Universal Lossless Source Coding Techniques for Images and Short Data Sequences

Nicklas Ekstrand
Ph.D. Thesis, April 6, 2001

LUND INSTITUTE OF TECHNOLOGY
Lund University

Nicklas Ekstrand
Department of Information Technology
Lund University
Box 118
S-221 00, Lund, Sweden
email: nicklas@ekstrand.org
url: http://www.ekstrand.org

ISRN LUTEDX/TEIT–01/1017–SE
ISBN 91–7167–020–3

Printed in Sweden
KFS AB, Lund 2001

Abstract

In this thesis various topics in universal lossless source coding are discussed and analyzed. The main focus in this work is on lossless data compression of grayscale still images. Such images are, for example, frequently occurring in medical imaging.
Based on theoretical considerations and empirical observations new compression algorithms are presented that are, in terms of compression performance, efficient compared to *traditional* methods.
This work includes research on how to use the *Context Tree Weighting* algorithm, linear prediction and probability assignment techniques in lossless data compression. The performance of these algorithms/methods is studied both asymptotically and for usage on *short* data sequences.
The presented techniques can be used separately or together when designing efficient lossless data compression systems.

Contents

Chapter 1

Introduction

1.1 Motivation

In recent years there has been a vivid interest in lossless compression of (electronic) picture data. In particular, picture compression is considered very useful in multi-media applications and in distributed information systems that operate in network environments. However, the vast majority of research and development of picture compression techniques is focused on lossy source coding. This is primarily due to the fact that by allowing a small amount of distortion in the reconstructed picture one can obtain much better compression performance than is possible for lossless source coding. From information theory we know that there is a tradeoff between the compression performance and the amount of distortion in the restored picture. This tradeoff is formalized in the rate-distortion theory. The basis for rate-distortion theory was formulated by Shannon in his landmark 1948 paper [Sha48]. For a summary of the research on lossy data compression (or source coding) since 1948 we refer to [BG98]. On the other hand lossless source coding techniques, which allow the exact recovery of a picture from its compressed version, can only provide moderate compression performance. Yet, there are situations where only lossless (or near lossless) picture compression can be used. This is for example the case when dealing with medical data such as X-ray, CT-scan, and MR-scan pictures (e.g. see Figure 1.1), where legal issues may forbid or make it undesirable to apply lossy coding techniques. On the other hand the amount of data that is generated each year in a large Swedish hospital environment such as the university hospital "Malmö Akademiska Sjukhus" (MAS) in Malmö is enormous and picture compression is of paramount importance to reduce storage costs

and transmission times. Again taking the case of MAS, one estimates the yearly[1] amount of picture data to be 7 Terabytes!
Besides medical applications it is foreseeable that in the future we want to compress data of pictures that are "signed" with electronic signatures or water-marks for copyright enforcement. Also in such a scenario the use of lossless or near lossless picture compression may be necessary. Another application of (near) lossless compression is that of commercial image databanks where prior to the selling of the images, the required customer quality/resolution is not known. Hence, there is need for research on improved lossless picture compression techniques. In particular, the technical goals of our research were inspired by the needs for picture compression at the new radiology physics laboratory at the MAS university hospital in Malmö. One of the goals for this laboratory is to centralize processing and storage of medical data and to provide fast access to data through high-performance networks.
In the historical perspective most work in lossless image compression has been mainly focused on rather ad-hoc decisions and experiments, e.g., the former lossless JPEG standard [PM93]. One problem, from the compression ratio perspective, is the lack of knowledge on how to apply the theoretical work in information theory and source coding to lossless image compression.

1.2 Existing Methods

Depending on the aim of (lossless) image compression we focus on different theories for our solutions. The performance criterias often used to compare/evaluate image compression schemes are:

- Compression ratio (in lossy image compression we also have to consider the introduced distortion).
- Complexity (hardware requirements, algorithmic complexity).
- Speed (time to compress/decompress).

The main criterion used in the work of this thesis is the **compression ratio**. We will also make some remarks about algorithmic complexity since it must be possible to use the presented methods in practice although we will not discuss the actual implementations.
If we again consider the case of medical images at a large Swedish hospital we see that the intention may be to store the individual images for a long time, e.g. up to 20 years. In this period of time an image will be

[1] Estimate for 1997.

Figure 1.1: An example of a medical image. This is the CT-image “thorax”.

retrieved about once a year or so. The equipment used in this context is what most people would call very sophisticated or advanced. We could therefore conclude that lossless image compression in this area is mainly focused on the compression ratio and that complexity and speed of compression are less important. Fast decompression is, however, desirable to obtain acceptable performance during image retrieval.
In other applications such as image processing/editing software it is of major importance to have both sufficient speed and a low complexity. This will often be at the expense of the compression ratio. In many applications we see a trade-off between complexity (memory requirement, compression/decompression speed) on one side and compression ratio on the other side.
At the time of writing of this thesis (spring 2001) there are two major lossless image compression standards:

- JPEG2000, JPEG-LS (JPEG=Joint Photographic Experts Group),
- PNG (PNG=Portable Network Graphics).

There are actually two different versions of lossless JPEG standards: lossless JPEG [PM93] and JPEG-LS [ISO98] (or LOCO-I[2] [WSS96] and [WSS00]). The lossless JPEG has been around for a couple of years but has not gained any public interest since the compression performance is very disappointing. The aim with the recently accepted standard JPEG-LS was to gain both in compression performance and in compression/decompression speed. On the other hand the PNG standard [Wor96] has been designed for network or web applications. The aim was therefore to make sure that it should be easy to implement on almost all systems, the compression performance should be good and the decompression speed should be high.

Example 1.1:
A comparison of the compression performance of the image in Figure 1.1:

Method	**compr. size**	**compr. time**	**decompr. time**
JPEG-LS (new)	79223	$\sim 0.5-1$ s	$\sim 0.5-1$ s
PNG	135599	$\sim 0.5-1$ s	$\sim 0.5-1$ s

The size of the image is 512x512 pixels and with 8 bits for representation of each pixel. Thus, totally 262144 bytes are required to describe the image in its original format. The compression and decompression times are almost the same for both schemes. ■

As a final remark about lossless image compression we note that the compression performance is very hard to improve upon. The public interest

[2] LOw COmplexity LOssless COmpression for Images

in this matter is therefore more concentrated on the speed performance than on the possible gain of saving a few bytes here and there. But for specialized applications and from a theoretical point of view we will see that there is still more to examine.

1.3 Main Goals

The main goals of this thesis are:

- Contribute to the understanding of lossless data compression of images.
- Apply the *Context Tree Weighting* (CTW) algorithm to image compression.
- Design and evaluate different estimation/probability assignment techniques for sources commonly used in image compression.
- Discuss compression performance for **short** data sequences and apply it to practical problems.

1.4 About this Thesis

This thesis consists of four major parts where the three later can be considered as (almost) independent papers:

- Chapter 2: this chapter is a summary of the "teknisk licentiat thesis" presented in 1998.
- Chapter 3: the concept of *Prediction and Probability Assignment* (PPA) is discussed and a technique for PPA is presented and analyzed.
- Chapter 4: in this chapter we define the *Context Mapping Function* and present a method along with experiments on how to optimize the CMF for several different sources.
- Chapter 5: this chapter concerns the compression of request data in ARQ protocols.

Most of the work in this thesis has already been presented at conferences etc. according to the following:

- Lossless Compression of Medical Images [Eks95].

- On the Estimation and Model Costs in Lossless Universal Image Data Compression by Context Weighting [ES96].
- Lossless Compression of Grayscale Images via Context Tree Weighting [Eks96].
- Weighting of Double Exponetial Distributed Data in Lossless Image Compression [ES98b].
- Notes on the P-Context Algorithm [ES98a].
- Some Results on Lossless Compression of Grayscale Images [Eks98].
- Some Notes on the Context Mapping Function in Lossless Data Compression [ES00a].
- A Technique for Prediction and Probability Assignment (PPA) in Lossless Data Compression [ES00b].
- The Qualitative Modeling and Compression of the Request Sequences in ARQ Protocols [ERSS01].

This work has been financially supported by:

- Swedish Research Council for Engineering Sciences (http://www.tfr.se).
- Swedish National Board for Industrial and Technical Development (http://www.nutek.se).

Some computational resources have been provided by:

- Center for Scientific and Technical Computing, LUNARC, Lund University, Sweden (http://www.lunarc.lu.se).
- National Supercomputer Center, NSC, Linköping University, Sweden (http://www.ncs.liu.se).

Chapter 2

Lossless Compression of Grayscale Images

This chapter is a summary of the "licentiat"-thesis *Some Results on Lossless Compression of Grayscale Images* [Eks98]. It is organized such that each building block of an image compression scheme (see Figure 2.2) is explained in each section. In Section 2.1 we introduce and define some basic concepts in lossless data compression and image compression. In Section 2.2 the topic of context tree models is covered and how to use these models in lossless image compression. In Section 2.3 we discuss how to convert the 2-dimensional image data into 1-dimensional data which is appropriate for our statistical/context tree modeling. In Section 2.4 we define and discuss how to use *prediction* in lossless image compression in order to reduce the complexity of the statistical modeling. In Section 2.5 we define and present some results on the statistical modeling which is used in combination with the context tree modeling. Finally, in Section 2.6 we make a simple study of the correlation between adjacent pixels in images. This is an important aspect since it motivates the use of prediction and context modeling.

2.1 Source Coding & Image Compression

In this section a general compression scheme for lossless image compression will be presented. This general description is not intended to cover all available compression schemes, but it will cover current state-of-the-art lossless compression schemes from an information theoretical perspective, e.g., we do not consider the implementation aspects of the

algorithms. Notations and concepts used in this chapter (and throughout the thesis) are similar or equal to the notations used in [Sme96][1]. It will be assumed that the reader is familiar with the concepts of source codes.

2.1.1 The Intractability of Optimal Lossless Image Compression

We start by noting two things that make image compression special: the image data (from a finite alphabet) is 2-dimensional and images have finite boundary, i.e., they have a fixed limited size. From the second observation we can state the following about optimal lossless compression of individual images when considering the average code word length:

Theorem 2.1:
For the set of all possible images $\mathcal{I} = \{I_1, I_2, \ldots, I_k\}$ of a fixed given size and with their corresponding set of probabilities of occurring $\mathcal{P} = \{P(I_1), P(I_2), \ldots, P(I_k)\}$ an *optimal uniquely decodable source code*, in terms of average redundancy, can be found by constructing a Huffman code based on $\mathcal{P}$. □

The theorem follows directly from the optimality of Huffman codes [CT91]. This construction would not have been possible if the images had infinite boundary, i.e., the number of possible images, k, would be infinite. Note also that, except for very small images, Huffman coding as used in Theorem 2.1 is not a practical technique since k will be a huge value. As an example, in this thesis we will often use images of size 512x512 pixels with 8 bits per pixel. The number of possible images with this size is thus $k = 256^{262144}$.

2.1.2 Lossless Image Compression Based on Sequential Universal Source Coding

Based on a basic (universal) lossless source coding scheme, see Figure 2.1, a general lossless image compression scheme will be presented. A probability assignment function $P_{Assign}(\cdot)$ is said to be universal for a class of sources if it holds that for any source S in the class that the average code word length per source symbol asymptotically approaches the entropy (per source symbol) of the source:

[1] However, all notations and concepts are defined or explained whenever introduced.

$$E_S\left[\frac{-\log(\prod_{i=1}^{n} P_{Assign}(m_i|m_1\cdots m_{i-1}))+\log(P_S(\mathbf{m}^n))}{n}\right]\to 0 \qquad (2.1)$$

when $n \to \infty$. Since $P_{Assign}(x|\cdot)$ should be a probability distribution it holds that $P_{Assign}(x|\cdot) \in [0,1]$, $\forall x \in \{0,1,\ldots,M-1\}$, where M is the source alphabet size. It also holds that:

$$\sum_{x=0}^{M-1} P_{Assign}(x|\cdot) = 1. \qquad (2.2)$$

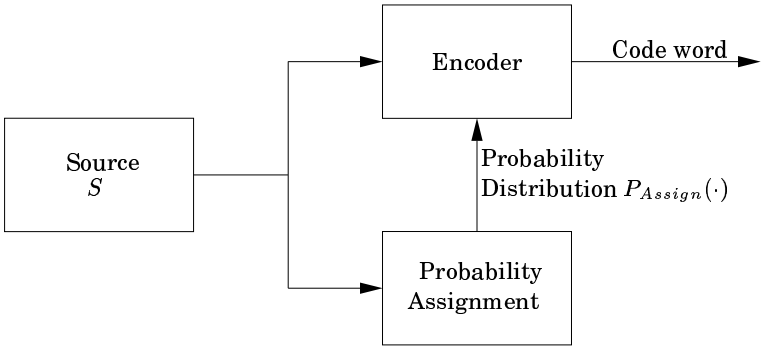

Figure 2.1: A basic lossless compression scheme.

The encoder in Figure 2.1 forms the code word[2] from the assigned probability distributions and the source symbols. This is usually done by an arithmetic encoder. For a binary arithmetic encoder we know that the maximum overhead (redundancy) will be less than 2 bits for any length of the source sequence, provided that the numerical precision problem in arithmetic coding can be ignored, e.g., see [WST95]. This will be further explained later in this chapter.
Derived from the general description in Figure 2.1, we propose the general lossless image compression scheme in Figure 2.2.
The conceptual principle for almost all lossless image compression schemes is based on four main components (see Figure 2.2): Firstly, convert the 2-dimensional image into a 1-dimensional string (source with serializer). Secondly, pre-process the data. This is mainly done to simplify the last

[2]Depending on the encoder the output may be either a single code word or several code words. However, we will often consider the output as a single code word.

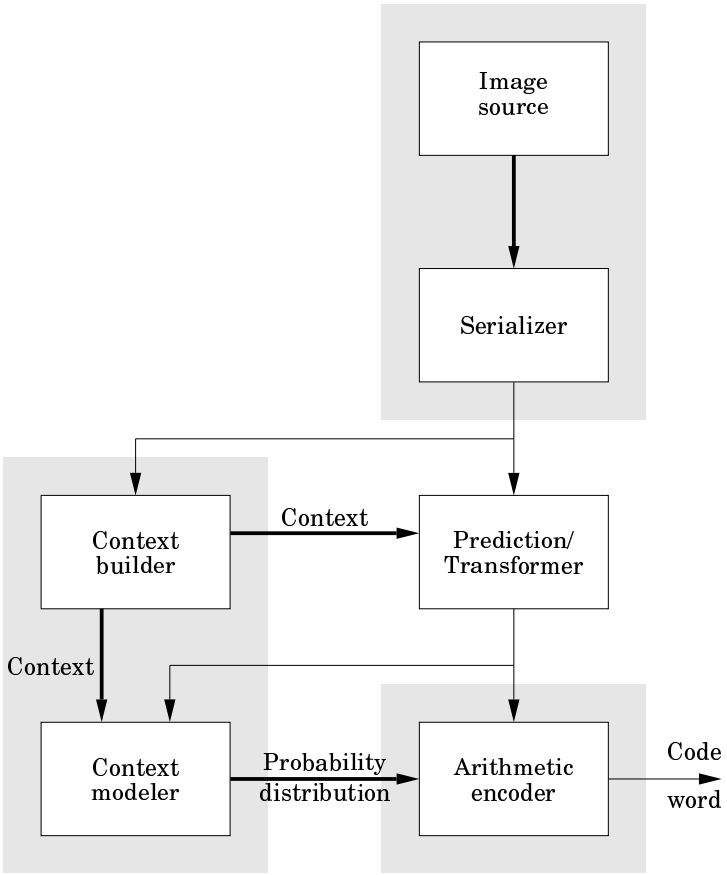

Figure 2.2: A general image compression scheme. Each gray box in this figure corresponds to one box in the basic lossless data compression scheme in Figure 2.1.

two components (prediction/transformer). Thirdly, assign a probability distribution for each pre-processed data symbol to be encoded (context builder/context modeler). Finally, the code word is constructed from the pre-processed data and probability distribution (arithmetic encoder).
Some comments to the general scheme: It should be noted that the choice of serializer, pre-processor, and probability assignment function must be done in a combined process since the individual components are not independent of each other. If one of them is changed the performance of the others will be affected. It should also be noted that from an implementation perspective the scheme may look different.

2.1.3 Some Basic Concepts in Lossless Data Compression

From information theory (see e.g., [CT91]) we know that the self-information $I(A)$ about an event A depends on the probability $P(A)$ for that event as

$$I(A) = -\log P(A), \tag{2.3}$$

where $\log$ is the two-logarithm[3] and the self-information is measured in bits. In data compression we are interested in describing the information and removing the redundancy from the data. In order to do this the probability for the data (or event) has to be determined, directly or indirectly. In our setting according to Figure 2.1 we determine the probability distribution according to some kind of statistical model.
The following main definitions are basic concepts throughout this thesis.

Definition 2.2 (Ideal Codeword Length):
The *ideal codeword length* (in bits) for a message m with known probability of occurring $P(m)$ is defined as:

$$-\log P(m). \tag{2.4}$$

■

In general it is not possible to encode according to the ideal codeword length. But through binary *arithmetic encoding* (e.g., [Pas76], [RM89], [Jon81], [BCW90]) it is theoretically possible to encode with an overhead of not more than 2 bits. The bound comes from the fact that it is possible to construct a binary prefix-free codeword with length w such that $w = \lceil -\log P(m) \rceil + 1$. This is discussed in the mentioned references on arithmetic encoding and is also described in [CT91] along with the Shannon-Fano coding. Arithmetic coding is derived from the Shannon-Fano coding.

[3] The two-logarithm, $\log_2$, is used throughout this thesis.

For measuring the compression performance with respect to theoretical limits some definitions of redundancy are necessary. The message $\mathbf{m} = [m_1, m_2, \ldots, m_n]$ consists of n symbols from the source alphabet $\mathcal{M}$. Finally, $S \in \mathcal{S}$ denotes the source in a class of sources $\mathcal{S}$, $P_{\mathcal{S}}(S)$ denotes the probability for source S in the class, and $P_S(\boldsymbol{\theta})$ denotes the probability distribution for the parameter vector $\boldsymbol{\theta}$ associated with source S. In the following $\hat{P}_S(\mathbf{m})$ and $\hat{P}_{\mathcal{S}}(\mathbf{m})$ denote the used coding probability distributions for which we would like to calculate the redundancy. With $\hat{P}_S(\mathbf{m})$ we refer to the case when the source S is known and with $\hat{P}_{\mathcal{S}}(\mathbf{m})$ only the class of sources $\mathcal{S}$ is known.
In the following *code* is the description of how to map source messages, $\mathbf{m}$, onto code words.

Definition 2.3:
For a code with code word lengths for the input messages according to the code word length function $L_S(\mathbf{m}|\boldsymbol{\theta})$ for the source S with known parameter vector $\boldsymbol{\theta}$ the *individual redundancy* is defined as:

$$\rho_S(\mathbf{m}|\boldsymbol{\theta}) = L(\mathbf{m}|\boldsymbol{\theta}) + \log P_S(\mathbf{m}|\boldsymbol{\theta}). \tag{2.5}$$

■

Sometimes we use the notation *coding probability*, $\hat{P}(\cdot)$. This is the distribution corresponding to the code word length function, $L(\cdot)$, i.e., in the above definition the corresponding coding probability is:

$$\hat{P}_S(\mathbf{m}|\boldsymbol{\theta}) = C^{-L_S(\mathbf{m}|\boldsymbol{\theta})}, \tag{2.6}$$

where C is the code word alphabet size. From the Kraft inequality (see [CT91]) we know that it for any uniquely decodable code:

$$\sum_{\mathbf{m}} \hat{P}_S(\mathbf{m}|\boldsymbol{\theta}) \leq 1. \tag{2.7}$$

Thus $\hat{P}_S(\cdot)$ may not sum up to 1. Although $\hat{P}_S(\cdot)$ does not necessarily form a probability distribution we will still denote it as coding probability.
From the individual redundancy for a single message we define the average over all messages according to:

Definition 2.4:
The *average individual redundancy per source symbol* for a code is defined as:

$$\bar{\rho}_S(\boldsymbol{\theta}) = \frac{1}{n} \sum_{\mathbf{m} \in \mathcal{M}^n} P_S(\mathbf{m}|\boldsymbol{\theta}) \rho_S(\mathbf{m}|\boldsymbol{\theta}). \tag{2.8}$$

■

We will in the sequel not focus much on the redundancy for the code construction since we will use arithmetic coding except when otherwise stated. The main concern will be the statistical modeling and thus the probability assignment. For the PA we will use the following definition to measure the performance:

Definition 2.5:
The *average redundancy per source symbol* for a probability assignment or estimation function for which the parameter vector is not known is defined as:

$$\bar{\rho}_S(\boldsymbol{\theta}) = \frac{1}{n} \sum_{\mathbf{m} \in \mathcal{M}^n} P_S(\mathbf{m}|\boldsymbol{\theta})(-\log \hat{P}_S(\mathbf{m}) + \log P_S(\mathbf{m}|\boldsymbol{\theta})). \tag{2.9}$$

■

Definition 2.6:
The *average redundancy per source symbol* for a probability assignment or estimation function for which the source and associated parameter vector are not known is defined as:

$$\bar{\rho}_{\mathcal{S}}(S, \boldsymbol{\theta}) = \frac{1}{n} \sum_{\mathbf{m} \in \mathcal{M}^n} P_{\mathcal{S}}(\mathbf{m}|S, \boldsymbol{\theta})(-\log \hat{P}_{\mathcal{S}}(\mathbf{m}) + \log P_{\mathcal{S}}(\mathbf{m}|S, \boldsymbol{\theta})). \tag{2.10}$$

■

Definition 2.7:
The *average parameter redundancy per source symbol* for a probability assignment/estimation function for which the parameter vector is not known is defined as:

$$\bar{\rho}_S = \int_{\boldsymbol{\theta} \in \Theta_S} P_S(\boldsymbol{\theta}) \bar{\rho}_S(\boldsymbol{\theta}) d\boldsymbol{\theta}. \tag{2.11}$$

■

Definition 2.8:
The *average parameter redundancy per source symbol* for a probability assignment/estimation function for which the source and corresponding parameter vectors are not known is defined as:

$$\bar{\rho}_{\mathcal{S}} = \sum_{S \in \mathcal{S}} P_{\mathcal{S}}(S) \int_{\boldsymbol{\theta} \in \Theta_S} P_S(\boldsymbol{\theta}) \bar{\rho}_{\mathcal{S}}(S, \boldsymbol{\theta}) d\boldsymbol{\theta}. \tag{2.12}$$

■

Definition 2.9:
The *worst case redundancy for an individual sequence* for a specific source S is defined:

$$\tilde{\rho}_S(\mathbf{m}) = -\log \hat{P}_S(\mathbf{m}) + \sup_{\boldsymbol{\theta} \in \Theta_S} \log P_S(\mathbf{m}|\boldsymbol{\theta}). \tag{2.13}$$

■

Definition 2.10:
The *worst case redundancy for an individual sequence* for a class of sources $\mathcal{S}$ is defined as:

$$\tilde{\rho}_{\mathcal{S}}(\mathbf{m}) = -\log \hat{P}_{\mathcal{S}}(\mathbf{m}) + \max_{S \in \mathcal{S}} \sup_{\boldsymbol{\theta} \in \Theta_S} \log P_S(\mathbf{m}|\boldsymbol{\theta}). \tag{2.14}$$

■

We will later, were applicable, divide the redundancy into two parts, a *source description cost* and a *parameter description cost*.
When considering sequences the following definition is useful:

Definition 2.11 (Block Probability):
For a sequence $\mathbf{x}^n = [x_1, x_2, \cdots, x_n]$, $x_i \in \mathcal{A}$, where $\mathcal{A}$ is a finite alphabet, of symbols with known probability distribution function $P(\cdot)$ with known parameter vector $\boldsymbol{\theta}$ the *block probability* is defined as:

$$P_b(\mathbf{x}^n|\boldsymbol{\theta}) = \prod_{i=1}^{n} P(x_i|\mathbf{x}^{i-1}, \boldsymbol{\theta}). \tag{2.15}$$

■

One important technique when coding data from a source with unknown parameter vector is *weighting*. To make a weighted distribution the block probability is used:

Definition 2.12 (Weighted Block Probability):
For a vector of q parameter vectors $\Theta = [\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_q]$, where the parameter vectors $\boldsymbol{\theta}_i = [\theta_1, \theta_2, \ldots, \theta_k]$ have k scalar parameters, and a sequence $\mathbf{x}^n$ the weighted block probability is defined as:

$$P_w^{\Theta}(\mathbf{x}^n) = \sum_{i=1}^{q} c_i P_b(\mathbf{x}^n|\boldsymbol{\theta}_i), \tag{2.16}$$

where $c_i \geq 0$ are weighting coefficients, and $\sum_{i=1}^{q} c_i = 1$. ■

The weighted block probability is a useful concept when coding sources with unknown parameters:

Example 2.1:
A binary memoryless source, S, generates its symbols, $\{a, b\}$, according to either one of two different distributions, P_1 or P_2:

	P_1	P_2
a	$3/10$	$9/10$
b	$7/10$	$1/10$

We further know that S generates symbols according to P_1 with probability $\frac{1}{2}$.
By Definition 2.12 we get the weighted block probability for the sequence $\mathbf{x}^n = [x_1, x_2, \ldots, x_n]$, $x_i \in \{a, b\}, i \in \{1, 2, \ldots, n\}$:

$$P_w(\mathbf{x}^n) = \frac{1}{2}\prod_{i=1}^{n} P_1(x_i) + \frac{1}{2}\prod_{i=1}^{n} P_2(x_i). \tag{2.17}$$

If a sequence starts with baa the probability distribution for the next symbol can from the weighting technique be found as:

$$P(a|baa) = \frac{P_w(baaa)}{P_w(baa)} \approx 0.6375, \tag{2.18}$$

$$P(b|baa) = 1 - P(a|baa) \approx 0.3625. \tag{2.19}$$

We note that the distribution matches neither P_1 nor P_2. ■

By calculating the weighted block probability as in Example 2.1 it is possible to asymptotically encode the source data with an arithmetic encoder[4] with an average rate equal to the entropy without knowing the actual parameter of the source. Another approach would be to guess the source parameter, i.e., for each symbol to be encoded make a (reasonable) guess of which one of the two sources has generated the sequence so far and encode according to that parameter.

Example 2.2 (Example 2.1 continued):
For the same source as in Example 2.1 we use the following simple scheme to guess the parameter of the source:

- If $n_a \geq n_b$ then assume P_2,
- Otherwise assume P_1,

where n_a denotes the number of a's seen so far and n_b the corresponding number for b. By using the guessing scheme we find that:

$$P(a|baa) = 9/10,$$
$$P(b|baa) = 1/10.$$

[4] Again we are ignoring any numerical problems in the arithmetic encoder.

Sequence	Weighting	Guessing
aaa	0.378	0.729
aab	0.072	0.081
aba	0.072	0.081
baa	0.072	0.027
abb	0.078	0.009
bab	0.078	0.003
bba	0.078	0.021
bbb	0.172	0.049

Table 2.1: Comparison between a weighting technique and a "guessing" scheme according to Example 2.2. In the table the individual redundancy in bits is shown for all possible sequences of length 3.

■

We finish this example by comparing the two schemes for all possible sequences of length 3 in Table 2.1. We conclude from the table that the weighting technique is sequence order independent[5] and that the guessing scheme is not. The fact that the sequence order changes the redundancy may be an undesirable property. With weighting techniques we have a controlled way of administrating the redundancy. We also conclude that the average redundancy per message, $\bar{\rho}_S(\boldsymbol{\theta})$, in this example should for any sequence length be less than or equal to 1 bit, i.e., one bit is required to describe which one of the two sources that is used since they appear with equal probability. This is obvious for the weighting technique since

$$\begin{aligned} -\log P_w(\mathbf{x}^n) &\leq -\log \frac{1}{2}\prod_{i=1}^{n} P_j(x_i) \\ &= 1 - \log \prod_{i=1}^{n} P_j(x_i), \end{aligned} \tag{2.20}$$

for both $j = 1$, and $j = 2$. But for the guessing scheme the redundancy could be much larger in the worst case.

Example 2.3 (Example 2.2 continued):
We start by calculating the average redundancy for each of the two tech-

[5] Meaning that if $\mathbf{s}$ is a sequence and $\Pi(\cdot)$ is a permutation of its symbols, it holds that $P_w(\mathbf{s}) = P_w(\Pi(\mathbf{s}))$ for all permutations $\Pi(\cdot)$.

niques and get:

$$\text{Weighting: } \bar{\rho}_S(7/10) \approx 0.201 \text{ bit/source symbol}, \quad (2.21)$$
$$\text{Weighting: } \bar{\rho}_S(1/10) \approx 0.213 \text{ bit/source symbol}, \quad (2.22)$$
$$\text{Guessing: } \bar{\rho}_S(7/10) \approx 0.899 \text{ bit/source symbol}, \quad (2.23)$$
$$\text{Guessing: } \bar{\rho}_S(1/10) \approx 0.042 \text{ bit/source symbol}. \quad (2.24)$$

Since the total redundancy should not be more than 1 bit, i.e., $\frac{1}{3}$ bit/source symbol, the guessing scheme fails in the worst case.
Considering the average over the parameters we get:

$$\text{Weighting: } \bar{\rho}_S \approx 0.208 \text{ bit/source symbol}, \quad (2.25)$$
$$\text{Guessing: } \bar{\rho}_S \approx 0.470 \text{ bit/source symbol}. \quad (2.26)$$

From the above discussion we conclude that for the guessing scheme the total description cost of the parameter is higher than the desired 1 bit.
Finally, we calculate the worst case redundancy for the sequence $[bab]$:

$$\text{Weighting: } \tilde{\rho}_S([bab]) \approx 0.914 \text{ bits}, \quad (2.27)$$
$$\text{Guessing: } \tilde{\rho}_S([bab]) \approx 5.615 \text{ bits}. \quad (2.28)$$

For the worst case redundancy it should also hold that the redundancy is less than 1 bit. ■

In order to keep the different techniques apart we say that the weighting technique uses a *probability assignment* technique and the guessing scheme is an *adaptive* technique.
This section about some basic concepts in lossless data compression ends with an important definition for sets:

Definition 2.13:
A set of sequences $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$ is said to be *prefix-free* if no sequence $\mathbf{x}_i$ is a prefix to any other sequence $\mathbf{x}_j$ with $i \neq j$ and $i, j \in \{1, 2, \ldots, m\}$.
The set $\mathcal{X} \subseteq \mathcal{A}^{n*}$ is *complete* if for all sequences $\mathbf{y} \in \mathcal{A}^n$ there exists an i such that $\mathbf{x}_i$ is a prefix to $\mathbf{y}$, where $\mathcal{A}^{n*} = \bigcup_{k=0}^{n} \mathcal{A}^k$. ■

2.2 Context Tree Modeling

This section concerns the context modeling and context building in the generic scheme, see Figure 2.2. The motivation for using context modeling in image compression is to take advantage of the correlation between

adjacent pixels. We come back to this in more detail in Section 2.6. Context modeling, used together with a prediction scheme and other preprocessing, may not have an intuitively natural approach. The intention of preprocessing is to reduce the complexity of the data. But, as was shown in [FM92], no prediction scheme will achieve total de-correlation, and thus a context model is necessary to utilize the remaining correlation. This will be studied further in Section 2.4.
Much of the contents in this section is inspired by the work of Rissanen on the Algorithm CONTEXT [Ris83] and, primarily, by the excellent work of Willems, Shtarkov and Tjalkens on the Context Tree Weighting Method [WST93] and [WST95]. An introduction to these algorithms along with possible extensions for applications in lossless image compression will be presented in this section.

2.2.1 Context Tree - Basics

To avoid confusion with other definitions used in the literature we define a context tree as follows, see also Figure 2.3.

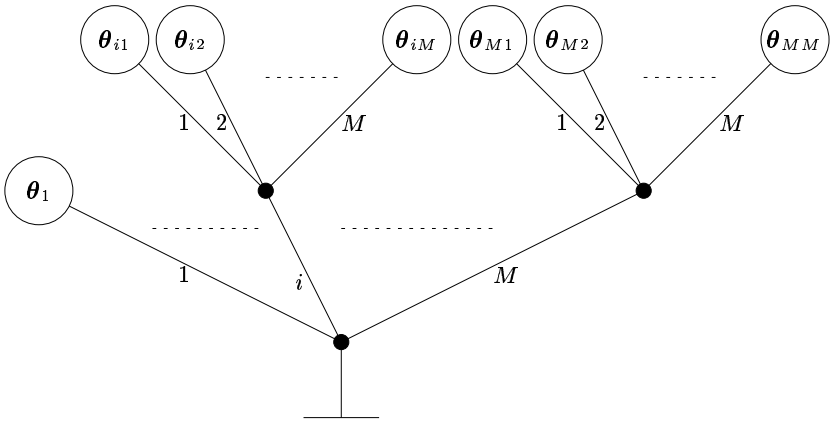

Figure 2.3: An example of a context tree.

Definition 2.14 (GMCT-f):
For a complete and prefix-free set of M-ary strings:

$$\mathcal{T} \subseteq \bigcup_{i=0}^{D} \{0, 1, \ldots, M-1\}^i, \tag{2.29}$$

a general M-ary context tree is defined as:

$$T = \{[c, \boldsymbol{\theta}_c] : c \in \mathcal{T}\}, \tag{2.30}$$

which is a set of parameter vectors $\boldsymbol{\theta}_c$ along with a description of the position in the tree. The parameter vectors apply to a memoryless probability distribution function $f(\cdot|\boldsymbol{\theta}_c)$, i.e., given a (long enough) context, a probability distribution will be found in the context tree. The maximum length D is fixed and limited. The tree type is denoted GMCT-f. ■

The fact that the set $\mathcal{T}$ is prefix-free makes it natural to regard each element of $\mathcal{T}$ as a leaf in a tree. Since the set $\mathcal{T}$ is complete there will be no context that does not point out a leaf in the tree. From the definition a source may be constructed. A context tree source will from a *given* context generate a source symbol according to the distribution given by the distribution function and the parameter vector corresponding to the prefix of the context that is an element in $\mathcal{T}$, i.e., a leaf in the context tree.

The FSMX source was defined by Rissanen in [Ris83]. The FSMX source is a special case of the more general GMCT-f in that the FSMX source will use the same alphabet for both the source and the context. The context is determined by the previous source symbols. For the FSMX source it also holds that it can be modeled by a Markov source (finite state machine).

A similar construction to the GMCT-f is a corresponding *context model*:

Definition 2.15 (GMCM-f):
A *General M-ary Context Model* is a GMCT without the requirement that $\mathcal{T}$ is prefix-free. The parameter for a context is found according to the longest matching string in $\mathcal{T}$. ■

The nodes in a context model, in contrast to a context tree, may therefore not necessarily be fully extended with all possible sons.

To visualize the difference between GMCT-f and GMCM-f we will look at an example:

Example 2.4 (GMCT-f vs GMCM-f):
For a binary probability distribution $f(\cdot|\theta)$, where θ denotes the probability for symbol 0, and $M = 3$, we have the following, see also Figure 2.4 and 2.5:

- a GMCT-f tree:
  $T_t = \{\ [0, \theta_0],\ [1, \theta_1],\ [20, \theta_{20}],\ [21, \theta_{21}],\ [22, \theta_{22}]\ \},$

- a GMCM-f tree:
  $T_m = \{\ [0, \theta_0],\ [1, \theta_1],\ [2, \theta_2],\ [20, \theta_{20}],\ [21, \theta_{21}],\ [220, \theta_{220}]\ \}.$

Given the context $220???\cdots$[6] we will get the probability distribution:

$$P(0|220???\cdots) = \theta_{22}, \tag{2.31}$$
$$P(1|220???\cdots) = 1-\theta_{22}, \tag{2.32}$$

for T_t and

$$P(0|220???\cdots) = \theta_{220}, \tag{2.33}$$
$$P(1|220???\cdots) = 1-\theta_{220}, \tag{2.34}$$

for T_m. For context $221???\cdots$ we get the parameter θ_{22} for T_t and θ_2 for T_m. ■

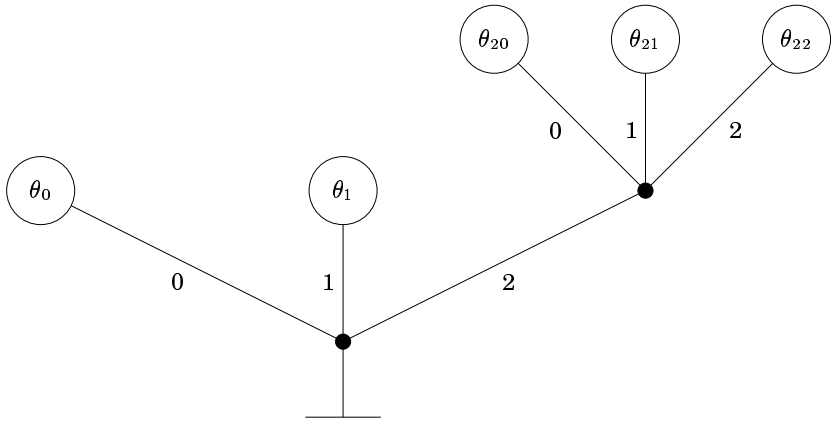

Figure 2.4: The context tree T_t in Example 2.4.

In order to describe a context tree source it is necessary and sufficient to describe the context tree $\mathcal{T}$ with its parameter vectors in T. In [Ris84] a lower bound for universal coding was given. Applied to a context tree source $\mathcal{T}$ with source alphabet size m and the parameter vectors $\boldsymbol{\theta}_c = [\theta_1, \theta_2, \ldots, \theta_{m-1}, 1-\sum_{i=1}^{m-1}\theta_i]$, $c \in \mathcal{T}$ we get the following theorem:

[6] The question marks, "?", denote unknown (arbitrary) symbols.

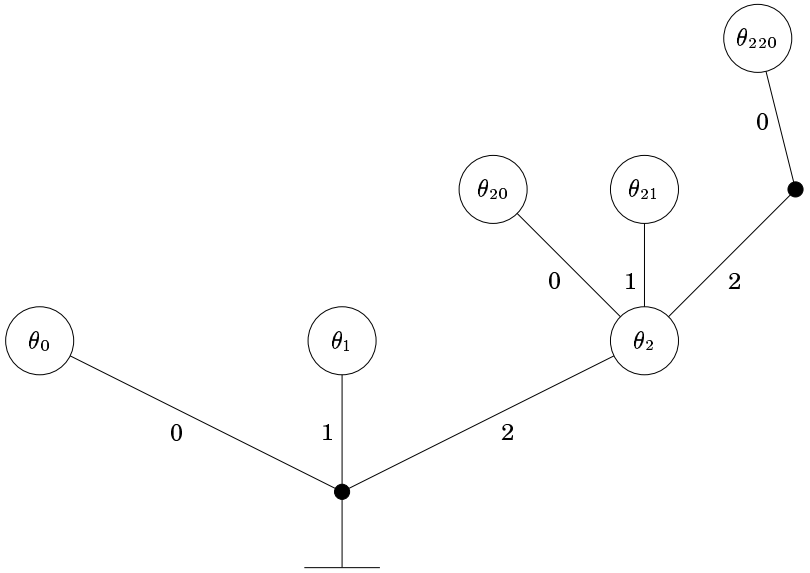

Figure 2.5: The context model T_m in Example 2.4.

Theorem 2.16 (Universal Coding of Context Tree Sources):
For coding of a sequence of length n, with corresponding contexts, from a context tree source T, with known leaves $\mathcal{T}$ with unknown parameters, and with source output alphabet size m, it holds, for all $\epsilon > 0$, that the average redundancy

$$\bar{\rho}_T \geq (1-\epsilon)\frac{|\mathcal{T}|(m-1)}{2}\frac{\log n}{n}, \tag{2.35}$$

for all parameter vectors in T, except for a small set whose volume goes to zero as $n \to \infty$. □

This important theorem was proved in [Ris84] (although, it was slightly different formulated). In [Sme96] this theorem is called *Rissanen's Converse*.
The description cost for describing the tree (tree description cost) is possible to make constant, i.e., independent of n. How this is done will be shown the in following sections. For the GMCT-f, the source alphabet does not affect the redundancy in the same way:

Corollary 2.17:
For the same conditions as in Theorem 2.16 with a GMCT-f source with $f(\cdot)$ having a parameterization with $\hat{m}$ independent real valued parameters the average redundancy fulfills

$$\bar{\rho}_T \geq (1-\epsilon)\frac{|\mathcal{T}|m_{min}}{2}\frac{\log n}{n}, \tag{2.36}$$

where $m_{min} = \min(m-1, \hat{m})$. □

The corollary follows from Theorem 2.16. It is (presumably) easy to find a parameterization where $\hat{m} \geq m-1$. But we would actually like to find the parameterization with the fewest parameters since:

Corollary 2.18:
For the same conditions as in Corollary 2.17 and with two different parameterizations with $\hat{m}_1$ and $\hat{m}_2$ independent real valued parameters respectively, it holds:

$$\bar{\rho}_T \geq (1-\epsilon)\frac{|\mathcal{T}|m_{min}}{2}\frac{\log n}{n}, \tag{2.37}$$

where $m_{min} = \min(\hat{m}_1, \hat{m}_2)$. □

However, finding the parameterization with a minimal number of parameters may be hard to determine.
When a tree only consists of the root node it is a memoryless source:

Corollary 2.19:
Under the same conditions as in Theorem 2.16 and Corollary 2.17 we get for a memoryless source:

$$\bar{\rho} \geq (1-\epsilon)\frac{m_{min}}{2}\frac{\log n}{n}. \tag{2.38}$$

□

Since Theorem 2.16 and Corollaries 2.17 and 2.19 only tell us how well it is possible to perform, we make the following definition which will be used as a goal in the sequel:

Definition 2.20:
A probability assignment (PA) function for memoryless sources is *asymptotically optimal with respect to average redundancy* if the following holds for some $N > 0$:

$$\bar{\rho} \leq \frac{m_{min}}{2}\frac{\log n}{n} + \frac{C_{PA}}{n}, \quad \forall n \geq N, \tag{2.39}$$

where C_{PA} is a constant. In the same way, a PA-function is *asymptotically optimal*, with respect to average redundancy, for a context tree source T, where no context in the tree $\mathcal{T}$ appear with probability 0, if the following holds:

$$\bar{\rho}_T \leq \frac{|\mathcal{T}|m_{min}}{2}\frac{\log n}{n} + \frac{|\mathcal{T}|C_{PA}}{n}, \quad \forall n \geq N, \tag{2.40}$$

and when the tree structure is unknown:

$$\bar{\rho}_T \leq \frac{|\mathcal{T}|m_{min}}{2}\frac{\log n}{n} + \frac{|\mathcal{T}|C_{PA}}{n} + \frac{C_{\mathcal{T}}}{n}, \quad \forall n \geq N, \tag{2.41}$$

where $C_{\mathcal{T}}$ is a constant which depends on the actual tree structure $\mathcal{T}$. ■

For clarity some additional definitions will be made:

Definition 2.21 (Nodes and Leaves):
For a tree T: $\mathcal{L}(T) \triangleq \mathcal{T}$ denotes the set of leaves, $\mathcal{L}_d(T)$ the set of leaves at depth d and $\mathcal{N}(T)$ the set of nodes or the set of all possible prefixes to $\mathcal{T}$. ■

Definition 2.22 (MDL-Context Tree):
The *Minimum Description Length (MDL) Context Tree* for a sequence is the context tree that, including tree description and parameter description costs, has the shortest corresponding codeword. The MDL-context tree may not be unique. ■

The definition of MDL was introduced in [Ris78]. Definition 2.22 is an extension applied to context trees, see also [Noh94].
In the sequel of this chapter sequential algorithms for estimation of unknown context trees with unknown parameters will be studied.

2.2.2 Algorithm CONTEXT

The Algorithm CONTEXT (AC) was first presented in [Ris83]. In e.g., [Noh94] applications to image compression were investigated. The most promising results for use on images where presented in [WRA96]. The following description will follow the latter reference.
The main idea of AC is to encode each symbol according to the so far "best" context model. This is done by comparing different context models by measuring the performance when test expansion/contraction of leaves are made in the context tree. The algorithm is therefore an adaptive algorithm that can be proven (e.g., see [Ris83], [Noh94]) to perform asymptotically optimal for context tree sources.

Algorithm 2.1 (Algorithm CONTEXT):
The algorithm CONTEXT uses a dynamic tree where each node, i, has an adaptive probability assignment function, $P(x|\hat{\boldsymbol{\theta}}_i)$, an efficiency variable, E_i, and a counter, n_i. It is the estimate of the parameter vector $\hat{\boldsymbol{\theta}}_i$ that is determined adaptively (e.g., by the Dirichlet estimator that is described in Section 2.5.1). Initially $E_i = 0$, $n_i = 0$, and $n_{Threshold}$ is a preselected threshold.

- Set the constants b_l and b_h, which are used as thresholds for a forgetting function when updating the efficiency variables.
- Initialize the tree with the root node.
- For each symbol to be encoded along with a corresponding context do the following:
  - Find the node along the context with $E_i < 0$, that is most distant from the root node, and where all nodes on the path to the root have efficiency variables less than 0. Use $P(x|\hat{\boldsymbol{\theta}}_i)$ in the found node for coding the symbol.
  - Update every node along the context from leaf to root.
  - If, in the leaf node i, it holds that $n_i > n_{Threshold}$ then expand the tree by adding new leaves to the previous leaf. Update the new leaf in the context.
- In order to update a node the following steps are carried out:
  - $E_i := w\left(E_i + \log \frac{P(x|\hat{\boldsymbol{\theta}}_j)}{P(x|\hat{\boldsymbol{\theta}}_i)}\right)$, where j corresponds to the father of i, and

$$w(x) = \begin{cases} b_h & \text{if } x > b_h, \\ x & b_l \leq x \leq b_h, \\ b_l & \text{if } x < b_l. \end{cases}$$

▷ Update $\hat{\boldsymbol{\theta}}_i$ according to the new symbol.

▷ $n_i := n_i + 1$

■

The original algorithm was proven in [Ris83] to be asymptotically optimal in terms of average redundancy for context tree sources. The main theorem of the [Ris83]-paper actually says that the algorithm works for any stationary ergodic source. It therefore holds for both FSMX and Markov sources. However, the algorithm does not, in the case when $M > 2$ necessarily find a context tree but a context model, i.e., not all inner nodes are fully extended. But this does not affect the final result of the algorithm. According to our definition of GMCM (Definition 2.15), Algorithm 2.1 will not find the context model of a source for all context model sources since there is a dependence between each node and its father and sons in the algorithm. Instead the algorithm will find a model that is an approximation of a GMCM with an over-fitted context model.

2.2.3 The Context Tree Weighting Algorithm

In this section follows a presentation of the basic algorithm of the Context Tree Weighting (CTW) algorithm along with some useful extensions. The main results of the CTW algorithm will also be presented[7].
The CTW algorithm does not, in contrast to AC, try to estimate the best context tree for each symbol to be encoded. The approach is to use a probability assignment technique via weighting. Consider the following example:

Example 2.5:
Consider all possible binary context trees with maximum depth 2 as shown in Figure 2.6.

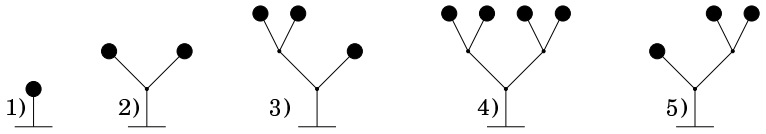

Figure 2.6: All possible binary context trees of maximum depth 2.

For each tree the probability estimation/assignment is done in the leaves. Thus we have the following block probabilities for each tree (where λ

[7]At the time of the writing of this thesis there is a web page [Wil97] describing the historical background of the CTW algorithm.

denotes the *empty* string corresponding to the root node):

$$P_1 = P_e^{\lambda}, \tag{2.42}$$
$$P_2 = P_e^{0} P_e^{1}, \tag{2.43}$$
$$P_3 = P_e^{00} P_e^{01} P_e^{1}, \tag{2.44}$$
$$P_4 = P_e^{00} P_e^{01} P_e^{10} P_e^{11}, \tag{2.45}$$
$$P_5 = P_e^{0} P_e^{10} P_e^{11}, \tag{2.46}$$

where P_e^s is the block probability for the probability assignment for the sequence in leaf s. By weighting the different trees with an associated weight, c_i, we get:

$$P_w = \sum_{i=1}^{5} c_i P_i = \tag{2.47}$$
$$= \frac{1}{2} P_e^{\lambda} + \frac{1}{8} P_e^{0} P_e^{1} + \frac{1}{8} P_e^{0} P_e^{10} P_e^{11} + \frac{1}{8} P_e^{00} P_e^{01} P_e^{1} + \frac{1}{8} P_e^{00} P_e^{01} P_e^{10} P_e^{11}.$$

■

The coefficients, c_i, in the example are chosen in a special way: $c_i = 2^{-|x_i|}$, where $|x_i|$ is the number of nodes and leaves in the tree i which are not at maximum depth. This choice of coefficients has both the property that $\sum c_i = 1$ and that $-\log c_i$ will correspond to a description cost for the tree i. The following algorithm is a simple way of describing a tree:

Algorithm 2.2 (Context Tree Description):
For an M-ary context tree with maximum depth $d_{max} \leq d$, run the following procedure starting with the root node:

- ▶ PROCEDURE Tree (node)
- ▶ IF node on a depth less than a predefined threshold d THEN
  - ▷ IF node is a leaf THEN
    - output the code symbol 0
  - ▷ ELSE
    - output the code symbol 1
    - FOR $i = 1$ TO M DO
      - ◦ Tree (son M)
    - END FOR
  - ▷ END IF

▶ END IF

▶ END PROCEDURE

■

It is obvious that the description cost for a tree T with this algorithm will be:

$$|\mathbf{x}_T| = |\mathcal{N}(T)| + |\mathcal{L}(T)| - |\mathcal{L}_d(T)|, \tag{2.48}$$

and in general it could also be formulated as:

Theorem 2.23:
For the output sequence $\mathbf{x}_T$, from the Algorithm 2.2, with the M-ary tree T as input, it holds that:

$$|\mathbf{x}_T| = \frac{M|\mathcal{L}(T)| - 1}{M - 1} - |\mathcal{L}_d(T)|. \tag{2.49}$$

□

Proof: The total number of bits required are $|\mathcal{N}(T)| + |\mathcal{L}(T)| - |\mathcal{L}_d(T)|$, i.e., one bit for each node and leaf except for those leaves on maximum depth.
Since the tree is complete we know that the number of leaves can be determined as a function of the inner number of nodes as: $|\mathcal{L}(T)| = |\mathcal{N}(T)|(M-1) - 1$. Thus,

$$\begin{aligned} |\mathbf{x}_T| &= |\mathcal{N}(T)| + |\mathcal{L}(T)| - |\mathcal{L}_d(T)| && (2.50)\\ &= \frac{|\mathcal{L}(T)| - 1}{M - 1} + |\mathcal{L}(T)| - |\mathcal{L}_d(T)| && (2.51)\\ &= \frac{M|\mathcal{L}(T)| - 1}{M - 1} - |\mathcal{L}_d(T)| && (2.52) \end{aligned}$$

■

By comparing the tree description cost in Theorem 2.23 with the coefficients in Example 2.5 it is possible to see the connection between the two. More formally we define the context tree weighting as:

Definition 2.24 (Context Tree Weighting):
The weighted block probability is defined as

$$P_w = \sum_{\mathcal{T} \in \mathcal{Q}_d} c_{\mathcal{T}} \prod_{t \in \mathcal{L}(\mathcal{T})} P_e^t, \tag{2.53}$$

where $\mathcal{Q}_d$ is the set of all possible context trees with maximum depth less than or equal to d and $c_{\mathcal{T}} = 2^{-|\mathbf{x}_{\mathcal{T}}|}$ where $\mathbf{x}_{\mathcal{T}}$ is the output from Algorithm 2.2. ■

In [WST95] a binary tree with binary alphabet was considered. As binary estimator the *Dirichlet estimator* (which is defined and analyzed in Section 2.5.1) was used. For this case the following was proved:

Theorem 2.25 ([WST95]):
For all sequences $\mathbf{x}^n$ of length n it holds for (2.53) with the Dirichlet estimator that:

$$\tilde{\rho}_{\mathcal{S}_d}(\mathbf{x}^n) \quad < \quad -\log c_{\mathcal{T}'} + |\mathcal{T}'|\gamma(\frac{n}{|\mathcal{T}'|}), \tag{2.54}$$

where

$$\gamma(z) \quad = \quad \begin{cases} z & \text{if } 0 \leq z < 1, \\ \frac{1}{2}\log(z) + 1 & \text{if } z \geq 1, \end{cases} \tag{2.55}$$

and $\mathcal{T}'$ is the MDL-tree, i.e.,

$$\mathcal{T}' \quad = \quad \arg \max_{\mathcal{T} \in \mathcal{S}_d} \left(c_{\mathcal{T}} \prod_{t \in \mathcal{L}(\mathcal{T})} P_e^t \right), \tag{2.56}$$

and $\mathcal{S}_d$ is the set of possible tree sources of maximum depth less than or equal to d. □

When considering the average redundancy and a source based on a GMCT-f we state the following:

Theorem 2.26:
For a CTW-scheme, using an asymptotically optimal probability assignment function PA, with respect to average redundancy, and for any source with tree structure $\mathcal{T}$ we state the following:

a) Under the same conditions as in Theorem 2.16 it holds:

$$\bar{\rho}_T(n) \geq (1-\epsilon)\frac{|\mathcal{T}|m_{min}}{2n}\log n, \tag{2.57}$$

when $n \to \infty$ and any $\epsilon > 0$.

b) For some $N > 0$ it holds:

$$\bar{\rho}_T(n) \leq \frac{1}{n}\left(\frac{|\mathcal{T}|m_{min}}{2}\log(n) + |\mathcal{T}|C_{PA} - \log c_{\mathcal{T}}\right), \quad \forall n \geq N, \tag{2.58}$$

where C_{PA} is a constant dependent on the PA-function.

□

Proof:
Part a) of the theorem follows directly from Theorem 2.16 and Corollary 2.17.

Part b) of the theorem is derived by observing that:

$$
\begin{aligned}
\bar{\rho}_T(n) = & \\
&= \int_{\boldsymbol{\theta}\in\Theta_{\mathcal{T}}} P_{\mathcal{T}}(\boldsymbol{\theta})\bar{\rho}_T(\boldsymbol{\theta})d\boldsymbol{\theta} \\
&= \int_{\boldsymbol{\theta}\in\Theta_{\mathcal{T}}} P_{\mathcal{T}}(\boldsymbol{\theta})\frac{1}{n}\sum_{\mathbf{m}\in\mathcal{M}^n} P_{\mathcal{T}}(\mathbf{m}|\boldsymbol{\theta}) - \log P_w(\mathbf{m}) + \log P_{\mathcal{T}}(\mathbf{m}|\boldsymbol{\theta})d\boldsymbol{\theta} \\
&= \int_{\boldsymbol{\theta}\in\Theta_{\mathcal{T}}} \frac{P_{\mathcal{T}}(\boldsymbol{\theta})}{n}\sum_{\mathbf{m}\in\mathcal{M}^n} P_{\mathcal{T}}(\mathbf{m}|\boldsymbol{\theta})\log P_{\mathcal{T}}(\mathbf{m}|\boldsymbol{\theta}) - \log\sum_{Q\in\mathcal{Q}_d} c_Q P_e^Q(\mathbf{m})d\boldsymbol{\theta} \\
&\leq \int_{\boldsymbol{\theta}\in\Theta_{\mathcal{T}}} \frac{P_{\mathcal{T}}(\boldsymbol{\theta})}{n}\sum_{\mathbf{m}\in\mathcal{M}^n} P_{\mathcal{T}}(\mathbf{m}|\boldsymbol{\theta})(\log P_{\mathcal{T}}(\mathbf{m}|\boldsymbol{\theta}) - \log(c_{\mathcal{T}}P_e^{\mathcal{T}}(\mathbf{m})))d\boldsymbol{\theta} \\
&= \frac{1}{n}\left(-\log c_{\mathcal{T}} + r(n)\right)) \\
&\leq \frac{1}{n}\left(\frac{|\mathcal{T}|m_{min}}{2}\log n + |\mathcal{T}|C_{PA} - \log c_{\mathcal{T}}\right), \qquad (2.59)
\end{aligned}
$$

where $P_e^{\mathcal{T}} = \prod_{t\in\mathcal{L}(\mathcal{T})} P_e^t$ and

$$
\begin{aligned}
r(n) = & \\
&= \int_{\boldsymbol{\theta}\in\Theta_{\mathcal{T}}} P_{\mathcal{T}}(\boldsymbol{\theta})\sum_{\mathbf{m}\in\mathcal{M}^n} P_{\mathcal{T}}(\mathbf{m}|\boldsymbol{\theta})(\log P_{\mathcal{T}}(\mathbf{m}|\boldsymbol{\theta}) - \log(P_e^{\mathcal{T}}(\mathbf{m})))d\boldsymbol{\theta} \\
&= \int_{\boldsymbol{\theta}\in\Theta_{\mathcal{T}}} P_{\mathcal{T}}(\boldsymbol{\theta})\sum_{\mathbf{m}\in\mathcal{M}^n} P_{\mathcal{T}}(\mathbf{m}|\boldsymbol{\theta})(\log P_{\mathcal{T}}(\mathbf{m}|\boldsymbol{\theta}) - \log(\prod_{t\in\mathcal{L}(\mathcal{T})} P_e^t))d\boldsymbol{\theta} \\
&= \int_{\boldsymbol{\theta}\in\Theta_{\mathcal{T}}} P_{\mathcal{T}}(\boldsymbol{\theta})\left(\sum_{t\in\mathcal{L}(\mathcal{T})}\bar{\rho}_t(\boldsymbol{\theta})\right)d\boldsymbol{\theta} \\
&\leq \int_{\boldsymbol{\theta}\in\Theta_{\mathcal{T}}} P_{\mathcal{T}}(\boldsymbol{\theta})\left(\sum_{t\in\mathcal{L}(\mathcal{T})}\frac{m_{min}}{2}\log n_t + C_{PA}\right)d\boldsymbol{\theta} \\
&= \sum_{t\in\mathcal{L}(\mathcal{T})}\frac{m_{min}}{2}\log n_t + C_{PA} \\
&\leq |\mathcal{T}|\frac{m_{min}}{2}\log n + |\mathcal{T}|C_{PA}, \qquad (2.60)
\end{aligned}
$$

where n_t is the number of observations and $\bar{\rho}_t(\boldsymbol{\theta})$ the average redundancy in node t. The second last inequality follows directly from Definition 2.20 for asymptotically optimal probability assignment together with the definition for average redundancy. ■

The computational complexity of calculating the weighted probability according to Definition 2.24 is not feasible for practical purposes because of the rapidly growing number of possible trees. The number of possible M-ary trees of depth less than or equal to D can recursively be calculated as:

$$X(D) = 1 + X^M(D-1), \tag{2.61}$$

where D is the maximum depth and $X(0) = 1$. This function grows as: $X(D) \geq 2^{M^{D-1}}$. In order to solve the problem of the exploding number of terms in the weighting efficiently the following algorithm was presented in [WST93]:

Algorithm 2.3 (CTW):
Start by constructing a tree with maximum depth d and where each node (or leaf) s initially has an estimated probability $P_e^s = 1$ and a weighted probability $P_w^s = 1$.

- For each symbol x to be encoded with context c update the tree from the leaf node pointed out by c to the root node according to:
  - Update $P_e^s := P_e^s P_{assign}(x|s)$.
  - Update $P_w^s := \frac{1}{2}\left(P_e^s + P_w^{s1} P_w^{s2} \dots P_w^{sM}\right)$ if s is a node and $P_w^s := P_e^s$ if s is a leaf.
  - Update the probability assignment function.

The output from the algorithm is the weighted probability in the root node, P_w^λ. ■

The algorithm updates $O(d)$ nodes for each symbol to be encoded compared to $O(X(d))$ (Equation 2.61) terms in Equation 2.53. To get the conditional probability for the next symbol from the CTW-algorithm a test update for each possible source symbol can be performed.
A related weighting method for context tree sources with (almost) arbitrary weights, i.e., choosing $c_{\mathcal{T}}$ in Equation 2.53 arbitrary, was presented in [Suz95]. The basis for the method is that the weighted probability is updated according to:

$$P_w^s := \beta(s) P_e^s + (1 - \beta(s)) P_w^{s1} P_w^{s2} \cdots P_w^{sM}, \tag{2.62}$$

where $0 \leq \beta(s) \leq 1$ denotes the a priori probability that s will be a leaf compared to being an inner node given that s will be in the tree. The weights $c_{\mathcal{T}}$ can not be really arbitrary since:

$$c_{\mathcal{T}} = \prod_{s \in \mathcal{N}(\mathcal{T})} (1 - \beta(s)) \prod_{s \in \mathcal{L}(\mathcal{T}) \cap \mathcal{L}_d(\mathcal{T})} \beta(s). \tag{2.63}$$

In conclusion we note that with the presented weighting techniques the cost or redundancy for describing a tree depends on the size of the tree and not on the sequence length. This means that the algorithm is asymptotically optimal with respect to the average redundancy.
After a sequence has been processed by the CTW-algorithm it is possible to find the MDL-context tree from the updated CTW-tree:

Algorithm 2.4 (CTW-MDL):
Recursively do the following, starting with the root node:

- ▶ PROCEDURE MDL (s)
- ▶ IF s is a leaf THEN
  - ▷ $P_m^s := P_e^s$
  - ▷ $I^s :=$ yes
- ▶ ELSE
  - ▷ FOR i = 1 TO M DO
    - MDL (si)
  - ▷ END FOR
  - ▷ $p := P_m^{s1} P_m^{s2} \cdots P_m^{sM}$
  - ▷ IF $\beta(s)P_e^s \geq (1-\beta(s))p$ THEN
    - $P_m^s := \beta(s)P_e^s$
    - $I^s :=$ yes
    - (remove all sons with subtrees)
  - ▷ ELSE
    - $P_m^s := (1-\beta(s))p$
    - $I^s :=$ no
  - ▷ END IF
- ▶ END IF
- ▶ END PROCEDURE

The output of the algorithm is the context tree with each node marked with I^s denoting whether the node is a possible leaf in the MDL-tree or not. The MDL-tree is the largest[8] tree where all nodes and leaves have $I^s =$ yes.

■

[8] With "largest" we refer to the tree with most number of nodes.

An important aspect when considering the CTW algorithm is to find the actual tree and parameter description costs. Since the CTW algorithm doesn't describe a single context tree no unique cost can be found. We will, however, estimate the tree and parameter description costs in two steps from the MDL-tree:

Algorithm 2.5 (Tree & Parameter Description Costs):
From the final updated CTW-tree, execute Algorithm 2.4 to get the MDL-tree then do the following:

- In the MDL-tree; set $\beta_s = 0$ if s is a node and $\beta_s = 1$ if s is a leaf. Recalculate the MDL-tree and recalculated P_m^λ denotes the block probability for the MDL-tree without the tree description cost.
- For each leaf recalculate the assigned block probability P_e^s with the "known" (i.e., best) parameters. Recalculate the MDL-tree. Then P_m^λ denotes the block probability for the MDL-tree without both tree and parameter description costs.

■

A similar method to find the MDL-tree online is the *Context Tree Maximizing Method*, e.g., see [VW95]. However, this method does not necessarily work for GMCT-f when the f-function is arbitrary or when extensions are applied (e.g., see Section 2.2.5). The method presented in the [VW95] was only considering a binary tree with binary alphabet. While using a Dirichlet estimator, it is possible to find the MDL-tree in an online algorithm. One of the nice properties shown in [VW95] with the CTW maximizing method was that it is possible to reduce the tree description cost as described in Algorithm 2.2. This reveals that it is possible to improve the bounds in Theorem 2.25 and Theorem 2.26.
Finally, an extension to the CTW algorithm will be made for applications on images. Since the images are bounded in a 2-dimensional rectangle the pixels on the boundary will lack some of the context pixels. This can be solved by extending each node with an additional parameter P_c which is a substitute for the updating of the sons. The following algorithm is basically the same as the CTW-algorithm but with the extension for missing contexts:

Algorithm 2.6 (CTW with missing contexts):
Start by constructing a tree with maximum depth d and where each node (or leaf) s initially has a estimated probability $P_e^s = 1$, a "cut" probability $P_c^s = 1$ and a weighted probability $P_w^s = 1$.

- For each symbol x to be encoded update the tree from the leaf node or the last node in the available context to the root node according to:
  - Update $P_e^s := P_e^s P_{assign}(x|s)$.
  - If node s is the last node in the available context update $P_c^s := P_c^s P_{assign}(x|s)$.
  - If s is a node update $P_w^s := \beta(s)P_e^s + (1-\beta(s))P_c^s P_w^{s1} P_w^{s2} \cdots P_w^{sM}$ or update $P_w^s := P_e^s$ if s is a leaf.
  - Update the probability assignment function.

The output from the algorithm is the weighted probability in the root node, P_w^λ. ■

The probability P_c is therefore a substitute for not updating any further down in the tree. This could be seen as updating according to one path from the cut-node with the same probability for each node. The difference is that the probability assignment functions will not be updated on greater depth in the tree.

2.2.4 The P-Context Algorithm

In [WS97] Weinberger and Seroussi introduced the concept of *permutation minimal trees*. The basic idea originates from applications in image compression and concerns a reduction of the parameter description cost. The idea is as follows: Given a tree source, consider all combinations of permutations of the alphabets in the leaves on a maximal depth D and their corresponding *minimal tree*. The minimal tree is found by merging all sibling leaves with equal parameter vectors. Since only tree sources are considered it is required that all siblings with the same father have the same parameter vector in order to make the merging. The smallest minimal tree is referred to as the permutation minimal tree.
The gain in finding the permutation minimal tree is that we will reduce the number of parameters to describe and in a restricted sense we will have a minimal number of parameters to describe. From Theorem 2.16 we know that the asymptotic parameter description cost, $\bar{\rho}_T(\boldsymbol{\theta})$, for a tree source with k leaves is bounded by

$$\bar{\rho}_T(\boldsymbol{\theta}) \geq (1-\epsilon)\frac{k(m-1)}{2n}\log n, \tag{2.64}$$

where m is the alphabet-size and n is the length of the sequence.
The traditional way of reducing the number of parameters is by prediction, e.g., see [TJR85], [ES96]. Although prediction gives very good

compression results, when, for example, combined with a universal encoding procedure as in Figure 2.2, it is not very appealing from a theoretical point of view. In particular, it is not clear why we should use the above mentioned two different model tools on the same information. The P-Context algorithm was introduced as a way to investigate this problem and to investigate how prediction and universal coding interact. However, the drawback with permutation minimal trees used in the P-context algorithm is that the actual permutation for each leaf must be described.

Example 2.6:
A binary tree source with a binary alphabet $\{a, b\}$, see Figure 2.7, has the following parameters: $P(a|\text{context } = 0) = \theta_0 = p$, $\theta_{10} = p$ and $\theta_{11} = 1 - p$. This tree is minimal, but if the binary alphabet in leaf 11 is permuted (i.e., $a' = b$ and $b' = a$), that leaf will get the new parameter $\theta_{11}^{'} = p$. All other leaves will remain with unchanged parameters. With this permutation we will get the minimal tree $\theta_{\lambda}^{'} = p$. This is also the permutation minimal tree, if we consider the permutation at depth $D \geq 1$. ■

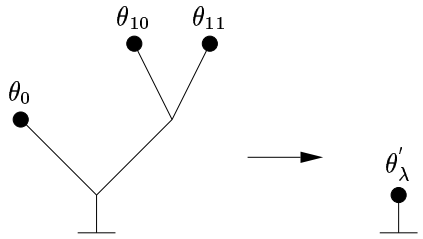

Figure 2.7: An example of a binary tree source along with its corresponding permutation minimal tree ($D \geq 1$) in Example 2.6. The parameters are $\theta_0 = \theta_{10} = p$, $\theta_{11} = 1 - p$ and $\theta_{\lambda}^{'} = p$. Observe that the permutation is done at depth D.

The algorithm P-Context, which is generalized in Figure 2.8, will asymptotically find, for almost all sequences, the permutation minimal tree. In Example 2.8 it is shown when it is possible for a prediction scheme to fail, and thus the P-Context algorithm. In [WS97] the following bound was derived:

$$\frac{1}{n} E_T(L(\mathbf{x}^n)) \leq H_n(T') + \frac{k'(m-1)}{2n} \log n + O(n^{-1}), \qquad (2.65)$$

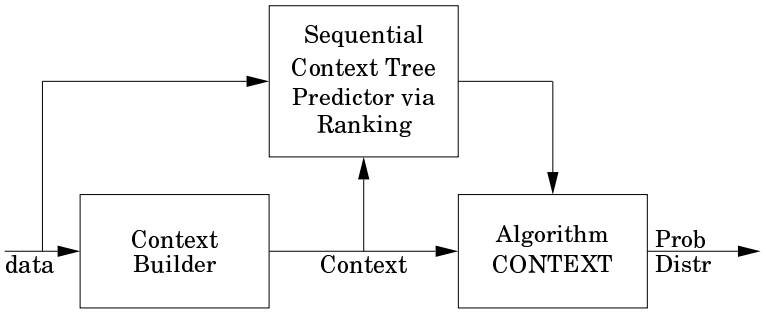

Figure 2.8: A general P-context algorithm.

where the left hand side is the expected code word length per input symbol for the tree source T with the permutation minimal tree source T'. On the right hand side is the entropy for the permutation minimal tree source $H_n(T')$ and k' is the number of leaves in the permutation minimal tree. From Figure 2.8 it is obvious that any asymptotically optimal context tree algorithm will have the same result as algorithm CONTEXT. The bound does, however, not hold for all sequences which will be shown in Example 2.8.
In [WS97] the bound (2.65) was also more accurately determined. Theorem 2 from [WS97]:

Theorem 2.27 ([WS97]):
Let T be an arbitrary, ergodic tree source. The expected code length $E_T\left(L(\mathbf{x}^n)\right)$ assigned by the P-Context algorithm to sequence $\mathbf{x}^n$ emitted by T satisfies

$$\frac{1}{n}E_T\left(L(\mathbf{x}^n)\right) \leq H_n(T') + \sum_{s\in S'} \alpha(s)\frac{\log n}{2n} + O(n^{-1}), \tag{2.66}$$

where $H_n(T')$ denotes the per-symbol binary entropy of n-vectors emitted by T, S' denotes the set of leaves of the permutations-minimal tree T' of T, $\alpha(s)$ is the minimum between $\alpha-1$ and the number of non-zero entries in the probability vector $\mathbf{p}(s)$, and the $O(n^{-1})$ term depends on T. □

In order to describe how the P-context algorithm works and its disadvantages we start by analyzing the *sequential prediction via ranking*. In the following algorithm RA denotes a vector with the ranking order of the alphabet, i.e., $RA[0]$ denotes the most (or one of the most) frequent symbol(s) and $RA[m-1]$ denotes the least (or one of the least) frequent symbol(s):

Algorithm 2.7:
Input: Sequence $\mathbf{x}^n = [x_0, x_1, \ldots, x_{n-1}]$, where $x_i \in \mathcal{M}$ and $|\mathcal{M}| = m$.

Output: The rank predicted sequence $\hat{\mathbf{x}}^n = [\hat{x}_0, \hat{x}_1, \ldots, \hat{x}_{n-1}]$, where $\hat{x}_i \in \{0, 1, \ldots, m-1\}$.

- ▶ Initialize the counters $t[i] = 0, i \in \mathcal{M}$.
- ▶ Initialize the RA to the lexicographical sorted alphabet $\mathcal{M}$. Position 0, i.e., $RA[0]$ will be equal to the first symbol in $\mathcal{M}$ and $RA[m-1]$ will be equal to the last symbol in $\mathcal{M}$.
- ▶ FOR $i = 0$ TO $n-1$ DO
  - ▷ $\hat{x}_i = ra(x_i)$, where $RA[ra(x_i)] = x_i$.
  - ▷ $t[x_i] := t[x_i] + 1$.
  - ▷ WHILE ($ra(x_i) > 0$ AND $t[x_i] > t[RA[ra(x_i) - 1]]$) DO
    - Swap positions $ra(x_i)$ and $ra(x_i) - 1$ in RA.
  - ▷ END WHILE
- ▶ END FOR

■

Example 2.7:
With Algorithm 2.7 we would like to get the rank predicted sequence from $[b, a, a, b, c, b]$ where the input alphabet is $\{a, b, c\}$. The result is shown in Table 2.2. The output sequence is: $[1, 1, 1, 1, 2, 1]$. We note that

Input	RA			Output
b	a:0	b:0	c:0	1
a	b:1	a:0	c:0	1
a	b:1	a:1	c:0	1
b	a:2	b:1	c:0	1
c	a:2	b:2	c:0	2
b	a:2	b:2	c:1	1
	b:3	a:2	c:1	

Table 2.2: The rank prediction algorithm. In the RA-column the order of the alphabet is given along with the counters for each symbol. The last row shows the RA after the encoding of the sequence.

it is possible to continue the encoding with any input symbols and thus it is possible to do the rank prediction online. ■

By using the sequential ranking predictor we may have trouble in the worst case as the following example shows:

Example 2.8:
In each leaf in the Context Tree Predictor (Figure 2.8) there is a sequential ranking predictor according to Algorithm 2.7. The output after the prediction is the index value of the input symbol from the leaf pointed out by the context.

Suppose for a ternary tree source we have the alphabet $\{a, b, c\}$. In the table below sample sequences occurring in some of the leaves are listed along with their rank predicted sequence[9]:

$$\begin{array}{lcl} [a,b,c]^\infty & \longrightarrow & [0,1,2]^\infty, \\ [c,b,a,a,b,c]^\infty & \longrightarrow & [2]^\infty, \\ [a,a,b,b,c]^\infty & \longrightarrow & [0,0,1,1,2]^\infty, \\ [b,a,a,b,c]^\infty & \longrightarrow & [1,1,1,1,2]^\infty. \end{array}$$

As this example shows, the rank predicted sequences may not at all have the desired properties, e.g., if the first two (or last two) sequences would occur in sibling nodes they could not be merged although the symbol sequences have the same symbol frequencies. ■

The example demonstrates that we have a possible problem with the worst case performance, e.g., we could get a tree expansion. If we consider short sequences, this effect might show itself in an undesirable way. Moreover, we would like to know the overhead, i.e., the size of the $O(n^{-1})$ term, of the bound in (2.65). In [ES98a] it was shown how to eliminate the sequence dependence by applying weighting techniques and an explicit bound was derived. The main theorem coincides with (2.65) and for practical purposes a two-pass algorithm was given. The following subsections will give the details.

A Two-pass P-Context Algorithm

We will consider a general case for tree sources where the alphabet size of the source is m and the alphabet size for the context is M. For example in [WRA96] and [ES98b], m=256 and M=2 was used. The prediction tree will have the depth D and the set of all possible prediction trees is denoted $\mathcal{A}$. The difference between the prediction trees is the alphabet mapping in the leaves.

By using weighting along with the CTW-algorithm, we get the weighted

[9] $[a, b, c]^\infty \triangleq [a, b, c, a, b, c, a, b, c, \ldots]$

probability for a sequence $\mathbf{x}^n$:

$$P_w^{\mathcal{A}}(\mathbf{x}^n) = \frac{1}{|\mathcal{A}|} \sum_{T \in \mathcal{A}} P_{ctw}^T(\mathbf{x}^n), \tag{2.67}$$

where

$$|\mathcal{A}| = (m!)^{|\mathcal{L}_D|}, \tag{2.68}$$
$$|\mathcal{L}_D| = M^D. \tag{2.69}$$

For this probability assignment we state the following theorem:

Theorem 2.28:
For the weighted probability in (2.67) we have the average redundancy (or description cost for the prediction tree, context tree and parameters), $\bar{\rho}_T$, bounded by:

$$\bar{\rho}_T \leq \frac{\rho_{pred} + \rho_{ctw}}{n} \tag{2.70}$$
$$\leq n^{-1} \left(\sum_{s \in \mathcal{L}} \log \frac{m!}{(m - \alpha(s))!} - \log(m!) \right) + \bar{\rho}_{T'}, \tag{2.71}$$

where $\alpha(s)$ is the effective alphabet-size in node s, $\mathcal{L}$ the set of leaves for the best prediction tree, i.e., the leaves at maximum depth D. The permutation minimal tree is denoted $\mathcal{T}'$ and the average redundancy for the CTW-algorithm is denoted $\bar{\rho}_{T'}$. □

Proof: We find the best prediction tree and the set of prediction trees that will perform equally well as:

$$\mathcal{A}_{T'}(\mathbf{x}^n) = \left\{ T \in \mathcal{A} : P_{ctw}^T(\mathbf{x}^n) = P_{ctw}^{T'}(\mathbf{x}^n) \right\}. \tag{2.72}$$

Thus a lower bound of the weighted probability is:

$$P_w^{\mathcal{A}}(\mathbf{x}^n) \geq \frac{|\mathcal{A}_{T'}(\mathbf{x}^n)|}{|\mathcal{A}|} P_{ctw}^{T'}(\mathbf{x}^n). \tag{2.73}$$

We get the description cost for the prediction tree as the difference between the total ideal code word length and that of the best CTW-tree without the cost for the prediction tree:

$$\rho_{pred}(\mathbf{x}^n) = -\log P_w^{\mathcal{A}}(\mathbf{x}^n) + \log P_{ctw}^{T'}(\mathbf{x}^n) \leq \log \frac{|\mathcal{A}|}{|\mathcal{A}_{T'}(\mathbf{x}^n)|}, \tag{2.74}$$

where

$$|\mathcal{A}_{T'}(\mathbf{x}^n)| \geq m! \prod_{s \in \mathcal{L}} (m - \alpha(s))! \tag{2.75}$$

With $\rho_{pred}(\mathbf{x}^n)$ solved and the bound for $\bar{\rho}_{T'}$ known from Theorem 2.26 the proof is complete. ■

The bound in Theorem 2.28 can be viewed as a consequence of the description of the prediction tree and the context tree along with parameters. We use this fact and construct a simple two pass algorithm:

Algorithm 2.8 (Two-pass P-context):
Do the following:

- Parse the input data and find the prediction tree.
- From the counters in the prediction tree find the best context tree, i.e., the tree with the shortest code word length including the description costs. The description of the context tree must be included in the beginning of the code word.
- Parse the data a second time and use the prediction tree and context tree found from the first scan. When new indexes arise in any of the leaves in the prediction tree, information about the original symbol must be sent.

■

We may omit the alphabet permutation description in one leaf, but if we use an occurrence sorted alphabet, as is the case for P-Context, we will increase the description cost from Theorem 2.28 by $\log M!$.

Improving the P-Context Algorithm

From the bound in Theorem 2.28 we note a possible high cost for the description of the alphabet permutations in the prediction tree. This will have especially bad effects when considering short data sequences. To decrease this description cost we would therefore also consider using a smaller prediction tree. We would then have to describe both the tree and the alphabet mapping in the leaves. In the P-Context algorithm only the mapping had to be described. In the same manner as in (2.67) we would like to use weighting over all possible prediction trees, $\mathcal{B}$, with all possible alphabet mappings:

$$P_w^{\mathcal{B}}(\mathbf{x}^n) = \sum_{Z \in \mathcal{B}} \frac{\beta(Z)}{|\mathcal{A}_Z|} \sum_{T \in \mathcal{A}_Z} P_{ctw}^T(\mathbf{x}^n), \tag{2.76}$$

where $\sum \beta(Z) = 1$ and $\beta(Z) > 0, \forall Z \in \mathcal{B}$. For this probability assignment a similar result to Theorem 2.28 can be derived:

Theorem 2.29:
For (2.76) the redundancy, $\bar{\rho}_T$, is bounded by:

$$\begin{aligned} n\bar{\rho}_T \quad &\leq \quad \text{“Prediction tree description cost”} \\ &+ \quad \text{“Mapping description cost”} \\ &+ \quad \text{“Context tree description cost”} \\ &+ \quad \text{“Parameter description cost”}, \end{aligned} \tag{2.77}$$

where the first three terms do not depend on the sequence length, n. The parameter cost has a $\log n$ behavior. □

A practical algorithm following this approach is a problem for future research.

2.2.5 Extended Context Trees

In the general scheme in Section 2.1 the prediction and context modeling where separated. In this section we will consider a way of extending the context tree algorithms to take care of the prediction as well. The basic idea was presented in [WRA96] and the following description is from [ES98b].
From an information theoretical perspective the prediction and probability assignment cannot be separated into independent parts since no prediction scheme will (in general) accomplish total decorrelation between the symbols. This was shown in e.g., [FM92]. In [ES98b] the notion *Prediction and Probability Assignment*, PPA, was used for describing a function that uses a combination of prediction and probability assignment in order to get down to a minimal number of parameters for a memoryless source with known probability distribution function where the parameters are not known. The natural extension of the CTW-algorithm is to use PPA instead in the probability assignment function, see Figure 2.9, and in the prediction/transformation scheme. This means that no prediction scheme as in Figure 2.2 is necessary. The extension with PPA of the CTW (or AC) algorithm does not affect its main function, i.e., providing a universal context modeling scheme. More about PPA can be found in Section 2.4 and 2.5.

2.3 Serializing Images

In this chapter some different approaches for serializing and context building for images will be presented. The serializing is strongly connected with the context building and the two must be constructed to-

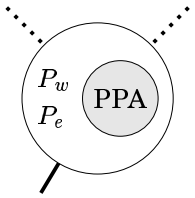

Figure 2.9: An extended CTW node which contains not only the probabilities P_w and P_e, but also an adaptive predictor and a probability assignment function, PPA.

gether and with full knowledge of each other in order to obtain maximum knowledge of the correlation between adjacent pixels.
In a recent paper [MNS00] the subject of serializing images has been investigated and the paper is a good complement to this section.
Note: In this thesis multi-resolution and successive resolution refinement techniques will not be considered. For this subject see for example [How93].

2.3.1 Methods for Serializing and Context Building

In this thesis some different scanning or serializing methods will be investigated. But it is the belief of the author that anyone who has tried compressing images will be able to come up with its own scanning method. It is not possible to determine or define a best strategy since the image data has a two-dimensional correlation which is unknown, but in the following the objectives and goals will be made clear and some reasonable solutions will be proposed.
Trying to formalize the objectives the following two goals are required for the serializer:

- It should be possible to take advantage of the local correlation between pixels.

- It should be possible to determine a similar located context of neighborhood pixels for every pixel except for pixels close to the boundary.

The first requirement comes from the discussion in Section 2.6 which implies that the correlation between pixels is local for most grayscale images. This may not be the case when considering for example half tone

images. To take advantage of the local correlation in the context modeling it is therefore necessary that the context can be built from neighborhood pixels as close as possible. The second goal is natural since it is important to have the same type of correlation and relations between every position in the context. The objectives are visualized in Figure 2.10. Note that it might be advantageous to select different kinds of context depending on the actual data in the neighborhood. However, this subject will not be addressed in this thesis.

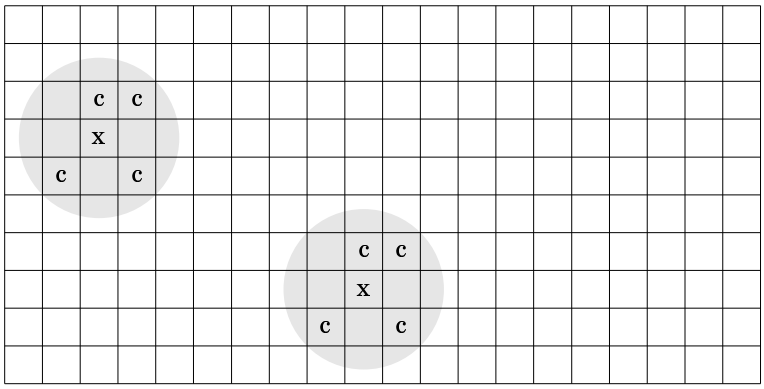

Figure 2.10: When serializing an image it is important that the pixel to be processed, x, with its neighborhood (shaded) and the context pixels, c, have the same relative positions for all pixels. The formation in this picture is not possible to use since there is no possible scanning-order to fulfill this requirement.

Raster Scan

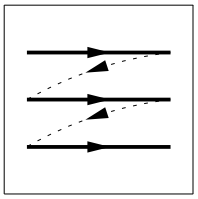

Figure 2.11: Raster scan.

The most common way of scanning images, e.g., [PM93], is simply by taking each row of pixels from top to bottom and scan it from left to

right, see Figure 2.11. Despite of its simplicity the method fulfills our goals with the scanning process. The interesting part is how to select the context from the neighborhood pixels. As shown in Figure 2.12 it is possible to use any of the previously scanned pixels. Two questions arise:

- In what order should the neighborhood pixels be selected for the context?
- Is the pixel order in the context independent of the actual data in the neighborhood?

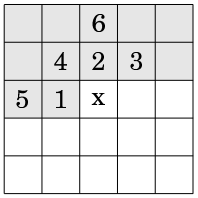

Figure 2.12: The shaded pixels are the available neighborhood pixels in raster scan. The numbers correspond to commonly appearing ordering of the pixels when used as a context. This ordering is based on distance to the x-point.

It was stated in [WRA96] that it would be reasonable to rank neighborhood pixel bits according to:

$$J(d, v) = \rho^d 2^{-v}, \tag{2.78}$$

where d is the distance to the pixel and v is the bit position. If the pixel value range is $[0, \ldots, m-1]$ then $v = 1$ is the most significant bit and $v = \log m$ is the least significant bit. The constant ρ depends on the type of image. The ranking is motivated by the fact that the correlation between pixels tends to decay exponentially with the distance. The ranking function was used in [WRA96] to determine the order in which the bits from neighborhood pixels would appear in a binary context. Note that this ranking does not result in an unique pixel order. In [Eks96] a slightly different scanning method was suggested where the scanning was actually done bitplane-wise instead of pixel-wise, see Figure 2.13. For some kind of images this method could prove useful since "forward" bits are available in the context. The drawback is the lack of more significant bits from the neighborhood. Another "drawback" is that the generic scheme in Figure 2.2 must be modified in the way that the transformation must be done before the serializing.

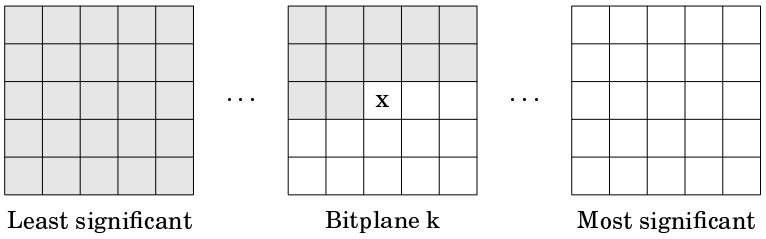

Figure 2.13: Scanning of the image on the basis of bitplanes.

From the above discussion some proposals for context selection will be made:

Proposal 2.1 (Raster scanning):
Scanning: Raster scan.
Context selection:

- Pixel context: The neighborhood pixels are sorted according to distance, e.g., the order in Figure 2.12.
- Binary context: Same as the previous but the context is taken from the individual bits in each pixel value with the least significant bit first.
- Sorted binary context: The context bits are selected according to the ranking function $J(d, v)$ in (2.78).

□

From the above proposals no selection method depends on the actual data which was one of the opening questions in Section 2.3.1.

Circular Scan

The strategy of the circular scan is the same as for raster scan as can be seen in Figure 2.14. The context can be found in a similar way as for raster scan, although rotated depending on the scan-direction. The difference between circular and raster scan is the number of times different context positions are unavailable due to the image boundary, see Table 2.3. The circular scan has in general more missing contexts than raster scan except for the first context position. This may under certain conditions be of importance. Another possible advantage is when the circular scanning is used on images with some kind of symmetric appearance, e.g., Figure 1.1, and together with an adaptive context model with forgetting.

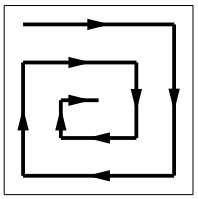

Figure 2.14: Circular scan.

Context position	**Raster scan**	**Circular scan**
1	h	1
2	w	$2w + 2h - 4$
3	$w + h$	$2w + 2h - 4$
4	$w + h$	$2w + 2h - 4$
5	$2h$	2
6	$2w$	$4w + 4h - 13$
$\sum$	$5(w + h)$	$10(w + h) - 22$

Table 2.3: Comparison of the unavailability of context pixels between raster and circular scan for an image of height h and width w.

The proposals for circular scan are the same as for raster scan except for the scanning:

Proposal 2.2:
Same as Proposal 2.1 but with circular scan. □

Peano-Hilbert Scan

To be able to take advantage of the local correlation and also to forget about the previously scanned and far away pixels in a simple way the Peano-Hilbert scan, Figure 2.15, appears to be an obvious choice, see e.g., [PR94]. The major drawbacks of this method is that it is not possible

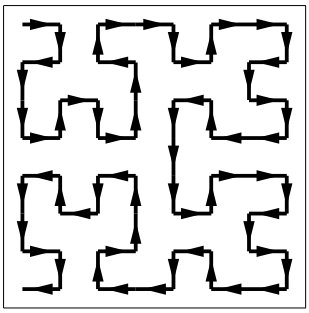

Figure 2.15: Peano-Hilbert scan.

to determine a similar context for all pixels and there are not as many close neighborhood pixels as for raster or circular scan. If the mutual relationship between adjacent pixels are not regarded this method could prove useful since the output data could actually be considered as serial data. From the serial data it would then be possible to apply traditional signal processing and data compression methods.

Proposal 2.3:
Scanning: Peano-Hilbert scan.
Context selection:

- Pixel context: The context is taken directly from the serial data in the order it appears from the Peano-Hilbert scan.
- Binary context: Same as the previous but the context is taken from the individual bits in each pixel value with the least significant bit first.
- Sorted binary context: The context bits are selected according to the ranking function $J(d, v)$ in 2.78.

□

Note that for the sorted binary context it is necessary to actually compute the distance to the individual pixels in the serial data stream.

2.3.2 Context Reduction Methods

In image compression, in contrast to for example text compression, the data is usually disturbed by noise[10]. The influence of the noise will be most noticed in the context modeling where observations will be spread out over more different paths in the context tree compared to a case without the influence of noise. This will in its turn result in fewer observations per node and thus a higher total parameter description cost. The possible gain with having more nodes is that each node could maybe more accurately determine its parameters if the input data were infinite. In Section 2.2.4 it was shown that prediction could reduce the number of parameters. This fact was also used in [ES96] to reduce the influence of noise by merging similar contexts together via prediction. In this section some different methods will be presented that will reduce the length of the context or that will merge contexts with similar characteristics together.

Context Merging via Prediction

One way of avoiding the influence of noise is to merge context with almost the same values together. Consider the following example:

Example 2.9:
We have the following three contexts with 4 pixel values each:

$$\begin{aligned} \mathbf{C}_1 &= [100, 101, 102, 99], \\ \mathbf{C}_2 &= [101, 102, 103, 100], \\ \mathbf{C}_3 &= [1, 2, 3, 0]. \end{aligned}$$

The only difference between the contexts is that they have different bias. Under some conditions these may be considered as the same context.
If we would like to perform a merging of these context we may subtract the (discretized) average value:

$$\begin{aligned} \mathbf{C'}_1 &= \mathbf{C}_1 - 100 = [0, 1, 2, -1], \\ \mathbf{C'}_2 &= \mathbf{C}_2 - 101 = [0, 1, 2, -1], \\ \mathbf{C'}_3 &= \mathbf{C}_3 - 1 = [0, 1, 2, -1], \end{aligned}$$

[10] Although one might consider spelling errors in written text as noise.

and all the contexts will be the same.
In many cases, however, it could be of interest to separate $\mathbf{C}'_3$ from the other two. ■

Another way to merge similarities together via prediction is to subtract a predicted value at each context position:

Example 2.10:
Same conditions as the previous example. For each position in the context a predicted value is calculated from the neighborhood:

$$\hat{\mathbf{C}}_1 = [101, 100, 100, 102], \tag{2.79}$$

the resulting context will thus be:

$$\mathbf{C}'_1 = \mathbf{C}_1 - \hat{\mathbf{C}}_1 = [-1, 1, 2, -3]. \tag{2.80}$$

The statistical behavior of the context values will be approximately the same as in Example 2.14. ■

Although the idea is to group similarities together in the same branches there will also be a loss of information. Another disadvantage with the second example is that a context model needs very long depths for some contexts and very short depths for others. To avoid this other ideas will be presented in the following subsections.

Context Reduction via Quantization

In Example 2.10 the context values will, for most images, be distributed according to the DE-distribution. This will, when used together with a context tree, result in heavy usage of contexts close around the zero value, and almost no usage at all for the contexts far away from the zero value. In order to make the usage more uniform and to be able to take advantage of the few samples in the outer range there was suggested a quantization method in [WRA96]. The idea is to do the following:

- For each context position subtract the value of the pixel to the left of the context pixel.
- The difference value is quantized in logarithmic sized intervals, e.g., for 8 bit data: [−255,−129], [−128,−65], [−64,−33], [−32,−17], [−16,−9], [−8,−5], [−4,−3], [−2,−1], [0,1], [2,3], [4,7], [8,15], [16,31], [32,63], [64,127], and [128,255].

Since there are 16 intervals we will have a four bit quantized context value instead of the original 8. By this approach we reduce the length of the context as well as we merge contexts that are similar and/or rarely used.

Context Reduction via Tree Transformations

A general approach to the context reduction problem is to have a mapping function that takes the original context and converts it into a new context with some desired property. In this way it would also be possible to have different context orderings depending on the data.

Example 2.11:
Consider a binary context tree source with binary source symbols and the parameters $\{\theta_{00} = p_1, \theta_{01} = p_2, \theta_{10} = p_1, \theta_{11} = p_2\}$.

By using a context mapping function that reorders the context positions 1 and 2 a universal context tree modeler would only have to estimate two parameters instead of all four which is necessary without the mapping function. ■

In order to utilize this method with a mapping function/tree transformation requires, however, very good or full knowledge of the source. In Section 2.2.4 one method was suggested with the P-Context algorithm. The final result in Theorem 2.28 was that it was possible to pay a constant penalty in order to reduce the number of unknown parameters. In a general case the optimal mapping function has similarities to asking as few questions as possible in a questionnaire.

2.3.3 Concluding Remarks About Serializing

There is no obvious way of selecting a (good) scanning method for images. Depending on the scanning method the statistical model will change and a comparison will be hard to evaluate. If we have a test set of images it would be possible to evaluate different scanning methods and statistical models on that set, but such a comparison may suffer from an unwanted bias due to the selected test set.

We will in this thesis use raster scan when not otherwise stated. We believe that raster scan is a reasonable method to use in combination with for example the CTW algorithm as discussed in this section. The main reason for this is due to the fact that the relative position of the context symbols will be the same throughout the image and we will thus have the same correlation between the context symbols and the observed data.

The subject of context reduction is further studied in Chapter 4 where an algorithm is presented that can compute a “good” context reduction.

2.4 Prediction

In Section 2.2.4 prediction was used as a tool for reducing the number of parameters. In Section 2.3.2 prediction was used to reduce the length the context string and merging similar paths together. In [ES96] the beneficial effects of prediction were shown for reducing the total number of parameters in the context modeling.
This chapter focuses on possible prediction methods for image data and the discrepancy between the the original data and the prediction. In analogy with Sections 2.2.4 and 2.3.2 it is the possible parameter reduction that will be of interest.

2.4.1 Prediction – Basics

It has been shown that no prediction scheme can accomplish total decorrelation between symbols, [FM92] and [FM94]. To show one of the main problem occurring in image compression consider the following example:

Example 2.12:
A ternary tree source with ternary alphabet, $\mathcal{A} = \{0, 1, 2\}$, has the following parameter vectors: $T = \{[0, \boldsymbol{\theta}_0], [1, \boldsymbol{\theta}_1], [2, \boldsymbol{\theta}_2]\}$, where $\boldsymbol{\theta}_0 = [\frac{1}{10}, \frac{2}{10}, \frac{7}{10}]$, $\boldsymbol{\theta}_1 = [\frac{2}{10}, \frac{3}{10}, \frac{5}{10}]$, and $\boldsymbol{\theta}_2 = [\frac{3}{10}, \frac{4}{10}, \frac{3}{10}]$, denote the symbol probabilities. As context the previous symbol is used.
A prediction scheme with full knowledge of the tree source re-map the tree source alphabet to a new ternary alphabet, $\mathcal{B} = \{a, b, c\}$, according to: a for the most probable symbol, c for the least probable and b otherwise.
The output symbols after the prediction are to be encoded by an entropy encoder, see Figure 2.16. One possible gain with the prediction scheme is to avoid the usage of a context model together with the entropy encoder, e.g., Figure 2.17. In this example, however, it is necessary to have knowledge of all the previous symbols in order to have an output rate equal to the entropy of the tree source, thus a memoryless estimator could not be used. ■

Example 2.13:
The same conditions as in Example 2.12, but the tree source will have the following parameter vectors: $\boldsymbol{\theta}_0 = [\frac{1}{10}, \frac{2}{10}, \frac{7}{10}]$, $\boldsymbol{\theta}_1 = [\frac{2}{10}, \frac{1}{10}, \frac{7}{10}]$, and $\boldsymbol{\theta}_2 = [\frac{7}{10}, \frac{1}{10}, \frac{2}{10}]$. For this source it is possible for the entropy encoder to use the probability distribution: $P(a) = \frac{7}{10}$, $P(b) = \frac{2}{10}$, and $P(c) = \frac{1}{10}$ for all symbols and thus no context modeling is necessary, i.e., as in Figure 2.16.

■

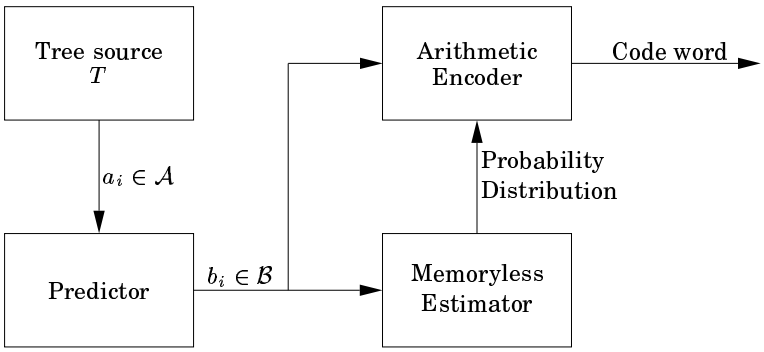

Figure 2.16: A combination of prediction, estimator and entropy encoder (e.g., arithmetic encoder) for encoding of the tree source T.

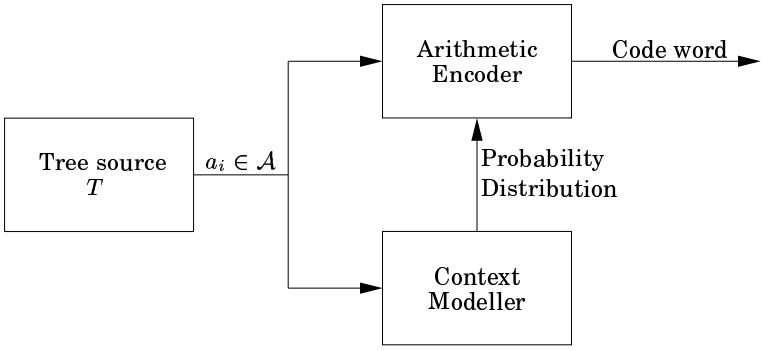

Figure 2.17: A combination of context modeling and an arithmetic encoder for encoding of the tree source T.

In the above examples all parameters where known and there is no possible gain in compression performance. If the tree source is unknown and a universal context modeling scheme is to be used, it could be of major importance to use a prediction scheme in order to reduce the number of parameters. In Example 2.13 the number of parameters would have been reduced from 6 to 2. In image compression it is usually possible to find a distribution with only two parameters which is, in most applications, considerably less than the source alphabet size M.

Example 2.14:
As a simple prediction scheme for images the prediction value is the pixel value left of the pixel to be encoded. Figure 2.18 shows the resulting probability distribution when the prediction scheme is applied to the image in Figure 1.1 which has $M = 256$. The typical probability distribu-

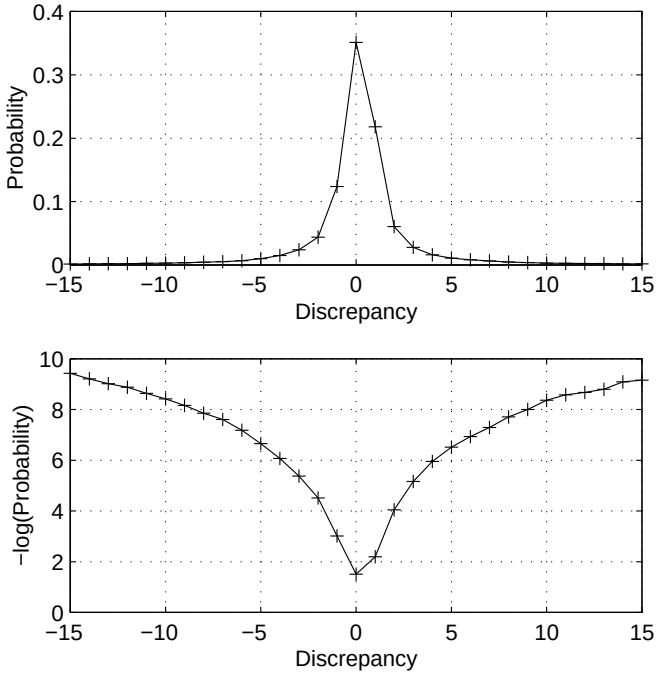

Figure 2.18: The upper graph is the probability distribution of the discrepancy between actual and predicted value when a simple prediction scheme (mode 1 for the JPEG standard, see Table 2.4) is applied to a medical image (thorax). The lower graph shows the negative logarithm of the probability distribution.

tion, when as in this example a simple prediction scheme is used, tends for medical images to be *double exponential* (DE)[11] with the mean value m and distribution parameter s:

$$f_{DE}(x) = \frac{1}{2s}\exp(-\frac{|x-m|}{s}), \quad s > 0, \tag{2.81}$$

and for more noisy photographic pictures the typical probability distribution tends to be Gaussian. The number of parameters for the double exponential or Gaussian distributions is only 2 in contrast to the 255 parameters necessary to describe the alphabet probability distribution. ■

Some comments to Figure 2.18:

- First of all it should be noted that the DE-distribution is only an approximation of the discrepancy.
- The variance of the distribution varies throughout the image and therefore we will not get the expected straight lines in the right graph in the figure.
- Finally, since some parts of the image contain data close to zero the distribution will not be symmetric for this example.

However, it should be noted that the distribution is dependent of the neighborhood pixels for the same reason as the symbols were dependent in Example 2.12. Thus it is necessary to describe 2 parameters in each context. Finally it is possible to describe only one parameter since most prediction schemes will, for these symmetric distributions, have an average discrepancy (between predicted and actual value) equal to zero.
This section ends with a definition of PPA:

Definition 2.30 (Prediction and Probability Assignment):
For a class of sources $\mathcal{S}$ a jointly optimized function, with respect to redundancy, of both prediction and probability assignment is denoted PPA($\mathcal{S}$) or just PPA. ■

The fact that prediction and probability assignment are jointly optimized is of major importance when adaptive algorithms are used since the two may otherwise work against each other. Important is also that the probability assignment takes full advantage of the data according to the probability distribution after the prediction. Finally, it should be noted that in the prediction scheme there are also parameters that will have an associated cost.

[11]The notation is from [Leh91]. In, for example [Mat96], this distribution is called two-sided exponential. In the image compression community it is often called Laplacian [How93].

2.4.2 Some Simple Predictors

In the lossless JPEG standard [PM93] a set of linear predictors are used, see Table 2.4. In principle any of the 7 predictors or no prediction can be

Mode	Prediction value: $\hat{x}(i,j)$
0	0 (No prediction)
1	$x(i-1,j)$
2	$x(i,j-1)$
3	$x(i-1,j-1)$
4	$x(i,j-1)+x(i-1,j)-x(i-1,j-1)$
5	$x(i,j-1)+(x(i-1,j)-x(i-1,j-1))/2$
6	$x(i-1,j)+(x(i,j-1)-x(i-1,j-1))/2$
7	$(x(i,j-1)+x(i-1,j))/2$

Table 2.4: The predictors in the (old) lossless JPEG. Raster scan is assumed and upper left corner is the coordinate (0,0).

used for any symbol. But the overhead of describing $\log 8 = 3$ bits for each symbol may not be motivated. Another approach could be to adaptively change the predictor according to the compression performance.
No matter which of the linear predictors is used the difference between actual and predicted value will behave essentially as in Example 2.14. The subtle difference between the predictors is the variance of the discrepancy. The variance will depend on the correlation between the pixels, which may change throughout the image. In [Eks95] a technique was presented to take care of the changing correlation by splitting the image into similar regions. This was done by using a quad-tree scheme. This approach was derived from [Noh94]. The method in [Eks95] was further developed in [Eks96] where the set of predictors was also extended to be of the more general form:

$$\hat{x}(i,j) = \frac{w_1 x(i-1,j) + w_2 x(i,j-1) + w_3 x(i-1,j-1) + w_4 x(i+1,j-1)}{w_1 + w_2 + w_3 + w_4}, \tag{2.82}$$

where $w_k \in \{1, 2, \cdots, 256\}$, $k = 1, 2, 3, 4$, and $\sum w_k > 0$. The coefficients w_k were in [Eks96] updated for each pixel to be encoded according to an pseudo optimization based on the neighborhood.
In contrast to the above linear predictors more *ad hoc* and empirical prediction techniques have been presented. In [MRS94] a foundation for non-linear prediction techniques was stated. In [MS95] some different, both linear and non-linear, prediction techniques were evaluated. The conclusion that can be made from these articles, is that a more accu-

rate/precise goal must be set for the prediction schemes since the prediction is not a goal by itself. The only thing that really matters is the code word length. For the time being, the ad hoc prediction schemes rules. We will further discuss this topic in Chapter 3.

2.4.3 Linear Prediction

As a way of finding good prediction schemes, linear prediction with real valued coefficients has been studied in e.g., [WRA96] and [ES98b]. The prediction, which is calculated as

$$\hat{x}_i = x_{i-1} + \sum_{k=1}^{r} a_k (x_{i-1-k} - x_{i-1}), \tag{2.83}$$

where $a_k \in \mathbb{R}$, $k = 1, 2, \cdots, r$, has nice mathematical properties and have been studied intensively in signal processing and control theory. A usual way of determining the coefficients, a_k, is by minimizing the quadratic sum of the errors between the predicted and actual values. If the prediction scheme is applied on images we have noted in the previous sections that the discrepancy usually tends to be double exponentially distributed. The variance for the DE-distribution is $2s^2$ and from statistical theory we know that it is possible to determine an unbiased estimation of the variance as:

$$\frac{1}{n-1}\left[\sum_{i=1}^{n}(x_i - \hat{x}_i)^2 - \frac{1}{n}\left(\sum_{i=1}^{n}(x_i - \hat{x}_i)\right)^2\right], \tag{2.84}$$

which can be simplified if the average is zero:

$$\frac{1}{n-1}\left[\sum_{i=1}^{n}(x_i - \hat{x}_i)^2\right], \tag{2.85}$$

thus the variance depends on the quadratic error between the predicted and actual values. The continuous entropy for a DE-distributed variable can be found to be:

$$-\int_{-\infty}^{\infty} f_{DE}(x) \log f_{DE}(x) dx = 1 + \frac{1}{\ln 2} + \log s. \tag{2.86}$$

To conclude, the entropy for a DE-distributed variable will decrease with s, $s > 0$. To minimize s, the variance must be minimized which leads to that the quadratic error must be minimized. The same will hold if we consider a Gaussian distribution.

In [WRA96] the linear predictor was used together with the Algorithm CONTEXT. The prediction coefficients were assumed to change depending on the neighborhood, and thus, it comes natural to have one predictor for each node in the AC-tree. This was further examined in [ES98b] by using PPA in each node of a CTW-tree. By using either of these two methods for prediction it is possible to utilize a linear prediction scheme and also to have different prediction coefficients depending on the context.

2.5 Probability Assignment Techniques for Memoryless Sources

For the lossless image compression in this chapter we focus on three different kinds of statistical models for the data: binary, m-ary alphabet, and double exponential. In this section we will present and analyze different probability assignment (or code word length assignment) techniques for universal coding of these different statistical models. Also, some adaptive estimation techniques will be presented for comparison. In order to clarify the subtle difference between assignment and estimation, consider Section 2.1.3 and the following example:

Example 2.15:
A random variable X has the probability distribution $f_X(x|s)$ which depends on the unknown parameter s. For online coding of the parameter the different techniques would do the following:

- **Assignment:** This technique will use a coding probability distribution for the next symbol, $g_X(x_{n+1}|\mathbf{x}^n)$, which will have a redundancy contribution (or parameter description cost) that matches the a priori assignment of cost for the actual parameter. The function $g_X(\cdot)$ may differ from $f_X(\cdot)$.
- **Estimation:** This method will estimate the parameter $\hat{s}$ that according to some measure will be the best. The coding probability distribution for the next symbol is therefore equal to $f_X(x|\hat{s})$.

■

The motivation for using assignment techniques in data compression is that it is a powerful way of actually controlling the parameter cost for individual sequences. With estimation techniques it is also possible to control the parameter cost, but generally only on the average over all possible sequences. The two techniques may in some cases actually be the same. By using the weighting technique we will have a powerful tool for making probability assignment with controlled redundancy. With weighting it is also possible to get a sequence order independence.

2.5.1 Binary Data

For binary data there is only one unknown parameter, θ. In the sequel the following is assumed: $P(1) = \theta$ and $P(0) = 1 - \theta$.
One way of estimating the parameter for the next symbol, $\hat{\theta}_{n+1}$, given a past sequence, $\mathbf{x}^n$, is by the following α-biased estimator:

$$\hat{\theta}_{n+1} = \frac{t_1(\mathbf{x}^n) + \alpha}{n + 2\alpha} = 1 - \frac{t_0(\mathbf{x}^n) + \alpha}{n + 2\alpha}, \tag{2.87}$$

where $t_0(\mathbf{x}^n)$ denotes the number of zeroes in the sequence $\mathbf{x}^n$ and $t_1(\mathbf{x}^n)$ the number of ones. The constant α could be chosen in different ways and it is set according to the a-priori knowledge of the parameter distribution.

Definition 2.31:
If $\alpha = 1/2$ in (2.87) the estimator is called the *Dirichlet estimator* and for $\alpha = 1$ it is called the *Laplace estimator*. ■

One thing to note about this α-biased estimator is that it is sequence order independent, which is a useful property. It could also be shown that both the Dirichlet and Laplace estimators are the result of using weighting techniques, and thus, they are not only estimators but they also fit under the definition of probability assignment functions. An alternative definition of the Dirichlet and Laplace estimators are therefore based on the block probability:

$$P_e(a, b) = \int_{\theta=0}^{1} w(\theta)(1 - \theta)^a \theta^b d\theta, \tag{2.88}$$

where $P_e(a, b)$ is the block probability for a zeroes and b ones and $w(\theta)$ is the a priori probability distribution for θ. For the Dirichlet estimator the prior is:

$$w(\theta) = \frac{1}{\pi\sqrt{(1 - \theta)\theta}}, \tag{2.89}$$

and for Laplace:

$$w(\theta) = 1. \tag{2.90}$$

In [KT81] it was shown that universal codes constructed from the Dirichlet estimator are asymptotically optimal with respect to average redundancy. For individual sequences we follow the presentation in [WST95] where the following was shown:

Theorem 2.32 ([WST95]):
For the block probability from the Dirichlet estimator, $P_e(a,b)$, it holds for all sequences with a zeroes and b ones ($a+b>0$):

$$\frac{1}{2}I(a,b) \leq P_e(a,b) \leq \sqrt{\frac{2}{\pi}}I(a,b), \tag{2.91}$$

where

$$I(a,b) = \frac{1}{\sqrt{a+b}}\left(\frac{a}{a+b}\right)^a\left(\frac{b}{a+b}\right)^b. \tag{2.92}$$

□

Stated in worst case redundancy for individual sequences we get:

Corollary 2.33 ([WST95]):
For the Dirichlet estimator it holds:

$$\tilde{\rho}_S(x(a,b)) \leq \frac{1}{2}\log(a+b)+1, \tag{2.93}$$

where S is a binary memoryless source and $x(a,b)$ is an arbitrary sequence with a zeroes and b ones ($a+b>0$). □

Proof: From Equation 2.91 we have:

$$P_e(a,b) \geq \frac{1}{2}\frac{1}{\sqrt{a+b}}\left(\frac{a}{a+b}\right)^a\left(\frac{b}{a+b}\right)^b. \tag{2.94}$$

The worst case redundancy is:

$$\begin{aligned}
\tilde{\rho}_S(x(a,b)) &= -\log P_e(a,b) + \log\left(\left(\frac{a}{a+b}\right)^a\left(\frac{b}{a+b}\right)^b\right) && (2.95)\\
&\leq -\log\frac{1}{2\sqrt{a+b}} && (2.96)\\
&= \frac{1}{2}\log(a+b)+1. && (2.97)
\end{aligned}$$

■

2.5.2 m-ary Alphabet

For m-ary alphabet coding, i.e., $m > 2$, it is possible to use a similar approach as in the binary case for assigning probabilities. The Dirichlet estimator has the following general form for arbitrary alphabet size:

Definition 2.34 (Dirichlet Estimator):
For the Dirichlet estimator the next symbol probability for symbol a given a past sequence $\mathbf{x}^n$ is defined as:

$$P(a|\mathbf{x}^n) = \frac{t_a(\mathbf{x}^n) + 1/2}{n + m/2}, \tag{2.98}$$

where m is the alphabet size and $t_a(\mathbf{x}^n)$ denotes the number of occurrences of symbol a in the sequence $\mathbf{x}^n$. ■

In correspondence to the binary case, it could also be possible to have a general α-biased form, i.e., $P(a|\mathbf{x}^n) = \frac{t_a(\mathbf{x}^n)+\alpha}{n+m\alpha}$, although the possible gain of adjusting the parameter α is hard to determine in the general case, and may in most cases be replaced by a more specialized assignment techniques. However, we note that the Dirichlet estimator for m-ary coding performs optimal in terms of average redundancy, e.g., see [KT81], [Sht87] and [STW95].

2.5.3 Double Exponential Data

In image compression it is often possible to model the data by a double exponential (DE) distribution, see e.g., Example 2.14 and [WRA96]. Since the DE-data is produced as the residual between the image data and some prediction scheme the PPA concept (Definition 2.30) becomes useful. In this section some possible methods for probability estimation and assignment of the DE-data will be presented.
The continuous DE-probability distribution is defined as:

$$f_{DE}(x|\hat{x}, s) = \frac{1}{2s} \exp(-\frac{|x - \hat{x}|}{s}), \tag{2.99}$$

where $s > 0$, and $\hat{x}$ is the mean value. However, it is of interest to consider the distribution for discrete data, i.e., a finite alphabet $\{0, 1, \ldots, m-1\}$:

Definition 2.35:
A discrete probability density function, p_{DE}, for the DE-distribution over the finite alphabet $\{0, 1, \ldots, m-1\}$, is defined as:

$$\begin{aligned} p_{DE}(x|\hat{x}, s) &= \\ &= \lim_{\Delta \to 0} \frac{f_\Delta(x|\hat{x}, s)}{\sum_{k=0}^{m-1} f_\Delta(k|\hat{x}, s)} \\ &= \frac{\exp(-\frac{|x-\hat{x}|}{s})(1 - \exp(\frac{1}{s}))}{\exp(\frac{-\hat{x}}{s}) + \exp(\frac{\hat{x}-m+1}{s}) - \exp(\frac{1-\delta}{s}) - \exp(\frac{\delta}{s})}, \end{aligned} \tag{2.100}$$

where $\delta = \hat{x} - \lfloor \hat{x} \rfloor$, and

$$f_\Delta(x|\hat{x}, s) = \int_{t=x-\Delta/2}^{x+\Delta/2} f_{DE}(t|\hat{x}, s)dt. \tag{2.101}$$

■

It should also be mentioned that in LOCO-I [WSS96] a slightly different approach was taken. Instead of the DE-distribution a two-sided geometrical distribution was used:

$$f_{TSG}(x|\theta, d) = C(\theta, d)\theta^{|x+d|}, \tag{2.102}$$

where

$$C(\theta, d) = (1-\theta)/(\theta^{1-d} + \theta^d), \tag{2.103}$$

is a normalizing factor, $0 < \theta < 1$, and $0 \leq d \leq 1$. This distribution is more restricted than the suggested DE-distribution, but for applications such as LOCO-I ([WSS96]) it will be possible to find a low complexity solution for universal coding, see further in [MSW96].
In this section a discussion on four different methods for probability estimation/assignment for the DE-distribution will be made:

A. Estimation by making an unbiased estimate of the parameter s, i.e.,

$$s_{e1}(\mathbf{x}^n, \hat{\mathbf{x}}^n) = \sqrt{\frac{1}{2(n-1)}\left[\sum_{i=1}^{n}(x_i - \hat{x}_i)^2 - \frac{1}{n}\left(\sum_{i=1}^{n}(x_i - \hat{x}_i)\right)^2\right]}, \tag{2.104}$$

and under the assumption that the mean value is 0 the estimate simplifies to:

$$s_{e2}(\mathbf{x}^n, \hat{\mathbf{x}}^n) = \sqrt{\frac{1}{2(n-1)}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2}. \tag{2.105}$$

B. Estimation by finding the parameter s that will have the shortest code word length, i.e.,

$$s_{e3}(\mathbf{x}^n, \hat{\mathbf{x}}^n) = \arg\min_{s>0}\left\{-\log\left(\prod_{i=1}^{n} p_{DE}(x_i|\hat{x}_i, s)\right)\right\}. \tag{2.106}$$

C. Assignment technique based on weighting.

D. Assignment technique based on *local optimization*.

In the following section some minor notes about the different techniques will be made.

A. Estimation 1 and 2 (e_1 and e_2)

One of the most simple ways of estimating the parameter for the DE-distribution is by making an unbiased estimate according to s_{e1} (Eq. 2.104) or s_{e2} (Eq. 2.105). Some of the disadvantages with these two methods are:

- It is an approximation since we will use a normalized discrete probability distribution $p_{DE}(\cdot)$ and not a continuous infinite function. This will be especially critical when the center value is close to either 0 or m, i.e., the outer part of the range.
- Since the estimation is made on the parameter for each symbol to be encoded, each symbol will be given a probability according to the DE-function and the corresponding estimate of the parameter. In this way there is no possible way of controlling the parameter cost, and, the scheme will be sequence order dependent.

B. Estimation 3 (e_3)

Compared to the estimations techniques 1 & 2 a more complex way of estimation the parameter s is by finding the parameter that will minimize the code word length according to s_{e3} (Eq. 2.106). This estimation technique will have the same drawbacks as estimation techniques 1 & 2 and it will not be as simple to calculate. However, it is possible to make an approximation in order to find a simple solution. We start by rewriting the expression for $s_{e3}(\mathbf{x}^n, \hat{\mathbf{x}}^n) = s_{e3}$:

$$\begin{aligned} s_{e3} &= \arg\min_{s>0} \left\{ -\log\left(\prod_{i=1}^{n} p_{DE}(x_i|\hat{x}_i, s) \right) \right\} \\ &= \arg\max_{s>0} \left(\prod_{i=1}^{n} p_{DE}(x_i|\hat{x}_i, s) \right) \\ &= \arg\max_{s>0} \left(\prod_{i=1}^{n} \frac{\exp(-\frac{|x_i-\hat{x}_i|}{s})(1-\exp(\frac{1}{s}))}{d_i} \right), \end{aligned} \tag{2.107}$$

where

$$d_i = \exp(\frac{-\hat{x}_i}{s}) + \exp(\frac{\hat{x}_i - m + 1}{s}) - \exp(\frac{1-\delta_i}{s}) - \exp(\frac{\delta_i}{s}), \tag{2.108}$$

and $\delta_i = \hat{x}_i - \lfloor \hat{x}_i \rfloor$. The expression for s_{e3} due to the product of dominators, $\prod d_i$, is infeasible to calculate. One possible solution to approximate the

expression could be to omit the normalizing factor, i.e.,

$$\tilde{s}_{e3} = \arg\max_{s>0} \left(\prod_{i=1}^{n} \frac{1}{2s} \exp(-\frac{|x_i - \hat{x}_i|}{s}) \right) = \frac{1}{n} \sum_{i=1}^{n} |x_i - \hat{x}_i|. \tag{2.109}$$

The approximation is worst when the center value, i.e., $\hat{x}_i$, is close to either of the boundaries 0 or $m-1$.

C. Weighting Technique

An assignment technique for the DE-data based on weighting was presented in [ES98b]. The idea is simple, we start by considering the block probability:

$$P_{DE}(\mathbf{x}^n|\hat{\mathbf{x}}^n, s) = \prod_{i=1}^{n} p_{DE}(x_i|\hat{x}_i, s). \tag{2.110}$$

From the block probability the weighted block probability is defined as:

$$P_{wDE}(\mathbf{x}^n|\hat{\mathbf{x}}^n) = \int_{s\in\mathcal{S}} \alpha(s) P_{DE}(\mathbf{x}^n|\hat{\mathbf{x}}^n, s) ds, \tag{2.111}$$

where $\mathcal{S} = (0, \infty)$, $\int_{s\in\mathcal{S}} \alpha(s) ds = 1$, and $\alpha(s) \geq 0$, $\forall s \in \mathcal{S}$. One requirement for being able to use this approach is that the block probability $P_{DE}(\cdot)$ can be expressed by a closed form expression. This is, however, not possible because of the denominator in $p_{DE}(\cdot)$, e.g., the same reason as why s_{e3} in the previous section was not possible to calculate. In [ES98b] a simplified approach was proposed where the weighting was done over a finite number of values of the parameter, i.e.,

$$\tilde{P}_{wDE}(\mathbf{x}^n|\hat{\mathbf{x}}^n) = \sum_{i=1}^{N} \alpha_i P_{DE}(\mathbf{x}^n|\hat{\mathbf{x}}^n, s_i), \tag{2.112}$$

where $\sum_{i=1}^{N} \alpha_i = 1$, and $\alpha_i \geq 0$, $s_i > 0$, $\forall i \in \{1, 2, \ldots, N\}$. For this weighting it is possible to make some estimate on the worst case redundancy:

Theorem 2.36:
For the weighted block probability in (2.112) the worst case redundancy for individual sequences of length n satisfies:

$$\tilde{\rho}_{DE}(\mathbf{x}^n|\hat{\mathbf{x}}^n) \leq -\log \alpha_a + \log \frac{P_{DE}(\mathbf{x}^n|\hat{\mathbf{x}}^n, s^*)}{P_{DE}(\mathbf{x}^n|\hat{\mathbf{x}}^n, s_a)}, \tag{2.113}$$

where

$$a = \arg\max_{i\in\{1,2,\ldots,N\}} \alpha_i P_{DE}(\mathbf{x}^n|\hat{\mathbf{x}}^n, s_i), \tag{2.114}$$

$$s^* = \arg\max_{s>0} P_{DE}(\mathbf{x}^n|\hat{\mathbf{x}}^n, s). \tag{2.115}$$

□

The theorem tells us that the worst case redundancy depends on two different things: parameter description from the weighting coefficient and a parameter mismatch. The proof is straight forward:

Proof: From (2.112) we get:

$$\begin{aligned}\tilde{P}_{wDE}(\mathbf{x}^n|\hat{\mathbf{x}}^n) &= \sum_{i=1}^{N} \alpha_i P_{DE}(\mathbf{x}^n|\hat{\mathbf{x}}^n, s_i) \\ &\geq \max_{i\in\{1,2,\ldots,N\}} \alpha_i P_{DE}(\mathbf{x}^n|\hat{\mathbf{x}}^n, s_i) \\ &= \alpha_a P_{DE}(\mathbf{x}^n|\hat{\mathbf{x}}^n, s_a).\end{aligned}$$

The worst case redundancy, $\tilde{\rho}_{DE}(\cdot)$, is then:

$$\begin{aligned}\tilde{\rho}_{DE}(\mathbf{x}^n|\hat{\mathbf{x}}^n) &= -\log(\tilde{P}_{wDE}(\mathbf{x}^n|\hat{\mathbf{x}}^n)) + \log(P_{DE}(\mathbf{x}^n|\hat{\mathbf{x}}^n, s^*)) \\ &\leq -\log(\alpha_a P_{DE}(\mathbf{x}^n|\hat{\mathbf{x}}^n, s_a)) + \log(P_{DE}(\mathbf{x}^n|\hat{\mathbf{x}}^n, s^*)) \\ &= -\log\alpha_a + \log\frac{P_{DE}(\mathbf{x}^n|\hat{\mathbf{x}}^n, s^*)}{P_{DE}(\mathbf{x}^n|\hat{\mathbf{x}}^n, s_a)}.\end{aligned}$$

■

D. Local Optimization

The concept of local optimization was introduced by Shtarkov, e.g., in [Sht87], and the idea, which is very simple, is most useful for sequential coding. According to local optimization the next symbol probability distribution should be calculated as ([Sht98]):

$$\vartheta(x_{n+1}|\mathbf{x}^n, \hat{\mathbf{x}}^{n+1}) = \frac{P^*_{DE}(\mathbf{x}^{n+1}|\hat{\mathbf{x}}^{n+1})}{\sum_{i=0}^{m-1} P^*_{DE}(\mathbf{x}^n i|\hat{\mathbf{x}}^{n+1})}, \tag{2.116}$$

where

$$P^*_{DE}(\mathbf{x}^n|\hat{\mathbf{x}}^n) = \max_{s>0} P_{DE}(\mathbf{x}^n|\hat{\mathbf{x}}^n, s). \tag{2.117}$$

With this method we again end up with having to calculate a closed form expression for a product of dominators which will not be possible. In order to find a possible solution we will therefore again approximate the distribution and omit the normalizing factor and hence:

$$\tilde{P}^*_{DE}(\mathbf{x}^n|\hat{\mathbf{x}}^n) = \max_{s>0} \tilde{P}_{DE}(\mathbf{x}^n|\hat{\mathbf{x}}^n, s) \tag{2.118}$$

$$= \max_{s>0} \prod_{i=1}^{n} \frac{1}{2s} \exp(-\frac{|x_i - \hat{x}_i|}{s}) \quad (2.119)$$

$$= \max_{s>0} \frac{1}{(2s)^n} \exp(-\frac{S_x(n)}{s}) \quad (2.120)$$

$$= \left(\frac{n}{eS_x(n)}\right)^n, \quad (2.121)$$

where $S_x(n) = \sum_{i=1}^{n} |x_i - \hat{x}_i|$. From the closed form expression we get the next symbol probability as ([Sht98]):

$$\tilde{\vartheta}(x_n|\mathbf{x}^{n-1}, \hat{\mathbf{x}}^n) = \frac{\left(\frac{n}{e(S_x(n-1)+|x_n-\hat{x}_n|)}\right)^n}{\sum_{i=0}^{m-1}\left(\frac{n}{e(S_x(n-1)+|i-\hat{x}_n|)}\right)^n} \quad (2.122)$$

$$= \left(S_x^n(n)\sum_{i=0}^{m-1}\left(\frac{1}{S_x(n-1)+|i-\hat{x}_n|}\right)^n\right)^{-1} \quad (2.123)$$

Without the approximation, the expected parameter description cost should be equal for all parameters since it is a property of local optimization. For this approximated distribution we could expect a slightly different behavior due to the limited range for the input symbols.

2.6 Pixel Correlation

In this section we will show that the correlation between pixels is very local. There are, however, some major problems associated with estimating the correlation between pixels: the input data is limited (i.e., not infinite), the correlation changes throughout the image and it is not the same for all images.
In order to estimate the correlation we use a context tree of depth 1. Each leaf will have a Dirichlet estimator. Based on this we estimate the codeword length by using the context tree and use different pixels as context. The result is shown in Figure 2.19.
The used image is a medical image with 512x512 pixels and with 8 bits representation of the gray scale for each pixel. Our conclusion from Figure 2.19 is that the correlation tends to be very local and there is a big difference already among the closest pixels. The result may be somewhat different when considering other types of images. For example when considering half tone images the correlation is “spread out“ in a systematic way determined by the technique used to construct the half tone image. This subject is discussed in, for example, [MF96].

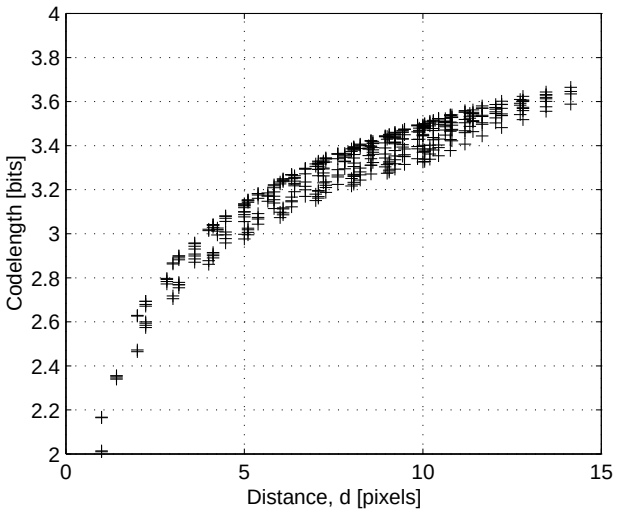

Figure 2.19: The figure shows an estimated codeword length based on the knowledge of a neighbor pixel at distance d away. This is for the medical image "skalle".

Chapter 3

A Technique for Prediction and Probability Assignment (PPA) in Lossless Data Compression

This chapter is based on the work presented in [ES00b].

3.1 Introduction

The aim in universal lossless data compression is to adapt the compression scheme to cope with the unknown source parameters. The best achievable performance, in terms of average redundancy, that we asymptotically can expect is stated by Rissanen's lower bound [Ris84] for universal coding. One way of constructing a sequential universal source coding scheme is to use the local optimization method. In local optimization the aim is to minimize the maximal individual redundancy for any sequence. This approach is well studied by, for example, Shtarkov [Sht87]. One important fact that we will pay attention to is that by studying individual redundancy we get a tool for short or limited sequences, i.e., we may get a desired performance from the first symbol to the last. This plays an important role in, for example, lossless image compression where the amount of data is, by nature, limited by the bounds of

the image. One might, however, argue that the amount of data is much larger in a typical image compression setting compared to a corresponding text compression setting. This may be true but as will be discussed further on in this chapter it is absolutely necessary to analyze methods and performance for short data sequences in order to squeeze a few extra bits (of redundancy) out of the compressed representation.
The lower bound for redundancy in universal data compression depends not only on the length of the sequence but also on the number of unknown parameters, K, i.e., the average redundancy satisfies:

$$\bar{\rho}(n) \geq (1-\epsilon)\frac{K}{2}\log n, \tag{3.1}$$

where $\epsilon > 0$, see also Theorem 2.16. Thus, for constructing a data compression scheme for practical applications it is the aim to find a parameterization of the source with a minimal number of unknown parameters without losing any information. It is well known in the lossless image compression community that (linear) prediction is an excellent tool for doing such a reduction of the number of unknown parameters, see e.g., [How93], [MRS94], [MS95], [ES96], [WRA96], [WSS96], [Wu96]. For this reason, we will focus on a compression scheme as in Figure 3.1. This is the kind of model that is often[1] used, directly or indirectly, in each node of a context tree in an image compression scheme. The probability assignment (PA) scheme is based on a memoryless model with a probability distribution with mean value equal to zero. In some schemes, for example [WSS96], a small bias to the mean is used to compensate for a possible rounding error when discretizing the real values in the prediction scheme.
Some of the above may sound strange. Do we not always use a minimal number of parameters, and what does "losing any information" mean? The answer is that many image compression schemes use a model similar to Figure 3.1 based on the double exponential (DE), Gaussian or similar probability distribution. For example, with the Gaussian distribution the parameterization will have only 2 unknown parameters:

$$p(x) \sim \exp\left(-\frac{(x-m)^2}{s}\right), \quad s > 0, \tag{3.2}$$

where m and s are unknown. This parameterization is, however, only useful if it is "correct". In general, for practical applications, we cannot tell if this is true or not. But there is still a gain since the ratio between $M-1$ and 2 (where M is the size of the alphabet) unknown parameters

[1] For example in some of the mentioned references.

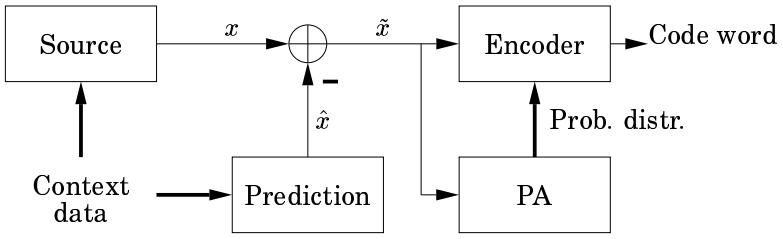

Figure 3.1: A compression scheme with separate prediction and probability assignment (PA). The predicted value is subtracted from the source data, i.e., $\tilde{x} = x - \hat{x}$. The output from the encoder is a code word based on the $\tilde{x}$-value and the probability distribution from the PA. (The thicker arrows in the figure corresponds to vector representation)

could be "very" big and the larger model can only perform better if the data sequence is "very" long. A simple example will show the possible gain:

Example 3.1:
We consider a source that emits source symbols according to the following discrete distribution,which is a distorted DE distribution with mean 128.3:

$$\tilde{p}(x) = c_x \exp\left(-\frac{|x - 128.3|}{s}\right), \quad s > 0, \tag{3.3}$$

where $x \in \{0, 1, \ldots, 255\}$ is the source symbol and c_x is a random value taken from a random variable which is uniformly distributed in the range $[\frac{1}{2}, 2]$. The parameter s is unknown.
From the distribution we make a probability distribution by normalizing:

$$p(x) = \frac{\tilde{p}(x)}{\sum_i \tilde{p}(i)}. \tag{3.4}$$

We will compare two different methods for probability assignment of the data from this source. On the one hand, we will use the Dirichlet estimator (see Section 2.5.2) for 256-ary alphabet, and on the other hand, we will use the probability assignment for DE-data that was derived in Section 2.5.3.
With the Dirichlet estimator we will have 255 unknown parameters to estimate, and with the probability assignment for DE-data we will have only one unknown parameter[2]. But since the distribution is not really

[2]The mean value is known and s is the only unknown parameter.

a DE-distribution, we will not perform asymptotically correct with only one parameter.
In Figure 3.2 we have simulated 100 sequences of length 10000 and compared the average results for the two different probability assignment methods.
We conclude that, in this example, it takes on the average approximately 7000 symbols before the Dirichlet estimator is superior. Thus, it will be of great advantage to use the one parameter approach for short sequences although the probability distribution for the source data is a distorted DE-distribution. ■

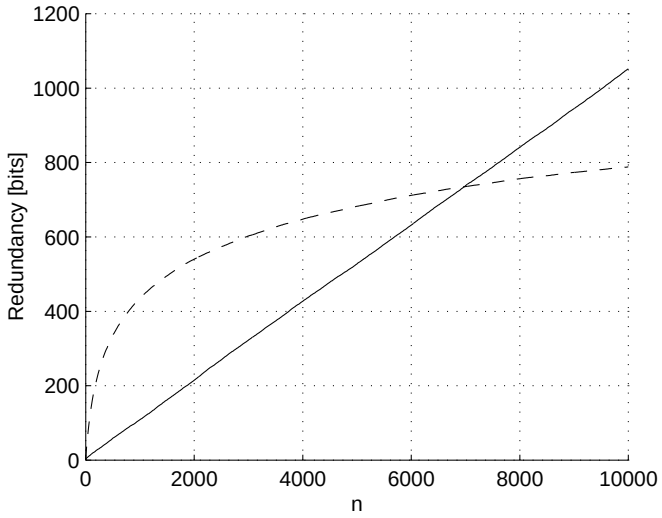

Figure 3.2: The result from Example 3.1. The y-axis shows the redundancy (averaged over 100 sequences) and the x-axis shows the sequence length. The solid line in the graph corresponds to probability assignment for DE-data, and the dashed line is the Dirichlet estimator.

Much work has been focused on different strategies for universal prediction schemes. These prediction schemes have often some kind of connection with universal data compression, e.g., [Ris84], [FM92], [FM94], [FS98], [MF98]. Despite the excellent results in that area, the application in lossless image compression requires some further investigation since we want to minimize the resulting codeword length. Minimizing the code word length may not necessarily be the same as minimizing the error from the prediction scheme.

In the way the data is treated in most image compression schemes, with independent prediction and probability assignment (or estimation), we cannot guarantee that it is possible to make a compression scheme that has an optimal behavior, i.e., minimal value of K in the average total redundancy:

$$\bar{\rho}(n) = \frac{K}{2} \log n + O(1), \tag{3.5}$$

when $n \to \infty$. For this reason, the *prediction and probability assignment* (PPA) concept was introduced in [ES98b] and [Eks98]. The aim with PPA is to optimize the prediction and the probability assignment together in order to control the behavior of the redundancy in a desired way. This is also of major importance since we usually use some kind of context tree model for our data, and the sequences in each node of a context tree tends to be very small, e.g., less than 100 samples, except for a few nodes at small depth. For sequences of limited length it could be disastrous to use a universal source coding scheme which **only** performs asymptotically correct according to Equation 3.5 and has a "bad" initial behavior.

In the sequel of this chapter we will describe a technique for construction of a PPA scheme for some commonly appearing sources in lossless data compression. The technique may very well be suited for other similar sources.

3.2 Context Linear Sources

We will focus our analysis on a special type of sources that are common in image compression (e.g., see [WRA96]), in various topics of signal processing etc. We will denote this class of sources as *Context Linear Sources* (CL-Sources). The name is based on the similarities with for example Context Tree Sources. A CL-source takes a context as input, and based on a linear combination of the context it will emit a source symbol. We will consider two different kinds of CL-sources, with or without feedback (see Figure 3.3). A subset of the class of CL-sources can also be regarded as *auto regressive processes* (AR-processes) [Mat96]. Further discussion on the AR-process and image compression can be found in [WRA96]. When we consider applications of image compression, we will assume that each node of the context tree observes a sequence from a CL-source without feedback.

The CL-source works as follow: the source takes an integer vector $\mathbf{y} = y_1 y_2 \dots y_r$ as input. From the input a temporary value m is calculated

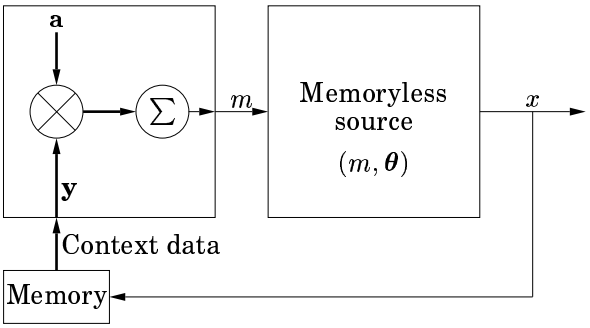

Figure 3.3: Context Linear Sources (CL-source) with feedback. If the context data would appear from another source, it would have been a CL-source *without* feedback.

according to:

$$m = m(\mathbf{y}, \mathbf{a}) = \sum_{i=1}^{r} a_i y_i, \tag{3.6}$$

where the coefficient vector $\mathbf{a} = a_1 a_2 \dots a_r$, $a_i \in \mathbb{R}$, is an unknown parameter of the source. From the m-value and a parameter vector $\boldsymbol{\theta}$ a probability distribution for the output alphabet is calculated, i.e., $P(m(\mathbf{y}, \mathbf{a}), \boldsymbol{\theta})$. It should be noted that, for simplicity and due to the nature of digital images, we will in the sequel only consider integer context data.

3.3 The PPA Concept

The PPA concept was informally defined in [Eks98], see Definition 2.30. In Chapter 2 it was discussed that it is important not to split the prediction and probability assignment into separate parts. When we consider the CL-source based on a Gaussian distribution, we will find several methods for estimating the distribution parameter s and the vector $\mathbf{a}$ in Figure 3.3. A simple approach is to use an estimate giving the least square sum error, i.e.,

$$\arg\min_{\mathbf{a}} \sum_{i=0}^{n-1} (x_i - m_i)^2. \tag{3.7}$$

From statistical theory there exists many (easy) ways of estimating the distribution parameter s in (3.2) for the continuous Gaussian probability

distribution. One way of doing this is by estimating the variance:

$$\frac{1}{n-1}\sum_{i=0}^{n-1}(x_i - m_i)^2, \tag{3.8}$$

and from the variance calculate the estimate of s. However, it should be noted that since we are working with a discrete limited alphabet, it is not always possible to find a *useful* closed form expression for the variance. This will be discussed later in this chapter.
The elementary method of doing PPA would be to use a weighting technique. A straight forward derivation of a weighted PPA method for the Gaussian distribution could be as follows:
First, we define our discrete Gaussian distribution according to:

$$P_G(x, \mathbf{y}, s, \mathbf{a}) = c(m(\mathbf{y}, \mathbf{a}), s)\exp(-\frac{(x - m(\mathbf{y}, \mathbf{a}))^2}{s}), \quad s > 0 \tag{3.9}$$

where $c(m, s)$ normalizes the function to a probability distribution, i.e.,

$$c(m, s) = \left(\sum_{x=0}^{M-1}\exp(-\frac{(x-m)^2}{s})\right)^{-1}. \tag{3.10}$$

The block probability for a sequence is thus:

$$P_{BG}(\mathbf{x}_0^{n-1}, \mathbf{Y}_0^{n-1}, s, \mathbf{a}) = \prod_{i=0}^{n-1} P_G(x_i, Y_i, s, \mathbf{a}), \tag{3.11}$$

where

$$\mathbf{x}_0^{n-1} = x_0 x_1 \cdots x_{n-1}, \tag{3.12}$$

$$\mathbf{Y}_0^{n-1} = Y_0 Y_1 \cdots Y_{n-1}, \tag{3.13}$$

$$Y_i = y_{i,1} y_{i,2} \cdots y_{i,r+1}. \tag{3.14}$$

From the block probability we define the weighted block probability according to:

$$P_{WBG}(\mathbf{x}_0^{n-1}, \mathbf{Y}_0^{n-1}) = \int_s \int_{\mathbf{a}} \alpha(s, \mathbf{a}) P_{BG}(\mathbf{x}_0^{n-1}, \mathbf{Y}_0^{n-1}, s, \mathbf{a}) d\mathbf{a} ds, \tag{3.15}$$

where the constraint on the a priori parameter distribution, $\alpha(\cdot)$, fulfills:

$$\int_s \int_{\mathbf{a}} \alpha(s, \mathbf{a}) d\mathbf{a} ds = 1. \tag{3.16}$$

From the weighted block probability we get the next symbol probability according to:

$$P_{GPPA}(x|\mathbf{x}_0^{n-1}, \mathbf{Y}_0^n) = \frac{P_{WBG}(\mathbf{x}_0^{n-1} x, \mathbf{y}_0^n)}{P_{WBG}(\mathbf{x}_0^{n-1}, \mathbf{y}_0^{n-1})}. \tag{3.17}$$

3.4 A PPA Technique Based on Local Optimization

The basic idea with local optimization is that every parameter value for the unknown parameters should have equal description cost, i.e., the redundancy for not knowing the parameters in advance is independent of the actual parameter value (within a specified range). By having this aim locally in every part and every symbol in the sequence, we do not only get an asymptotically optimal behavior but also a redundancy behavior according to Equation 3.5 for every sequence. With optimal in this case we refer to individual redundancy which also means optimal for average redundancy.

Construction of a local optimization scheme for the Gaussian distribution is done by first calculating the maximum probability for a sequence:

$$P^*_{BG}(\mathbf{x}_0^{n-1}, \mathbf{Y}_0^{n-1}) = \max_{s,\mathbf{a}} \beta(s,\mathbf{a}) P_{BG}(\mathbf{x}_0^{n-1}, \mathbf{Y}_0^{n-1}, s, \mathbf{a}), \tag{3.18}$$

where $\beta(\cdot)$ is the a priori distribution on the parameters. In order to give all parameter values equal description cost within the input range the $\beta(\cdot)$-function should be chosen as a constant in that range. Note that $\beta(\cdot)$ may not necessarily be a probability distribution which is important when we have infinite (or semi-infinite) parameter range, e.g., $s > 0$.

From the maximum probability we get the individual next symbol probabilities by normalization:

$$P_{LOG}(x|\mathbf{x}_0^{n-1}, \mathbf{Y}_0^n) = \frac{P^*_{BG}(\mathbf{x}_0^{n-1}x, \mathbf{Y}_0^n)}{\sum_{i=0}^{M-1} P^*_{BG}(\mathbf{x}_0^{n-1}i, \mathbf{Y}_0^n)}. \tag{3.19}$$

This technique has been shown by Shtarkov [Sht87] to perform optimal in terms of (individual and average) redundancy for essentially any probability distribution and any number of unknown parameters.

The drawback with the presented technique is when, for example, applying it on the Gaussian distribution since it is not possible to calculate the maximum probability in an efficient way (closed form expression). This is due to the fact that the normalization $c(\cdot)$ in (3.10) cannot be calculated without knowing the mean value m for every sample. But m depends on the parameter vector $\mathbf{a}$ which is, and will stay, unknown in the local optimization scheme. Thus, we will have to construct an approximate local optimization scheme:

Algorithm 3.1:
In order to find $P^*_{BG}(\mathbf{x}_0^{n-1}, \mathbf{Y}_0^{n-1})$ do the following:

▶ Calculate the best estimate of $\mathbf{a}$ according to a least square sum criteria, i.e.,

$$\mathbf{a}^* = \arg\min_{\mathbf{a}} S_{sq}(\mathbf{x}_0^{n-1}, \mathbf{a}, \mathbf{Y}_0^{n-1}) \tag{3.20}$$

$$= \arg\min_{\mathbf{a}} \sum_{i=0}^{n-1} (x_i - m(Y_i, \mathbf{a}))^2 . \tag{3.21}$$

▶ Determine the parameter s^* giving the maximal probability, i.e.,

$$s^* = \arg\max_{s} \beta(s, \mathbf{a}) P_{BG}(\mathbf{x}_0^{n-1}, \mathbf{Y}_0^{n-1}, s, \mathbf{a}^*). \tag{3.22}$$

▶ Limit the parameter s^* to a preselected range. Denote this value s^{**}.

▶ Find the max-probability according to:

$$P^*_{BG}(\mathbf{x}_0^{n-1}, \mathbf{Y}_0^{n-1}) = \beta(s^{**}, \mathbf{a}^*) P_{BG}(\mathbf{x}_0^{n-1}, \mathbf{Y}_0^{n-1}, s^{**}, \mathbf{a}^*). \tag{3.23}$$

■

Two approximations are introduced:

- The parameters $\mathbf{a}$ and s are optimized individually. This has the advantage of not only simplifying the calculations but it also makes it possible to take advantage of known results in linear prediction, e.g., [FS98].

- The parameter s is limited to a preselected range in order to avoid the problems associated with, for example, $s = 0$ (where the function is not defined).

We should also note that since the $\mathbf{a}$-vector is determined by the least square sum criterion, the a priori distribution $\beta(\cdot)$ will not have any affect on $\mathbf{a}^*$.
The reason for doing these approximations are not just to be able to solve the problem. We note that if the probability distribution behaves "nicely" and the predicted values (m-values) are located not too close to the edges of the the range $[0, M-1]$, the $\mathbf{a}$-vector will have little influence on the normalizing factor of the probability distribution (at least if we consider DE or Gaussian distribution as described previously in the chapter).

3.4.1 PPA in the Gaussian Case

When considering the construction of the approximate PPA described in Algorithm 3.1 for the Gaussian distribution, we will have to introduce further approximations since it is not possible to find a useful discrete probability distribution. For this reason we will use an approximation based on the continuous Gaussian probability distribution:

$$\tilde{P}_G(x, \mathbf{y}, s, \mathbf{a}) = \frac{1}{\sqrt{\pi s}} \exp(-\frac{(x - m(\mathbf{y}, \mathbf{a}))^2}{s}). \tag{3.24}$$

The step to get the block probability is trivial:

$$\begin{aligned} P_{BG} &= \tilde{P}_{BG}(\mathbf{x}_0^{n-1}, \mathbf{Y}_0^{n-1}, s, \mathbf{a}^*) \\ &= \prod_{i=0}^{n-1} \tilde{P}_G(x_i, Y_i, s, \mathbf{a}^*) \qquad (3.25) \\ &= \left(\frac{1}{\sqrt{\pi s}}\right)^n \exp\left(-\frac{S_{sq}(\mathbf{x}_0^{n-1}, \mathbf{a}^*, \mathbf{Y}_0^{n-1})}{s}\right). \qquad (3.26) \end{aligned}$$

From the block probability we derive an expression for the parameter s that maximizes the probability:

$$\begin{aligned} \tilde{s}^* &= \arg\max_s \tilde{P}_{BG}(\mathbf{x}_0^{n-1}, \mathbf{Y}_0^{n-1}, s, \mathbf{a}^*) \qquad (3.27) \\ &= \frac{2S_{sq}(\mathbf{x}_0^{n-1}, \mathbf{a}^*, \mathbf{Y}_0^{n-1})}{n}. \qquad (3.28) \end{aligned}$$

With (3.26) and (3.28) used in Algorithm 3.1 we get an approximate PPA scheme for the Gaussian distribution.
By considering (3.26) we note why it is natural to find the best **a**-vector before the best s parameter in Algorithm 3.1. The square sum of the prediction error $S_{sq}(\cdot)$ in (3.26) is independent of the s parameter.

3.4.2 PPA in the DE Case

In [Eks98] a probability assignment technique for DE-data based on local optimization was derived. That method has, however, the drawback that the mean value, m_i, must be known for each symbol to be encoded. Another drawback when considering the DE-distribution is that the aim should be to minimize the sum of the absolute error and not the sum of the squared error.
In this chapter we will not investigate the DE distribution further but conclude that it is possible to derive a PPA scheme similar to the Gaussian case with the same type of approximations.

3.5 Some Experimental Results

Based on the methods presented in this chapter we will investigate how they perform in practice. In all experiments we use a CL-source without feedback. The context data symbols are independent of each other. The source and context data alphabet sizes are both 256, which corresponds to the gray scale images commonly used for comparison in lossless data compression and throughout this thesis.

We start by considering a CL-source with the Gaussian distribution according to (3.2). It must be noted from a practical point of view that, since we will use arithmetic encoding for the actual coding, there will be a smallest probability entity that can be represented. As an example we could consider an arithmetic coder with 15 bits numerical precision leading to the lowest entity of representable probability of 2^{-15}. If we use the Gaussian distribution and have the *shape* parameter $s = 15$ and mean value $m = 128$, one can find that only 23 symbols will have probability larger than 2^{-15}. Thus, the remaining 233 symbols must be encoded with a (much) higher coding distribution than the "correct" probability distribution. Normalizing the coding distribution to the 15 bit precision yields that we will get an coding redundancy of 0.010 bits/source symbol.

The first experiments, resulting in Figure 3.4 and 3.5, show how the redundancy increases with the length of the data sequence for $r = 2$ and $r = 8$. Each line, corresponding to one value of s, is an average of 1000 sequences. The values of the s parameter in this experiment lie in the range $[1, 12]$. The $\mathbf{a}$-vector is chosen in such a way that every element of the vector is in the range $[0, 1]$. A new $\mathbf{a}$-vector was created for each sequence.

The dash-dotted lines correspond to the case with separate estimation of the $\mathbf{a}$-vector and the parameter s. The dashed lines corresponds to a case where the parameter s is estimated together with the $\mathbf{a}$-vector, i.e., from the estimated $\mathbf{a}$-vector a squared sum of errors is calculated based on the sequence and s is estimated from this sum. The solid line corresponds to the approximate local optimization method presented in this chapter.

One can note that independent estimation of $\mathbf{a}$ and s is more or less useless in comparison to the other two cases. We also note a large difference between the other estimator approach and the local optimization technique. This difference is also possible to observe in Figure 3.6 where the redundancy after 128 symbols, with $r = 8$, are shown with respect to the s parameter.

Within the range, $[1, 12]$, of the s parameter we note that the redundancy is almost "constant" with respect to s, i.e., the redundancy after 128 symbols is almost independent of s. It is, however, possible to change the

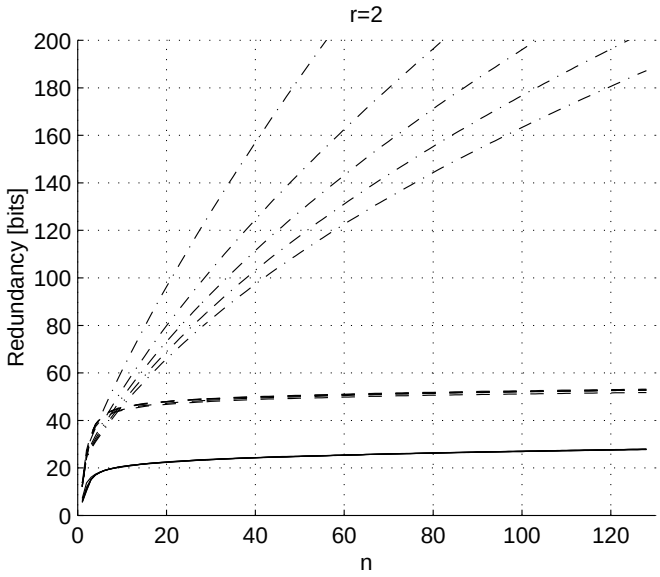

Figure 3.4: Redundancy vs sequence length for $r = 2$. The worst performance is for independent estimation of the $\mathbf{a}$-vector and the s parameter. The best performance is by the approximate local optimization scheme. Each line corresponds to one value of of the parameter s, and is an average over 1000 sequences.

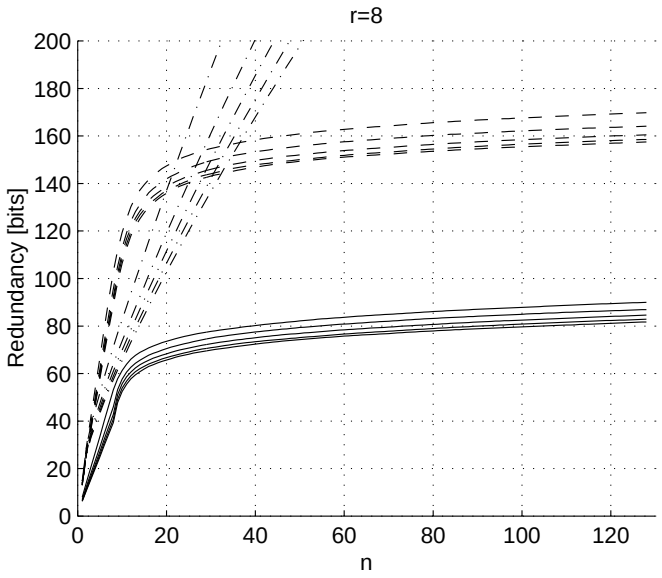

Figure 3.5: Same as the previous figure but $r = 8$.

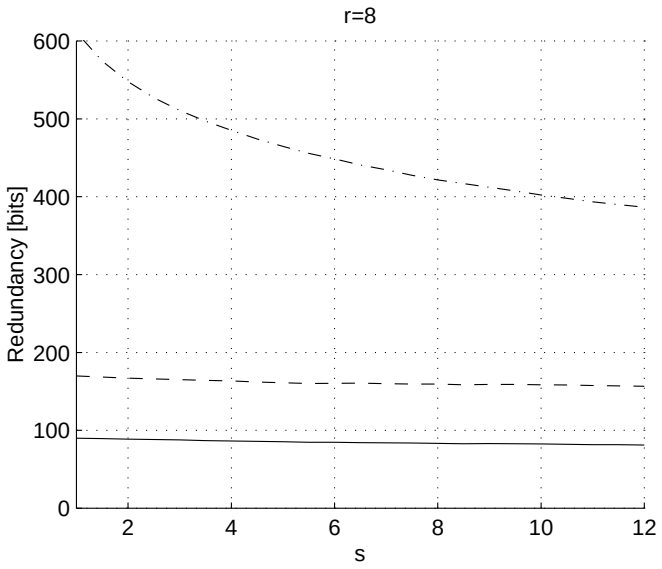

Figure 3.6: Redundancy after 128 source symbols vs the parameter s. ($r = 8$)

behavior for the local optimization scheme. This is done by changing the a priori information or parameter description cost $\beta(\cdot)$ in (3.18).
Our presented approximate PPA scheme can, however, not take full advantage of $\beta(\cdot)$ since the $\mathbf{a}^*$ in the algorithm is determined by the least square sum criterion only, and thus without consideration of $\beta(\cdot)$. This might be a major drawback in some applications, e.g., where the $\mathbf{a}$-vector is restricted to a certain range.

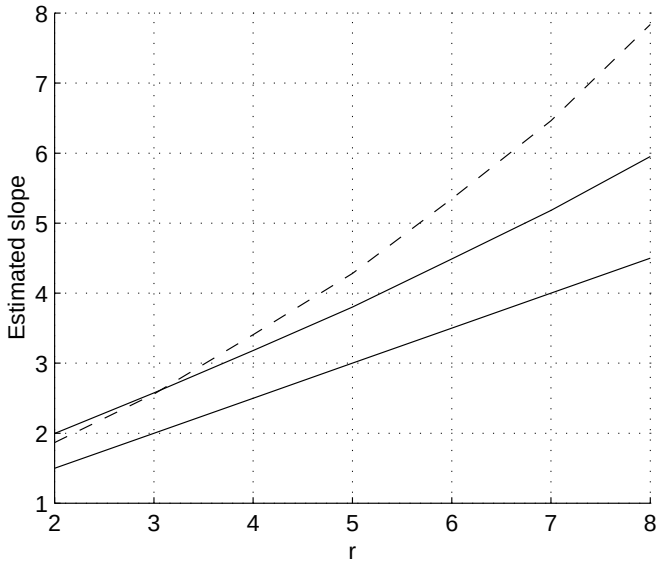

Figure 3.7: The slope vs r when considering the redundancy vs log(sequence length). The dashed line corresponds to the estimation technique, the middle line is the local optimization technique, and the lower line is the theoretical bound, i.e., "K/2".

An important aspect of the compression scheme is that it should perform like $K/2 \log n + O(1)$ in terms of redundancy, where K is the number of unknown parameters. In Figure 3.7 the slope "$K/2$" behavior is studied, i.e., $\rho(n)/\log(n)$. In our setup the number of unknown parameters is $K = r + 1$. The lower line in the figure shows the ideal performance, and the dashed and solid line correspond to the same cases as in the previous figures. The scheme with independent estimation of $\mathbf{a}$ and s is omitted in this figure.
We note a slightly worse performance for the two PPA schemes than the desired behavior. In some sense we may consider this as if our PPA

scheme uses more parameters than necessary. This is the penalty for the introduced approximations.
Some further notes about the two estimation techniques: When we use separate prediction and PA, the performance is really bad compared to estimation of s based on the estimate of $\mathbf{a}$-vector. This is because the error in the estimate of $\mathbf{a}$ will always increase the squared sum of errors after the prediction. This will, in its turn, make us estimate s too large. In order to make a good estimate of s we will therefore need to use the information of the estimated $\mathbf{a}$-vector.
It must be pointed out that for lossless image compression it is usually not the case that the context data symbols are independent of each other.

3.6 Conclusions

In this chapter we have studied prediction and probability assignment (PPA) in the context of lossless image compression and similar areas.
From some basic definitions we have presented and motivated an approximate PPA scheme. We have shown that this scheme performs well in terms of average redundancy on a test setting based on the Gaussian probability distribution. Similar results are to be expected for similar distributions such as the DE distribution.
However, the approximate scheme for the Gaussian case does not perform optimally (in terms of average redundancy).

Chapter 4

The Context Mapping Function (CMF) in Lossless Data Compression

This chapter is based on [ES00a].

4.1 Introduction

In many lossless image compression schemes there exists some kind of prediction scheme along with a context modeling scheme, e.g., [WSS96], [Wu96], [WRA96], [Eks98]. One of the main goals with prediction is to reduce the number of unknown parameters in the context modeling, e.g., see [ES96], since using fewer parameters will have a major impact on the redundancy, at least asymptotically. This is due to Rissanen's bound for universal modeling [Ris83], from which we find our aim for the redundancy for stationary ergodic sources (this was also discussed in Chapter 2 and 3):

$$\rho(n) \leq \frac{km}{2}\frac{\log n}{n} + \frac{O(1)}{n}, \quad n \to \infty, \tag{4.1}$$

where k is the number of states (or nodes if we have a tree representation) in the model and m is the number of unknown parameters in each state, e.g., $m = 1$ for binary data.

By using prediction in lossless data compression we are not only able to reduce the number of states k. We may also use a statistical model with fewer parameters than the alphabet size minus one, e.g., for 8-bit data we may be able to reduce from $m = 255$ to $m = 2$ by using the double exponential or the two sided geometrical distribution. This was discussed in detail in Chapter 3.

However, one problem still remains and that is the topic of selecting an appropriate context from the observed data for the context modeling scheme. In [WSS96], [Wu96], [WRA96], [Eks96] etc., we have seen a variety of ad-hoc context selection schemes with the aim of improving the compression performance.

In [Eks98] the *Context Mapping Function* (CMF) concept was introduced as a general tool of describing how the context selection was done. It was also stated that a well selected CMF could make great improvements to lossless image compression[1]. This is motivated by the fact that the correlation between adjacent pixels and bit planes in images tends to be very high locally, but decreases rapidly as the distance increases, see Section 2.6. For this reason it is very hard to take advantage of long contexts, since the gain in information per context data symbol cannot compensate the increased loss when the number of states grows exponentially with the length of the context.

The question still remains what motivates us to try to find the "best" CMF, and, in case we would like to find the best CMF, how to do it? Some of these issues will be discussed in this chapter along with some experimental results.

Related work on this subject has been carried out in [WLZ92] and [WS96]. But already in [RL81] a similar approach to the CMF was introduced: the *structure function*. The structure function, $z = f(\mathbf{s})$, defines a state, z, from the previous observed data, $\mathbf{s}$. It is stated in [RL81] that "to find an optimum structure function is an undecidable problem" . The basis for the work in [WLZ92] is to use a kind of hypothesis testing to find an ordering of the context data adaptively. In [WS96] an approach is made using weighting of CTW models, each model using different ordering of the context symbols. However, both these approaches lack the ability to adapt to models where there exists a dependence between the context data symbols since they will only "change" the order of the context data. Another drawback of these algorithms is the computational complexity. In [WS96] it is stated that using a depth larger than only a few symbols is exceedingly intractable.

[1] Without giving any explicit details.

4.2 The CMF Concept

In the sequel we will assume a universal source coding scheme according to Figure 4.1 as our main tool. The *Context Tree Weighting* (CTW) algorithm [WST95] is known to perform optimally both in terms of average and individual redundancy. The study of individual sequences is of major importance when considering compression of short data sequences.

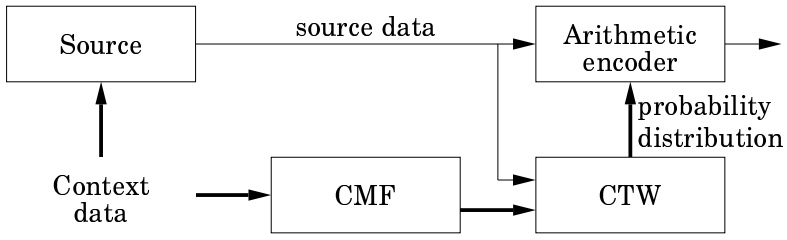

Figure 4.1: A universal source coding scheme based on the Context Tree Weighting (CTW) algorithm.

We define the CMF according to:

Definition 4.1:
Given context data $\mathbf{X} = X_1 X_2 \cdots X_R$, where $X_i \in \mathcal{X}, i = 1, 2, \ldots, R$ are context symbols, the *context mapping function* (CMF) defines a corresponding meta-context $\mathbf{Y} = Y_1 Y_2 \cdots Y_Q = \text{CMF}(\mathbf{X})$, where $Y_i \in \mathcal{Y}, i = 1, 2, \ldots, Q$.
The context data alphabet $\mathcal{X}$ may be any alphabet and $\mathcal{Y}$ is a discrete alphabet of finite size. ■

Based on the definition, we give a simple example:

Example 4.1:
Quantization of real valued data into a finite alphabet with a 16 bit numerical precision is one example of a CMF, i.e., $\mathcal{X} \in \mathbb{R}$, $\mathcal{Y} \in \{0, 1\}$, $R = 1$, and $Q = 16$. ■

We will in the sequel denote the size of the input vector (i.e., R) as *CMF input size* and correspondingly for the output vector (of size Q) the *CMF output size*. We will for practical purposes and due to the nature of digital data restrict our analysis to the case with binary alphabets, i.e., $\mathcal{X}, \mathcal{Y} \in$

$\{0, 1\}$.
In order to describe the influence of the CMF we consider a simple example. In Figure 4.2 we have a *general binary context tree source* (see Section 2.2.1) where the data alphabet is binary and all contexts appear with equal probability, i.e., $1/8$.

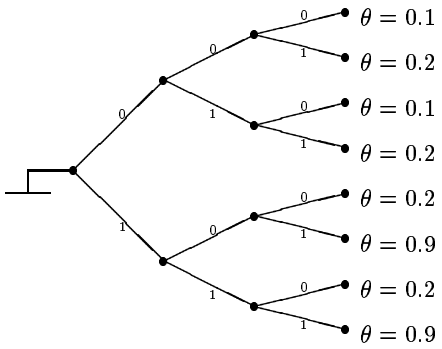

Figure 4.2: A general binary context tree source with binary output symbols. The parameters denote the probability for one of the data symbols.

The first observation we make is that the second context bit is of no importance for our source, thus the source could be simplified. From our universal source coding perspective, however, we do not know in advance the parameters of the source. But with the CMF in Table 4.1 our CTW algorithm would be able to find the tree in Figure 4.3.

Input: $X_1X_2X_3$	**Output:** $Y_1Y_2Y_3$
000	000
001	010
010	001
011	011
100	100
101	110
110	101
111	111

Table 4.1: The CMF for removing the influence of the second context bit, i.e., context bits two and three are swapped.

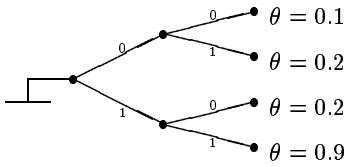

Figure 4.3: The general binary context tree source when the second context bit is omitted.

When we study the tree i Figure 4.3 we note that it will be possible to further reduce the number of nodes in the tree by merging together the states with equal parameter values. But the corresponding CMF could be constructed in 3 different ways, see Table 4.2.

Input: $X_1X_2X_3$	**CMF1:** $Y_1Y_2Y_3$	**CMF2:** $Y_1Y_2Y_3$	**CMF3:** $Y_1Y_2Y_3$
000	000	000	000
001	010	100	100
010	001	001	001
011	011	101	101
100	010	100	100
101	100	110	010
110	011	101	101
111	101	111	011

Table 4.2: The three possible CMFs to obtain a resulting tree model with 3 leaves.

It should be clearly stated that this kind of merging of tree leaves is only successful when the parameters are equal and we consider the performance asymptotically. For **limited** length sequences the case is, however, slightly more complicated.
Leaving the simple example, we take a closer look at the general idea. If we make the comparison in terms of code word length when using two separate nodes, B and C, with parameters θ_B and θ_C respectively, and only using one node A which is a merge of B and C, we get:

$$l_A = nh(P_B\theta_B + (1-P_B)\theta_C) + \frac{1}{2}\log(n) + c_1, \tag{4.2}$$

$$l_B + l_C = n_B h(\theta_B) + \frac{1}{2}\log n_B + c_2 + n_C h(\theta_C) + \frac{1}{2}\log n_C + c_3, \tag{4.3}$$

where n_B and n_C are the number of observations in the nodes, $n = n_B + n_C$, $h(x) = -x\log x - (1-x)\log(1-x)$ the binary entropy function, $P_B = n_B/(n_B + n_C)$ and c_i, $i = 1, 2, 3$, are constants. From this we can derive when the one node model is superior to the two node model, and we end up with an expression of the type:

$$l_A < l_B + l_C, \tag{4.4}$$

$$c_4 n < c_5 + \frac{1}{2}\log n, \tag{4.5}$$

where n is the total number of observations, and $c_4 \geq 0$ and c_5 are constants. The conclusion is that the one node model **may** be **better** initially but will always perform **worse** than the two node model asymptotically, except when $c_4 = 0$, i.e., when $\theta_B = \theta_C$. This result is well known and it is an important result for our application where the amount of data is most limited and where "asymptotic behavior" may not at all be applicable.
Returning to our example we may ask ourself which one of the three CMFs is the best? From the asymptotic analysis we would state that the CMFs are equally good. But for short data sequences we might think otherwise. In Figure 4.4 the three CMFs are compared. Since CMF1 will

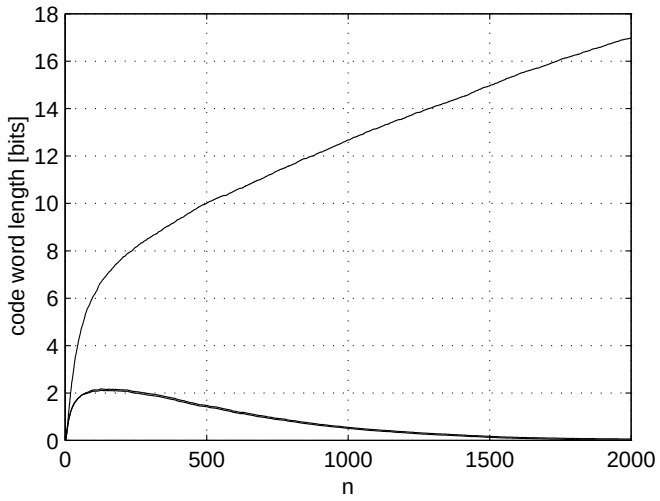

Figure 4.4: A comparison when using no CMF (top), CMF2, and CMF3 (almost indistinguishable from CMF2). The comparison is calculated relative to CMF1. An average over 1000 sequences is made.

clearly perform best, the comparison is shown as the extra code word

length compared to CMF1. The CMF2 and CMF3 perform almost indistinguishable, in terms of code word length, in this example. We also note the extra penalty for not using well chosen CMFs. Why do we care about these few bits in difference? There are two things we should keep in mind:

- The gain would have been bigger if the parameters were closer.
- In a binary context modeling scheme with tree depth 15 we may have up to 65535 useful nodes and leaves. If we use such a scheme for lossless image compression and are able to save a few bits for many of the nodes or leaves we make an improvement that could be of great interest.

Another gain that is very important, and which we get for free, is that we could use a shorter context and thus reduce the computational complexity of the context modeling.
Since we will only consider the case when the CMF is used together with the CTW algorithm, it will be natural to define the CMF from a tree structure instead of a table as in Table 4.1 and 4.2. Let us return to the example with CMF1 in Table 4.2. In order to determine the first output symbol, Y_1, we find the Boolean function[2]:

$$Y_1 = X_1 \wedge X_3. \tag{4.6}$$

For the second symbol we have two cases depending on the first symbol, Y_1:

$$\begin{aligned} Y_{2|0} &= X_1 \vee X_3, & (4.7)\\ Y_{2|1} &= 0, & (4.8) \end{aligned}$$

where $Y_{2|0}$ denotes Y_2 given that $Y_1 = 0$ (and $Y_{2|1}$ correspondingly for Y_2 and $Y_1 = 1$). Finally, the third and last symbol, Y_3, depends on the previous two and can be found to be:

$$\begin{aligned} Y_{3|00} &= X_2, & (4.9)\\ Y_{3|01} &= X_2, & (4.10)\\ Y_{3|10} &= X_2, & (4.11)\\ Y_{3|11} &= X_2. & (4.12) \end{aligned}$$

We can conclude that the last symbol does not depend on the first two which was already observed in the beginning of this example since X_2 does not affect the source.

[2]Logical "and" is $\wedge$ and logical "or" is $\vee$.

In the sequel we will denote each of the above functions as a *map rule* for each node of a tree, see Figure 4.5. In the next section we will describe a method to find *good* map rules for some sample data. We will use map rules with a fixed number of input variables thus restricting the required size for the CMF.

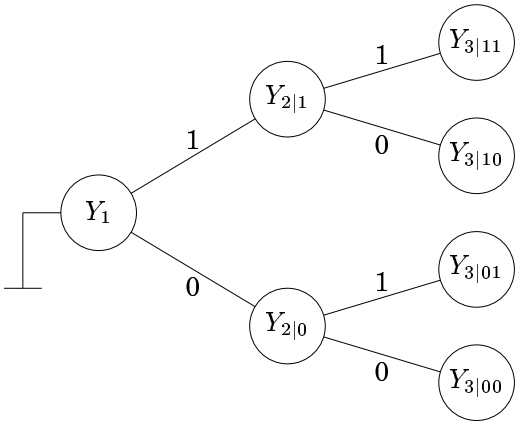

Figure 4.5: A CMF-tree with one map rule for each node (and leaf) in the tree.

4.3 An Off-line Algorithm for Construction of Good CMFs

The problem of finding good CMFs have some similarities to finding good test or decision algorithms. The main differences between the principles of CMF and decision algorithms are that we may merge nodes in the CMF and that CMF should be constructed off-line, which may not necessarily be the case for decision algorithms. With off-line in this case we mean that the CMF will be constructed on a test data set. Once it is constructed it is assumed to be known and we can use it in our desired application without having to construct/describe the CMF again.
An investigation on how to modify and apply the CTW algorithm for use as an on-line decision algorithm was presented in [Vol94]. For use in lossless data compression we would, in contrast to the work in [Vol94], like to construct a CMF off-line. This implies that we will make the construction based on a set of test files or test data. From this set, we will extract a test sequence consisting of both observations and the corresponding context for the observation.

In order to construct an off-line scheme for finding a good CMF we will first have to define a measure which will enable us to compare two CMFs. The basic approach is to apply the CMFs to the source coding scheme in Figure 4.1 and compare which one is producing the shortest code word. It should be noted that two CMFs are only comparable if the same test sequence is applied to both of them and the result will thus be dependent of the actual test sequence. Since every non-trivial[3] source will be able to produce for example only zeroes or only ones, for a limited length sequence, we recognize a problem with comparing CMFs if the test sequence is too short.

A simplified version of the CMF-comparison will be used when we are searching for a good CMF since the CMFs will only differ in the last level, i.e., at maximum depth. We will therefore not use the CTW algorithm to evaluate the performance. Instead we use a tree with counters for the observations. For each leaf in the counter-tree we calculate a pseudo code word length, $l(a,b)$, and corresponding pseudo rate, $r(a,b)$, according to:

$$l(a,b) = (a+b)h(\frac{a}{a+b}) + \frac{1}{2}\log(a+b), \quad a+b>0, \tag{4.13}$$

$$r(a,b) = \frac{l(a,b)}{a+b}, \tag{4.14}$$

where a and b are the counters for the binary symbols. By definition we have $l(0,0)=0$.

The problem of finding the best CMF can be done by exhaustive search, but that is, of course, not reasonable except for trivial cases. What we will do is to use *simulated annealing*, which is a basic method from combinatorial optimization, see e.g., [AK89]. We will also "optimize" at only one level at a time, i.e., first optimize at depth 1, then at depth 2 and so on until the maximum depth is reached. This will not result in the global optimum, except for special cases, but will hopefully give us a reasonable result.

Based on the basic simulated annealing scheme we propose the following algorithm for finding a good CMF for given test data:

Algorithm 4.1 (Basic optimization of CMF):

In the following pseudo code, `maximum depth` denotes the maximum depth of the desired CTW-tree. The parameter `iterationLength` is the number of steps in the simulated annealing for each depth of the algorithm. For each step in the simulated annealing scheme a `C`-value is calculated as an exponentially decreasing function with constants `Ca` and `Cb`. These

[3] Non-trivial here refers to sources where no probability parameter equals 0 or 1, and is non-deterministic.

constants has to be chosen is such a way that the desired range of C is obtained.

```
Initialize TestSequence
Initialize BestCMF
FOR currentdepth from 1 to maximum depth DO
  Increase the depth of BestCMF to currentdepth
  Evaluate (BestCMF,TestSequence)
  PreviousCMF = BestCMF
  CurrentCMF = BestCMF
  Permute CurrentCMF
  FOR counter from 1 to iterationLength DO
    C=Ca*exp(-Cb*counter)
    Evaluate (CurrentCMF,TestSequence)
    BestCMF = Mix (BestCMF,CurrentCMF,0)
    CurrentCMF = Mix (PreviousCMF,CurrentCMF,C)
    PreviousCMF = CurrentCMF
    Permute CurrentCMF
  OD
OD
Output BestCMF
```

■

Some notes about the algorithm. Each CMF-tree contains the rules for the mappings represented in a tree along with the counters at depth currentdepth. When the Evaluate function is applied the counters are updated according to the mapping rules and the test sequence:

Algorithm 4.2 (Evaluate):
As input to Evaluate, the parameters TestSequence and CMF are given. The parameter TestSequence consists of both TestData and ContextData.

```
Initialize Counters
FOR i from 1 to |TestSequence| DO
  k = CMF (ContextData[i])
  Increase Counters[k] according to TestData[i]
OD
Output Counters
```

In the pseudo code |TestSequence| denotes the length of TestSequence. As output for the algorithm the counters in the leaf nodes are given. ■

The Mix function makes, for each couple of sibling leaves (with common father), a comparison on which mapping rules to keep according to the simulated annealing principle. The comparison determines the mapping

rule at the depth just below the depth where the leaves resides. When the parameter `C` is equal to 0 the new rule is only accepted if it is better (lower rate) than the old one. In the other case, i.e., `C`>0, the new rule is accepted if the difference in rate fulfills:

$$r(\text{new}) - r(\text{old}) \leq \text{random value in the range } [0, \texttt{C}), \tag{4.15}$$

where $r(\text{new}) = (l(a_0, b_0) + l(a_1, b_1))/(a_0 + b_0 + a_1 + b_1)$ and correspondingly for $r(\text{old})$. The counters a_0 and b_0 denote the observations in one of the leaf nodes and a_1 and b_1 the other leaf node. This implies that the new rule is always accepted if it is better than the old one.

The `Permute` function makes a permutation of each of the mapping rules at the depth `currentdepth`-1. In order to make this feasible to solve, we cannot have a complete function table in each node, e.g., if we have a total of 50 bits as input to the CMF we would have to store the function table of size 2^{50} bits in each node. As a simplification, we restrict all mapping rules to have only a *few* bits of input. This will allow us to make the function table of reasonable size. The `Permute` function will therefore either make a change in the function table or change one of the input variables.

By using the basic simulated annealing scheme as presented in Algorithm 4.1, it is possible to find a good CMF as will be shown in Section 4.4. However, the convergence of the algorithm will be shown in Section 4.4.1 to be slow or the improvement of the optimized CMF will be low. One reason for this is due to the fact that small changes (or permutations) in the CMF may not imply small changes on the resulting pseudo rate. This will be discussed more in Section 4.4.1.

One more thing to consider in Algorithm 4.1 is how to use it when we are going to optimize over a test data set, e.g., files from the *Calgary Corpus*. In order to find optimized CMF for multiple sources or a test data set and improve the convergence rate we propose the following algorithm:

Algorithm 4.3 (Optimizing CMF for multiple sources):
The setup is the same as in Algorithm 4.1 except that multiple test sequences are used, one for each source. As a consequence multiple CMF-trees with counters must be used.

```
FOR f from 1 to number of sources DO
  Initialize TestSequence[f]
OD
Initialize BestCMF
FOR currentdepth from 1 to maximum depth DO
  Increase the depth of BestCMF to currentdepth
  FOR f from 1 to number of sources DO
    CurrentCMF[f] = BestCMF
    Evaluate (CurrentCMF[f],TestSequence[f])
  OD
  BestCMF = InitCounters (CurrentCMF)
  PreviousCMF = BestCMF
  TmpCMF = BestCMF
  Permute TmpCMF
  FOR f from 1 to number of sources DO
    CurrentCMF[f] = TmpCMF
  OD

  WHILE continue condition ok DO
    FOR counter from 1 to iterationLength DO
      C=Ca*exp(-Cb*counter)
      FOR f from 1 to number of sources DO
        Evaluate (CurrentCMF[f],TestSequence[f])
      OD
      BestCMF = Mix (BestCMF,CurrentCMF,0)
      PreviousCMF[f] = Mix (PreviousCMF,CurrentCMF,C)

      TmpCMF = CurrentCMF[0]
      Permute TmpCMF
      FOR f from 1 to number of sources DO
        CurrentCMF[f] = TmpCMF
      OD
    OD
  OD

OD
Output BestCMF
```

The output of the algorithm is a CMF. ■

The differences between Algorithm 4.1 and Algorithm 4.3 are the following:
We use multiple test sequences in the second algorithm. Thus we must make a small change in the `Mix`-function. We calculate the rate according

to:

$$r(\text{new}) = \frac{\sum_f l(a_0[f], b_0[f]) + l(a_1[f], b_1[f])}{\sum_f a_0[f] + b_0[f] + a_1[f] + b_1[f]}, \tag{4.16}$$

and accept the new rule in the same way as in Equation 4.15.
Another change in from Algorithm 4.1 to Algorithm 4.3 is that we use several/many restarts on the C-value, e.g., we run the cooling scheme more than once. The reason for doing this is to avoid getting trapped in one of the many local minimum. It is easy to get into a local minimum since the change in rate is *too* big between neighboring permutations. On the line: "`WHILE continue condition ok DO`" we have an unspecified stop criteria. For example, in the experiments carried out on text and image files in the next section, we use consumed CPU-time as stop criteria.

4.4 Experimental Results

We use the source in Figure 4.2 to examine the behavior of Algorithm 4.1. In Table 4.3 the resulting MDL-trees are extracted (see Section 2.2.3) from the CTW algorithm when different lengths of the test sequences are applied.

Seq length	**Node**	**#0**	**#1**	$\hat{\theta}$
100	0	9	67	0.123
	1	22	2	0.900
1000	00	40	340	0.106
	01	71	301	0.192
	1	220	28	0.886
10000	00	218	2216	0.090
	01	1023	3998	0.204
	1	2310	235	0.908

Table 4.3: The extracted MDL-tree for various lengths of the test sequence.

From Table 4.3 conclude that our algorithm is able to find the expected CMF for this simple source if the test sequence is long enough (in this example somewhere between 100 and 1000). It should, however, once more, be pointed out that the CMF is dependent on the test sequence and thus we may always risk to run into some "bad luck".
We will in the sequel of this section investigate a more advanced tree source example and how to apply our suggested algorithms to text and image compression.

4.4.1 Tree Sources

To further study the performance of our suggested CMF optimization scheme in Algorithm 4.3 we will study a somewhat more complex situation where we have two binary sources.

Source 1: As context input the source uses the last 5 emitted symbols, $c_1, c_2, \ldots, c_5$. From the context the following is calculated[4]:

$$x_1 = c_1 \oplus c_3 \oplus c_4, \tag{4.17}$$

$$x_2 = c_1 \oplus c_5. \tag{4.18}$$

The x_1 and x_2 are used as context to a context tree source with depth 2. The parameters for the context tree source are:

x_1x_2	$P(0\|x_1x_2)$
00	0.70
01	1.00
10	0.15
11	0.00

Source 2: This source works in the same way as source 1 and use similar equations, i.e., $x_1 = y_2$ and $x_2 = y_3$. This source has depth 3:

$$y_1 = c_2 \oplus c_3, \tag{4.19}$$

$$y_2 = c_1 \oplus c_3 \oplus c_4, \tag{4.20}$$

$$y_3 = c_1 \oplus c_5. \tag{4.21}$$

$y_1y_2y_3$	$P(0\|y_1y_2y_3)$
000	0.90
001	0.65
010	0.20
011	0.55
100	1.00
101	0.05
110	0.00
111	0.85

The question is whether we obtain a common CMF for both these sources? The short answer is yes! But the follow up question is: how do we get there?

In the following we have used Algorithm 4.3 for optimizing the CMF. We will only make one run of the simulated annealing scheme, i.e., the

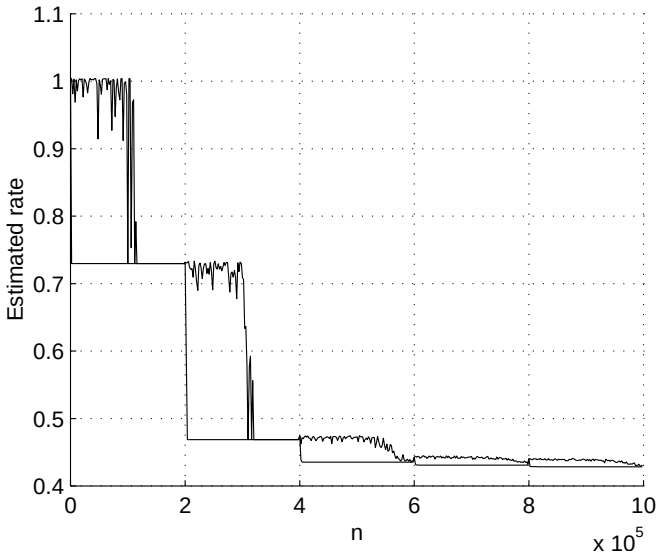

Figure 4.6: Estimated code rate vs simulation time. The lower line corresponds to `BestCMF` and the upper to `CurrentCMF`. The value of `iterationLength` is $2 \cdot 10^5$, thus the sharp steps are taken when the depth of the CTW and the CMF is increased.

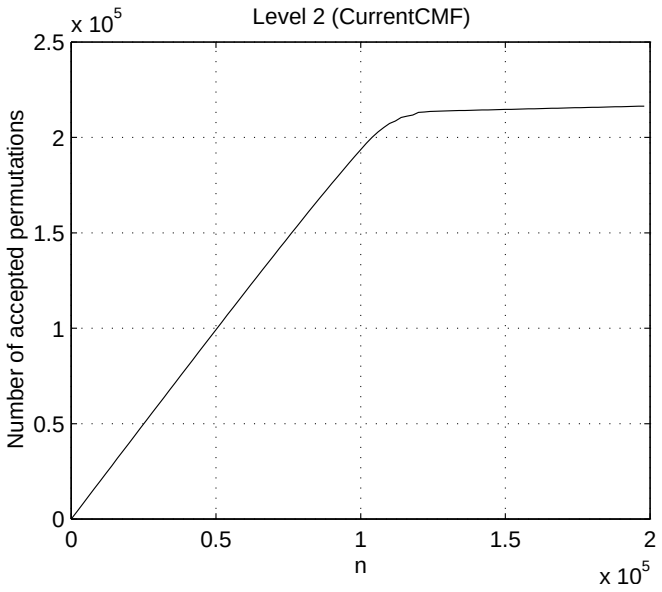

Figure 4.7: Number of accepted tries for `CurrentCMF` for level 2.

"continue condition" will only be valid once. The map rule input size is chosen to be 3 bits.

We start by observing the estimated code rate for the test data versus simulation time in Figure 4.6. We note that whenever the depth increases there is an instant response in the rate. We also note that approximately halfway on each depth there is a big change in the rate for the `CurrentCMF`. If we study this phenomenon further by looking at the acceptance rate for `CurrentCMF` in Figure 4.7 and for `BestCMF` in Figure 4.8 we note that after approximately halfway on that level only few updates are made on the `CurrentCMF`. But for `BestCMF` there is a big increase in the number of updates. The point for this "break" depends on the source and the choice of the parameters `Ca` and `Cb`[5].

What we conclude from Figure 4.7 and Figure 4.8 together with Figure 4.6 is that we should seek to use `C`-values that are in the "middle" range. In this region we have both a high acceptance rate for the `CurrentCMF` and `BestCMF`. The reason why such a low acceptance rate

[4] Logical "xor" or modulo-2 sum is denoted $\oplus$, i.e., $a \oplus b = (a' \wedge b) \vee (a \wedge b')$, where a' and b' are the (binary) inverse of a and b respectively.

[5] The parameters in this example are deliberately chosen in such a way that these figures should get the break point approximately halfway.

for `CurrentCMF` in the second part of the simulation interval is that every permutation makes a *too* big change in the rate and thus can't be accepted. This is like being trapped in a box, i.e., a local minimum. Therefore we will have to do something to be able to jump out of the box. In Algorithm 4.3 we do this by running through the entire range several/many times so that larger jumps may be accepted.
Although the relatively large number of acceptance in `BestCMF` in the later part of the simulation interval there is no big impact on the code rate. This is interpreted as the input variables do not change but changes in the function tables of map rules are made.

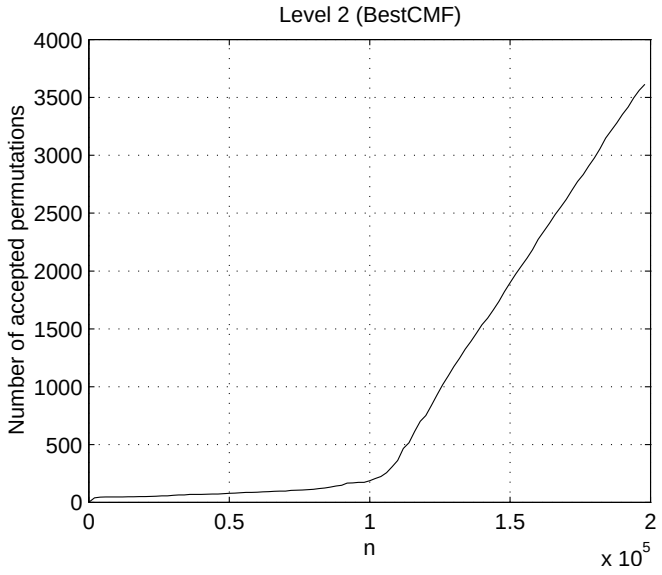

Figure 4.8: Number of accepted tries for `BestCMF` for level 2.

From Algorithm 4.3 we obtain an optimized CMF for these sources. The resulting map rules for this example turns out to be the following (presented as a Boolean expressions):

$$\hat{x}_1 = c_1 \oplus c_5, \tag{4.22}$$
$$\hat{x}_{2|0} = c_2 \oplus c_4 \oplus c_5, \tag{4.23}$$
$$\hat{x}_{2|1} = c_3 \oplus c_4 \oplus c_5, \tag{4.24}$$
$$\hat{x}_{3|00} = c_2 \oplus c_3, \tag{4.25}$$

$$\hat{x}_{3|01} = c_2 \oplus c_3, \tag{4.26}$$
$$\hat{x}_{3|10} = c_2 \oplus c_3, \tag{4.27}$$
$$\hat{x}_{3|11} = c_2 \oplus c_3. \tag{4.28}$$

We identify the functions from the source definitions although they do not appear in the same order as in the definition since we have optimized over two sources. Using the optimized CMF and the compression scheme from Figure 4.1 we compare the compression performance (code word length) for each of the sources in Figure 4.9 and Figure 4.10. We conclude that the optimized CMFs yields a results in this case that is comparable to using the "known" CMFs, i.e., the algorithm has been able to find a good approximation.

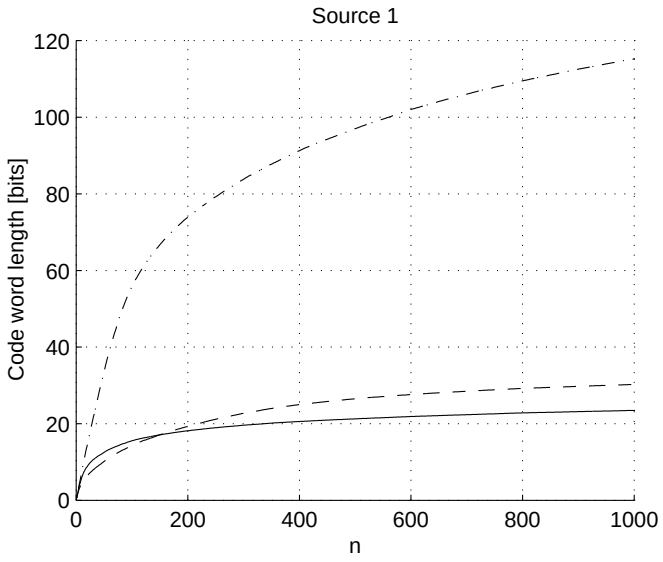

Figure 4.9: Code word length as a function of the number of observations for source 1. The solid line corresponds to using the known CMF from the source definition. The dashed line corresponds to the optimized CMF and the dash-dotted line to the raw context without a CMF.

4.4.2 Text Compression

The presented method in Algorithm 4.3 for optimizing the CMF has been tested on the well known *Calgary Corpus*. This test set consists of vari-

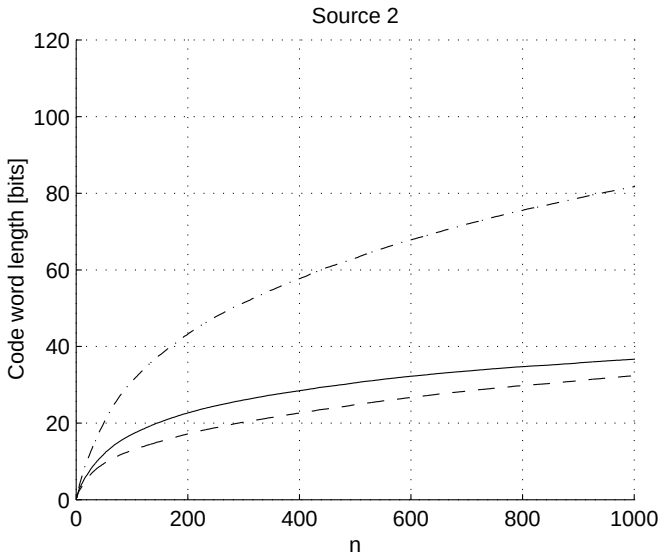

Figure 4.10: Code word length as a function of the number of observations as in Figure 4.9 but for source 2.

ous text and data files and has been used for many years for comparing data compression methods.
In Tables 4.4 and 4.5 we have optimized the CMFs on the entire set of 14 files. We have used 100000 characters (bytes) from each file for the simulated annealing scheme. Since the size of many of the files are less than 100000 bytes we have used resampling to collect data and context data from the files. Each of the 8 bitplanes uses its own CMF and own CTW. The maximum depth in the CTW trees is (only) 18. The input size of the mapping rules is 4 binary symbols taken from a 56 bit[6] context.
The possible gain of using optimized CMF is obvious compared to a "traditional" CMF. The average rate decreases from 2.953 to 2.498 bits per input character. The result is, however, not much better for the optimized CMF. We have also compared our results with a PPM scheme with a maximum depth of 5 from [Åbe99] (Table 7.10). We observe that the results of this PPM scheme is clearly better than our optimized CMF.

D	bib	book1	book2	geo	news	obj1	obj2	paper1
1	6.261	5.559	5.706	6.092	6.034	6.850	7.100	5.848
2	5.956	5.297	5.447	5.734	5.832	6.425	6.947	5.649
3	5.452	4.956	5.197	5.469	5.524	6.262	6.732	5.426
4	5.219	4.663	4.944	5.189	5.280	6.048	6.563	5.191
5	4.764	4.392	4.703	4.619	5.041	5.805	6.351	4.906
6	4.331	4.065	4.391	4.374	4.788	5.549	6.068	4.585
7	3.910	3.926	4.228	4.191	4.626	5.261	5.719	4.359
8	3.564	3.779	4.027	4.051	4.420	4.935	5.308	4.101
9	3.331	3.660	3.847	3.966	4.284	4.580	4.938	3.867
10	3.106	3.506	3.655	3.898	4.112	4.208	4.594	3.646
11	2.909	3.379	3.469	3.837	3.943	3.843	4.258	3.426
12	2.721	3.261	3.296	3.792	3.789	3.540	3.955	3.204
13	2.564	3.163	3.137	3.761	3.643	3.319	3.692	3.034
14	2.419	3.060	2.973	3.734	3.498	3.193	3.484	2.883
15	2.303	2.974	2.839	3.711	3.381	3.133	3.338	2.774
16	2.215	2.904	2.728	3.695	3.283	3.110	3.240	2.696
17	2.157	2.846	2.639	3.682	3.209	3.100	3.179	2.646
18	2.116	2.803	2.576	3.672	3.157	3.096	3.141	2.615
PPM	1.812	2.257	1.941	4.491	2.331	3.714	2.383	2.298

Table 4.4: Rate for the Calgary Corpus files when the CMF is optimized for all 14 files together. The first column, D, corresponds to the depth of the CTW.

We will further analyze why the improvement when using optimized

[6] Corresponding to 7 bytes.

D	paper2	pic	progc	progl	progp	trans	Avg	Basic
1	5.582	1.344	6.189	6.006	6.127	6.621	5.809	5.873
2	5.292	1.255	5.977	5.728	5.943	6.231	5.551	5.656
3	4.999	1.153	5.598	5.249	5.406	5.974	5.243	5.387
4	4.718	1.062	5.270	4.722	4.974	5.593	4.960	5.102
5	4.434	1.002	4.974	4.309	4.642	5.207	4.653	4.868
6	4.125	0.976	4.709	3.938	4.293	4.707	4.350	4.605
7	3.968	0.954	4.468	3.752	4.008	4.308	4.120	4.438
8	3.776	0.932	4.198	3.508	3.714	3.867	3.870	4.171
9	3.618	0.906	3.964	3.278	3.391	3.502	3.652	4.001
10	3.424	0.893	3.690	3.034	3.087	3.149	3.429	3.810
11	3.247	0.880	3.441	2.817	2.805	2.816	3.219	3.671
12	3.100	0.874	3.214	2.590	2.550	2.539	3.030	3.544
13	2.970	0.868	3.019	2.392	2.355	2.288	2.872	3.432
14	2.843	0.863	2.875	2.222	2.223	2.095	2.740	3.316
15	2.746	0.859	2.770	2.107	2.133	1.958	2.645	3.211
16	2.668	0.857	2.708	2.021	2.075	1.855	2.575	3.104
17	2.614	0.856	2.667	1.967	2.043	1.789	2.528	3.027
18	2.581	0.855	2.645	1.932	2.026	1.753	2.498	2.953
PPM	2.271	0.805	2.352	1.641	1.656	1.398	2.239	-

Table 4.5: Continuation of the previous table. The second last column, Avg, corresponds to the average rate over all 14 files. The last column, Basic, is the average result if we just use the plain context data without any optimized CMF.

CMF is not better. In Table 4.6 only the file paper1 is considered and the CMF is optimized only for that file. Since the size of the file is only 53161 bytes it will probably not make any sense to a use a CTW depth larger than, let say, 16-17, since the number of observations per node on maximum depth will be too small. The first observation we make is that the lowest rate obtained for paper1 in Table 4.6 is 1.798 and from Table 4.4 we got 2.615. The difference in rate of 0.635 bits/source symbol is substantial. The only explanation for this difference is that the files in Calgary Corpus are not similar enough to optimize the CMFs for all files together.

Another observation we make from Table 4.6 is that a map rule input size of 3 bits gives the best result. It would have been reasonable to an input size of 5 bits would give at least equal result since all possible functions with 3 input bits is just a subset of the functions with input size 5. The reason in this case why input size 4 and 5 gives worse result is that the same iteration length and stop criteria are used in all cases. But since the function space grows exponentially with the input size the larger map functions do not get enough time to converge in the same way as for example input size 3 in this case.

Depth	1	2	3	4	5	basic
1	5.726	5.525	5.471	5.437	5.555	5.895
2	5.415	5.117	5.097	5.015	5.215	5.729
3	5.127	4.727	4.718	4.702	4.846	5.495
4	4.808	4.454	4.415	4.399	4.510	5.274
5	4.461	4.189	4.111	4.128	4.219	5.098
6	4.177	3.885	3.807	3.834	3.899	4.840
7	3.889	3.592	3.496	3.493	3.564	4.722
8	3.600	3.311	3.177	3.163	3.238	4.440
9	3.322	3.033	2.866	2.837	2.891	4.261
10	3.042	2.773	2.597	2.547	2.561	4.023
11	2.800	2.540	2.360	2.303	2.315	3.870
12	2.593	2.346	2.168	2.119	2.124	3.737
13	2.443	2.190	2.027	1.983	2.011	3.625
14	2.337	2.073	1.924	1.899	1.933	3.492
15	2.267	1.998	1.862	1.844	1.887	3.380
16	2.229	1.941	1.819	1.817	1.870	3.254
17	2.208	1.910	1.798	1.801	1.861	3.170

Table 4.6: Comparison of the rate for paper1 with various depth on the CTW scheme and various input bit size of the mapping functions.

A final observation we make from these experiments is that when optimizing over all the files in Calgary Corpus we gain in rate for every increase in depth of the CTW model. Thus, we are not able to actually reduce the useful length of the meta-context to something as short as 17 bits. One might also argue whether it is reasonable to use the same CMF for all files since some of the files are text files, there is one image file, one file with numerical data etc. The result may be improved upon if we could use one CMF for text files, one for image files etc. However, there will be an additional cost of describing which CMF is actually used.

4.4.3 Image Compression

In Table 4.7 we have compared the compression performance for various medical images and the famous *lena* image with and without the usage of optimized CMF. The images are tested in four different ways: **Raw** = the original image and a basic CMF, **Raw+CMF** = the original image and an optimized CMF, **Pred** = the original image is transformed via a simple[7] prediction scheme and a basic CMF is used, **Pred+CMF** = the same as **Pred** but an optimized CMF is used. From the results we note

Image	Raw	Raw+CMF	Pred	Pred+CMF
lena	5.436	3.987	4.288	3.917
backen	2.543	1.269	1.620	1.317
buk	3.131	1.765	2.144	1.727
skalle	2.002	1.050	1.188	0.906
thorax	3.873	2.243	2.518	1.985
Avg	2.887	1.582	1.868	1.484

Table 4.7: Results in (calculated) code word rate when applying CMFs to the source coding scheme. All images are of size 512x512 with 8-bit gray levels. Except for the first image these are all medical images. The last line is the average rate for the medical images.

the beneficial effect the prediction has on the compression performance. The gain is, however, lower when the optimized CMFs are used.
It should be noted that these results are not comparable with results from other image compression schemes since the CMFs are optimized individually for each image. To be able to use this kind of construction for a practical application the CMF must be determined beforehand on

[7] Simple in this case means that we use an average of the 4 closest neighbors as a prediction.

a test image set. This is done in Table 4.8. For comparison we give the corresponding compression results on the test images for JPEG-LS and PNG in Table 4.9.

Depth	backen	buk	skalle	thorax	Avg
1	1.791	2.343	1.645	2.811	2.148
2	1.654	2.161	1.412	2.590	1.955
3	1.591	2.081	1.298	2.489	1.865
4	1.555	2.035	1.239	2.432	1.815
5	1.517	1.997	1.191	2.380	1.771
6	1.497	1.971	1.164	2.345	1.744
7	1.480	1.951	1.142	2.316	1.722
8	1.467	1.936	1.123	2.292	1.704
9	1.458	1.925	1.108	2.276	1.692
10	1.453	1.918	1.100	2.266	1.684
11	1.448	1.913	1.094	2.258	1.678
12	1.445	1.910	1.090	2.254	1.675
13	1.443	1.908	1.087	2.251	1.672
14	1.442	1.907	1.086	2.249	1.671
15	1.441	1.906	1.085	2.248	1.670
16	1.440	1.905	1.084	2.247	1.669

Table 4.8: The rate for the medical images when the CMF is optimized for all images together. After depth 8 the improvement is very small.

We note that, in contrast to the text compression case, the performance in Table 4.8 does not really improve after depth 8.
Another difference compared to the text compression case is that we in image compression are actually able to outperform the JPEG-LS scheme on these medical images.

4.5 Conclusions

The benefit of using optimized CMFs, as presented in this chapter, is that we can systematically search for good CMFs without having to rely on ad-hoc decisions when constructing a lossless data compression scheme. Another benefit we get for free is that we could use the CMF to shorten the length of the meta-context and thus reduce the computational complexity of the context modeling.
We conclude that it is possible to construct good CMFs off-line via simulated annealing. Our presented test results clearly show the great ad-

Image	JPEG-LS	PNG	Opt CMF
lena	4.237	4.611	-
backen	1.554	2.568	1.440
buk	2.028	3.343	1.905
skalle	1.358	2.645	1.084
thorax	2.418	4.139	2.247
Avg	1.840	3.174	1.669

Table 4.9: Compression rate per pixel for the test images with the JPEG-LS, and the PNG compression schemes. For comparison the results from Table 4.8 are given in the last column.

vantage of "optimized" CMF. We also conclude that the scheme presented in Figure 4.1 can be applied to text, image or any other data. The only thing that has to change is the CMF. It will also be possible to change the CMF for different kinds of languages etc. in text compression. However, the extra description cost for the chosen CMF and the extra memory requirements for the compression/decompression algorithms must be investigated.
As a side effect the presented results also show the danger of using certain test sets of images or "text corpuses" when comparing compression algorithms. By using optimized CMF we are able to build in a fair amount of a priori knowledge about the test set.

Chapter 5

Compression of the Request Sequences in ARQ Protocols

In this chapter we present simple yet efficient compression methods based on universal source coding ideas for the repeat request data in ARQ protocols. Beside the compression methods themselves, we give an in-depth motivation for the chosen encoder models. The choice of method is backed-up by simulations. This work was presented in [ERSS01].

5.1 Introduction

In this chapter we address the problem of compressing the repeat request data in ARQ protocols (ARQ = automatic repeat request). Such protocols are used in different types of data networks (see, e.g., [BG87]). A modern example of a radio network where ARQ is applied is the UMTS system as being standardized in the 3rd Generation Partnership Program (3GPP) [3rd99]. The UMTS system is a radio network that operates at high transmission rates and is targeted to be used in mobile communication. The ARQ protocol data we are considering here are the repeat request (binary) sequences which should be processed (compressed) independently. Due to lack of knowledge about the statistical properties of the data, most of the practical algorithms are *universal*, i.e., algorithms that can adapt to unknown parameters of a known (or even partially unknown) statistical model of the data. A larger length of the data sequence decreases the influence of the lack of information about the statistical

data properties, and vice versa. The problem of unknown statistics (and the universality of the algorithm) is very manifest since the lengths of the repeat request sequences are rather small.
In Section 5.2 we will describe some properties of ARQ and we detail the compression problem that will be studied. The problem of unknown statistics of the repeat request data is considered in Section 5.3. In Section 5.4 the compression algorithms are proposed and their performances are studied through simulations in Section 5.5.

5.2 Defining the problem

In many modern communication systems data of several sources is multiplexed into blocks that are transmitted. In the transmission process the transmitted blocks may get corrupted in spite of a feed-forward error-correction mechanism. Through the use of cyclic redundancy checks (CRC) the receiver can correctly detect, with high probability, the erroneous blocks and request for retransmission of the blocks in error. For our study of the sequence of repeat requests or just request sequences (Rs) we ignore all the involved details of such an automatic repeat request scheme. We let a 1 denote that a block has been successfully received and let a 0 denote a block in error or a block not received. Figure 5.1 illustrates the steps from which the Rs arise. If no errors occur on

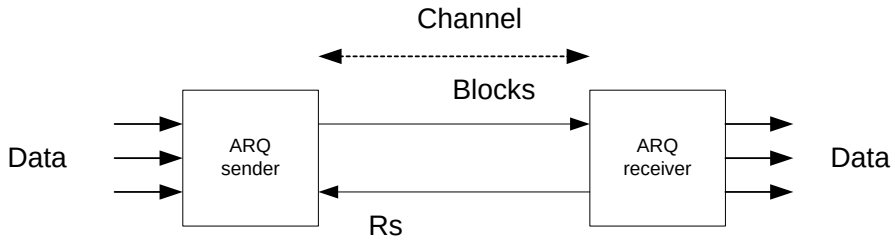

Figure 5.1: The origin of the repeat sequence (Rs) in a ARQ protocol.

the transmission channel then the Rs is a binary sequence $11\ldots100\ldots0$ of given length N. Any such sequence is described entirely by its length $n^* < N$ of run of ones, and it suffices to encode (describe) n^* by a uniform encoding (of length $\log N$) or a variable length prefix code. However, the advantages of variable length coding are not obvious (and have definitely no essential impact). Therefore we shall assume a uniform coding which gives a rate of $(\log N)/N$.

Correctly received blocks result in the substitution of zeros into ones in the Rs. When we receive more blocks we will thus get more ones in our Rs described by the binary sequence x^n. The position of the last one in x^n will therefore increase. However, the value of N does not change, i.e., the length of run of zeroes decreases. Therefore we now must describe the value of n **and** x^n (which we will refer to as the *bitmap*). The description should be as short as possible. This is the main task for our data compression scheme. The most efficient coding of x^n is based on known probabilities of all bitmaps of given length n. Unfortunately, these probabilities are neither known nor constant, i.e., they will vary with time. Sometimes the knowledge of a source model or of a rather narrow set of such models is used for increasing the coding efficiency. But for the considered problem this does not work: the statistical properties of bitmaps are defined by the statistical properties of the error sequence $e^n = e_1, \ldots, e_n$, $e_i \in \{0, 1\}$, and by the ARQ algorithm, and both these "components" are practically unknown and/or very complex to describe. In fact, error sequences are "generated" by many sources of different nature, and their mixture in the different proportions defines a very wide class of possible sources. But even for a small set of models (or just one model) we can not recalculate the statistics of error sequence into statistics of bitmaps; the complexity of any ARQ algorithm leaves us no chances of solving this problem. Thus the problem is formulated in the following way: we must encode (compress) *the output of the black box* (ARQ algorithm) *with unknown input signal* (error sequence).

5.3 The Statistical Model

From our data compression perspective we will consider the simplified model shown in Figure 5.2.

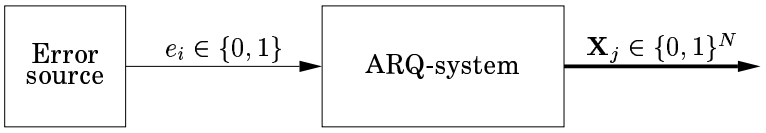

Figure 5.2: The system under consideration from the data compression perspective.

For this model we note the following important properties:

- The error source generates a continuous stream of binary symbols, e_i, $i = 0, 1, 2, \ldots$

- The ARQ-system generates bitmaps of binary symbols and with a predefined size, $\mathbf{X}_j$, $j = 0, 1, 2, \ldots$
- The amount of data on the input and output of the ARQ-system is not necessarily the same. Actually, the ARQ-system tries to keep the number of bitmaps as low as possible under the constraint that the communication system should work.
- The first position in the ARQ bitmap is always a zero, i.e., denoting a not received (or not correctly received) block. In order for this to be possible the ARQ-system also has a special mechanism to provide a description of the first position of the bitmap. This description is not considered in this work.
- There is a high correlation between bitmaps but since there are errors on the channel sending the bitmaps we will be unable to use this correlation.

Since our aim is to compress the bitmaps generated by the ARQ-system we must know what kind of properties we can use in this data. We may know the properties of the communication system and thus the properties of the error source. The commonly used models for the error source are based on *independent* errors and *burst* errors respectively (e.g. see [BG87]). A reasonable assumption is that the independent errors will introduce isolated zeros and burst errors will generate runs of zeros in the ARQ bitmap. These properties will be confirmed in the studies in Section 5.5.

5.4 Some Compression Algorithms

The traditional methods (we again refer to [BG87]) of describing the bitmaps of the ARQ-system are based on the two models of block error, i.e., independent and burst.
Describing (many) independent errors in a bitmap can not be expected to be done efficiently, no matter what compression scheme we use. It is the same as compressing data from a binary memoryless source with an unknown parameter. Based on the results in [WST95] we know "exactly" what we can achieve with a Dirichlet estimator and an arithmetic encoder. For this reason the "conventional" method has been to just describe the position of the last one in the bitmap and then a raw description of the bitmap. This is an appealing method from a complexity perspective. We will in the sequel denote this as the *bitmap*-description (BM).

When we consider the case of having burst errors we will (most likely) get runs of ones and zeros in the bitmap (for independent errors we may, however, also get runs of ones followed by a single zero). The "traditional" method in this case is to use a *list*-description. This method describes, in a list, the positions where the bitmap changes from a run of one symbol to a run of the other symbol. This could also be done by just describing the length of each run in the bitmap. The codeword for each such description could either be described by a fixed number of bits or by using arithmetic encoding where the likelihood is uniformly distributed over the possible range of values. Using the simple method of a fixed precision numbers for the run lengths we get a low complexity method. Using arithmetic encoding is somewhat more complex but will on the other hand yield a significantly better result if we have *many* runs in the bitmap. This is due to the fact that we will pay an extra (unnecessary) cost for each description since we will reserve code word space for events that will not occur. In the sequel when referring to the list description we will use the method based on arithmetic encoding.
In an actual mobile communication systems we will, however, have a mixture of independent errors and burst errors. A *natural* extension to the bitmap and list descriptions would in this case be to spend one extra bit describing which method is used (bitmap or list) and then use that method. In fact, this method is actually the one which is used in the UMTS standard. The goal would be to use the method resulting in the shortest description. Note that for the bitmap representation we have no actual compression except for the description of the last position where we have a one in the bitmap.
There are two other description methods worth considering based on the principles above:

- List/bitmap-mixture: it is possible to describe parts of the bitmap with the list description and other parts, where the list description works poorly, with a bitmap description. Such a approach will, however, result in a very computationally complex algorithm since we have to find a good (or optimal) partitioning of the bitmap.

- Universal coding of the runs: based on the list description we may estimate the probability for each length of run. In the list description previously defined we used a uniform description for all lengths. The problem with this approach is the number of unknown parameters. If we have k possible lengths of the runs we have $k-1$ unknown probabilities (parameters). From previous chapters we know that the number of unknown parameters are of major importance for the compression performance. In this case k will be

of the same order as the maximum length of the bitmap and thus making this an intractable approach if we cannot reduce the number of unknown parameters in some way. For example, if we have prior knowledge about the lengths being exponentially distributed we may use results from Section 2.5 and Chapter 3.

Getting back to the main issue we must thus construct a compression scheme that will be able to handle a mixture of independent and burst errors with a reasonable number of unknown parameters. How do we then distinguish independent from burst errors? In Figure 5.3 we have two similar Markov models.

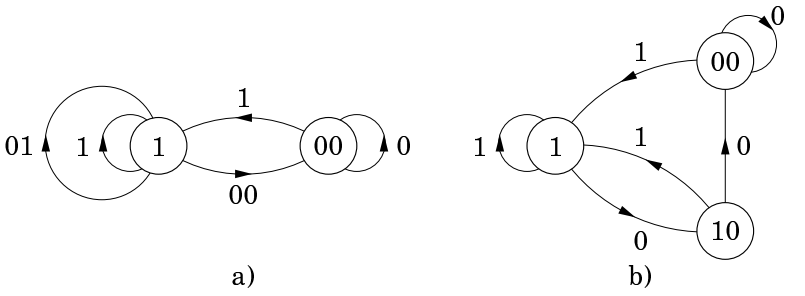

Figure 5.3: Two Markov models with 3 unknown parameters. In the left model an alphabet extension is made to take care of the two observation of either two zeros or a zero and a one. In the right model we use a conventional Markov model.

If we start by considering Figure 5.3a we have the two states in the model where each state corresponds to "the last symbol", i.e., erroneous or correctly received block. If the last symbol was one we have two possibilities, the next symbol will also be a one (e.g. we are in a burst of correctly received blocks) or we will enter some runs of zeros. In this case we distinguish between an independent error and stay in the one state or we have a burst error and enter the 00-state. If we enter the "burst error state", i.e., 00, we stay there until we get a one, i.e., end of burst. Figure 5.3b describes essentially the same thing. When we are in the one state and get a zero we enter the "error" state 10. Based on the next symbol we know if it was an independent error or a burst error and change state accordingly to 1 or 00, respectively. The model in Figure 5.3b is straight forward to convert to a tree model (compare for example with the results of FSMX sources by Rissanen [Ris83]).

Based on these Markov models and their corresponding tree models we will investigate the compression performance using fixed tree models and compare the results with the CTW algorithm.

5.5 Experiments

The basis for our tests are different models of the bit errors occurring on the transmission channel. We have used both our own idealized models and also data obtained from a UMTS system under different test conditions.
The error models that we will consider are based on independent errors and burst errors. These models will help us determine some important characteristics of the ARQ-system and they are also often the major concern in a mobile communication system.
From the bit errors we have produced the corresponding block errors and used these in a "simple" selective repeat ARQ system.
Since the output from the ARQ system differs "very much" according to the retransmission strategy used, we make the actual bitmaps available on the web [Eks00]. This allows others to apply their compression algorithms and compare their results with ours.

5.5.1 Verifying the Statistical Properties

We will compare the statistical properties of the input and the output data of the ARQ-system, i.e., the error sequence e_i and the bitmaps $\mathbf{X}_j$ in Figure 5.2. What we are interested in is whether the ARQ-system (the "black box") preserves the burst of runs from the input data to the bitmaps as previously assumed. Therefore, we will use test error sequences with a wide range of different lengths of the bursts. Those burst models we will consider are generated by the following three sources:

IND independent block errors with probability P_e.

MB "medium burst" block errors. The average block error probability is P_e. The lengths of the runs are distributed almost uniformly in $\{1, 2, \ldots, 10\}$. See remark below.

LB "large burst" block errors. The only difference compared to MB is the lengths of runs: $\{1, 2, \ldots, 100\}$.

A remark about the burst errors. Since we generate the errors on the bit level which will be grouped together to blocks, the probability for block errors will not be uniformly over the burst lengths $\{1, 2, \ldots, M\}$, $M =$

$\{10, 100\}$. It holds, approximately, that $P(1) + P(M+1) = P(i) = 1/M$, $i = 2, 3, \ldots, M$.

We have generated 2500 bitmaps, each of length 1024. The probabilities for the different lengths of bursts, m, are given in Table 5.1 (IND-model) and Table 5.2 (MB-model). A corresponding table for the LB-model does not fit in this thesis since it would occupy too many pages.

In the tables for the IND-model and MB-model we have used four different values on the block error probability P_e. We compare the probability distribution on the input and the output of the ARQ-system.

	Independent errors							
m	1%		5%		10%		15%	
1	98.8	89.5	94.1	92.3	89.3	89.1	84.8	86.1
2	1.1	6.0	5.5	6.2	9.5	9.2	12.9	11.4
3	0.0	2.9	0.3	1.0	1.0	1.3	2.0	2.0
4	0.0	1.2	0.0	0.3	0.1	0.3	0.3	0.4
5	0.0	0.3	0.0	0.1	0.0	0.1	0.1	0.1
6	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
11	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
13	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
14	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 5.1: The results of ARQ testing for the IN models. The probability distributions are given in percent.

A subjective conclusion from Table 5.2 and Table 5.1 (and from corresponding results with LB) is that the ARQ-system conserves the burst distribution from the input error sequence to the bitmaps since the probability distribution seems similar in these tables. Furthermore, the average length of the actually used bitmap, n, increases when P_e increases for the IND errors. We will not discuss this behavior of the ARQ-system in this thesis but it is a subject of interest when optimizing the ARQ-system to use bitmaps that are as few and as short[1] as possible.

[1] Short here refers to the used size of the bitmaps n.

	Medium burst							
m	1%		5%		10%		15%	
1	5.7	6.7	5.2	7.2	4.9	8.5	4.8	10.1
2	10.5	22.5	10.3	16.3	10.1	15.5	9.6	15.1
3	9.3	15.7	9.5	13.0	9.7	12.5	9.6	12.5
4	9.8	10.5	9.7	10.3	9.8	10.5	9.2	10.2
5	9.9	7.7	9.3	8.4	9.7	9.2	9.9	9.4
6	10.5	7.3	10.2	8.6	9.8	8.6	10.0	8.7
7	10.1	7.0	9.9	8.0	9.8	8.0	9.9	7.9
8	9.8	6.4	9.6	7.8	9.9	7.7	9.8	7.3
9	10.2	6.7	10.0	7.9	9.8	7.4	9.8	6.9
10	9.2	6.2	10.4	8.1	9.8	7.2	10.1	6.9
11	4.8	3.0	5.1	4.0	5.3	3.8	5.0	3.4
12	0.1	0.1	0.2	0.1	0.3	0.2	0.4	0.3
13	0.0	0.0	0.1	0.1	0.2	0.2	0.4	0.3
14	0.0	0.0	0.1	0.1	0.3	0.2	0.4	0.2

Table 5.2: The results of ARQ testing for the MB model. The probability distributions are given in percent.

5.5.2 Compression efficiency

The same simulation procedure and the same error source models, as in Section 5.2, were used for the estimation of the efficiency of the compression algorithms. We have once more used 2500 bitmaps of the length $N = 1024$. The results of the various compression schemes are presented in the Table 5.3.

The first and the second columns describe the error source model and the value of P_e (in percent) respectively. The next 8 columns contain the values of average length of codeword for one Rs (including $\log N = 10$ bits for the description of variable value of n) for 8 different compression algorithms.

BM is bitmap coding with codeword length $\approx \log N + (\bar{n} - 2)$, where $\bar{n}$ is the average value of n over 2500 bitmaps. When constructing the bitmap code words the first and the last symbol in the bitmap are known, i.e., the first is always 0 and the last (pointed out by n) is always 1 (except when $n = 0$).

"List" is a basic method of describing runs of zeroes and ones. The method just points out those positions in the bitmap where changes from a run of one symbol to the other occurs. Each such point requires $\log b$

	e	BM	List	MC0	MC1	MC2	CT3	CW2	CW3
IND	1	316.5	98.8	**43.0**	44.6	46.9	45.3	44.5	45.4
IND	5	521.6	385.7	**136.8**	138.6	141.6	139.9	138.3	139.2
IND	10	626.3	604.5	**207.8**	209.3	212.5	210.9	209.1	209.9
IND	15	680.4	767.2	**260.5**	261.3	263.8	263.0	261.1	261.8
MB	1	174.8	33.6	48.2	**24.0**	25.3	24.5	24.9	25.2
MB	5	333.6	76.5	112.8	**50.2**	52.6	51.1	52.2	52.8
MB	10	463.0	139.4	190.5	**87.0**	89.5	87.8	89.0	89.5
MB	15	544.3	192.2	250.1	**116.9**	119.3	117.6	118.7	119.3
LB	1	36.2	14.1	27.4	13.5	13.7	13.6	**12.9**	13.0
LB	5	102.0	19.8	67.2	**17.7**	18.4	18.0	17.8	18.1
LB	10	187.4	28.3	121.5	**24.3**	25.6	25.0	25.2	25.8
LB	15	262.8	37.5	171.5	**31.5**	33.4	32.5	33.0	33.8
Mix	-	471.4	168.2	195.8	96.1	95.2	**93.5**	95.1	96.0
tc1	10	349.4	54.9	170.1	**41.4**	42.4	42.5	42.7	43.3
tc2	10	289.3	48.4	157.3	**36.9**	38.1	37.9	38.2	38.8
tc3	10	294.3	48.2	156.3	**36.8**	38.1	37.8	38.2	39.0
tc4	10	337.9	53.0	168.1	**40.0**	41.1	41.1	41.3	42.0

Table 5.3: Some results of the compression efficiency.

	e	BM	List	MC0	MC1	MC2	CT3	CW2	CW3
IND	1	99.0	40.2	**14.3**	14.6	15.0	14.7	14.5	14.5
IND	5	167.2	91.9	**29.1**	29.2	29.4	29.4	29.2	29.2
IND	10	176.2	172.2	**53.1**	**53.1**	53.3	53.3	**53.1**	**53.1**
IND	15	178.1	236.0	71.3	71.0	**70.9**	71.2	**70.9**	**70.9**
MB	1	155.2	20.4	36.6	**12.4**	13.3	12.8	13.6	13.8
MB	5	163.1	34.5	51.4	**20.7**	21.2	20.9	21.1	21.2
MB	10	216.3	52.9	71.1	30.6	30.8	**30.5**	30.6	30.7
MB	15	232.5	69.1	89.3	**39.2**	39.4	**39.2**	39.3	39.3
LB	1	79.2	6.4	51.4	**4.7**	5.3	5.0	5.6	6.0
LB	5	155.2	12.3	94.6	**9.4**	10.4	9.9	10.8	11.3
LB	10	201.0	17.4	123.7	**13.7**	15.0	14.4	15.2	15.6
LB	15	234.3	22.4	142.8	**17.9**	19.1	18.6	19.1	19.4
Mix	-	213.5	92.0	112.4	45.5	43.3	**43.2**	43.6	43.7
tc1	10	197.2	26.6	57.0	16.1	14.5	15.9	14.7	**14.3**
tc2	10	192.7	27.5	80.5	17.5	16.5	17.5	16.7	**16.4**
tc3	10	195.0	25.7	73.3	16.5	**15.6**	16.4	15.7	**15.6**
tc4	10	183.3	28.3	57.9	16.8	14.9	16.6	15.1	**14.5**

Table 5.4: The (unbiased) estimate of the standard deviation of the compressed size in Table 5.3 for the different compression methods.

bits to describe[2], where b is the number of possible remaining positions.
The MC0, MC1 and MC2 are Markov chain encoders of depth 0, 1 and 2 respectively (MC0 corresponds to a memoryless model with 1 state), CT3 is the Context Tree Markov model (see Figure 5.3b), CW2 and CW3 are Context Tree Weighting algorithms for maximal memory depth 2 and 3 respectively (see [WST95]). Thus, any row of Table 5.3 contains the average lengths of Rs description for all chosen compression methods with given error source model and value of P_e.
It was clear beforehand that MC0 is the best algorithm for the IND model, but it is much worse for the MB and the LB models than the other algorithms. Therefore it should not be used in practice.
The last 5 algorithms has a similar compression performance. MC1 and MC2 are slightly better and slightly worse, respectively, than the other ones. In particular, MC2 is worse than MC1, CT3 and CW2 for *any* error model (except Mix) and for *any* value of P_e since the replacing of state "1" in MC1 and CT3 by states "01" and "11" in MC2 gives us no advantages but increases the number (and the "cost") of unknown parameters.
Since the efficiency of the proposed algorithms almost coincides, we should pay attention to other properties of the algorithms. It seems almost obvious that the *flexibility*, i.e., ability to adapt to unpredictable situations, is the most important property after the efficiency. For example, CTW *weights* all CT models with restricted memory depth, i.e., takes into account all CT models *simultaneously*. Therefore, it is not surprising, that CW2 and CW3 give almost optimal results for all error models and all values of P_e. It also illustrates that the best encoders for different conditions are *different*.
The algorithm "List" is the most efficient one for describing runs of large length. But even in this case it is slightly worse than some of the previously discussed algorithms. Furthermore, the important advantages of the proposed algorithms consist of the absence of any preliminary analysis of bitmaps and in a *uniform* coding process.
Thus we suppose that CT3 or CW2 is the best choice. The number of unknown parameters for the CT3 model is, of course, three. For CW2 we may say that it is variable with 1, 2, 3 or 4 parameters. Considering the computational complexity and storage requirement there are, however, 3 and 7 parameters respectively.
The used error models match rather well with the above considered testing problem. In spite that the error models cover a wide range of possible situations, the *mixture* of different error sequences should be considered too. The row "Mix" of Table 5.3 contains the results for such a mixture of IND, MB and LB errors. The probabilities for each of these errors

[2] If we use an arithmetic encoder.

changes throughout the data sequence in a way that is similar to a "real world" situation of mobile communication.
Finally, we report the results from 4 test-cases (tc1-tc4) which were derived from simulations of a mobile communication system. These 4 cases should correspond to "real world" situations and are defined by the 3GPP. We note that the results are fairly good for the list compression and conclude that this data contains very few independent errors compared to the number of errors generated by burst errors.
The results confirm our conclusion; CT3 and CW2 give the best results along with MC2 if we consider both compression efficiency and robustness against "independent" errors. As a measure of robustness we consider the standard deviation, s, for the code words, i.e.,

$$s = \sqrt{\frac{1}{n-1}\left(\sum_{i=1} nL_i^2 - \frac{1}{n}\left(\sum_{i=1} nL_i\right)^2\right)}, \tag{5.1}$$

where L_i, $i = 1, 2, \cdots, n$, denote the length of the code words. In Table 5.4 we present the standard deviation for the different algorithms.

5.6 Conclusions

To conclude, in this chapter we have studied the problem of compressing the representation of the repeat request sequences in ARQ protocols. By carefully limiting our scope and by carefully studying data we formulated encoder models, which were then the basis for a study of several compression methods which we have evaluated. Our method based on the CT3 model gives a better performance than existing, more or less ad-hoc compression methods that uses "list" type methods. Yet, the CT3 method has roughly the same computational complexity as these existing ones and can be implemented in only about 1 to 1.5 A4 pages of C code. Hence the method is both simple and performs very well.
Compared to the mixed list/BM scheme the CT3 method requires no analysis step of the bitmap as it can encode the bitmap directly, symbol by symbol.
Finally, following the procedure used in our work it is straightforward to address similar communication settings where feedback from the receiver to the sender is used to realize efficient data transmission.

Bibliography

[3rd99] 3rd Generation Partnership Project. Technical specification group radio access network; RLC protocol specification TS 25.322. Technical report, Release 1999.

[AK89] Emilie Aarts and Jan Korst. *Simulated annealing and Boltzmann machines: a stochastic approach to combinatorial optimization and neural computing*. Chichester Wiley Corp., 1989.

[BCW90] T. Bell, J. Cleary, and I. Witten. *Text Compression*. Prentice Hall, 1990.

[Åbe99] Jan Åberg. *A Universal Source Coding Perspective on PPM*. PhD thesis, Lund University, 1999.

[BG87] Dimitri Bertsekas and Robert Gallager. *Data networks*. Prentice-Hall, 1987.

[BG98] Toby Berger and Jerry Gibson. Lossy source coding. *IEEE Transaction on Information Theory*, 44(6):2693–2723, October 1998.

[CT91] Thomas Cover and Joy Thomas. *Elements of Information Theory*. John Wiley & Sons Inc., 1991.

[Eks95] Nicklas Ekstrand. Lossless compression of medical images. Technical report, Department of Information Theory, 1995. Available through the author.

[Eks96] Nicklas Ekstrand. Lossless compression of grayscale images via context tree weighting. In J. Storer and M. Cohn, editors, *Proceedings Data Compression Conference (DCC)*, pages 132–139. IEEE, April 1996.

[Eks98] Nicklas Ekstrand. *Some Results on Lossless Compression of Grayscale Images*. Teknisk Licentiat Thesis, Lund University Sweden, 1998.

[Eks00] Nicklas Ekstrand. Compression of request sequences in ARQ protocols. WWW: http://www.ekstrand.org/ARQ, 2000.

[ERSS01] Nicklas Ekstrand, Béla Rathonyi, Yuri Shtarkov, and Ben Smeets. The qualitative modeling and compression of the request sequences in ARQ protocols. In J. Storer and M. Cohn, editors, *Proceedings Data Compression Conference (DCC)*. IEEE, Mars 2001. Too appear.

[ES96] Nicklas Ekstrand and Ben Smeets. On the estimation and model costs in lossless universal image data compression by context weighting. In Peter Linde and Gunnar Sparr, editors, *Proceedings Symposium on Image Analysis*, pages 54–58. Swedish Soc. for Automated Image Analysis, March 7-8 1996.

[ES98a] Nicklas Ekstrand and Ben Smeets. Notes on the P-context algorithm. In *Proceedings International Symposium on Information Theory (ISIT) 1998*, page 543. IEEE, April 1998.

[ES98b] Nicklas Ekstrand and Ben Smeets. Weighting of double exponetial distributed data in lossless image compression. In J. Storer and M. Cohn, editors, *Proceedings Data Compression Conference (DCC)*, page 543. IEEE, 1998.

[ES00a] Nicklas Ekstrand and Ben Smeets. Some notes on the context mapping function in lossless data compression. In J. Storer and M. Cohn, editors, *Proceedings Data Compression Conference (DCC)*, page 553. IEEE, April 2000.

[ES00b] Nicklas Ekstrand and Ben Smeets. A technique for prediction and probability assignment (PPA) in lossless data compression. In *Proceedings International Symposium on Information Theory (ISIT) 2000*, page 72. IEEE, June 2000.

[FM92] Meir Feder and Neri Merhav. Universal prediction of individual sequences. *IEEE Transactions on Information Theory*, 38(4):1258–1270, July 1992.

[FM94] Meir Feder and Neri Merhav. Relations between entropy and error probability. *IEEE Transactions on Information Theory*, 40(1):259–266, January 1994.

[FS98] Meir Feder and Andrew Singer. Universal data compression and linear prediction. In J. Storer and M. Cohn, editors, *Proceedings Data Compression Conference (DCC)*, pages 511–520. IEEE, 1998.

[How93] Paul Glor Howard. *The Design and Analysis of Efficient Lossless Data Compression Systems*. PhD thesis, Brown University, Rhode Island, USA, 1993.

[ISO98] ISO/IEC JTC1/SC29 WG1 (JPEG/JBIG). FCD 14495, Lossless and near-lossless coding of continuous tone still images (JPEG-lS), 1998.

[Jon81] Christopher Jones. An efficient coding system for long source sequences. *IEEE Transaction on Information Theory*, 27(3):280–291, May 1981.

[KT81] Raphail Krichevsky and Victor Trofimov. The performance of universal encoding. *IEEE Transaction on Information Theory*, 27(2):199–207, March 1981.

[Leh91] E. L. Lehmann. *Theory of point estimation*. Wadsworth & Brooks/Cole, 1991.

[Mat96] Mathematical Society of Japan. *Encyclopedic Dictionary of Mathematics*. The MIT Press, second edition, 1996.

[MF96] Bo Martins and Søren Forchhammer. Bi-level image compression with tree coding. In J. Storer and M. Cohn, editors, *Proceedings Data Compression Conference (DCC)*, pages 270–279. IEEE, April 1996.

[MF98] Neri Merhav and Meir Feder. Universal prediction. *IEEE Transaction on Information Theory*, 44(6):2124–2147, October 1998.

[MNS00] Nasir Memon, David Neuhoff, and Sunil Shende. An analysis of some common scanning techniques for lossless image coding. *IEEE Transactions on Image Processing*, 9(11):1837–1848, November 2000.

[MRS94] Nasir Memon, Sibabrata Ray, and Khalid Sayood. Differential lossless encoding of images using non-linear predictive techniques. In *Proceedings ICIP-94*, pages 841–845, 1994.

[MS95] N. Memon and K. Sayood. Lossless image compression: A comparative study. In *Proceedings-of-the-SPIE*, pages 8–20, 1995.

[MSW96] Neri Merhav, Gadiel Seroussi, and Marcelo Weinberger. Lossless compression for sources with two-sided geometric distributions. 1996. Parts of this paper was presented at the 1996 International Conference on Image Processing, Lausanne, Switzerland, and at the 1997 International Symposium on Information Theory, Ulm, Germany.

[Noh94] Ragnar Nohre. *Some Topics in Descriptive Complexity*. PhD thesis, Linköping University, Sweden, 1994.

[Pas76] R. C. Pasco. *Source Coding Algorithms for Fast Data Compression*. PhD thesis, Stanford University, CA, USA, 1976.

[PM93] W. Pennebaker and J. Mitchell. *JPEG, still image data compression*. Van Nostrand Reinhold, 1993.

[PR94] Joseph Provine and Rangaraj Rangayyan. Lossless compression of peanoscanned images. *Journal of Electric Imaging*, 3(2):176–181, April 1994.

[Ris78] Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, September 1978.

[Ris83] Jorma Rissanen. A universal data compression system. *IEEE Transactions on Information Theory*, 29(5):656–664, September 1983.

[Ris84] Jorma Rissanen. Universal coding, information, prediction, and estimation. *IEEE Transactions on Information Theory*, 30(4):629–636, July 1984.

[RL81] Jorma Rissanen and Glen Langdon. Universal modeling and coding. *IEEE Transaction on Information Theory*, 27(1):12–23, January 1981.

[RM89] Rissanen and Mohiuddin. A multiplication-free multialphabet arithmetic code. *IEEE Transaction on Communications*, 32(4):93–98, July 1989.

[Sha48] Claude Elwood Shannon. A mathematical theory of communication. *Bell System Technical Journal*, pages 379–423; 623–656, July and October 1948.

[Sht87] Yuri Shtarkov. Universal sequential coding of single messages. *Problems of Information Transmission (english translation)*, July-Sept 1987.

[Sht98] Yuri Shtarkov. Private communications, 1998.

[Sme96] Ben Smeets. *An Introduction to Source Coding*. Department of Information Technology, March 1996. Lecture Notes.

[STW95] Y. Shtarkov, T. Tjalkens, and F. Willems. Multi-alphabet universal coding of memoryless sources. *Problems of Information Transmission*, 31(2):20–35, 1995.

[Suz95] Joe Suzuki. On some relationship between the context tree weighting and general model weighting techniques for tree sources. 1995. This paper was partially presented at the Fourth Japan-Benelux Coding and Information Theroy Workshop, Eindhoven, Netherland, 1994.

[TJR85] S. Todd, G.G. Langdon Jr., and J. Rissanen. Parameter reduction and context selection for compression of gray scale images. *IBM Jl. Res. Develop*, 29:188–193, March 1985.

[Vol94] Paul Volf. Deriving MDL-decision trees using the context maximizing algorithm. Master's thesis, Eindhoven University of Technology, 1994.

[VW95] Paul Volf and Frans Willems. On the context tree maximizing algorithm. September 1995.

[Wil97] Frans Willems. The context-tree weighting project. WWW: http://ei1.ei.ele.tue.nl/~frans/ResearchCTW.html, 1997.

[WLZ92] Marcelo Weinberger, Abraham Lempel, and Jacob Ziv. A sequential algorithm for the universal coding of finite memory sources. *IEEE Transaction on Information Theory*, 38(3):1002–1014, May 1992.

[Wor96] World Wide Web Consortium. PNG (Portable Networks Graphics) Specification. WWW: http://www.w3.org/, 1996.

[WRA96] Marcelo J. Weinberger, Jorma J. Rissanen, and Ronald B. Arps. Applications of universal context modeling to lossless compression of gray-scale images. *IEEE Transactions on Image Processing*, 5(4):575–586, April 1996.

[WS96] Frans Willems and Yuri Shtarkov. Context weighting for general finite-context sources. *IEEE Transaction on Information Theory*, 42(5):1514–1520, September 1996.

[WS97] Marcelo J. Weinberger and Gadiel Seroussi. Sequential prediction and ranking in universal context modeling and data compression. *IEEE Transaction on Information Theory*, 43(5):1697–1706, September 1997.

[WSS96] M. Weinberger, G. Seroussi, and G. Sapiro. LOCO-I: a low complexity, context-based, lossless image compression algorithm. In J. Storer and M. Cohn, editors, *Proceedings Data Compression Conference (DCC)*, pages 140–149. IEEE, April 1996.

[WSS00] Marcelo Weinberger, Gadiel Seroussi, and Guillermo Sapiro. The LOCO-I lossless image compression algorithm: Principles and standardization into JPEG-LS. *IEEE Transactions on Image Processing*, 9(8):1309–1324, August 2000.

[WST93] Frans M.J. Willems, Yuri M. Shtarkov, and Tjalling J. Tjalkens. Context weighting: General finite context sources. In *Proceedings 14th Symposium on Information Theory in the Benelux*, May 17 & 18 1993. Veldhoven, The Netherlands.

[WST95] Frans Willems, Yuri Shtarkov, and Tjalling Tjalkens. The context-tree weighting method: Basic properties. *IEEE Transaction on Information Theory*, 41(3):653–664, May 1995.

[Wu96] Xiaolin Wu. An algorithmic study on lossless image compression. In J. Storer and M. Cohn, editors, *Proceedings Data Compression Conference (DCC)*, pages 150–159. IEEE, April 1996.

Congratulations, you have reached the end of this thesis. I hope you have enjoyed reading it! /Nicklas