG2 compression rates in most, if not all, of the cases. The proposed method is efficient in terms of the memory required for the codebook as well as the analytical and empirical complexities.

5.2 Future Work

In Section 2.10, we illustrated protagonist and antagonist blocks with respect to the proposed lossless compression method. However, there exist seemingly protagonist blocks with basic geometric constructs which do not appear in the codebook and are, therefore, considered antagonists. For example, not all 8×8 blocks containing 63 zeros and a 1 are in the codebook. In such cases RCRC will certainly compress at a maximum rate equal to 73.4%. Moreover, such cases instigate premises to extend the proposed lossless compression method into a lossy method. For example, substituting the 1 with 0 in the case discussed here yields a white block, which can then be efficiently compressed via the codebook using only 1 bit. Extending the proposed method to lossy coding is a plausible future area of research.

In light of the latter extension, the proposed method can be employed for interactively reconstructing broken regions in binary images. If certain pixels are missing in some input image, then the region comprising the missing bits is referred to as a broken region. If one considers the missing bits as "don't care" bits, then the RCRC algorithm can be modified to accept such bits to determine the best 8×8 block that would reconstruct a particular portion of the broken region. In addition, if the RCRC-decoded block still contains 'don't care' bits, the codebook model can be used to search for the best match of the block. Notice that the best codebook match, if it exists, has the highest probability as the codebook entries are sorted in that fashion. Hence, it may be surmised that chances are that the reconstructed block for the particular portion of the missing region is the most probable block that exists to fill that region. This process may reconstruct blocks which are not necessarily suitable for the missing region. The latter observation brings about the interactive facet of the reconstruction, in which case a user can accept or reject a suggested reconstruction, or can modify the preconditions per the perception of how a fully reconstructed binary image would look like.

In addition, with a slight modification to the row-column reduction coding algorithm, the proposed method can be applied to compress Very-Large-Scale Integration (VLSI) circuitry test data. VLSI data consists of 0-1 matrices as well as "don't care" bits. If we view don't care bits as "wild card" bits, then the codebook may be search to find the match with the shortest code. On the other hand, RCRC may be modified to deal with "don't care bits" based on the following observation. Three row (or column) vectors, $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$ do not satisfy the Euclidean relation, i.e. $\mathbf{v}_1 R \mathbf{v}_2 \wedge \mathbf{v}_1 R \mathbf{v}_3 \nrightarrow \mathbf{v}_2 R \mathbf{v}_3$. Thus, "don't care" bits should be replaced with care bits in a way that maximizes the number of eliminated rows (or columns).

Published Material

Portions of Chapters 2 and 3 appeared in conference proceedings [36] and [37].

Bibliography

[1] D Salomon, *Variable-Length Codes for Data Compression* Springer, 2007

[2] K Sayood, *Introduction to Data Compression* San Francisco, CA, USA Morgan Kaufmann Publishers Inc , 3 ed , 2005

[3] A Moffat, T C Bell, and I H Witten, "Lossless Compression for Text and Images," *International Journal of High Speed Electronics and Systems*, vol 8, no 1, pp 179–231, 1997

[4] C E Shannon, "A Mathematical Theory of Communication," *The Bell Systems Technical Journal*, vol 27, pp 379–423, 1948

[5] R M Gray, *Entropy and Information Theory* Springer, 1990

[6] R M Gray, *Probability, Random Processes, and Ergodic Properties* Springer, 2 ed , 2009

[7] K J Balakrishnan and N A Touba, 'Relationship Between Entropy and Test Data Compression," *IEEE Transactions on Computer-Aided Design*, vol 23, no 4, pp 386–395, 2007

[8] A Bookstein and S T Klein, 'Is Huffman Coding Dead?,' *Computing*, vol 50 no 4, pp 279–296, 1993

[9] P G Howard, F Kossentini, B Martins, S Forchhammer, S -R Forchhammer, W J Rucklidge, and F Ono, The Emerging JBIG2 Standard,' *IEEE Trans Circuits and Systems for Video Technology*, vol 8, pp 838–848, 1998

[10] S B Gray, "Local Properties of Binary Images in Two Dimensions,' *IEEE Transactions on Computers*, vol C-20, no 5, pp 551–561, 1971

[11] M Buro, ' On the Maximum Length of Huffman Codes,' *Information Processing Letters*, vol 45, pp 219–223, 1993

[12] G Hansel, D Perrin, and I Simon, Compression and Entropy," in *STACS '92 Proceedings of the 9th Annual Symposium on Theoretical Aspects of Computer Science*, (London, UK) pp 515–528, Springer-Verlag, 1992

[13] J. R. Pierce, *An Introduction to Information Theory: Symbols, Signals and Noise.* Dover Publications, Inc., 2 ed., 1980.

[14] W. Feller, *An Introduction to Probability Theory and Its Applications*, vol. 1. Wiley, 3 ed., 1968.

[15] P. G. Howard and J. S. Vitter, "Fast and Efficient Lossless Image Compression," in *Data Compression Conference*, pp. 351–360, IEEE, 1993.

[16] A. Moffat, J. A. Storer, and J. H. Reif, "Two-Level Context Based Compression of Binary Images," in *Data Compression Conference*, pp. 382–391, IEEE, 1991.

[17] J. Zobel and A. Moffat, "Adding Compression to a Full-Text Retrieval System," *Software – Practice and Experience*, vol. 25, no. 8, pp. 891–903, 1995.

[18] J. L. Mitchell, "Facsimile Image Coding," in *Conf. Proc. National Computer*, (Anaheim, CA, USA), pp. 423–426, 1980.

[19] P. G. Howard and J. S. Vitter, "Practical Implementations of Arithmetic Coding," in *Image and Text Compression* (J. A. Storer, ed.), pp. 85–112, Kluwer Academic Publishers, 1992.

[20] University of Northern British Columbia, Geographic Information Systems Lab, "Topographic Map of British Columbia," 2009.

[21] P. Fränti, E. Ageenko, P. Kopylov, S. Gröhn, and F. Berger, "Map Image Compression for Real-Time Applications," in *Proc. Spatial Data Handling 2002 Symposium SDH 2002 (part of 2002 Joint International Symposium on Geospatial Theory, Processing and Applications*, 2002.

[22] P. Fränti, P. Kopylov, and E. Ageenko, "Evaluation of Compression Methods for Digital Map Images," in *Proc. Automation, Control, and Information Technology - 2002* (M. H. Hamza, O. I. Potaturkin, and Y. I. Shokin, eds.), (Novosibirsk, Russia), June 10–13 2002.

[23] R. G. Gallagher, "Variations on a Theme by Huffman," *IEEE Trans. on Information Theory*, vol. 24, no. 6, pp. 668–674, 1978.

[24] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic Coding for Data Compression," *Communications of the ACM*, vol. 30, pp. 520–540, 1987.

[25] A. Moffat and J. Zobel, "Coding for Compression in Full-Text Retrieval Systems," in *Proceedings of the Data Compression Conference DCC'92*, 1992.

[26] M. Kunt and O. Johnsen, "Block Coding of Graphics: A Tutorial Review," *Proceedings of the IEEE*, vol. 68, no. 7, pp. 770–786, 1980.

[27] X. Wu and N. Memon, "Context-Based, Adaptive, Lossless Image Coding," *IEEE Trans. Communications*, vol. 45, no. 4, pp. 437–444, 1997.

[28] B Y Kavalerchik, "Generalized Block Coding of Black and White Images," *IEEE Trans Image Processing*, vol 1, pp 518–520, 1992

[29] P Franti and O Nevalainen, "Compression of Binary Images by Composite Methods Based on Block Coding," *Journal of Visual Communication and Image Representation*, vol 6, no 4, pp 366–377, 1995

[30] P Franti and O Nevalainen, "A Two-Stage Modeling Method for Compressing Binary Images by Arithmetic Coding," *The Computer Journal*, vol 36, no 7, pp 615–622, 1993

[31] P Franti, "A Fast and Efficient Compression Method for Binary Images," *Signal Processing Image Communication (Elsevier Science)*, vol 6, pp 69–76, 1994

[32] A Quddus and M M Fahmy, "Binary Text Image Compression Using Overlapping Rectangular Partitioning," *Pattern Recognition Letters*, vol 20, no 1, pp 81–88, 1999

[33] ISO/IEC JTC1/SC29 JBIG Committee, "JBIG2 Working Draft WD14492," Tech Rep ISO/IEC JTC1/SC29, International Organization for Standardization, August 1998

[34] P Kopylov and P Franti, "Compression of Map Images by Multilayer Context Tree Modeling," *IEEE Trans on Image Processing*, vol 14, no 1, pp 1 11, 2005

[35] A Podlasov and P Franti, "Lossless Image Compression via Bit-Plane Separation and Multilayer Context Tree Modeling," *Journal of Electronic Imaging*, vol 15, no 4, p 043009, 2006

[36] S Zahir and A Borici, 'A Fast Lossless Compression Scheme for Digital Map Images Using Color Separation," in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '10)*, (Dallas, Texas, USA), pp 1318–1321, IEEE Computer Society, 14 19 March 2010

[37] S alZahir and A Borici, "Lossless Compression of Maps, Charts, and Graphs via Color Separation," in *Proceedings of the Data Compression Conference (DCC '10)*, (Snowbird, Utah, USA) p 518, IEEE Computer Society, 24 26 March 2010

[38] I Adler, S Oren, and S M Ross, "The Coupon-Collector's Problem Revisited,' *Journal of Applied Probability*, vol 40, pp 513 518, 2003

[39] E Ageenko, P Kopylov, and P Franti, "On The Size and Shape of Multi-Level Context Templates for Compression of Map Images,' in *Proceedings International Conference on Image Processing*, vol 3, (Thessaloniki, Greece), pp 458 461, 2001

[40] A. Akimov, A. Kolesnikov, and P. Fränti, "Lossless Compression of Map Contours by Context Tree Modeling of Chain Codes," *Pattern Recognition*, vol. 40, pp. 944–952, 2007.

[41] S. Alcaraz-Corona, R. A. Neri-Calderón, and R. M. Rodríguez-Dagnino, "Efficient Bilevel Image Compression by Grouping Symbols of Chain Coding Techniques," *Optical Engineering*, vol. 48, no. 3, p. 037001, 2009.

[42] R. Arnold and T. Bell, "A Corpus for the Evaluation of Lossless Compression Algorithms," in *DCC '97: Proceedings of the Conference on Data Compression*, p. 201, IEEE Computer Society, 1997.

[43] A. L. Berger, S. A. Della Pietra, and V. J. Della Pietra, "A Maximum Entropy Approach to Natural Language Processing," *Computational Linguistics*, vol. 22, no. 1, pp. 39–71, 1996.

[44] A. Bookstein and S. T. Klein, "Models of Bitmap Generation," *Information Processing & Management*, vol. 28, pp. 795–806, 1992.

[45] P. A. Bromiley, N. Thacker, and E. Bouhova-Thacker, "Shannon Entropy, Renyi Entropy, and Information," Internal Memo 2004-004, School of Cancer and Imaging Sciences, The University of Manchester, U.K., 2004.

[46] T. M. Cover and J. A. Thomas, *Elements of Information Theory.* Wiley Series in Telecommunications and Signal Processing, Wiley-Interscience, 2 ed., 2006.

[47] A. H. El-Maleh, E. Khan, and S. alZahir, "A Geometric-Primitives-Based Compression Scheme for Testing Systems-on-a-Chip," in *VTS '01: Proceedings of the 19th IEEE VLSI Test Symposium*, (Marina del Rey, CA, USA), p. 54, IEEE Computer Society, 2001.

[48] R. Estrada and R. P. Kanwal, *Asymptotic Analysis: A Distributional Approach.* Boston, MA: Birkhäuser, 1994.

[49] S. T. Klein and D. Shapira, "Huffman Coding with Non-sorted Frequencies," in *DCC '08: Proceedings of the Data Compression Conference*, p. 526, IEEE Computer Society, 2008.

[50] H. S. Malvar, "Fast Adaptive Encoder for Bi-Level Images," in *DCC '01: Proceedings of the Data Compression Conference*, p. 253, IEEE Computer Society, 2001.

[51] P. D. Miller, *Applied Asymptotic Analysis*, vol. 75 of *Graduate Studies in Mathematics.* American Mathematical Society, 2006.

[52] S. Zahir and R. Chowdhury, "A Hybrid 2-3-3 Bits-Based Image Encoding Scheme," in *IEEE International Symposium on Signal Processing and Information Technology (ISSPIT '06)*, (Vancouver, BC, Canada), pp. 781 786, IEEE Computer Society, 27–30 August 2006.

[53] L. Zhou and S. Zahir, "A New Efficient Algorithm for Lossless Binary Image Compression," in *IEEE Canadian Conference on Electrical and Computer Engineering CCECE'06*, pp. 1427–1431, 2006.

Appendix A

Models and Derivations

Everyone engaged in research must have had the experience of working with feverish and prolonged intensity to write a paper which no one else will read or to solve a problem which no one else thinks important and which will bring no conceivable reward—which may only confirm a general opinion that the researcher is wasting his time on irrelevancies.

– Noam Chomsky

A.1 Waiting Probabilities

Let $\mathcal{S}$ denote the set of coupons (or items, in general) having cardinality $||\mathcal{S}|| = N$. Coupons are collected with replacement from $\mathcal{S}$. Then, the problem poses the following two questions:

(i) What is the probability of waiting more than n trials in order to observe all N coupons?

(ii) What is the expected number of such trials?

Let $\mathcal{M}$ denote the set of observed coupons and T be a random variable. More accurately, $\mathcal{M}$ is a *multiset* of coupons because coupons are drawn from $\mathcal{S}$ with replacement. To answer the first question, it is more convenient to consider the probability of collecting more than n coupons, i.e. $P(T > n)$. The required probability

$P(T = n)$ is easily derived as $P(T = n) = P(T > n-1) - P(T > n)$ The following model gives $P(T > n)$ [14]

$$P(T > n) = \sum_{i=1}^{N-1} (-1)^{i+1} \binom{N}{i} \left(\frac{N-i}{N}\right)^n \qquad \text{(A 1)}$$

From formula (A 1), we have

$$P(T = n) = P(T > n-1) - P(T > n) \qquad \text{(A 2)}$$

Formula (A 2) gives the probability of waiting for n samples before observing N coupons and that answers the first question posed above

To determine the expected number of trials, $E[T]$ required to collect n coupons we use the following model

$$E[T] = nH_n \ , \qquad \text{(A 3)}$$

where H_n is the harmonic number $H_n = \sum_{i=1}^{n} \frac{1}{i}$ Based on the asymptotic expansion of harmonic numbers, one may derive the following asymptotic approximation for the expectation given in (A 3)

$$E[T] = n \ln n + \gamma n + \frac{1}{2} + o(1) \ , \quad \text{as} \quad n \to \infty \ , \qquad \text{(A 4)}$$

where $\gamma \approx 0\ 5772$ is the Euler-Mascheroni constant If we have $n = 2^{64}$ coupons, then the expected number of trials bounded by (A 4) is equal to $8\ 29 \times 10^{20}$, which implies a practically unattainable number of trials

Since T takes nonnegative values, we can employ formula (A 3) to bound the probability given in (A 1) using Markov's Inequality for $a > 0$

$$P(T \geq a) \leq \frac{E[T]}{a} \qquad \text{(A 5)}$$

See [14, 38] for more details on the Coupon-Collector's Problem.

A.2 Asymptotic Expansion on the Entropy Discrepancy

A.2.1 General Asymptotic Analysis

Let N be the number of 8×8 blocks and p_i, $\forall i = 1, 2, \ldots, N$ be the theoretical probability distribution of the N blocks.

Define the discrete function L: $L(q_i)$ to be the observed average code length given by:

$$L = \sum_{i=1}^{N} q_i \log_2 \frac{1}{q_i} \ , \tag{A.6}$$

where q_i denotes the empirical probability distribution of the N 8×8 blocks. Similarly, define H: $H(p_i)$ to be the theoretical average code length function for the theoretical probabilities p_i:

$$H = \sum_{i=1}^{N} p_i \log_2 \frac{1}{p_i} \ . \tag{A.7}$$

Function H is the entropy of the theoretical distribution p_i. We want to asymptotically study the error between the observed and the theoretical entropies given respectively by (A.6) and (A.7). For that purpose, we examine the absolute error model:

$$E = L - H = \sum_{i=1}^{N} \left(q_i \log_2 \frac{1}{q_i} - p_i \log_2 \frac{1}{p_i} \right) \ . \tag{A.8}$$

Define the discrepancy ϵ_i, $\forall i = 1, 2, \ldots, N$, between the empirical and theoretical probabilities as:

$$\epsilon_i = q_i - p_i \ . \tag{A.9}$$

Then, we derive an asymptotic expansion on ϵ_i in order to observe how E in formula (A.8) behaves asymptotically.

First, observe that based on (A.9), $L(q_i) = H(p_i + \epsilon_i)$ and equation (A.8) may be expressed as $E = H(p_i + \epsilon_i) - H(p_i)$, $\forall i = 1, 2, \ldots, N$. The first derivative of function $H(p_i) = -p_i \log_2 p_i$ is:

$$H'(p_i) = -\frac{1}{\ln 2} - log_2 p_i \ . \tag{A.10}$$

Then, we look for an expression such as the following:

$$[H(p_i + \epsilon_i) - H(p_i)] \sim H'(p_i)\epsilon_i \quad \text{as} \quad \epsilon_i \rightarrow 0 \ , \tag{A.11}$$

where the left-hand side of the relation represents the error function E given in (A.8). Note that this expression is the discrete version of the first-order Taylor series defined as follows:

Definition A.1. ***Taylor Series***

For a function $f(x)$ defined in a set $\mathbb{D}$ the corresponding Taylor Series (or Taylor Expansion) of the function about a point $x_0 \in \mathbb{D}$ is given by the following formula:

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n \ , \tag{A.12}$$

where $f^{(n)}(x_0)$ is the n^{th} derivative of function f at $x = x_0$.

If we let $f \equiv H$, $x = p_i + \epsilon_i$, and $x_0 = p_i$ in (A.12), we get the following expression:

$$H(p_i + \epsilon_i) = \sum_{i=1}^{N} \left[\sum_{n=0}^{\infty} \frac{H^{(n)}(p_i)}{n!} \epsilon_i^n \right] \ . \tag{A.13}$$

The dominant term in the series (A.13) is function $H(p_i)$. Subtracting this from

$H(p_i + \epsilon_i)$ yields the following series:

$$H(p_i + \epsilon_i) - H(p_i) = \sum_{i=1}^{N} \left[\sum_{n=1}^{\infty} \frac{H^{(n)}(p_i)}{n!} \epsilon_i^n \right] . \tag{A.14}$$

The left-hand side of equation (A.14) is the error function given in (A.8). Hence, the Taylor series serves as a suitable approximation to the absolute error function E. That is,

$$E \sim \sum_{i=1}^{N} \sum_{n=1}^{\infty} \frac{H^{(n)}(p_i)}{n!} \epsilon_i^n \quad \text{as} \quad \epsilon_i \to 0 . \tag{A.15}$$

We may write equation (A.15) as follows:

$$E = \sum_{i=1}^{N} \left[\sum_{n=1}^{\infty} \frac{H^{(n)}(p_i)}{n!} \epsilon_i^n + o(\epsilon_i^n) \right] \quad \text{as} \quad \epsilon_i \to 0 . \tag{A.16}$$

For most purposes, a second-order approximation is appropriate to provide useful insight into errors. That is, if we consider the first and second derivatives of function H, we may write (A.16) as follows:

$$E = \sum_{i=1}^{N} \left[-\epsilon_i \left(\frac{1}{\ln 2} + \log_2 p_i \right) - \frac{1}{2 \ln 2} \frac{\epsilon_i^2}{p_i} + o(\epsilon_i^2) \right] \quad \text{as} \quad \epsilon_i \to 0 . \tag{A.17}$$

Rearranging the terms in the summation and by the properties of asymptotic estimates, we write (A.17) in the following form:

$$E = -\sum_{i=1}^{N} \left[\frac{1}{\ln 2} \left(\epsilon_i + \frac{\epsilon_i^2}{2p_i} \right) + \epsilon_i \log_2 p_i \right] + o \left(\max_{i \in \{1, \dots, N\}} \{\epsilon_i^2\} \right) \quad \text{as} \quad \epsilon_i \to 0 . \tag{A.18}$$

See [51] for a rigorous treatment of asymptotic analysis.

A.2.2 Error Analysis for the Constructed Codebook

The asymptotic expansion exposed in Section A.2.1 applies to the empirical and theoretical average code lengths given, respectively, in formulas (A.6) and (A.7), wherein the summation bound is equal to the number of blocks, N. However, as stated in Section 2.3.2, the cardinality of the constructed codebook is equal to 6952 entries. Therefore, the probability terms in formula (A.6) are summed up to 6952 and the sum is thereafter considered to equal zero. This affects the discrepancy in formula (A.8) in that an additional error term equal to $-\sum_{i=6953}^{N} p_i \log_2 p_i$ is to be added. Since the theoretical probabilities p_i are fixed, this additional error term may be added to the model in formula (A.18). Here, we provide a more elaborate discussion on the additional error incurred on the constructed codebook of 6952 entries. We first consider the theoretical implication of the Principle of Maximum Entropy when applied on unknown distributions. Then, we establish a bound on the additional error term.

Let N denote the total number of blocks and M the number of blocks included in the constructed codebook, $M < N$. The Principle of Maximum Entropy (PME) instructs one to assume a uniform probability distribution over all symbols that have not been observed in a given data sample, but which are part of some alphabet [43]. In our case, we consider $q_i = 0$, for $i = M+1, M+2, \ldots, N$. Based on PME, we should consider smoothing the empirical probability distribution and consider all unobserved empirical probabilities as uniformly distributed. That is, $q_i = q^*$, for $i = M+1, M+2, \ldots, N$ and for some fixed probability value $q^* > 0$. In order to achieve a uniform distribution, we need to smoothen the observed probability values $q_i = 0$, for $i = 1, 2, \ldots, M$. Because the number N is very large, any smoothing method cannot be practically applied as it would disrupt information about observed blocks. For example, the probability of 1 valued 8×8 blocks in the constructed codebook is equal to 26%. A smoothing method, such as Laplacian smoothing, would replace this observed probability by a very small value. Then, the constructed Huffman code would not realistically represent the observed distribution of the 1-valued 8×8 block. For the purpose of determining a bound, however, it is theoretically possible to consider the implications of applying PME on the models described above.

The error model in formula (A.8) may be rewritten as follows:

$$\begin{aligned} E &= \sum_{i=1}^{M} \left(q_i \log_2 \frac{1}{q_i} - p_i \log_2 \frac{1}{p_i} \right) && \text{(A.19)} \\ &+ \sum_{i=M+1}^{N} \left(0 - p_i \log_2 \frac{1}{p_i} \right) . && \text{(A.20)} \end{aligned}$$

The asymptotic approximation we derived in formula (A.18) applies to formula (A.19). We focus on determining a bound for formula (A.20), denoted hereafter as E_2.

Based on the Principle of Maximum Entropy, the following inequality holds:

$$\sum_{i=M+1}^{N} \left(p_i \log_2 \frac{1}{p_i} \right) \leq \sum_{i=M+1}^{N} \left(p^* \log_2 \frac{1}{p^*} \right) = (N - M) p^* \log_2 \frac{1}{p^*} , \tag{A.21}$$

where p^* is some uniform probability value. Multiplying both sides of inequality (A.21) by -1, we have:

$$E_2 = - \sum_{i=M+1}^{N} \left(p_i \log_2 \frac{1}{p_i} \right) \geq -(N - M) p^* \log_2 \frac{1}{p^*} . \tag{A.22}$$

In other words, $E_2 \geq -(N-M)p^* \log_2 \frac{1}{p^*}$. Inequality (A.22) establishes a lower bound for the additional error term E_2.

Now, suppose that the empirical probabilities q_i, $i = M+1, \ldots, N$, are uniformly distributed with a probability value q^*. In this case, the empirical entropy value is maximized and we would have

$$E_2 < (N-M)q^* \log_2 \frac{1}{q^*} - \sum_{i=M+1}^{N} \left(p_i \log_2 \frac{1}{p_i}\right) \tag{A 23}$$

If we assume that $p_i = p^*$, $i = M+1, \ldots, N$, then the following inequality holds

$$\begin{aligned} E_2 &\leq (N-M)q^* \log_2 \frac{1}{q^*} - (N-M)p^* \log_2 \frac{1}{p^*} \\ &< (N-M)q^* \log_2 \frac{1}{q^*} - \sum_{i=M+1}^{N} \left(p_i \log_2 \frac{1}{p_i}\right) \end{aligned} \tag{A 24}$$

because $p^* \log_2 \frac{1}{p^*} > \sum_{i=M+1}^{N} \left(p_i \log_2 \frac{1}{p_i}\right)$

Inequality (A 24) establishes an upper bound for the additional error term E_2 Based on inequalities (A 22) and (A 24), E_2 is bounded by below and above as follows

$$-(N-M)p^* \log_2 \frac{1}{p^*} \leq E_2 \leq (N-M)\left[q^* \log_2 \frac{1}{q^*} - p^* \log_2 \frac{1}{p^*}\right] \tag{A 25}$$

Let $\epsilon = q^* - p^*$ Then, we can provide a second-order asymptotic expansion on ϵ in order to approximate the upper bound of E_2 in inequality (A 25) Using a simplified version of the asymptotic expansion given in formula (A 18), we have the following

$$\begin{aligned} \varphi(\epsilon) &= q^* \log_2 \frac{1}{q^*} - p^* \log_2 \frac{1}{p^*} \\ &= -\frac{1}{\ln 2}\left(\epsilon + \frac{\epsilon^2}{2p^*}\right) - \epsilon \log_2 p^* + o\left(\epsilon^2\right) \quad \text{as} \quad \epsilon \to 0 \end{aligned} \tag{A 26}$$

For $\epsilon \to 0$, $\varphi(\epsilon)$ is negligible Inequality (A 25) may now be written as

$$-(N-M)p^* \log_2 \frac{1}{p^*} \leq E_2 \leq (N-M)\varphi(\epsilon) \tag{A 27}$$

The empirical probabilities q_i, $i = 1, 2, \ldots, M$, do not follow a uniform distribution, as noted in Section 2 3 2 The asymptotic approximation in formula (A 18) suggests that the theoretical probabilities, too, are not uniformly distributed in the strict sense. Suppose that, for the purpose of finding a more reasonable lower bound, we wish to smoothen the probabilities p_i, $i = M+1, \ldots, N$, with the caveat that the probability values for $i = 1, 2, \ldots, M$ are not modified. Suppose that, after smoothing, the resulting probability value is p^*. Then, the following inequality holds:[1]

$$\frac{1}{N+\delta} \leq p^* < \frac{1}{N-M} , \tag{A.28}$$

for $|\delta| < M$. We focus on the left-hand side of inequality (A.28):

$$p^* \geq \frac{1}{N+\delta} . \tag{A.29}$$

Taking the logarithm base 2 of both sides in (A.29), we have:

$$\log_2 p^* \geq \log_2 \frac{1}{N+\delta} . \tag{A.30}$$

Multiplying both sides in (A.30) by $-p^*$, we have:

$$\begin{aligned} &\left(-p^* \log_2 p^* \leq -p^* \log_2 \frac{1}{N+\delta}\right) \\ \iff &\left(p^* \log_2 \frac{1}{p^*} \leq p^* \log_2 (N+\delta)\right) . \end{aligned} \tag{A.31}$$

Multiplying both sides in (A.31) by $-(N-M)$, we have:

$$-(N-M)p^* \log_2 \frac{1}{p^*} \geq -(N-M)p^* \log_2 (N+\delta) . \tag{A.32}$$

[1]To see why this is the case (for theoretical purposes), let $S = \sum_{i=1}^{M} p_i$. Then, a uniform probability value p^* for probabilities p_i, $i = M+1$ to $i = N$, could be an average value $p^* = \frac{1}{N-M} \sum_{i=M+1}^{N} p_i$. This value may be rewritten as $p^* = \frac{1-S}{N-M}$, which is less than $\frac{1}{N-M}$.

Multiplying both sides of inequality (A 29) by $-(N - M)\log_2(N + \delta)$ gives

$$-\frac{N - M}{N + \delta}\log_2(N + \delta) \geq -(N - M)p^* \log_2(N + \delta) \tag{A 33}$$

For $|\delta| < M$, the following holds

$$-\frac{N - M}{N + \delta}\log_2(N + \delta) > -\log_2(N + \delta) \tag{A 34}$$

From the right-hand side of inequality (A 28), we have $(N - M)p^* < 1$ Multiplying both sides by $-\log_2(N + \delta)$ gives

$$-(N - M)p^* \log_2(N + \delta) > -\log_2(N + \delta) \tag{A 35}$$

Taking the logarithm to base 2 of the terms in inequality (A 29) and multiplying both sides by $(N - M)p^*$ yields the following

$$-(N - M)p^* \log_2 \frac{1}{p^*} \geq -(N - M)p^* \log_2(N + \delta) \tag{A 36}$$

From formulas (A 32) and (A 35), we have

$$-(N - M)p^* \log_2 \frac{1}{p^*} > -\log_2(N + \delta) \tag{A 37}$$

From inequalities (A 27) and (A 37), we may bound error E_2, given in formula (A 20), as follows

$$-\log_2(N + \delta) < E_2 \leq (N - M)\varphi(\epsilon) \; , \tag{A 38}$$

where $\varphi(\epsilon)$ is given in formula (A 26)

In the case of 8×8 blocks, the bounds in formula (A 38) suggest that the additional error incurred on the constructed codebook is negligible For the lower bound, consider the worst case when δ is very close to $M = 6952$ and let $N = 2^{64}$. Then, $\log_2(N + \delta) \approx 64$ bits. The upper bound, on the other hand, tends to zero as $\epsilon \to 0$.

Appendix B

Test Images

What most experimenters take for granted before they begin their experiments is infinitely more interesting than any results to which their experiments lead.

– NORBERT WIENER

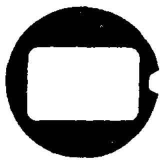
B.1 Binary Images

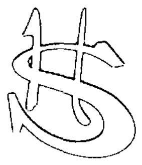
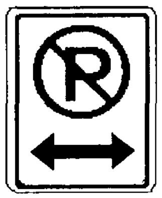
Here, we exhibit over 100 binary images (source [41]) employed for compression via the proposed method and JBIG2. Displayed underneath each image are the dimensions and the compression results. Overall, the proposed method outperforms JBIG2 by 5.33% with arithmetic coding and 5.45% with $\mathcal{C}_A$. In addition, $\mathcal{C}_A$ outperforms $\mathcal{C}_0$ in all images. It can be noticed that Huffman and Arithmetic coding yield close compression ratios.

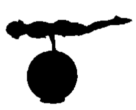
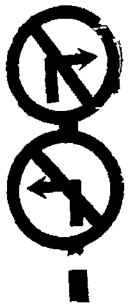
Table B.1: Solid test images

001	**002**	**003**	**004**	**005**
274 × 208	1006 × 669	900 × 899	512 × 800	1024 × 768
E_p 84 2	E_p 93 11	E_p 91 23	E_p 93 25	E_p 88 35
E_p^* 88 66	E_p^* 96 78	E_p^* 95 43	E_p^* 95 99	E_p^* 92 89
AC 90 06	AC 96 6	AC 95 31	AC 95 73	AC 93 92
JBIG2 81 18	JBIG2 95 46	JBIG2 94 27	JBIG2 94 69	JBIG2 89 06
006	**007**	**008**	**009**	**010**
2126 × 1535	640 × 492	2400 × 3000	1061 × 1049	2329 × 854
E_p 89 97	E_p 93 29	E_p 92 56	E_p 87 9	E_p 93 47
E_p^* 96 23	E_p^* 96 56	E_p^* 97 59	E_p^* 92 31	E_p^* 96 68
AC 95 75	AC 96 41	AC 96 98	AC 94 05	AC 96 86
JBIG2 95 78	JBIG2 95 32	JBIG2 98 34	JBIG2 88 44	JBIG2 94 83
011	**012**	**013**	**014**	**015**
800 × 524	575 × 426	535 × 518	498 × 395	1986 × 1500
E_p 91 77	E_p 90 85	E_p 89 57	E_p 88 4	E_p 91 32
E_p^* 95 47	E_p^* 94 45	E_p^* 94 85	E_p^* 92 74	E_p^* 96 5
AC 94 96	AC 95 29	AC 94 51	AC 93 69	AC 96 12
JBIG2 94 3	JBIG2 92 82	JBIG2 92 84	JBIG2 87 58	JBIG2 95 37

Table B 1 (*continued*)

	016	**017**	**018**	**019**	**020**
					
	400 × 400	542 × 493	483 × 464	791 × 663	448 × 444
E_p	77 45	87 27	89 67	91 68	89 12
E_p^*	81 9	93 27	93 73	94 84	94 63
AC	86 17	93 98	94 78	95 3	94 54
JBIG2	74	90 8	89 8	91 81	92 29

	021	**022**	**023**	**024**	**025**
					
	360 × 441	315 × 394	241 × 490	542 × 564	1500 × 845
E_p	88.49	84 38	86 9	94 4	93 28
E_p^*	90 38	89 66	89 44	97 31	96 88
AC	90 98	93 32	91 03	97 19	96 78
JBIG2	85 94	91 94	84 36	97 01	95 92

	026	**027**	**028**	**029**	**030**
					
	3000 × 2400	3000 × 2150	2400 × 1920	196 × 390	167 × 252
E_p	94 1	94 14	94 37	84 85	87 28
E_p^*	97 5	97 26	97 73	89 17	92 79
AC	97 2	96 92	97 47	91 61	93 69
JBIG2	97 14	94 96	98	81 59	86 67

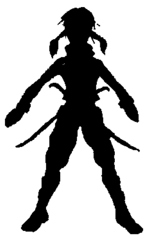
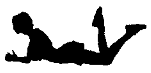
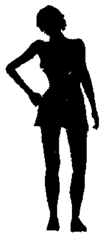
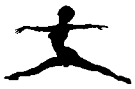
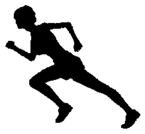
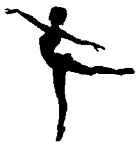
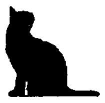
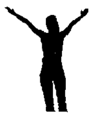
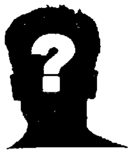
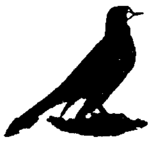
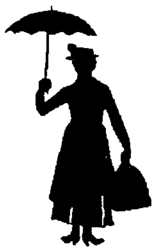
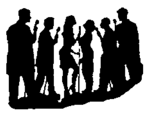
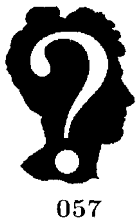
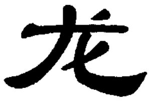
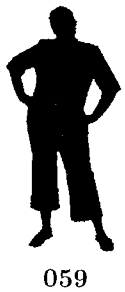
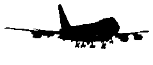
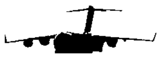
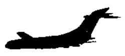
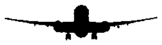
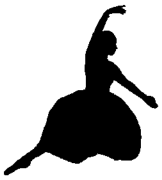
031	**032**	**033**	**034**	**035**
263 × 318	603 × 337	205 × 207	372 × 217	527 × 354
E_p 90.41	E_p 88.28	E_p 86.84	E_p 81.71	E_p 89.33
E_p^* 93.95	E_p^* 92.65	E_p^* 92.43	E_p^* 85.56	E_p^* 94.67
AC 94.39	AC 93.47	AC 93.52	AC 87.16	AC 95.37
JBIG2 89.77	JBIG2 88.97	JBIG2 86.92	JBIG2 80.1	JBIG2 92.7
036	**037**	**038**	**039**	**040**
1103 × 1088	357 × 281	226 × 418	805 × 447	300 × 300
E_p 90.18	E_p 84.59	E_p 84.99	E_p 85.67	E_p 92.12
E_p^* 96.06	E_p^* 91.73	E_p^* 91.52	E_p^* 91.12	E_p^* 95.38
AC 95.68	AC 92.52	AC 93.53	AC 91.72	AC 96.18
JBIG2 94.98	JBIG2 87.85	JBIG2 90.07	JBIG2 87.5	JBIG2 91.4
				
041	**042**	**043**	**044**	**045**
340 × 493	490 × 481	150 × 149	391 × 282	287 × 481
E_p 84.47	E_p 86.04	E_p 83.88	E_p 88.94	E_p 88.55
E_p^* 89.68	E_p^* 89.92	E_p^* 89.02	E_p^* 93.22	E_p^* 92.51
AC 90.55	AC 90.96	AC 91.76	AC 94.17	AC 93.58
JBIG2 85.38	JBIG2 84.99	JBIG2 78.92	JBIG2 88.91	JBIG2 85.81

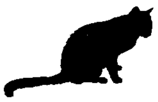
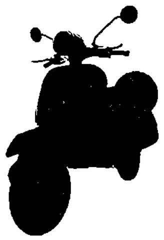
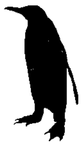
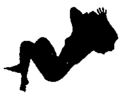
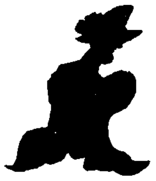
Table B 1 (*continued*)

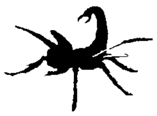
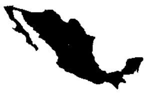
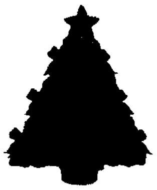
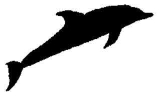
	046	047	048	049	050
					
	325 × 195	225 × 375	325 × 195	275 × 275	325 × 299
E_p	90 46	90 63	90 79	90 55	92 03
E_p^*	93 92	94 26	93 7	93 75	94 83
AC	95 09	94 87	94 91	94 81	95 26
JBIG2	89 18	90 75	89 21	86 71	91 36

	051	052	053	054	055
					
	350 × 229	360 × 270	350 × 340	263 × 233	216 × 348
E_p	91 75	92 76	88 65	87 87	87 76
E_p^*	95 75	95 4	94 29	92 31	92 1
AC	95 46	95 29	94 34	93 09	92 81
JBIG2	93 69	92 29	91 92	85 14	87 81

	056	057	058	059	060
					
	450 × 360	180 × 210	300 × 300	200 × 329	586 × 193
E_p	85 44	83 55	91 36	88 99	90 38
E_p^*	91 16	89 31	94 87	93 72	93 76
AC	92 69	91 35	96 22	94 58	94 34
JBIG2	86 86	80 17	90 03	88 58	90 88

	061	**062**	**063**	**064**	**065**
					
	640 × 439	640 × 412	640 × 439	640 × 439	576 × 640
E_p	93 98	94 62	93 71	94 59	91 61
E_p^*	96 39	97.09	96 76	96 77	96 14
AC	96 7	97 23	96 91	96 93	95 86
JBIG2	96 01	96 42	96 24	96 84	93 88

	066	**067**	**068**	**069**	**070**
					
	922 × 649	403 × 616	460 × 460	784 × 536	524 × 641
E_p	92 36	88 46	93 16	93 03	90 58
E_p^*	96 48	94 46	96 49	96 69	95 63
AC	96 06	94 36	96 44	96 5	95 12
JBIG2	96 2	91 76	95 02	95 21	94 08

	071	**072**	**073**	**074**	**075**
					
	545 × 393	729 × 434	434 × 365	203 × 247	790 × 480
E_p	90 72	91 63	93 36	86 9	92 78
E_p^*	93 75	95 54	96 65	92 66	96 5
AC	94 22	95 3	96 49	93 15	96 25
JBIG2	89 82	93 64	95 8	87 68	94 8

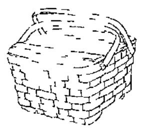
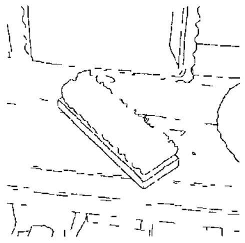
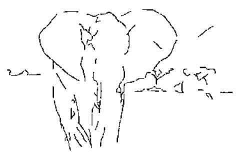
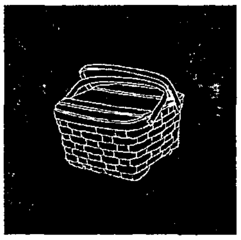
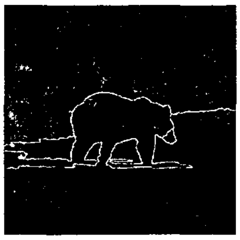
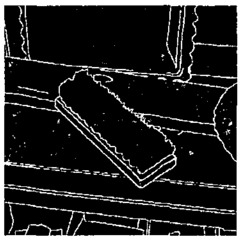
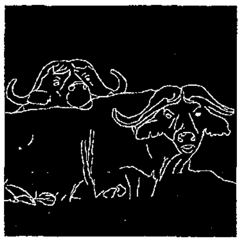
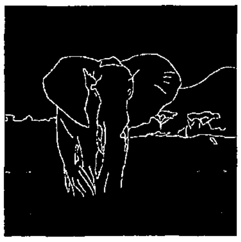
	076	**077**	**078**	**079**	**080**
	245 × 226	450 × 295	285 × 504	245 × 158	491 × 449
E_p	86.24	88.47	90.65	85.35	91.86
E_p^*	92.75	95.65	94.64	91.42	95.71
AC	93.41	95.38	95.1	91.96	95.86
JBIG2	86.82	94.83	90.71	84.91	92.17
	081	**082**	**083**	**084**	**085**
	245 × 248	491 × 526	354 × 260	240 × 394	167 × 405
E_p	89.2	92.21	88.93	87.85	86.9
E_p^*	92.84	96.33	95.29	92.52	92.55
AC	93.3	96.08	95.48	92.63	93.62
JBIG2	86.69	94.31	92.24	89.31	87.7

Table B 1 (*continued*)

086	**087**	**088**	**089**	**090**
335 × 500	447 × 459	1203 × 1200	610 × 763	350 × 357
E_p 91 12	E_p 89 89	E_p 91 12	E_p 90 39	E_p 86 68
E_p^* 95 88	E_p^* 96 19	E_p^* 95 88	E_p^* 95 85	E_p^* 95 02
AC 95 62	AC 95 73	AC 95 3	AC 95 3	AC 94 8
JBIG2 94 97	JBIG2 93 85	JBIG2 95 94	JBIG2 94 36	JBIG2 92 18
091	**092**	**093**	**094**	**095**
381 × 497	500 × 500	516 × 687	1018 × 486	680 × 449
E_p 88 65	E_p 90 78	E_p 93 04	E_p 92 43	E_p 92 86
E_p^* 93 65	E_p^* 96 08	E_p^* 96 02	E_p^* 96 42	E_p^* 96 06
AC 93 42	AC 96 11	AC 96 02	AC 96 32	AC 96 41
JBIG2 91 35	JBIG2 94 43	JBIG2 94 87	JBIG2 96 57	JBIG2 95 02
096	**097**	**098**	**099**	**100**
516 × 687	510 × 727	561 × 339	889 × 567	765 × 486
E_p 93 6	E_p 94 06	E_p 91 39	E_p 94 64	E_p 95 54
E_p^* 97 08	E_p^* 97 19	E_p^* 94 86	E_p^* 97 22	E_p^* 97 53
AC 97 06	AC 96 88	AC 95 55	AC 97 36	AC 97 59
JBIG2 97 9	JBIG2 96 87	JBIG2 92 86	JBIG2 97 16	JBIG2 97 71

Table B 2 displays the six binary images (source [53]) comprising boundary lines and the inverted counterparts. Results for these images are exhibited in Tables 3.9 and 3.10 in Section 3.1.

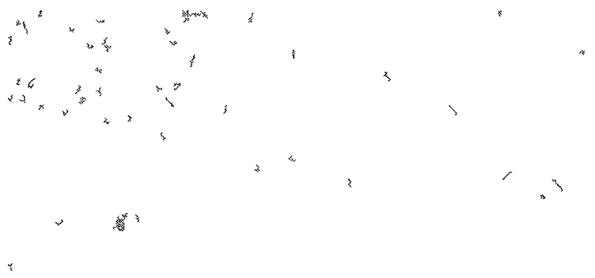
Table B.2: Test images with boundary lines

 **101-w**	**102-w**	 **103-w**
104-w	 **105-w**	**106-w**
 **101-b**	 **102-b**	 **103-b**
 **104-b**	 **105-b**	**106-b**

B.2 Selected Discrete-Color Images

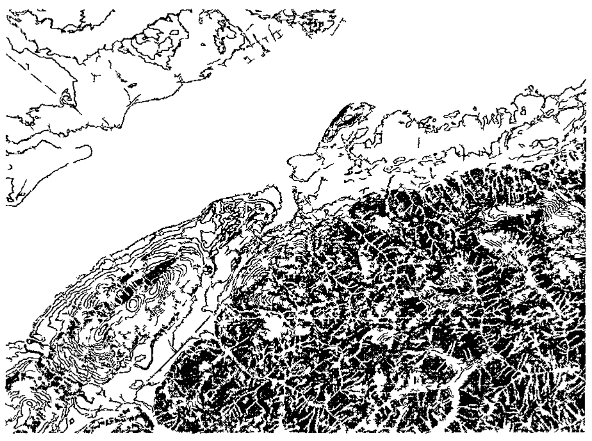
Table B 3 exhibits the three topographic maps (source [20]) used in Section 3 2 Each map contains four layers at 24-bit depth and 200 dpi resolution Compression results for these maps are exposed in Table 3 12

Table B.3: Topographic maps



Map 1

Map 2



Map 3

Table B 4 illustrates the charts and graphs used in Table 3 13 of Section 3 2

Table B.4: Charts and graphs

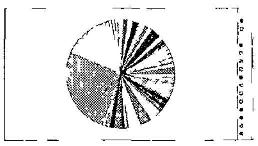
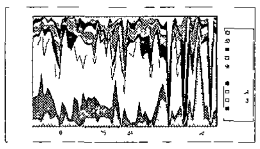
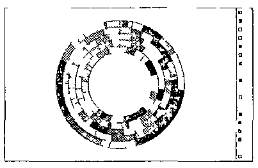
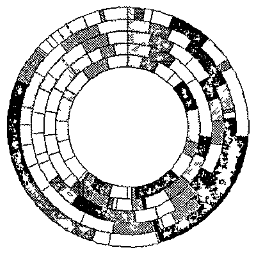
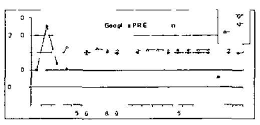
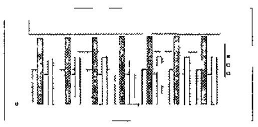
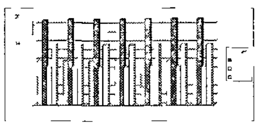
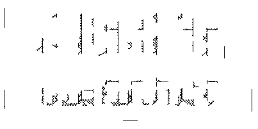
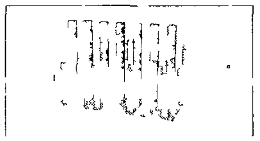
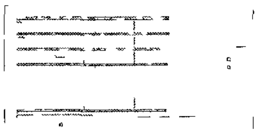
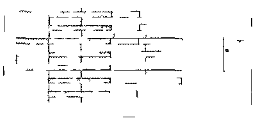
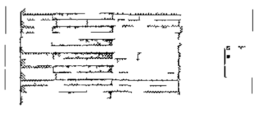
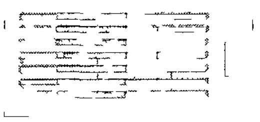
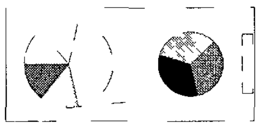
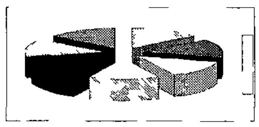
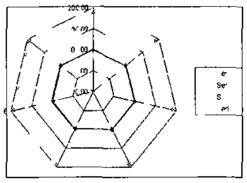
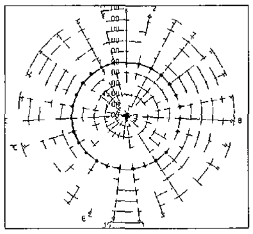
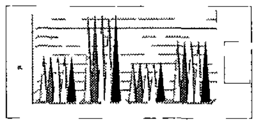
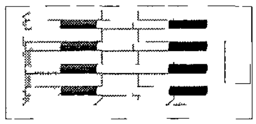
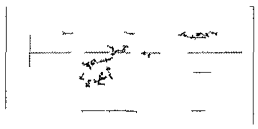
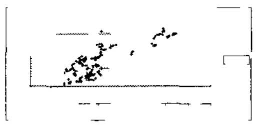
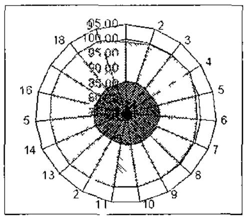
		
1 114 × 221	**2** 97 × 181	**3** 97 × 174
		
4 103 × 164	**5** 86 × 86	**6** 73 × 163
		
7 74 × 157	**8** 98 × 209	**9** 74 × 157
		
10 74 × 157	**11** 90 × 169	**12** 93 × 177
		
13 74 × 157	**14** 74 × 157	**15** 74 × 157

Table B 4 (*continued*)

		
16	**17**	**18**
74 × 157	74 × 157	74 × 98
		
19	**20**	**21**
105 × 114	74 × 157	74 × 157
		
22	**23**	**24**
101 × 205	74 × 157	74 × 83

Index